Universidade Católica do Salvador
Bacharelado em Engenharia de Software

Denilson Xavier Oliveira, Diego Andrade Deiró, João Victor Aziz Lima de Santana, Loren Vitória Cavalcante Santos, Neillane de Carvalho.

Janus: Uma Arquitetura de Middleware para Detecção e Anonimização de PII em Grandes Modelos de Linguagem no Contexto da LGPD

Salvador
2025

Denilson Xavier Oliveira, Diego Andrade Deiró, João Victor Aziz Lima de Santana, Loren Vitória Cavalcante Santos, Neillane de Carvalho.

Janus: Uma Arquitetura de Middleware para Detecção e Anonimização de PII em Grandes Modelos de Linguagem no Contexto da LGPD

Trabalho de Conclusão de Curso apresentado à Universidade Católica do Salvador como parte dos requisitos necessários para a obtenção do Título de Engenheiro de Software.
Orientador: Prof. Me. Mario Jorge Pereira

Universidade Católica do Salvador

Salvador
2025

Denilson Xavier Oliveira, Diego Andrade Deiró, João Victor Aziz Lima de Santana, Loren Vitória Cavalcante Santos, Neillane de Carvalho.

Janus: Uma Arquitetura de Middleware para Detecção e Anonimização de PII em Grandes Modelos de Linguagem no Contexto da LGPD

Trabalho de Conclusão de Curso apresentado à Universidade Católica do Salvador como requisito parcial para a obtenção do título de Engenheiro de Software.

Salvador, 16 de janeiro de 2026

Banca Examinadora:

Prof. Me. Mario Jorge Pereira
Universidade Católica do Salvador
Orientador

Prof. Me. Semiramis Ribeiro de Assis
Universidade Católica do Salvador

Prof. Me. Flavio Dusse
Universidade Católica do Salvador

Dedicamos este trabalho a todos que acreditaram no nosso potencial e que, de forma direta ou indireta, contribuíram para o nosso crescimento acadêmico e pessoal ao longo desta jornada.

Agradecimentos

Agradecemos primeiramente ao coordenador do curso, Osvaldo Requião, cuja dedicação, e gestão foram essenciais para garantir uma formação de qualidade junto com um ambiente acadêmico acolhedor. Agradecemos de forma especial ao orientador da equipe, Mario Jorge Pereira, pela paciência, pelas orientações e pelo conhecimento compartilhado durante todo o desenvolvimento deste Trabalho de Conclusão de Curso. Estendemos nossos agradecimentos a todos os professores que contribuíram para a trajetória acadêmica, transmitindo conhecimento, valores e experiências que levaremos para a vida profissional.

"Argumentar que você não se importa com o direito à privacidade porque não tem nada a esconder não é diferente de dizer que você não se importa com a liberdade de expressão porque não tem nada a dizer."

Edward Snowden

Resumo

A integração de Grandes Modelos de Linguagem (LLMs) em ambientes corporativos oferece ganhos significativos de produtividade, mas introduz riscos críticos à privacidade e à conformidade com a Lei Geral de Proteção de Dados (LGPD), especialmente no manuseio de Informações Pessoais Identificáveis (PII). Este trabalho propõe e valida a arquitetura Janus, um *middleware* de segurança projetado para interceptar, anonimizar e restaurar dados sensíveis em interações com provedores de IA externos. A solução adota uma estratégia de "Defesa em Profundidade", orquestrando três camadas de filtragem: determinística (Expressões Regulares), probabilística (Reconhecimento de Entidades Nomeadas - NER) e semântica (LLM Local Llama 3). A metodologia experimental envolveu testes de estresse progressivos, culminando na análise de 500 *prompts* do domínio de Recursos Humanos (RH), totalizando mais de 4.500 entidades processadas. Os resultados demonstraram que o sistema atinge uma estabilidade operacional com um *F1-Score* global próximo a 0.60 em cenários de alta complexidade. A análise evidenciou que, embora o filtro NER apresente desafios de precisão, a arquitetura prioriza a segurança (*Recall*), mitigando o risco de vazamento de dados. Conclui-se que o sistema Janus oferece uma solução viável de "Privacidade por Design", permitindo o uso seguro de IA Generativa sem comprometer o sigilo das informações corporativas.

Palavras-Chave: LGPD. Large Language Models. Privacidade de Dados. Mascaramento de PII. Arquitetura de Software.

Abstract

The integration of Large Language Models (LLMs) into corporate environments offers significant productivity gains but introduces critical risks to privacy and compliance with the General Data Protection Law (LGPD), particularly regarding the handling of Personally Identifiable Information (PII). This work proposes and validates the Janus architecture, a security middleware designed to intercept, anonymize, and restore sensitive data in interactions with external AI providers. The solution adopts a "Defense in Depth" strategy, orchestrating three filtering layers: deterministic (Regular Expressions), probabilistic (Named Entity Recognition - NER), and semantic (Local LLM Llama 3). The experimental methodology involved progressive stress testing, culminating in the analysis of 500 prompts from the Human Resources (HR) domain, totaling over 4,500 processed entities. Results demonstrated that the system achieves operational stability with a global F1-Score near 0.60 in high-complexity scenarios. The analysis highlighted that, although the NER filter presents precision challenges, the architecture prioritizes security (Recall), effectively mitigating data leakage risks. It is concluded that the Janus system offers a viable "Privacy by Design" solution, enabling the secure use of Generative AI without compromising corporate information confidentiality.

Keywords: LGPD. Large Language Models. Data Privacy. PII Masking. Software Architecture.

Lista de figuras

Lista de tabelas

Sumário

1 Introdução

O crescimento exponencial dos Grandes Modelos de Linguagem (LLMs) tem evidenciado sua rápida e crescente adoção no ambiente corporativo no intuito de otimização de tarefas nos mais diversos setores presentes dentro de uma empresa, seja no passo a passo de um desenvolvimento de um software específico, com explicações intuitivas e códigos, ou na gestão e administração automatizando a criação e sumarização de relatórios. No entanto, a integração de LLMs em ambientes corporativos promove desafios críticos de privacidade. A submissão de dados via *prompts* permite que provedores acessem informações sensíveis e segredos industriais.

Segundo Chong et al. (CHONG et al., 2024), a arquitetura de "caixa-preta"(*black-box*) desses serviços impede que os usuários tenham visibilidade ou controle sobre o processamento e armazenamento de suas informações. Essa insegurança é corroborada pela análise de diversas políticas de privacidade, que preveem o compartilhamento de dados de usuários com terceiros para fins comerciais.

Tal prática eleva substancialmente o risco de vazamento de Informações Pessoais Identificáveis (PII), evidenciando o conflito existente entre a utilidade tecnológica e a segurança da informação.

1.1 Aplicabilidade e Motivação

Esta pesquisa surge da necessidade de alinhar a eficiência das *Large Language Models* (LLMs) com as exigências de privacidade e governança de dados no ambiente corporativo brasileiro. O foco principal é a conformidade com a Lei Geral de Proteção de Dados (LGPD) – Lei nº 13.709/2018 (Brasil. Senado Federal, 2024). Essa legislação define regras obrigatórias para o tratamento de dados pessoais em organizações públicas e privadas, exigindo garantias de segurança, transparência e controle por parte dos usuários. Portanto, a adoção de ferramentas de IA nas empresas não deve buscar apenas produtividade, mas também respeitar rigorosamente as normas de proteção de dados atuais.

Para validar a proposta na prática, escolheu-se o setor de Recursos Humanos (RH) como cenário de aplicação, devido ao grande volume de dados sensíveis manipulados diariamente. Nesse contexto, o uso de assistentes de IA para automatizar consultas salariais e cadastrais traz riscos, pois essas interações envolvem o processamento direto de Informações Pessoais Identificáveis (PII). Entre os dados expostos nessas operações estão o nome completo, CPF, endereço, telefone e salário, tornando essencial a implementação de camadas de segurança que impeçam o envio dessas

informações para os provedores do modelo externo.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem como objetivo desenvolver um *middleware* para interceptar e anonimizar PII em consultas a LLMs, assegurando proteção de dados e conformidade com a LGPD.

1.2.2 Objetivos Específicos

Para alcançar o objetivo geral, foram definidas as seguintes etapas de desenvolvimento:

- Análise de Requisitos e Riscos: Identificar as categorias de dados pessoais (PII) comuns em sistemas de RH e analisar os riscos de vazamento ao enviá-los para modelos de linguagem. Com base nessa análise, definir os requisitos de segurança e as regras da LGPD necessárias para a criação dos filtros de proteção e do módulo de proxy;
- Definição da Arquitetura: Projetar a arquitetura de um *proxy* local que funcione como uma camada intermediária de segurança entre o usuário e a API da IA externa, garantindo que o sistema continue responsivo e fácil de usar;
- Implementação da Sanitização: Desenvolver os mecanismos de tratamento de texto combinando expressões regulares, reconhecimento de entidades (NER) e uma LLM local. O objetivo é detectar, classificar e mascarar dados sensíveis nos *prompts* antes que eles sejam enviados para a internet;
- Validação Experimental: Realizar testes utilizando conjuntos de dados (*datasets*) com exemplos típicos do setor de Recursos Humanos, avaliando a precisão da detecção dos filtros e garantindo que os dados originais foram mascarados corretamente.

2 Fundamentação Teórica

2.1 A Era dos LLMs e os Desafios de Privacidade

Atualmente, os *Large Language Models* (LLMs) representam um avanço significativo na Inteligência Artificial, tornando-se amplamente acessíveis por meio de interfaces desenvolvidas por grandes provedores, como o Gemini (Google), Copilot (Microsoft) e ChatGPT (OpenAI). O setor corporativo tem passado por um intenso processo de integração dessas ferramentas para aprimorar processos internos. Essa adoção é sustentada pela evolução contínua em aprendizado de máquina, redes neurais e, especificamente, na arquitetura de Transformadores (*Transformers*), que viabilizou o processamento de linguagem natural em larga escala e alta performance (IBM, 2025).

A ascensão dos LLMs transformou o cenário empresarial, sendo aplicados em tarefas críticas como sumarização de documentos, tradução técnica e assistência na escrita de código (SICHMAN, 2021). Contudo, o uso predominante de modelos baseados em nuvem (*SaaS*) introduz sérios riscos de privacidade. Os *prompts* enviados pelos colaboradores podem conter involuntariamente segredos industriais ou informações sensíveis, expondo a organização ao vazamento de dados ao transitarem por APIs proprietárias de terceiros (CARPENTIER et al., 2025).

O principal desafio reside na natureza desses sistemas, onde os usuários não possuem controle sobre como os dados são processados e armazenados, podendo gerar desconfiança. As políticas de privacidade dos provedores frequentemente confirmam a coleta de dados para fins comerciais, o pode levar ao vazamento de Informações Pessoais Identificáveis (PII) e criar um conflito entre a utilidade da tecnologia e a segurança dos dados (CHONG et al., 2024).

2.2 Abordagens para Proteção de Dados em LLMs

A crescente preocupação com a privacidade em LLMs impulsionou o desenvolvimento de mecanismos de segurança defensiva, abrangendo desde técnicas de ofuscação textual até a implementação de arquiteturas robustas de interceptação de dados.

Um desafio central na sanitização é o equilíbrio (*trade-off*) entre privacidade e utilidade: a alteração excessiva de um *prompt* pode degradar o contexto semântico, resultando em respostas imprecisas. Carpentier et al. (CARPENTIER et al., 2025) destacam que é difícil para o usuário prever a eficácia de um *prompt* sanitizado, o que pode levar a ineficiências operacionais. Buscando mitigar esse problema, o tra-

balho de Roy Chowdhury et al. (CHOWDHURY et al., 2025) aborda a categorização seletiva de dados sensíveis. Os autores aplicam técnicas distintas conforme o tipo de dado: Criptografia com Preservação de Formato (FPE) para identificadores rígidos (como CPF) e Privacidade Diferencial Métrica (mDP) para atributos numéricos (idade e salário), visando preservar a qualidade da inferência.

Para garantir que informações críticas não deixem o perímetro controlado da organização, arquiteturas de filtragem executadas no lado do cliente (*client-side*) demonstram-se eficazes. O sistema Casper, proposto por (CHONG et al., 2024), estabelece uma referência fundamental neste domínio. A solução propõe uma arquitetura de sanitização em três camadas complementares: (1) filtros baseados em regras (Regex); (2) Reconhecimento de Entidades Nomeadas (NER); e (3) identificação de tópicos sensíveis via LLM local. Essa abordagem multicamadas foi projetada para operar com baixa latência, funcionando como uma extensão de navegador independente de processamento em nuvem.

2.3 Conceitos Fundamentais

- *Large Language Models* (LLM): Modelos de Inteligência Artificial baseados em aprendizado profundo (*Deep Learning*), predominantemente construídos sobre a arquitetura *Transformer*. Esta arquitetura introduziu o mecanismo de autoatenção (*self-attention*) para processar eficientemente dados sequenciais (SICHMAN, 2021; PORTELA, 2025). Treinados em vastos *corpora* de texto (bilhões de parâmetros extraídos da web e livros), os LLMs aprendem padrões linguísticos complexos, capacitando-os a realizar tarefas de Processamento de Linguagem Natural (PLN) como geração de código, tradução e análise de sentimento. O ciclo de vida desses modelos envolve o pré-treinamento não supervisionado (previsão de tokens) e, frequentemente, o refinamento via Aprendizado por Reforço com Feedback Humano (RLHF) para alinhamento das respostas (PORTELA, 2025).

- Informações Pessoais Identificáveis (PII): Referem-se a qualquer dado relacionado a uma pessoa natural identificada ou identificável. O conceito abrange informações que, isoladamente ou combinadas, permitem a individualização de um sujeito (GARZA et al., 2025; PATEL, 2023). No cenário brasileiro, classificam-se em identificadores diretos (ex: Nome Completo, CPF, RG, e-mail) e identificadores indiretos ou sensíveis (ex: dados biométricos, endereço IP, geolocalização e informações financeiras). A proteção desses dados é o foco central dos mecanismos de sanitização abordados neste trabalho.

- Lei Geral de Proteção de Dados (LGPD): Legislação federal (Lei nº 13.709/2018) que regula o ciclo de vida do tratamento de dados pessoais no Brasil, inspi-

rada no GDPR europeu (ARANTES, 2022). A norma impõe aos agentes de tratamento (Controlador e Operador) a obediência a dez princípios norteadores, sendo os mais críticos para sistemas de IA: a finalidade (propósitos legítimos), a necessidade (minimização de dados) e a segurança (medidas técnicas de proteção). Além disso, a lei institui a Autoridade Nacional de Proteção de Dados (ANPD) e garante direitos aos titulares. Destaca-se o Artigo 20, que assegura o direito à revisão de decisões tomadas unicamente com base em tratamento automatizado, exigindo explicabilidade e transparência em sistemas algorítmicos.

2.4 Reconhecimento de Entidades Nomeadas (NER)

O Reconhecimento de Entidades Nomeadas (*Named Entity Recognition* - NER) constitui uma subtarefa central no Processamento de Linguagem Natural (PLN) e na extração de informações (ALBUQUERQUE et al., 2023). O objetivo fundamental desta técnica é localizar e classificar menções a entidades predefinidas em textos não estruturados. As categorias mais comuns incluem nomes de pessoas (PER), organizações (ORG), localizações (LOC), datas (DATE) e valores monetários (MONEY), variando conforme o domínio de aplicação (ALBUQUERQUE et al., 2023; DIAS et al., 2020).

Historicamente, as estratégias para NER evoluíram de sistemas baseados em regras e léxicos para modelos probabilísticos e neurais. As abordagens iniciais dependiam de dicionários manuais e padrões gramaticais rígidos. Embora úteis em domínios restritos, esses sistemas apresentavam dificuldades com a ambiguidade natural da linguagem e exigiam manutenção custosa (ALBUQUERQUE et al., 2023). Com o avanço do aprendizado de máquina, surgiram modelos estatísticos como Máquinas de Vetores de Suporte (SVM) e Campos Aleatórios Condicionais (CRF), que aprendem padrões a partir de dados anotados. Mais recentemente, o estado da arte foi redefinido por arquiteturas de *Deep Learning*, como redes recorrentes (Bi-LSTM-CRF) e modelos baseados em *Transformers* (como o BERT). Essas arquiteturas modernas capturam contextos complexos e dependências de longo prazo, resultando em maior precisão na classificação (ALBUQUERQUE et al., 2023; DIAS et al., 2020).

No âmbito da segurança da informação, o NER é uma ferramenta indispensável para a detecção automatizada de Informações Pessoais Identificáveis (PII) (DIAS et al., 2020; FERETZAKIS; VERYKIOS, 2024). Ao identificar padrões como CPFs, nomes completos, endereços e dados de saúde em *prompts* ou documentos, os sistemas de segurança podem aplicar mecanismos de mascaramento ou anonimização antes que os dados deixem o ambiente corporativo. Essa capacidade é essencial para prevenir a exfiltração de dados sensíveis para APIs de terceiros (CHONG et al., 2024; FERETZAKIS; VERYKIOS, 2024).

A aplicação de NER para a língua portuguesa, contudo, ainda enfrenta desafios

específicos quando comparada ao inglês (ALBUQUERQUE et al., 2023). O principal obstáculo é a escassez relativa de recursos linguísticos públicos, como *corpora* anotados em larga escala e modelos pré-treinados para variantes do idioma (brasileiro e europeu). Além disso, a complexidade morfológica e a flexibilidade sintática do português impõem dificuldades adicionais aos modelos (ALBUQUERQUE et al., 2023). Apesar disso, pesquisas recentes, como as de Dias et al. (DIAS et al., 2020), demonstram resultados promissores no uso de abordagens híbridas para a extração de dados sensíveis, indicando um caminho viável para a construção de filtros de privacidade robustos no contexto nacional.

2.4.1 Corpora e Desafios de Avaliação em Português

A eficácia de modelos de Reconhecimento de Entidades Nomeadas (NER) é intrinsecamente dependente da qualidade e representatividade dos dados utilizados para treinamento e avaliação. Para a língua portuguesa, o *Golden Standard* histórico é o corpus HAREM (Primeiro e Segundo HAREM), desenvolvido especificamente para avaliar sistemas de extração de informação em textos lusófonos (SANTOS; CARDOSO, 2007).

Embora o HAREM seja fundamental para a literatura, ele apresenta limitações quando aplicado ao contexto de assistentes de IA corporativos:

1. **Domínio Textual:** O corpus é composto majoritariamente por textos jornalísticos, sites e documentos públicos, cuja estrutura sintática difere substancialmente de *prompts* conversacionais curtos ou comandos de sistemas de RH;
2. **Escassez de PII Sensível:** Devido a restrições éticas, corpora públicos raramente contêm alta densidade de dados sensíveis reais (como salários associados a CPFs válidos), dificultando testes de estresse de segurança.

Para superar a escassez de dados anotados em domínios específicos, a literatura recente tem validado o uso de *Dados Sintéticos* gerados por Grandes Modelos de Linguagem (LLMs). Estudos demonstram que datasets gerados por modelos como GPT-4, quando devidamente curados, preservam as propriedades estatísticas da linguagem natural e permitem a criação de cenários de teste controlados e seguros, sem violar a privacidade de indivíduos reais.

2.5 Padrões Arquiteturais para Processamento de Dados

A escolha de uma arquitetura de software adequada é determinante para a manutenibilidade e escalabilidade de sistemas que lidam com fluxo contínuo de dados.

No contexto de processamento de linguagem natural e sanitização de informações, o padrão arquitetural *Pipe and Filter* (Duto e Filtro) destaca-se pela sua capacidade de decompor tarefas complexas em uma série de etapas de processamento independentes e sequenciais (SHAW; GARLAN, 1996).

Segundo Buschmann et al. (BUSCHMANN et al., 1996), este padrão estrutura o sistema em componentes funcionais denominados "Filtros", que são conectados por canais de comunicação chamados "Pipes". Cada filtro possui uma responsabilidade única: recebe os dados na entrada, executa uma transformação local (como normalização, extração ou substituição) e entrega o resultado para o próximo componente da cadeia.

As principais características que justificam a adoção deste padrão em sistemas de segurança de dados incluem:

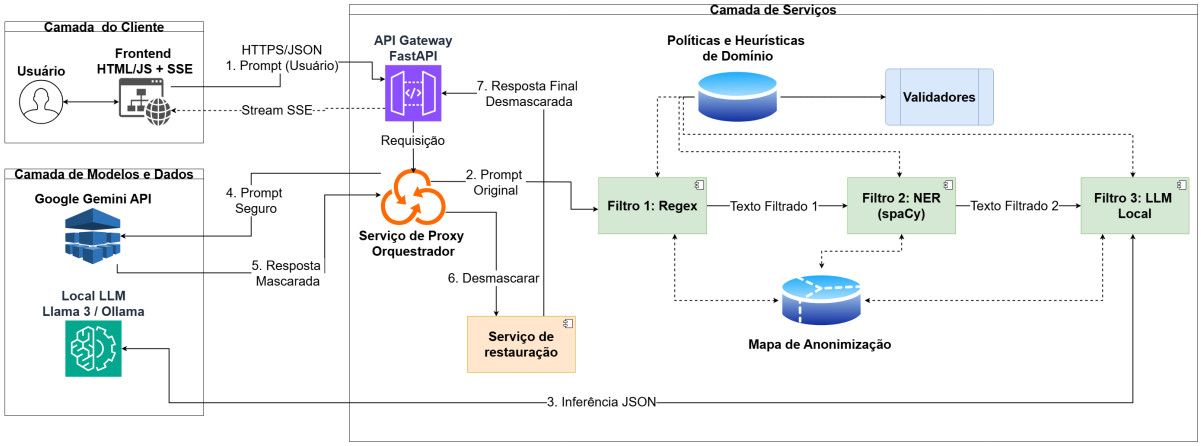
- **Desacoplamento:** Os filtros são independentes e desconhecem a existência dos demais. Um módulo de Regex, por exemplo, não precisa saber se o próximo passo é uma inferência de rede neural ou uma gravação em banco de dados;
- **Reusabilidade:** Componentes individuais podem ser reutilizados em diferentes contextos ou pipelines;
- **Flexibilidade:** A ordem de execução pode ser alterada e novos filtros podem ser inseridos (ex: adicionar um novo validador de CPF) sem a necessidade de refatorar o núcleo do sistema.

Esta abordagem é amplamente utilizada em compiladores e sistemas de processamento de texto, sendo ideal para a implementação de camadas de defesa em profundidade, onde cada filtro atua como uma barreira de segurança incremental.

2.6 Arquitetura de Proxy de Sanitização

A Arquitetura de Proxy de Sanitização configura-se como um padrão arquitetural de segurança projetado para atuar como intermediário entre o cliente (usuário ou aplicação) e um serviço de inferência externo (API de LLM). Seu objetivo primordial é implementar uma camada de prevenção contra vazamento de dados (DLP) (CHONG et al., 2024). Neste modelo, o componente *Janus* intercepta o tráfego de rede, inspecionando e modificando os pacotes de dados para remover, mascarar ou anonimizar informações sensíveis antes que estas deixem o perímetro de controle da organização.

Figura 1 – Diagrama da arquitetura do sistema Janus.



Fonte: Elaboração própria (2025).

No fluxo de interação com LLMs baseadas em nuvem, o funcionamento do Janus segue as etapas descritas a seguir:

1. Interceptação da Requisição: No momento em que o usuário submete um *prompt*, o proxy intercepta a chamada HTTP/HTTPS antes que ela trafegue para a internet, mantendo os dados ainda dentro do ambiente seguro (local ou corporativo) (CHONG et al., 2024).

2. Análise e Sanitização: O sistema aplica filtros de segurança (baseados em Expressões Regulares, NER ou classificadores de tópicos) para escanear o conteúdo textual. Ao identificar Informações Pessoais Identificáveis (PII), o proxy realiza a substituição desses dados por marcadores genéricos ou *placeholders* (ex: substituir um CPF real por `[CPF_1]`), garantindo a preservação da estrutura semântica do pedido sem expor o dado real (CHONG et al., 2024).

3. Encaminhamento à API Externa: O *prompt* sanitizado — agora livre de dados sensíveis — é encaminhado para o servidor da LLM (como OpenAI ou Google) para processamento e geração da resposta.

4. Interceptação da Resposta: O retorno gerado pela IA é capturado novamente pelo proxy antes de ser exibido ao usuário final.

5. Reconstrução de Contexto (RestorationService): O sistema implementa uma etapa mandatória de "re-hidratação"dos dados através do serviço de restauração. Durante a sanitização, os dados originais são armazenados em um estado temporário e substituídos por *placeholders* indexados sequencialmente (ex: `[CPF_1]`, `[NOME_2]`). Após o processamento pela LLM externa, o serviço intercepta a resposta e realiza a substituição inversa, mapeando cada marcador de

volta ao seu valor original. Isso assegura que o usuário final receba uma resposta íntegra e contextualizada, sem perceber a intervenção de segurança.

6. Entrega ao Cliente: O resultado final processado é exibido na interface do usuário.

2.7 Trabalhos Relacionados e Análise Comparativa Arquitetural

Embora a estratégia de filtragem em três camadas (Regex, NER e LLM) utilizada neste trabalho compartilhe semelhanças metodológicas com o sistema *Casper* proposto por Chong et al., as topologias de implementação divergem fundamentalmente. Esta seção estabelece um comparativo crítico entre a abordagem distribuída (*Client-Side*) e a abordagem centralizada (*middleware*) adotada pelo sistema Janus.

2.7.1 Abordagem Distribuída (Extensão de Navegador)

O sistema *Casper* opera como uma extensão de navegador, executando todo o processo de sanitização diretamente na máquina do usuário final antes que a requisição HTTP seja formada.

- Vantagem (Privacidade Máxima): O modelo de ameaça assume que a rede interna pode não ser confiável. Como a sanitização ocorre no dispositivo (*endpoint*), o dado sensível jamais trafega pela rede, nem mesmo pela Intranet corporativa.
- Desvantagem (Limitação de Hardware): A execução de modelos de IA locais (como LLMs para detecção de contexto) consome recursos significativos de CPU e memória RAM. Delegar essa carga para o dispositivo do usuário pode degradar a experiência de navegação ou inviabilizar o uso em dispositivos móveis e computadores corporativos com hardware modesto.
- Desafio de Governança: Em um cenário corporativo com milhares de colaboradores, a gestão de atualizações e a garantia de que a extensão está ativa e não foi desabilitada pelo usuário representam um desafio logístico de TI.

2.7.2 Abordagem Centralizada (*Middleware* Corporativo)

A arquitetura Janus, proposta no presente trabalho, centraliza a lógica de segurança em um servidor *proxy* dentro do perímetro seguro da organização.

- Vantagem (Soberania e Auditoria): A centralização facilita a conformidade com a LGPD, permitindo auditorias unificadas e a aplicação de políticas de segurança globais instantâneas. Além disso, a solução é agnóstica ao dispositivo, protegendo requisições vindas de navegadores, aplicativos móveis ou sistemas legados.
- Vantagem (Offloading Computacional): Ao mover o processamento para um servidor dedicado com hardware robusto (GPU), é possível executar modelos de detecção mais complexos e precisos (como o Llama 3 8B) sem impactar a performance da estação de trabalho do usuário.
- Trade-off de Privacidade: Diferente da abordagem *Client-Side*, o dado original transita não-anonimizado do computador do usuário até o servidor do proxy. No entanto, como esse tráfego ocorre dentro da rede privada (Intranet/VPN) da empresa, o risco é considerado controlado e aceitável em troca dos benefícios de governança.

A Tabela 1 sintetiza as diferenças estruturais entre as duas abordagens.

Tabela 1 – Comparativo Arquitetural: Casper vs. Janus

Característica	**Casper (Client-Side)**	**Janus (Middleware)**
Local de Execução	Dispositivo do Usuário	Servidor Corporativo
Custo Computacional	Alto para o Cliente	Baixo para o Cliente
Gestão de Atualização	Descentralizada (Complexa)	Centralizada (Simples)
Dependência de Hardware	Limitada pelo Endpoint	Escalável no Servidor
Perímetro de Dados	Nunca sai do dispositivo	Trafega na Intranet

Fonte: Elaboração própria (2025).

A principal vantagem dessa arquitetura reside na garantia de que dados confidenciais jamais transitem para servidores de terceiros, mitigando riscos de armazenamento indevido ou uso dos dados para retreinamento de modelos (CHONG et al., 2024). A implementação dessa lógica no lado do cliente (*client-side*), como em extensões de navegador, oferece o nível máximo de privacidade, pois a sanitização ocorre no próprio dispositivo do usuário, eliminando a dependência de servidores intermediários externos. O sistema Casper exemplifica essa abordagem, executando filtros de regras e modelos de NER localmente para proteger a interação com *chatbots web* (CHONG et al., 2024).

2.8 Métricas de Avaliação em Segurança da Informação

A avaliação de sistemas de detecção e classificação, como o proposto neste trabalho, baseia-se tradicionalmente em métricas de Recuperação de Informação (IR) derivadas da Matriz de Confusão. Conforme definido por Baeza-Yates e Ribeiro-Neto (BAEZA-YATES; RIBEIRO-NETO, 2011), as métricas fundamentais são:

- **Precisão (Precision):** Representa a fração de instâncias recuperadas que são relevantes. No contexto de anonimização, indica o quanto o sistema evita marcar textos inofensivos como sensíveis (evita Falsos Positivos);
- **Revocação (Recall):** Representa a fração de instâncias relevantes que foram efetivamente recuperadas. Indica a capacidade do sistema de encontrar todos os dados sensíveis presentes no texto (evita Falsos Negativos);
- **F1-Score:** A média harmônica entre Precisão e Revocação, utilizada para buscar um equilíbrio entre as duas métricas.

2.8.1 O Trade-off Privacidade-Utilidade

Embora o equilíbrio (F1-Score) seja desejável em tarefas genéricas de NLP, em sistemas de Prevenção contra Perda de Dados (DLP - *Data Loss Prevention*), existe uma assimetria crítica no custo do erro.

A falha em detectar um dado sensível (Falso Negativo) resulta em vazamento de informação, violação da LGPD e potenciais sanções legais. Em contrapartida, a classificação errônea de um termo comum como sensível (Falso Positivo) resulta apenas em uma ocultação desnecessária (*Over-masking*), o que pode degradar levemente a legibilidade, mas mantém a segurança íntegra. Portanto, a literatura de segurança preconiza que arquiteturas de privacidade devem priorizar a maximização da Revocação *Recall*, tolerando taxas mais baixas de Precisão em favor da garantia de não-vazamento (MANNING; RAGHAVAN; SCHÜTZE, 2008).

3 Metodologia

A metodologia adotada para o desenvolvimento e avaliação do *middleware* Janus envolve a construção de um protótipo baseado no padrão arquitetural *Pipe and Filter*, seguida de uma validação quantitativa com *datasets* curados do domínio de Recursos Humanos. A eficácia da solução é mensurada através de métricas de Precisão, Revocação e F1-Score, visando comprovar a viabilidade da anonimização *on-premise* em conformidade com a LGPD.

3.1 Enquadramento Metodológico

Quanto à sua natureza, esta pesquisa classifica-se como aplicada, pois tem como objetivo gerar uma solução tecnológica concreta para um problema prático: a incompatibilidade entre o uso de LLMs públicas e os requisitos de privacidade corporativa. O conhecimento gerado não se restringe à teoria, sendo materializado na implementação de uma arquitetura funcional.

Em relação aos objetivos, o estudo apresenta uma abordagem híbrida:

- Exploratória: Devido à necessidade de investigar o problema em técnicas de sanitização e compreender os riscos emergentes de privacidade em IA Generativa, uma área onde a literatura técnica ainda está em consolidação.
- Explicativa: Ao propor uma nova arquitetura de software e analisar detalhadamente como cada componente (Regex, NER, LLM) contribui para a mitigação dos riscos identificados.

Do ponto de vista dos procedimentos técnicos, adota-se o método Experimental. A validação da hipótese é realizada mediante a construção de um protótipo controlado, submetido a cenários de teste que simulam dados reais do setor de Recursos Humanos. A eficácia da solução é verificada quantitativamente através de estudos de ablação para isolar e mensurar o desempenho individual de cada filtro de segurança.

3.2 Materiais e Ferramentas Utilizadas

A seleção do *stack* tecnológico priorizou o desempenho em tempo real, a modularidade e a capacidade de processamento local para garantir a soberania dos dados.

3.2.1 Recursos de Software

A implementação do protótipo baseou-se em um conjunto tecnológico selecionado para priorizar o desempenho em tempo real e a modularidade. As principais ferramentas e bibliotecas utilizadas foram:

- Linguagem de programação Python 3.10+, escolhida pela robustez de seu ecossistema em Ciência de Dados e facilidade de integração com bibliotecas de inteligência artificial.
- *Framework* de orquestração FastAPI, utilizado para a construção do serviço de *proxy*. Sua escolha justifica-se pelo suporte nativo ao padrão *ASGI* (*Asynchronous Server Gateway Interface*), essencial para gerenciar alta concorrência e minimizar a latência durante a interceptação das requisições.
- Biblioteca *spaCy*, empregada para o reconhecimento de entidades nomeadas (NER). Utilizou-se o modelo `pt_core_news_lg` (otimizado para o português) em conjunto com o componente *EntityRuler*, permitindo uma abordagem híbrida que combina modelos estatísticos com regras gramaticais personalizadas.
- Plataforma Ollama, utilizada como motor de inferência para a execução local (*self-hosted*) do modelo de linguagem. O modelo selecionado foi o Llama 3 (8B), escolhido por oferecer um equilíbrio viável entre capacidade de raciocínio contextual e eficiência computacional para *hardware* de consumo.
- Biblioteca *Pydantic* e validadores aritméticos, responsáveis pela definição estrita de contratos de dados e validação de esquemas em tempo de execução. A implementação incluiu algoritmos de verificação de dígitos (como o módulo 11 para CPF e CNPJ) para eliminar falsos positivos comuns em detecções baseadas puramente em *Regex*.

3.2.2 Recursos de Hardware

Considerando que a execução de LLMs locais demanda recursos computacionais intensivos, a especificação do ambiente de testes é crítica para a reprodutibilidade dos tempos de inferência. Os experimentos foram conduzidos em uma estação de trabalho com as seguintes características:

- Processador (CPU): Processador Intel Core i5-11400F;
- Memória RAM: 32GB (2x16GB), 3200MHz, DDR4;
- Acelerador Gráfico (GPU): NVIDIA RTX 3060 Ti 8GB;
- Armazenamento: SSD Kingston NV3 1TB M.2 2280 NVMe Gen4.

3.3 Etapas da Pesquisa

O desenvolvimento deste trabalho foi estruturado em quatro etapas sequenciais, alinhadas às práticas de Engenharia de Software Experimental:

3.3.1 Etapa 1: Levantamento Bibliográfico e Requisitos

A fase inicial consistiu em uma revisão sistemática da literatura para mapear o estado da arte em privacidade para LLMs, com ênfase em ataques de injeção de *prompt* e técnicas de sanitização, utilizando como referência sistemas como o *Casper*. Paralelamente, realizou-se a engenharia de requisitos com base na LGPD, definindo as regras de negócio para a detecção de PIIs.

3.3.2 Etapa 2: Modelagem Arquitetural e Estratégia de Defesa

Nesta etapa, definiu-se a arquitetura da solução baseada no padrão *Pipe and Filter* (SHAW; GARLAN, 1996), visando o desacoplamento das responsabilidades. A estratégia de defesa foi projetada seguindo o princípio da "Defesa em Profundidade", estabelecendo três camadas de proteção (Regex, NER e Análise Semântica) para mitigar as falhas intrínsecas de cada método isolado.

3.3.3 Etapa 3: Implementação Iterativa do Protótipo

O desenvolvimento seguiu uma abordagem incremental. A implementação iniciou-se pelos módulos de detecção individuais (Filtros), seguidos pela construção do orquestrador em FastAPI e do mecanismo de reconstrução de contexto (*RestorationService*). Durante esta fase, foram integradas bibliotecas de validação rigorosa e o modelo local Llama 3, garantindo que o processamento de dados sensíveis ocorresse estritamente em ambiente local.

3.3.4 Etapa 4: Protocolo Experimental e Validação (Dataset)

A etapa final dedicou-se à validação quantitativa da solução. Para garantir a conformidade ética e evitar o manuseio de dados reais de terceiros durante a fase de desenvolvimento, optou-se pela construção de um Dataset Sintético Heterogêneo.

Geração e Proveniência dos Dados: O conjunto de dados é composto por 500 *prompts* curados do domínio de Recursos Humanos. Para mitigar o viés linguístico e assegurar a diversidade sintática, os dados foram gerados utilizando-se três modelos de IA generativa distintos: GPT-4 (OpenAI), Claude 3 (Anthropic) e Gemini (Google). Cada modelo foi instruído a assumir diferentes *personas* (ex: Recrutador, Gerente de

Benefícios, Auditor), resultando em uma variabilidade de estilos de escrita que simula um ambiente corporativo real.

Normalização e Integridade do Ground Truth: Uma limitação conhecida de modelos generativos é a imprecisão na contagem de caracteres (tokenização), o que poderia corromper o gabarito de teste. Para solucionar isso, desenvolveu-se um *pipeline* de pós-processamento determinístico (script Python). Este algoritmo realiza duas funções críticas:

1. Normalização NFC: Padroniza a codificação de caracteres acentuados (Unicode Normalization Form C), garantindo consistência entre diferentes sistemas operacionais.

2. Recálculo de Spans: Varre o texto gerado para identificar os índices exatos de início e fim de cada entidade nomeada, assegurando que o *Ground Truth* corresponda perfeitamente aos dados de entrada.

Essa abordagem garante que a avaliação das métricas de Precisão e Recall reflita exclusivamente o desempenho do sistema Janus, eliminando ruídos causados por erros na anotação dos dados.

3.4 Métricas de Avaliação

Para mensurar a eficácia dos filtros e superar o desafio da discrepância de granularidade entre a percepção humana e a detecção de máquina, foram adotadas métricas padronizadas de Recuperação de Informação.

A validação confronta as entidades mascaradas pelo *proxy* com um gabarito rotulado (*Ground Truth*) (JURAFSKY; MARTIN, 2024), classificando as ocorrências conforme as definições abaixo:

- Verdadeiro Positivo (VP): Ocorre quando uma PII presente no texto original é corretamente identificada e mascarada pelo sistema.

- Falso Positivo (FP): Ocorre quando o sistema aplica máscara em um trecho de texto inofensivo (erro por excesso de zelo).

- Falso Negativo (FN): Ocorre quando o sistema falha em detectar uma PII existente, permitindo seu vazamento (erro crítico de segurança).

Com base na matriz de confusão gerada, calculam-se os seguintes indicadores:

1. Precisão (*Precision*): Mensura a confiabilidade das detecções ($\frac{VP}{VP+FP}$). Uma alta precisão indica que o sistema preserva a legibilidade do *prompt*, evitando mascaramento desnecessário.

2. Revocação (*Recall*): Mensura a cobertura de segurança ($\frac{VP}{VP+FN}$). No contexto de proteção de dados, esta métrica é priorizada, pois indica a capacidade do sistema de blindar as informações sensíveis.

3. F1-Score: A média harmônica entre precisão e revocação, utilizada para demonstrar o equilíbrio operacional do sistema.

4 Projeto

4.1 Implementação do Sistema

A solução foi arquitetada como um *middleware* de segurança de alto desempenho, exposto via API RESTful para garantir interoperabilidade e escalabilidade. A fundação tecnológica do *backend* baseia-se no *framework* Python FastAPI, selecionado por sua estrita aderência à especificação ASGI (*Asynchronous Server Gateway Interface*). Esta natureza assíncrona constitui um requisito não-funcional crítico, assegurando alta vazão (*throughput*) e latência mínima no processamento de E/S, características indispensáveis para um sistema que atua como intermediário em tempo real entre o usuário e provedores de IA.

O design do software segue rigorosamente os princípios de Separação de Responsabilidades (*Separation of Concerns*) e Inversão de Controle (IoC), implementados através de interfaces agnósticas. A lógica central de orquestração foi isolada em um serviço de domínio dedicado, que coordena a execução sequencial dos mecanismos de segurança (Regex, NER e LLM) adotando o padrão arquitetural *Pipe and Filter*. Essa estrutura modular assegura que a lógica de sanitização permaneça desacoplada da camada de transporte HTTP, facilitando testes unitários e a manutenibilidade do código.

Para a camada de apresentação e validação, desenvolveu-se uma interface cliente reativa. Um diferencial técnico desta implementação é a utilização do protocolo *Server-Sent Events* (SSE) para a comunicação unidirecional em tempo real. Essa abordagem permite que o *Janus* transmita o estado do processamento passo a passo para a interface, promovendo a transparência algorítmica (explicabilidade). Dessa forma, o operador pode visualizar exatamente qual filtro está atuando sobre o *prompt* e como os dados estão sendo anonimizados antes do envio final para a nuvem.

4.1.1 Implementação do Filtro 1: Detecção Determinística (Regex)

O primeiro estágio do *pipeline* de segurança consiste em um módulo de filtragem determinística baseado em Expressões Regulares (Regex). Conforme definido na arquitetura do sistema, este componente atua como o "Guardião Determinístico", sendo responsável pela identificação de Informações Pessoais Identificáveis (PII) que possuem estrutura sintática rígida e universalmente validável, tais como números de documentos (CPF, CNPJ), endereços de e-mail e padrões telefônicos.

A execução deste filtro segue um fluxo sequencial de quatro etapas, desenhado para maximizar a precisão da detecção:

1. Varredura de Padrões (*Pattern Matching*): O texto de entrada é submetido a uma inspeção iterativa contra um repositório centralizado de definições Regex. Nesta fase, o sistema busca por correspondências sintáticas brutas para cada categoria de dado sensível mapeada.

2. Validação Aritmética e Lógica: Para mitigar a incidência de Falsos Positivos — uma limitação comum em abordagens puramente baseadas em Regex —, as correspondências identificadas passam por uma camada de validação rigorosa. O sistema implementa algoritmos de verificação de dígitos (como o cálculo de Módulo 11 para CPF e CNPJ) para atestar a autenticidade do documento. Sequências numéricas que respeitam o formato visual, mas falham na verificação matemática, são descartadas. Essa etapa assegura que números aleatórios ou códigos de protocolo não sejam mascarados indevidamente, elevando a precisão (*Precision*) do filtro para níveis próximos de 100% para dados estruturados.

3. Resolução de Conflitos e Sobreposição: O sistema trata ocorrências onde múltiplos padrões coincidem na mesma região do texto (ex: um número de RG contido dentro de uma sequência bancária) através de uma heurística de priorização. Cada tipo de entidade possui um peso definidor; em casos de colisão, o algoritmo preserva a entidade de maior especificidade e prioridade, evitando o mascaramento parcial ou corrompido dos dados.

4. Indexação e Geração de Estado: As entidades validadas são substituídas no texto original por marcadores de posição (*placeholders*) indexados sequencialmente (ex: `[CPF_1]`, `[EMAIL_2]`). Simultaneamente, o sistema gera um "Mapa de Restauração"— uma estrutura de dados que armazena o valor original, o tipo da entidade e sua posição exata. Este artefato é essencial para garantir a reversibilidade do processo, permitindo que o *RestorationService* reconstrua a resposta final do modelo sem perda de informações para o usuário.

Embora este filtro apresente alta precisão e baixo custo computacional, ele possui baixo *recall* para dados não estruturados ou dependentes de contexto (como nomes em meio a frases), tarefa que é delegada aos filtros subsequentes (NER e LLM).

4.1.2 Implementação do Filtro 2: Detecção Probabilística e Híbrida (NER)

Enquanto o primeiro filtro se ocupa de dados estruturados, o segundo estágio do *pipeline* de segurança endereça a detecção de PII de natureza não estruturada e semântica — como nomes de pessoas, localizações geográficas e organizações. Devido à variabilidade contextual desses dados, a abordagem determinística (Regex) torna-se

insuficiente. Portanto, implementou-se um módulo de Reconhecimento de Entidades Nomeadas (NER) baseado em inferência estatística.

A arquitetura deste componente é híbrida, combinando a generalização de modelos de *Deep Learning* com a precisão de regras baseadas em dicionários. A base tecnológica utiliza a biblioteca *spaCy* com o modelo `pt_core_news_lg` (Large), selecionado por possuir vetores de palavras (*word vectors*) mais densos, o que garante maior acurácia na desambiguação de contexto em língua portuguesa.

A implementação técnica divide-se em três camadas de processamento:

1. Extensão Híbrida (*EntityRuler*): Para mitigar limitações do modelo estatístico genérico — que tende a falhar em terminologias específicas de RH ou truncar nomes de cargos compostos —, o *pipeline* padrão foi estendido com o componente *EntityRuler*. Este módulo é injetado antes da inferência estatística e opera com uma base de conhecimento customizada para "Profissões" e "Cargos". Esta estratégia assegura um alto índice de revocação (*High Recall*) para dados críticos do domínio, corrigindo o problema de truncamento (ex: garantir a captura de "Engenheiro de Software Sênior" em vez de apenas "Engenheiro").

2. Heurísticas de Pós-Processamento e Supressão de Ruído: Dado que modelos probabilísticos são inerentemente suscetíveis a Falsos Positivos, implementou-se uma camada de refinamento que aplica regras de validação sobre as entidades detectadas:

    - Sanitização de Artefatos: Entidades que não atendem a critérios mínimos de integridade (como comprimento inferior a três caracteres ou compostas exclusivamente por números e símbolos) são descartadas automaticamente.
    - Lista de Exclusão (*Blocklist*): Um mecanismo de filtragem que ignora tokens comuns da língua portuguesa frequentemente classificados erroneamente como nomes próprios pelo modelo (ex: "Bom", "Atenção").
    - Validação Estrutural: Remoção de entidades que contêm caracteres indicadores de metadados ou rótulos de formulários (ex: ":"), prevenindo a quebra de estrutura do documento.

3. Orquestração Sequencial e Preservação de Estado: O Filtro 2 opera ciente do estado gerado pelo Filtro 1. O algoritmo foi instruído a ignorar regiões de texto já anonimizadas (ex: `[CPF_1]`), economizando recursos computacionais e evitando conflitos de reclassificação.

Para resolver conflitos internos (como a sobreposição entre uma entidade detectada pelo modelo estatístico e outra pelo *EntityRuler*), o sistema adota a estratégia

de Maximização de Cobertura (*Longest Match Strategy*). O algoritmo prioriza a entidade que abrange o maior número de *tokens*, assegurando que a informação seja capturada em sua totalidade semântica.

Concluído o processamento, as entidades validadas são convertidas em *placeholders* tipados (ex: `[NOME_1]`, `[ORG_2]`) e integradas ao mapa de restauração unificado, garantindo a reversibilidade total dos dados na etapa de pós-processamento.

4.1.3 Implementação do Filtro 3: Análise Contextual (LLM Local)

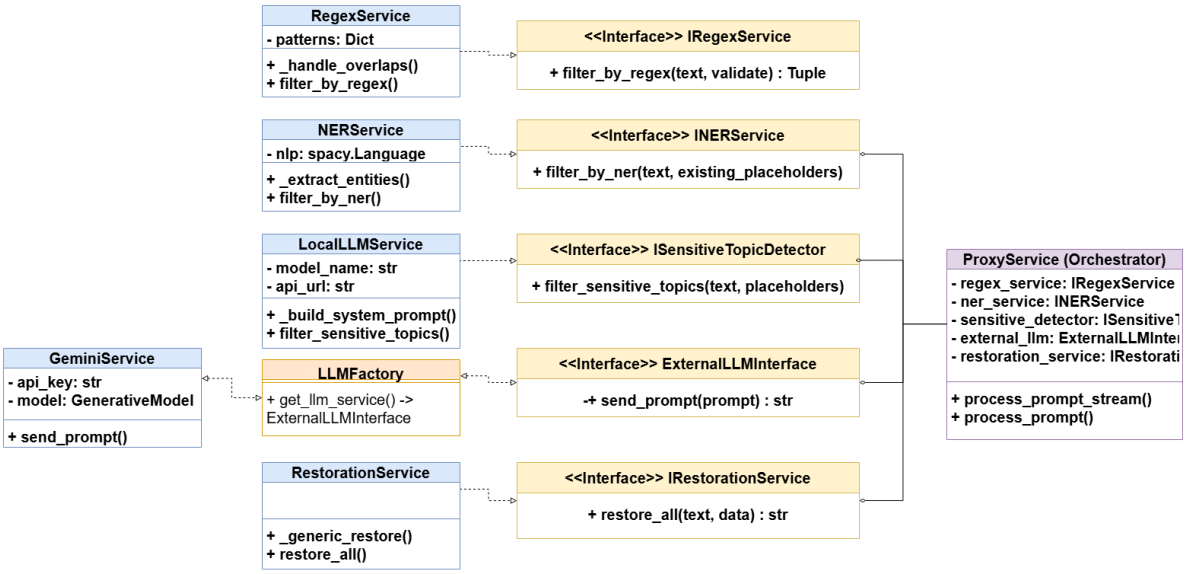
O terceiro estágio do pipeline, denominado "Analista Semântico", mitiga as limitações dos métodos determinísticos. Para operacionalizar essa análise mantendo a privacidade, utiliza-se uma instância local do modelo Llama 3 (8B) via Ollama. A implementação, contida na classe `LocalLLMService`, baseia-se em:

1. Engenharia de Prompt e JSON Enforcement: O núcleo deste filtro é um *System Prompt* especializado que define a persona do modelo como um "Especialista em LGPD". Diferente de interações de chat padrão, o sistema força o modelo a operar em Modo JSON (`"format": "json"`). Isso garante que a saída seja sempre uma estrutura de dados previsível (`sensitive_fragments`), eliminando a necessidade de *parsing* complexo de texto livre e facilitando a integração com o backend em Python.

2. Precedência e Não-Sobreposição: O algoritmo implementa uma lógica de verificação espacial (`_find_placeholder_spans`). Antes de mascarar um fragmento identificado pela LLM, o sistema verifica se as coordenadas (índices de início e fim) colidem com *placeholders* já inseridos pelos filtros de Regex ou NER. Se houver sobreposição, a detecção da LLM é descartada, priorizando a precisão dos métodos determinísticos anteriores.

4.2 Orquestração e Gerenciamento de Dependências

Figura 2 – Diagrama de Classes da arquitetura Janus, evidenciando o desacoplamento via interfaces e o padrão Factory.



Fonte: Elaboração própria (2025).

O núcleo operacional do sistema reside no módulo de orquestração (`ProxyService`), responsável por coordenar o ciclo de vida integral das requisições. A arquitetura deste componente foi concebida para maximizar a modularidade, utilizando extensivamente o padrão de Injeção de Dependência (DI) provido nativamente pelo ecossistema do FastAPI.

Em consonância com o Princípio da Inversão de Dependência (DIP) da metodologia SOLID, o orquestrador não depende de implementações concretas dos filtros. Em vez disso, o sistema é programado orientado a contratos (Interfaces), como `IRegexService` e `INERService`. Essa abordagem assegura que a lógica central do *pipeline* permaneça agnóstica às tecnologias subjacentes, permitindo, por exemplo, a substituição transparente do motor de LLM local ou a atualização de bibliotecas de NER sem impactar o fluxo de controle principal.

O *endpoint* de processamento executa a estratégia de filtragem sequencial (*Pipe and Filter*), gerenciando não apenas o fluxo de texto, mas também a serialização do estado (o mapa de dados anonimizados) entre os estágios:

1. Estágio Determinístico (Regex): O *prompt* bruto é submetido ao filtro de padrões rígidos. O orquestrador recebe o texto parcialmente mascarado e o fragmento inicial do mapa de restauração.

2. Estágio Probabilístico (NER): O texto resultante é injetado no serviço de NER. Crucialmente, este serviço opera ciente das máscaras já aplicadas (ex: `[CPF_1]`), evitando o reprocessamento de entidades já tratadas e garantindo a eficiência computacional.

3. Estágio Semântico (LLM Local): O artefato textual, já sanitizado estrutural e nominalmente, é submetido à análise contextual do modelo local para a detecção de dados sensíveis remanescentes.

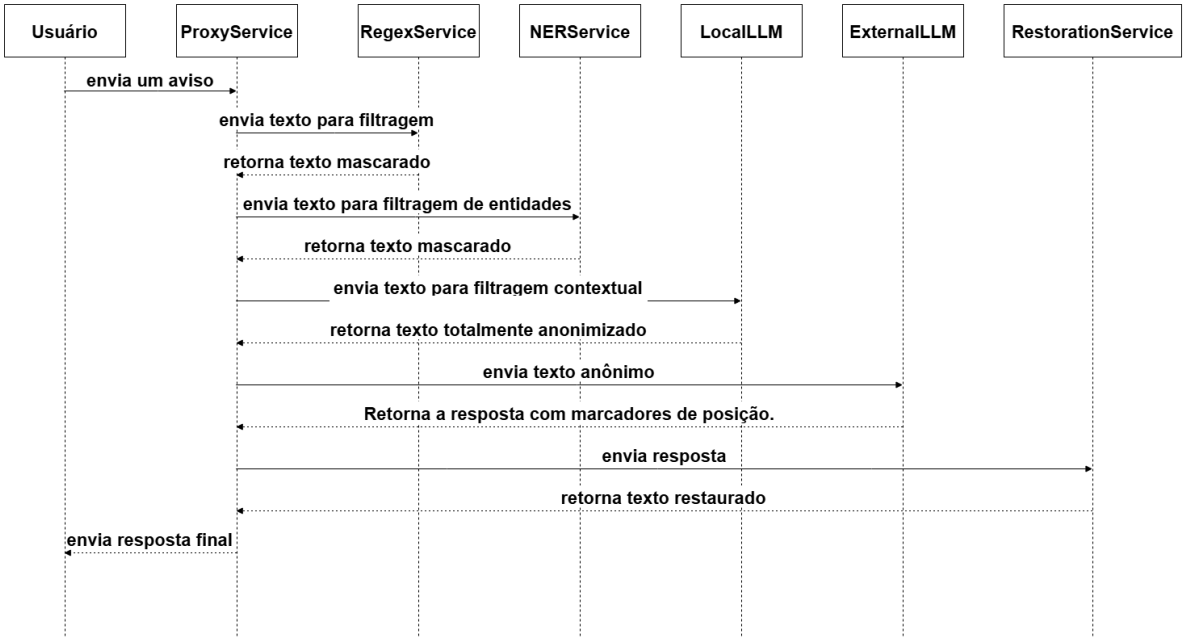
4. Despacho Seguro: Somente após a validação bem-sucedida em todos os filtros, o *payload* final — agora estritamente anônimo — é transmitido à API da LLM externa (Provedor de Inferência).

5. Reconstrução de Estado (Restauração): Ao receber a resposta da nuvem (que mantém os *placeholders*), o orquestrador invoca o `RestorationService`. Este serviço utiliza o "Mapa de Dados Unificado" — acumulado durante as três etapas anteriores — para realizar a substituição reversa, entregando ao usuário uma resposta contextualizada e inteligível.

Um diferencial técnico da implementação é a adoção do protocolo *Server-Sent Events* (SSE) para a comunicação com o cliente. Diferente de requisições HTTP síncronas tradicionais (que mantêm o usuário em espera opaca), o sistema estabelece um canal de *streaming* de eventos unidirecional. Isso permite notificar a interface gráfica em tempo real sobre o progresso de cada filtro (ex: *"Anonimizando Entidades...", "Aguardando LLM Externa..."*), promovendo a transparência algorítmica e mitigando a percepção de latência durante o processamento complexo da segurança.

4.3 Lógica de Negócios e Concorrência Assíncrona

Figura 3 – Diagrama de Sequência detalhando o fluxo de mensagens e a ordem de execução dos filtros de sanitização.



Fonte: Elaboração própria (2025).

A lógica de negócios é encapsulada na camada `ProxyService`, que atua como o núcleo de execução da *pipeline*. O maior desafio técnico desta orquestração é a integração de componentes síncronos e computacionalmente intensivos (CPU-Bound) em uma arquitetura web assíncrona (I/O-Bound) baseada em ASGI.

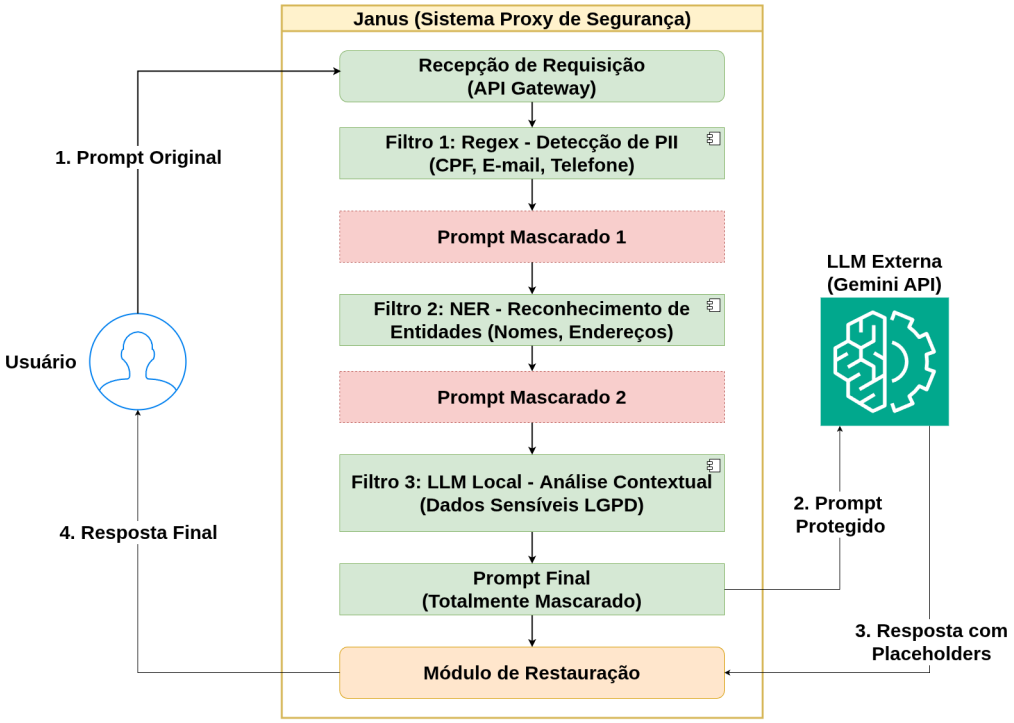
Operações de PLN, como a inferência do modelo spaCy ou chamadas HTTP bloqueantes para a LLM local, poderiam congelar o *Event Loop* principal do servidor Python, degradando severamente a capacidade de atender múltiplos usuários simultâneos — especialmente considerando a latência média de 3,5 segundos da camada semântica.

Para mitigar esse gargalo sem reescrever as bibliotecas de IA, o sistema adota o padrão de Delegação para *Thread Pool*. As tarefas pesadas são encapsuladas e despachadas para *threads* trabalhadoras (*worker threads*) separadas, utilizando a primitiva `asyncio.to_thread` (ou `run_in_executor`). Isso libera o loop principal para continuar aceitando novas conexões de rede enquanto o processamento pesado ocorre em paralelo, garantindo que a latência de um usuário não bloqueie a disponibilidade do serviço para outros.

O serviço gerencia a passagem de estado entre os filtros, seguindo o fluxo abaixo:

1. O prompt original é enviado ao `IRegexService`.
2. O texto resultante, parcialmente anonimizado, e a lista de *placeholders* gerados são passados ao `INERService`.
3. O texto processado pelo NER, juntamente com a lista acumulada de *placeholders* (Regex + NER), é enviado ao `ISensitiveTopicDetector` (LLM Local).
4. O prompt final, totalmente filtrado, é enviado ao `IExternalLLM`.

Figura 4 – Fluxograma do pipeline de processamento e transformação de dados no Sistema Janus.



Fonte: Elaboração própria (2025).

A Figura 4 detalha visualmente o fluxo de execução descrito acima, evidenciando a transformação progressiva do *prompt* à medida que atravessa as camadas de filtragem e o subsequente processo de restauração.

Na etapa de retorno, o serviço unifica os resultados, consolidando os mapas de anonimização individuais de cada filtro em um único conjunto de dados, que é então passado ao `IRestorationService` para a reversão final dos *placeholders* na resposta do LLM. Adicionalmente, a camada de serviço gera eventos de log em tempo real, transmitidos ao cliente via *Server-Sent Events* (SSE), provendo observabilidade sobre a execução de cada estágio da pipeline.

4.3.1 Justificativa para a Persistência de Estado e Restauração

Uma decisão arquitetural que distingue o sistema Janus de ferramentas tradicionais de Prevenção contra Perda de Dados (DLP) é a implementação da capacidade de Re-hidratação de Dados. Enquanto soluções de mascaramento estático focam apenas na obstrução da informação (redação destrutiva), a proposta deste trabalho prioriza a utilidade semântica da resposta gerada pela IA.

A justificativa para essa abordagem reside na manutenção da inteligibilidade do diálogo. Considere-se o seguinte cenário de uso no domínio de RH:

- Solicitação do usuário: “Gere uma carta formal de promoção para o funcionário associado ao CPF 123.456.789-00, informando seu novo salário de R$ 5.000,00.”
- Processamento (sem restauração): o sistema mascara o identificador e envia o *prompt* à LLM. O modelo gera o texto completo, incorporando o marcador na estrutura frasal.
- Resposta da LLM: “Prezado colaborador portador do documento `[CPF_1]`, temos o prazer de comunicar sua promoção...”

Sem a etapa de restauração, o documento gerado, embora gramaticalmente coerente e bem estruturado, torna-se inutilizável para fins administrativos, pois a referência legal de identidade foi perdida. Ao implementar a re-hidratação, o sistema Janus entrega o texto finalizado: “...portador do documento 123.456.789-00...”, fechando o ciclo de valor da informação e entregando um artefato pronto para uso.

Análise de *Trade-off* (Stateful vs. Stateless)

A implementação dessa funcionalidade impõe um requisito não-funcional de arquitetura: a necessidade de o *proxy* operar de maneira Stateful. Diferentemente de APIs REST puramente *Stateless*, o Janus precisa manter o “Mapa de Dados” em memória (RAM) durante o ciclo de vida da requisição. Embora essa estratégia introduza um leve aumento na complexidade de gerenciamento de memória e no consumo de recursos, tal *overhead* é justificado como um investimento necessário para viabilizar a usabilidade prática do sistema em cenários corporativos, onde a rastreabilidade da resposta é tão crítica quanto a privacidade da pergunta.

4.4 Ciclo de Restauração e Reconstrução de Contexto

A etapa de Desanonimização ("Re-hidratação") é executada pelo `RestorationService`, garantindo que a resposta gerada pela IA externa seja entregue ao usuário com os va-

lores originais.

4.4.1 Estratégia de Substituição Reversa

O mecanismo fundamenta-se no Mapa de Dados Unificado (`RestorationData`), que agrega os mapeamentos gerados pelos três filtros. Para assegurar a integridade da reconstrução, o algoritmo adota a seguinte lógica:

1. Ordenação Decrescente de Spans: Os mapeamentos são ordenados inversamente à sua posição no texto (do último para o primeiro caractere). Essa técnica, implementada no método `_generic_restore`, evita que a substituição de um *placeholder* altere os índices de outros ainda não processados, prevenindo a corrupção do texto.

2. Tolerância a Falhas *(Hallucination Handling)* : O serviço implementa uma verificação de existência antes da substituição. Caso a LLM externa “alucine” ou altere a grafia de um *placeholder* (ex: devolvendo `[NOME_99]` inexistente), o sistema detecta a ausência da chave no texto de resposta, registra o evento em *log* (nível *WARNING*) e preserva a integridade do restante da mensagem, evitando exceções em tempo de execução.

3. Limpeza de Artefatos: Como etapa final, o método `_cleanup_duplicate_labels` remove redundâncias comuns geradas por concatenação textual (ex: "CPF CPF 123..."), refinando a legibilidade da resposta final.

4.5 Modelagem de Dados e Contratos de Interface

Para assegurar a integridade transacional, a previsibilidade do fluxo de processamento e a interoperabilidade entre os microsserviços (Filtros, Orquestrador e API), o sistema implementa uma rigorosa camada de modelagem de dados. Utilizando a biblioteca *Pydantic*, foram definidos esquemas de validação estrita (*strict typing*) que atuam como guardiões nas fronteiras da aplicação. Esta abordagem estabelece "contratos formais"para a comunicação interna e externa:

1. Contratos de API (DTOs): Os modelos `PromptRequest` e `ProcessedResponse` atuam como Objetos de Transferência de Dados (DTOs), definindo, respectivamente, o esquema de entrada (o *prompt* bruto) e a estrutura de saída (a resposta restaurada enriquecida com metadados de auditoria). A validação em tempo de execução na camada de entrada impede que requisições malformadas propaguem erros para o núcleo da aplicação.

2. Modelo Canônico de Mapeamento (`PIIMapping`): Este é o artefato de dados central da arquitetura, servindo como a estrutura unificada para o "mapa de anonimização". Independentemente da origem da detecção (Regex, NER ou LLM), toda entidade identificada é normalizada em uma instância deste modelo, que encapsula:

   - `original_value`: O dado sensível bruto.
   - `placeholder`: O marcador substituto indexado.
   - `type`: A categorização da entidade (ex: CPF, NOME).
   - `span`: As coordenadas cartesianas (índices de início e fim) da entidade no texto.

   A persistência do atributo `span` é crítica para a reversibilidade do sistema, pois permite a ordenação espacial das entidades e a resolução determinística de conflitos de sobreposição durante a etapa de restauração.

3. Modelo de Validação (`PIIGroundTruth`): Para suportar a metodologia experimental, definiu-se uma estrutura para o “gabarito” (*Ground Truth*). Este modelo viabiliza a comparação programática entre as detecções do *Janus* e o *dataset* de referência, automatizando o cálculo das métricas de desempenho (Precisão, Revocação e F1-Score) detalhadas no Capítulo de Metodologia.

4.6 Repositório de Conhecimento e Heurísticas de Domínio

Um dos diferenciais da solução proposta é a sua especialização no contexto brasileiro e no domínio de Recursos Humanos (RH). Para alcançar esse nível de especificidade sem comprometer a manutenibilidade do código, adotou-se a estratégia de Externalização de Regras de Negócio. Desenvolveu-se um repositório centralizado de constantes e heurísticas que desacopla a lógica de detecção da implementação dos algoritmos, alimentando os três filtros de segurança:

1. Especialização do Filtro 1 (Regex): Curadoria da biblioteca `PII_PATTERNS`, otimizada para documentos nacionais. O repositório abrange padrões de alta precisão para identificadores fiscais (CPF, CNPJ) e formatos de contato locais, considerando variações de formatação (com ou sem pontuação).

2. Especialização do Filtro 2 (NER): A adaptação do modelo estatístico para o domínio de RH baseia-se em três mecanismos de injeção de conhecimento:

- Normalização Taxonômica: O mapa `NER_ENTITY_TYPE_MAPPING` traduz as etiquetas genéricas do *spaCy* (ex: PER, ORG) para a taxonomia interna do projeto, garantindo consistência semântica.
- Extensão via *EntityRuler*: Utilização da lista `NER_PROFESSION_PATTERNS` para capturar entidades negligenciadas pelo modelo base, como “Cargos” e “Funções”, cruciais em documentos corporativos.
- Supressão de Falsos Positivos: Implementação de uma *Blocklist* (`NER_FALSE_POSITIVES`) baseada em evidências empíricas, filtrando termos ambíguos (ex: "Bom dia", "RH") para elevar a precisão do sistema.

3. Direcionamento do Filtro 3 (LLM Local): Definição de um léxico semântico (`SENSITIVE_CATEGORIES`) que orienta a engenharia de *prompt*. Esta lista instrui o modelo local sobre quais contextos específicos buscar (ex: `CONDICAO_DE_SAUDE`, `HISTORICO_DISCIPLINAR`), ao mesmo tempo que reforça restrições negativas para ignorar padrões já tratados pelos filtros anteriores.

4.7 Abstração do Provedor Externo (Padrão Factory)

Visando assegurar o baixo acoplamento e a neutralidade tecnológica da arquitetura (*Vendor Agnosticism*), a integração com o LLM de destino é gerenciada através do Padrão de Projeto Factory Method.

Este componente abstrai a complexidade de instanciação dos clientes de API. A lógica central do *proxy* (`ProxyService`) não possui dependência direta de implementações concretas (como bibliotecas da OpenAI ou Google), interagindo apenas com a interface genérica `ExternalLLMInterface`.

O módulo “Fábrica” inspeciona as variáveis de ambiente durante a inicialização do sistema (ex: `EXTERNAL_LLM_PROVIDER="gemini"`) e injeta dinamicamente a implementação correspondente. Essa arquitetura permite a troca ou adição de novos provedores de IA sem a necessidade de refatoração na lógica de segurança ou no *pipeline* de anonimização.

4.8 Engenharia de Contexto e Robustez Semântica

A comunicação com a LLM externa é encapsulada em um *template* que injeta instruções de segurança (*System Prompt*). O sistema instrui explicitamente o modelo a tratar os *placeholders* (ex: `[NOME_1]`) como entidades imutáveis.

Testes empíricos demonstraram que, devido ao mecanismo de atenção (*Self-Attention*) da arquitetura Transformer, a LLM externa é capaz de manter a coerência sintática da resposta mesmo operando sobre texto mascarado. O sistema confia na ro-

bustez semântica do modelo para gerar respostas contextualizadas, delegando ao `RestorationService` a tarefa determinística de reinserir os dados reais apenas na fronteira de saída do sistema.

5 Resultados e Análise Experimental

A validação experimental da arquitetura proposta foi conduzida através de uma estratégia incremental, desenhada para avaliar não apenas a eficácia pontual dos filtros, mas a estabilidade do sistema sob diferentes cargas de complexidade (escalabilidade). O protocolo de testes abrangeu cinco cenários distintos, totalizando a análise de mais de 4.500 entidades sensíveis.

5.1 Fase 1: Prova de Conceito (Small-Scale)

Utilizando esse conjunto de dados controlado, inicialmente com 10 *prompts* (127 PIIs esperadas), a arquitetura integrada atingiu um *F1-Score* de 0.7511.

Embora promissor, este resultado inicial indicou um cenário otimista, típico de amostras reduzidas, onde a variabilidade dos dados de entrada ainda não refletia a complexidade real do domínio de Recursos Humanos.

5.2 Fase 2: Evolução e Convergência Estatística

Para determinar o desempenho real do sistema, o experimento foi escalado progressivamente através de quatro cenários adicionais: 50, 100, 250 e 500 *prompts*. Esta progressão permitiu observar o fenômeno de estabilização das métricas frente ao aumento da entropia dos dados.

5.2.1 Consolidação dos Resultados da Pipeline Integrada

A Tabela 2 resume a evolução do desempenho global da arquitetura (todos os filtros ativos) ao longo dos cinco cenários de teste.

Tabela 2 – Evolução do Desempenho da Pipeline Completa (10 a 500 Prompts)

Dataset (N)	Total PII	Precisão	Recall	F1-Score
10	127	0.8091	0.7008	**0.7511**
50	465	0.7075	0.6086	**0.6543**
100	882	0.6894	0.6190	**0.6523**
250	2235	0.6514	0.5776	**0.6123**
500	4560	0.6313	0.5603	**0.5937**

Fonte: Elaboração própria com base nos dados experimentais (2025).

Análise de tendência: Observa-se um comportamento de estabilidade assintótica. Após a queda inicial natural (de $N = 10$ para $N = 50$), as métricas não colapsaram com o aumento exponencial do volume de dados. Entre $N = 250$ e $N = 500$, a variação do *F1-Score* foi inferior a 2 pontos percentuais ($0.61 \rightarrow 0.59$), indicando que o sistema encontrou seu patamar operacional robusto em torno de 0.60, mesmo processando milhares de entidades.

5.3 Análise Detalhada por Componente (Cenário N=500)

Para compreender a contribuição individual de cada camada de defesa, analisa-se detalhadamente o cenário de maior estresse (Relatório Técnico 5.0), onde o sistema foi desafiado a identificar 4.560 PIIs.

5.3.1 Filtro 1: A Base Determinística (Regex)

O filtro baseado em Expressões Regulares confirmou-se como o alicerce da arquitetura.

- Desempenho: Manteve consistência absoluta em todos os testes. No cenário final, processou 1.081 entidades com Precisão de 99.26% e *F1-Score* de 0.9940.
- Impacto arquitetural: A capacidade deste filtro de resolver quase 25% de toda a carga de PII (1.081 de 4.560) com custo computacional marginal valida a estratégia de *offloading*. Sem esta camada, os modelos probabilísticos subsequentes estariam sobrecarregados.

5.3.2 Filtro 2: O Desafio Probabilístico (NER)

O filtro de Reconhecimento de Entidades Nomeadas apresentou o maior desafio de calibração.

- Alta sensibilidade, baixa precisão: No teste de 500 *prompts*, o NER detectou 1.882 ocorrências, das quais 1.267 eram Falsos Positivos. Isso resultou em uma precisão de apenas 32.68%.
- Interpretação: O modelo demonstrou ser excessivamente zeloso (*Over-sensitive*), classificando termos corporativos comuns e ambiguidades linguísticas como nomes próprios ou organizações. Embora isso prejudique a precisão, em um contexto de segurança, essa “paranoia” do modelo contribui para o *Recall* global, garantindo que poucas entidades passem despercebidas, ainda que ao custo de maior ofuscação do texto.

5.3.3 Filtro 3: O Analista Semântico (LLM Local)

O modelo Llama 3 (8B) assumiu a maior carga de trabalho complexa, sendo responsável pela varredura de contextos subjetivos.

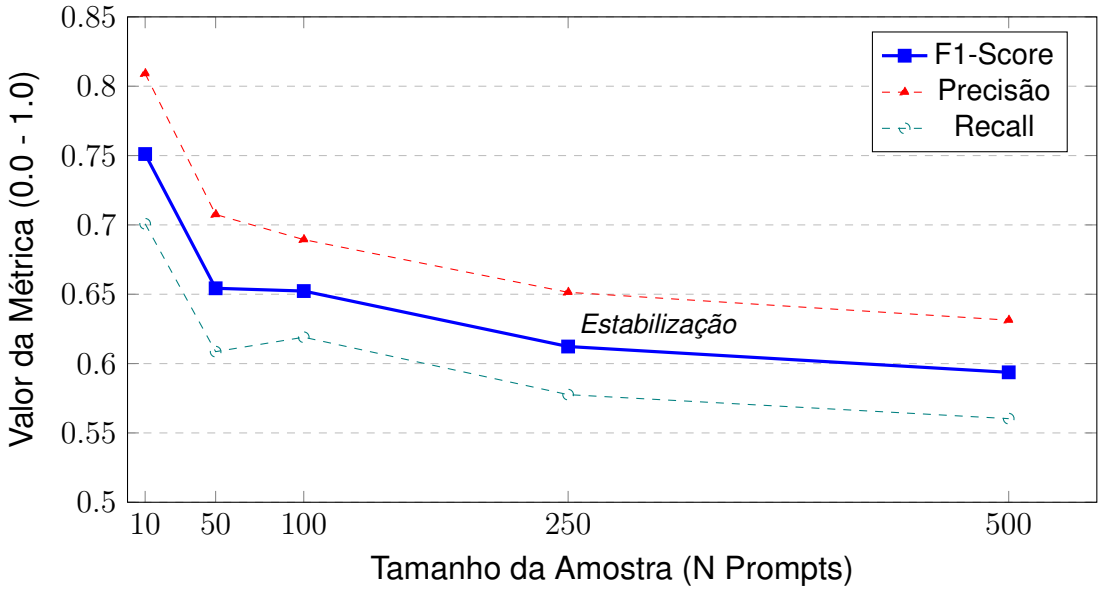
- Volume de processamento: Esperava-se a detecção de 2.713 entidades contextuais.
- Métricas: O modelo atingiu um *F1-Score* de 0.4736, com um equilíbrio quase simétrico entre Precisão (50.02%) e *Recall* (44.97%).
- Análise: Considerando a complexidade da tarefa (interpretar nuances de RH) e o tamanho reduzido do modelo (8 bilhões de parâmetros), o resultado demonstra a viabilidade da inferência local, embora evidencie que modelos maiores poderiam refinar a distinção entre dados sensíveis e contexto geral.

5.4 Discussão dos Resultados

5.4.1 Suficiência Estatística e Regressão à Média

A curva de desempenho que ilustra o fenômeno estatístico da Regressão à Média pode ser observada na Figura 5. A queda acentuada do *F1-Score* entre a Fase 1 (0.75) e a Fase 2 (0.65) não indica degradação do *software*, mas sim a correção do viés de amostragem inicial. A estabilização das métricas nos testes subsequentes (100 a 500) comprova a suficiência estatística do experimento: o sistema comporta-se de maneira previsível independentemente do volume de dados, validando a robustez da solução para ambientes de produção.

Figura 5 – Curva de convergência das métricas de desempenho em função do tamanho da amostra.



Fonte: Elaboração própria (2025).

5.4.2 Trade-off: Segurança vs. Legibilidade em DLP

A análise global (Tabela 2) revela que a *pipeline* completa tende a priorizar a segurança em detrimento da precisão textual absoluta. O elevado número de Falsos Positivos gerados pelo filtro NER (1.267 em $N = 500$), resultando em uma precisão de aproximadamente 32%, indica a ocorrência de *Over-masking* (mascaramento excessivo).

No entanto, no domínio de Prevenção contra Perda de Dados (*DLP*), existe uma assimetria fundamental no custo do erro:

1. Custo do falso positivo: A ocultação de uma palavra inofensiva (ex: mascarar o nome de um projeto público achando que é um cliente) pode reduzir levemente a clareza do *prompt*.

2. Custo do falso negativo: A falha em ocultar um dado real (ex: vazar um CPF) constitui uma violação direta da LGPD, acarretando riscos jurídicos e financeiros severos.

Portanto, a baixa precisão do NER não deve ser interpretada como falha operacional, mas como uma calibração intencional de "Privacidade por Design" (*Privacy by Design*). A arquitetura Janus foi otimizada para maximizar a Revocação (cobertura de risco), aceitando o ruído na legibilidade como o custo necessário para a garantia da conformidade legal.

5.4.3 Robustez Semântica e Preservação de Utilidade

Uma preocupação adjacente à baixa precisão é a inteligibilidade do *prompt* resultante para a LLM externa. Testes qualitativos realizados durante o desenvolvimento com a API do Google Gemini demonstraram que a substituição de entidades por *placeholders* tipados (ex: `[NOME_1]`, `[CARGO_2]`) preserva a estrutura morfossintática das sentenças.

Observou-se que modelos generativos modernos operam com alta robustez contextual. A instrução *"Analise o desempenho de [NOME_1]"* é processada pelo modelo externo com a mesma eficácia lógica de *"Analise o desempenho de João"*, visto que o mecanismo de atenção (*Self-Attention*) foca nas relações entre o sujeito e o predicado, e não no valor literal do substantivo. Isso valida a hipótese de que a utilidade da ferramenta de IA é mantida mesmo em cenários de *Over-masking*, desde que a coerência gramatical do *prompt* seja preservada pelos filtros.

5.4.4 Eficácia do Ground Truth e Curadoria de Dados

A consistência dos resultados do filtro Regex (*F1-Score* ≈ 0.99 em todas as fases) serve como um controle de qualidade para o *dataset* utilizado. O fato de o sistema ter lidado com 4.560 entidades rotuladas manualmente sem apresentar anomalias estatísticas reforça a qualidade do *Ground Truth* construído, isolando as variações de desempenho como características intrínsecas dos modelos de IA, e não como falhas na metodologia de teste.

5.5 Análise de Latência e Custo Computacional

A viabilidade de um sistema de segurança em tempo real não depende apenas da eficácia da detecção (*F1-Score*), mas também de sua latência intrínseca. Para mensurar o impacto da arquitetura proposta na experiência do usuário, conduziu-se um teste de desempenho mensurando o tempo de processamento isolado de cada componente em 10 requisições aleatórias.

A Tabela 3 apresenta os dados detalhados referentes aos tempos de resposta (médio, mínimo e máximo) obtidos no ambiente de testes (GPU NVIDIA RTX 3060 Ti).

Tabela 3 – Latência de Processamento por Componente (Média de 10 Requisições)

Componente	**Média (s)**	**Min (s)**	**Max (s)**	**Impacto (%)**
Filtro 1: Regex	0.0004	0.0003	0.0006	$\approx 0.01\%$
Filtro 2: NER	0.0277	0.0216	0.0363	$\approx 0.77\%$
Filtro 3: LLM Local	3.5605	1.9238	6.3073	$\approx 99.22\%$
Pipeline Total	**3.5641**	**1.6245**	**6.4611**	**100%**

Fonte: Elaboração própria (2025).

Análise de gargalo: Os dados evidenciam uma assimetria extrema no custo computacional. Os filtros determinísticos e probabilísticos (Regex e NER) operam em tempo quase real (< 30ms combinados), validando a eficiência da estratégia de "Defesa em Profundidade": a maior parte das ameaças estruturadas é neutralizada com custo computacional negligenciável.

O gargalo do sistema reside inteiramente na inferência semântica (LLM Local), que adiciona uma latência média de 3.56 segundos. Embora alto para padrões de sistemas *web* tradicionais, este valor é considerado aceitável no contexto de uma ferramenta de auditoria de segurança assíncrona, representando o “Imposto de Privacidade” (*Privacy Tax*) necessário para garantir o processamento de dados sensíveis em perímetro local (*On-Premise*).

5.6 Síntese dos Resultados Experimentais

A validação quantitativa, escalada progressivamente até o cenário de estresse com 500 *prompts* e mais de 4.500 entidades analisadas, permitiu consolidar as seguintes constatações:

- Robustez da base determinística: O filtro Regex confirmou-se como o pilar de eficiência do sistema, mantendo precisão virtualmente perfeita ($> 99\%$) e resolvendo aproximadamente 25% da carga de detecção com custo computacional marginal.

- Estabilidade em escala e calibração defensiva: A convergência do *F1-Score* global para o patamar de 0.60 nos cenários de alta complexidade demonstra a estabilidade operacional do sistema. A análise dos componentes revela uma dicotomia: enquanto o filtro NER apresentou uma calibração excessivamente sensível (gerando muitos falsos positivos e impactando a precisão), a camada semântica (LLM) atuou como fator limitante para a cobertura total (*Recall*). O resultado final reflete um sistema que opera com "privacidade por padrão"nas camadas iniciais, mas cuja capacidade de detecção contextual ainda depende da evolução dos modelos de inferência locais.

- Viabilidade técnica: O sistema provou ser capaz de operar como um "Firewall de Privacidade"funcional para o domínio de Recursos Humanos, mitigando riscos de exposição de dados sensíveis em provedores de nuvem pública, estabelecendo uma base sólida para a adoção segura de IA Generativa.

6 Conclusões

A presente pesquisa validou a hipótese de que é possível compatibilizar o uso de Grandes Modelos de Linguagem (LLMs) em ambientes corporativos com os rigorosos requisitos da Lei Geral de Proteção de Dados (LGPD), mediante a interposição de uma arquitetura de *middleware* de segurança. O desenvolvimento do *proxy* de anonimização multicamadas atingiu seu objetivo primário, entregando uma solução capaz de interceptar, mascarar e restaurar Informações Pessoais Identificáveis (PII) com consistência operacional.

Do ponto de vista arquitetural, a estratégia de "Defesa em Profundidade" — orquestrando filtros Determinísticos (Regex), Probabilísticos (NER) e Semânticos (LLM Local) — demonstrou-se superior a abordagens monolíticas. A implementação do ciclo de vida completo da informação, culminando na "re-hidratação" dos dados via lógica LIFO (*Last-In, First-Out*), assegurou que a camada de segurança permanecesse transparente para o usuário final, preservando a utilidade da ferramenta de IA sem comprometer o sigilo das informações.

6.1 Trabalhos Futuros

Considerando as limitações identificadas, especialmente a latência média introduzida pela camada semântica e a necessidade de reduzir a taxa de falsos positivos, sugerem-se as seguintes linhas de investigação e desenvolvimento para a evolução natural da plataforma:

- Otimização de inferência com *Small Language Models*: O modelo atual provou ser o gargalo de processamento devido ao seu tamanho. Trabalhos futuros devem explorar a substituição por modelos menores e mais eficientes (*Small Language Models* - SLMs), como as famílias *Phi-3* ou *Gemma*, combinados com técnicas de quantização agressiva (ex: *GGUF 4-bit*). O objetivo é reduzir o tempo de inferência para a faixa de sub-segundo em *hardware* de consumo, mantendo a capacidade de raciocínio contextual necessária para a tarefa.

- Refinamento do modelo probabilístico (*Domain Adaptation*): Substituição do modelo de reconhecimento de entidades genérico por um modelo submetido a *Fine-Tuning* com um *corpus* específico de documentos de Recursos Humanos brasileiros. Essa adaptação visa mitigar a alta taxa de falsos positivos observada nos resultados experimentais e melhorar a distinção semântica entre cargos corporativos e nomes próprios, aumentando a precisão global do sistema.

- Técnicas avançadas de preservação (*Format-Preserving Encryption*): Integração de algoritmos de criptografia com preservação de formato (*FPE*) para dados estruturados. Isso permitiria que o modelo externo recebesse dados fictícios que mantêm as propriedades estatísticas e de formatação dos originais (ex: um CPF falso, mas matematicamente válido), melhorando a qualidade da inferência gerada sem expor os dados reais, superando a limitação dos *placeholders* genéricos.

Referências

ALBUQUERQUE, H. O. et al. Named entity recognition: a survey for the portuguese language. *Procesamiento del Lenguaje Natural*, v. 70, p. 171–185, 2023. Acesso em: 29 jul. 2025. 19, 20

ARANTES, F. A. N. *Impactos da Lei Geral de Proteção de Dados nas Relações de Trabalho: Tomada de Decisão por Inteligência Artificial nos Processos de Recrutamento e Seleção. Reflexões sobre o Direito do Trabalhador de Rever o Algoritmo de Decisão*. Dissertação (Dissertação de Mestrado) — Universidade Federal de Minas Gerais, Faculdade de Direito, Belo Horizonte, 2022. Acesso em: 11 nov. 2025. 19

BAEZA-YATES, R.; RIBEIRO-NETO, B. *Modern Information Retrieval: The Concepts and Technology behind Search*. 2nd. ed. Harlow, England: Addison-Wesley Professional, 2011. Acesso em: 02 set. 2025. ISBN 978-0321416919. 25

Brasil. Senado Federal. *Lei Geral de Proteção de Dados Pessoais: Lei nº 13.709/2018*. Brasília, DF: Senado Federal, Coordenação de Edições Técnicas, 2024. Edição atualizada até junho de 2024. Acesso em: 30 jul. 2025. 15

BUSCHMANN, F. et al. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. New York, NY: John Wiley & Sons, 1996. Acesso em: 27 out. 2025. ISBN 978-0471958697. 21

CARPENTIER, R. et al. *Preempting Text Sanitization Utility in Resource-Constrained Privacy-Preserving LLM Interactions*. 2025. Acesso em: 10 set. 2025. Disponível em: <https://arxiv.org/abs/2411.11521>. 17

CHONG, C. J. et al. *Casper: Prompt Sanitization for Protecting User Privacy in Web-Based Large Language Models*. 2024. Acesso em: 15 jul. 2025. Disponível em: <https://arxiv.org/abs/2408.07004>. 15, 17, 18, 19, 21, 22, 24

CHOWDHURY, A. R. et al. *Pr$\epsilon\epsilon$mpt: Sanitizing Sensitive Prompts for LLMs*. 2025. Acesso em: 22 ago. 2025. Disponível em: <https://arxiv.org/abs/2504.05147>. 18

DIAS, M. et al. Named entity recognition for sensitive data discovery in portuguese. *Applied Sciences*, v. 10, n. 7, p. 2303, 2020. Acesso em: 14 set. 2025. 19, 20

FERETZAKIS, G.; VERYKIOS, V. S. Trustworthy ai: Securing sensitive data in large language models. *AI*, v. 5, n. 4, p. 2773–2800, 2024. Acesso em: 18 out. 2025. 19

GARZA, L. et al. *PRvL: Quantifying the Capabilities and Risks of Large Language Models for PII Redaction*. 2025. Acesso em: 05 nov. 2025. Disponível em: <https://arxiv.org/abs/2508.05545>. 18

IBM. *O que é LLM (large language models)?* 2025. <https://www.ibm.com/br-pt/think/topics/large-language-models>. Acesso em: 12 out. 2025. 17

JURAFSKY, D.; MARTIN, J. H. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. 3. ed. [S.l.]: Pearson, 2024. Draft of the 3rd edition available online. Acesso em: 03 nov. 2025. 29

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. Acesso em: 04 set. 2025. ISBN 978-0521865715. Disponível em: <https://nlp.stanford.edu>. 25

PATEL, K. Revolutionizing consumer data analysis: The development and impact of a unique customer identifier. *International Journal of Computer Trends and Technology*, v. 71, n. 12, p. 61–74, 2023. Acesso em: 01 ago. 2025. 18

PORTELA, A. K. E. *Um Estudo Exploratório Sobre o Uso de LLMs como Recurso para o Aprendizado de Fundamentos de Programação*. Dissertação (Trabalho de Conclusão de Curso) — Universidade Federal do Ceará, Campus de Quixadá, Quixadá, 2025. Acesso em: 08 set. 2025. 18

SANTOS, D.; CARDOSO, N. (Ed.). *Reconhecimento de entidades mencionadas em português: Documentação e actas do HAREM, a primeira avaliação conjunta na área*. Braga: Linguateca, 2007. Acesso em: 30 nov. 2025. ISBN 978-989-20-0773-1. Disponível em: <https://www.linguateca.pt>. 20

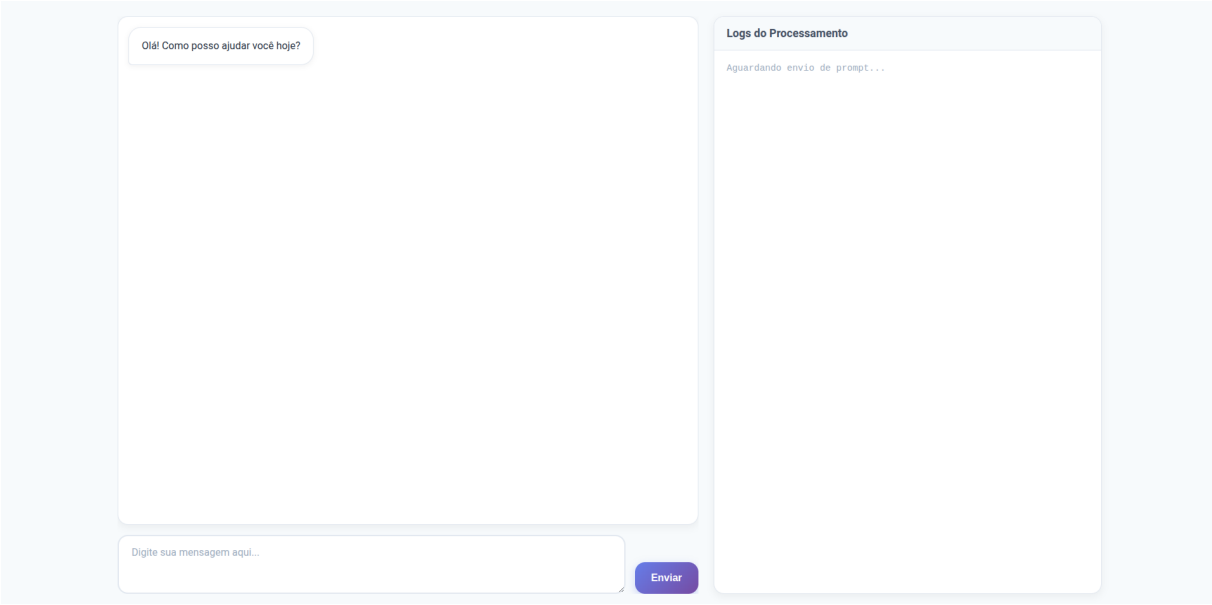
SHAW, M.; GARLAN, D. *Software architecture: perspectives on an emerging discipline*. Upper Saddle River, NJ: Prentice Hall, 1996. 21, 28

SICHMAN, J. S. Inteligência artificial e sociedade: avanços e riscos. *Estudos Avançados*, v. 35, n. 101, p. 37–49, 2021. Acesso em: 25 out. 2025. 17, 18

APÊNDICE A – FLUXO DE EXECUÇÃO E INTERFACE GRÁFICA

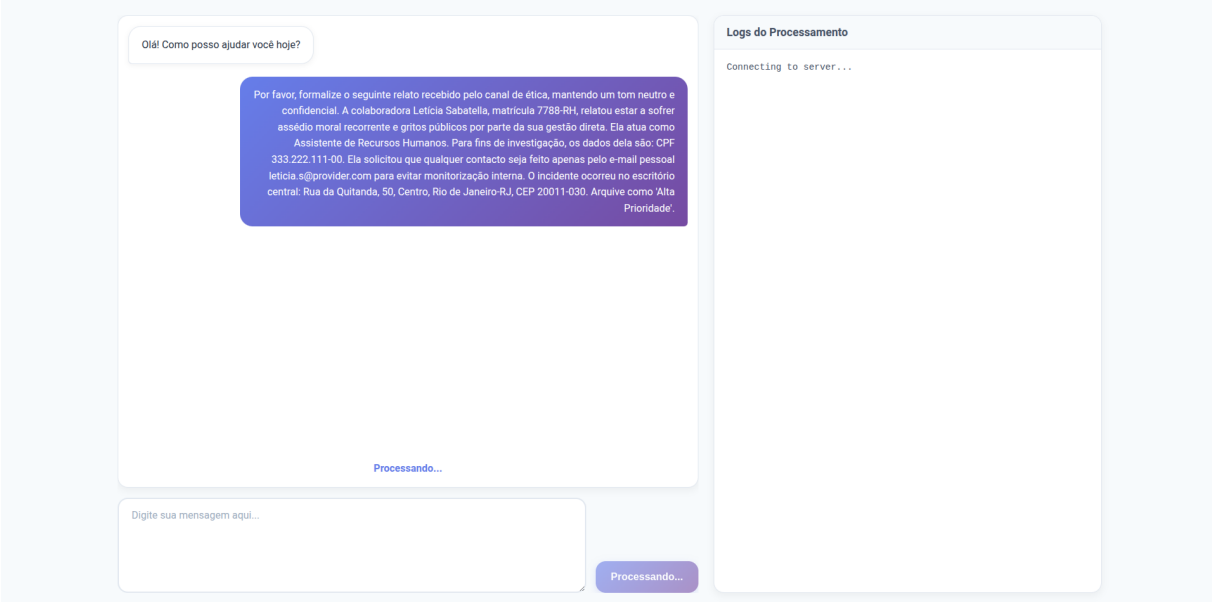
Este apêndice documenta o fluxo completo de uma requisição no sistema Janus, desde a entrada de dados até a recepção da resposta sanitizada e restaurada. As capturas de tela abaixo evidenciam a transparência do processamento através dos logs transmitidos em tempo real via *Server-Sent Events* (SSE).

Figura 6 – Estado inicial da interface do usuário, aguardando entrada.



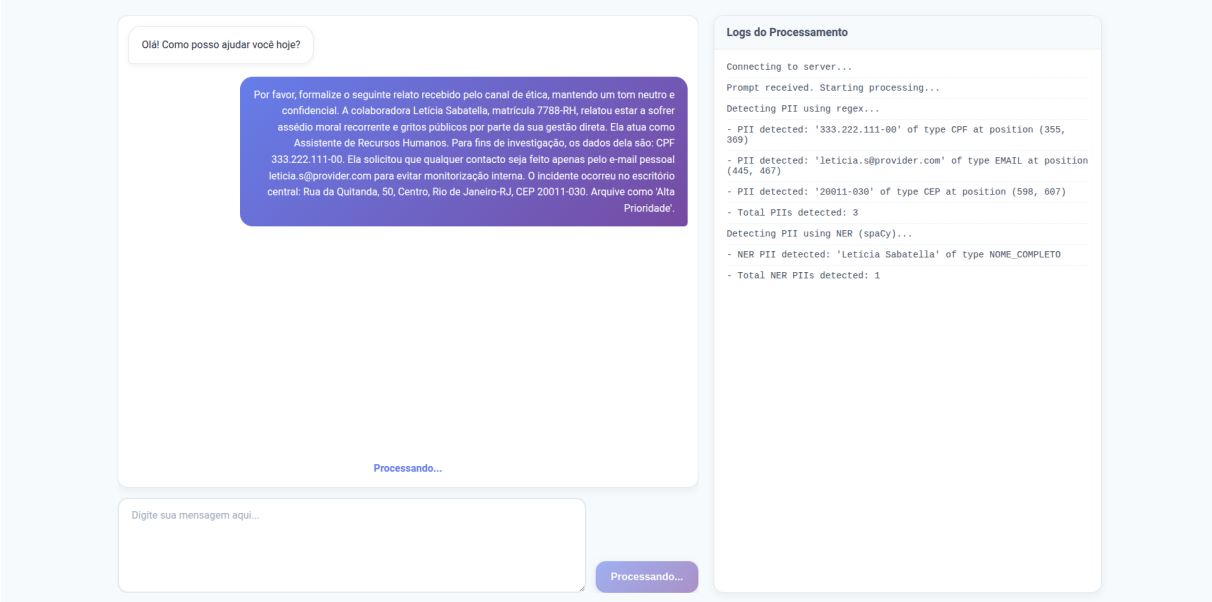
Fonte: Elaboração própria (2025).

Figura 7 – Inserção de prompt contendo dados sensíveis (PII) simulados para teste de estresse (CPF, E-mail, Endereço).



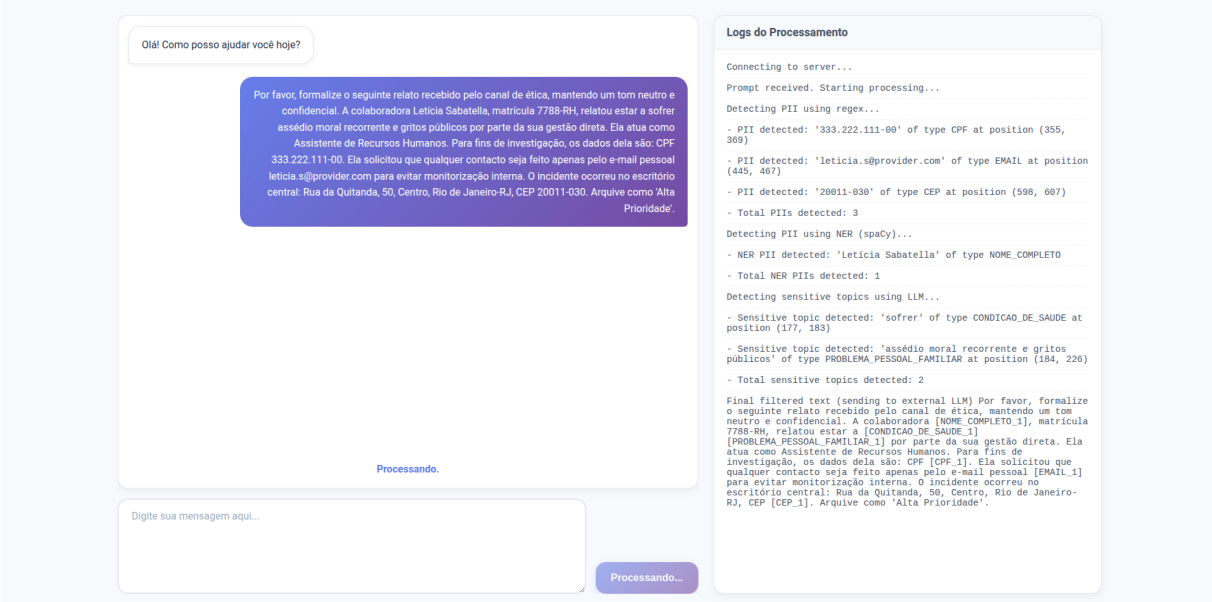
Fonte: Elaboração própria (2025).

Figura 8 – Início do processamento: O painel lateral direito exibe a conexão SSE e a detecção imediata de padrões via Regex (CPF e E-mail).



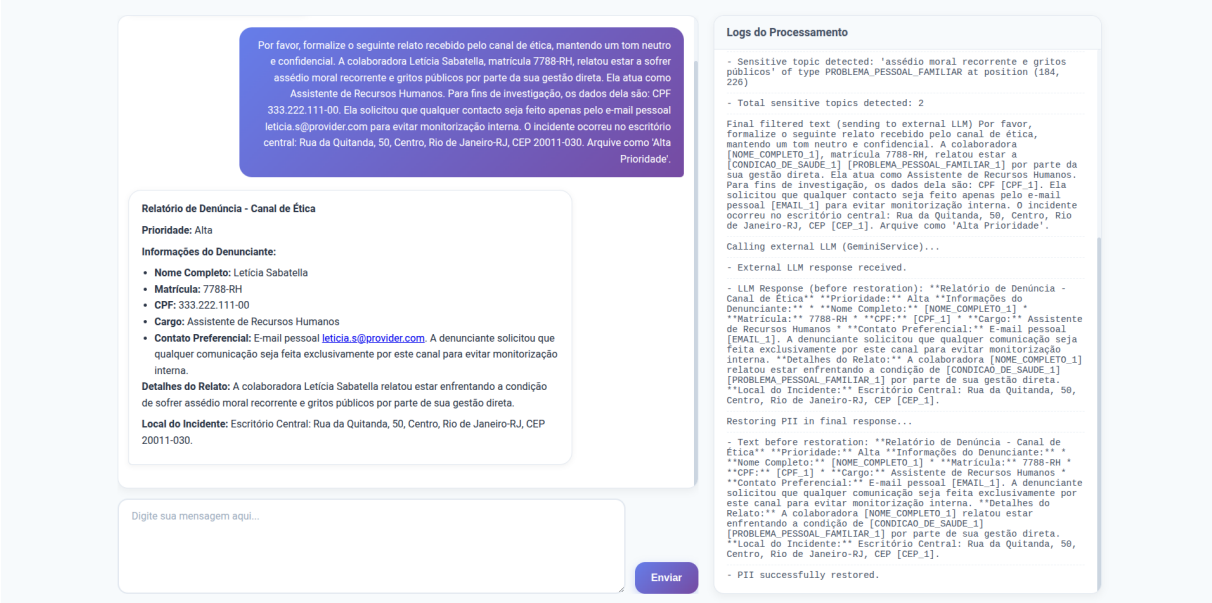
Fonte: Elaboração própria (2025).

Figura 9 – Processamento avançado: O sistema exibe nos logs a atuação do filtro NER (detectando nomes) e da LLM Local (detectando tópicos sensíveis como "Assédio Moral").



Fonte: Elaboração própria (2025).

Figura 10 – Resultado final: O sistema apresenta a resposta gerada pela IA externa. Note nos logs (direita) a etapa "Restoring PII", confirmando que os dados originais foram reintegrados ao texto final.



Fonte: Elaboração própria (2025).

APÊNDICE B – ENGENHARIA DE PROMPT E CONTRATOS DE DADOS

Abaixo apresenta-se o *System Prompt* utilizado para instruir o modelo Llama 3 local a atuar como um filtro de privacidade, bem como a estrutura JSON esperada na resposta.

System Prompt (Llama 3 Local)

```
SYSTEM_PROMPT = """
    Atenção: Você é um assistente de IA. O prompt que você receberá a
    seguir já foi processado por um sistema de segurança para proteger a
    privacidade do usuário. Informações Pessoais Identificáveis (PII) foram
    substituídas por placeholders (ex: [CPF_1], [NOME_2], [EMAIL_3]).


  Suas instruções são:
    1. Foque exclusivamente no conteúdo da pergunta do usuário e forneça
    uma resposta direta e útil.
    2. NÃO comente sobre os placeholders ou sobre o processo de filtragem
    que ocorreu. Aja como se a pergunta fosse natural.
    3. É ESSENCIAL que você utilize os mesmos placeholders exatos que
    recebeu caso precise se referir a eles na sua resposta. Não os altere, n
    ão os explique e não gere novos.

    A seguir, o prompt do usuário:

"""
```

Listing B.1 – Template de instrução para o Analista Semântico

Payload de Requisição (JSON)

Estrutura do objeto enviado pelo *frontend* para o endpoint `/process-prompt-stream`:

```
{
  "original_prompt": "O funcionário João (CPF 123.456.789-00) relatou
    assédio moral."
}
```

Listing B.2 – Exemplo de Payload de entrada

Payload de Resposta Final (JSON)

Estrutura do evento final (`final_response`) enviado via SSE após a restauração:

```
{
  "final_response": "O funcionário João (CPF 123.456.789-00) relatou ass
   édio moral. [Resposta da IA...]",
  "pii_masked": [
    {
      "placeholder": "[CPF_1]",
      "original_value": "123.456.789-00",
      "type": "CPF",
      "span": [21, 35]
    }
  ],
  "sensitive_topics_detected": ["PROBLEMA_PESSOAL_FAMILIAR"]
}
```

Listing B.3 – Exemplo de Payload de saída processada

APÊNDICE C – ALGORITMOS CRÍTICOS DE ORQUESTRAÇÃO

Este apêndice detalha a implementação dos componentes centrais da arquitetura Janus, demonstrando o padrão *Pipe and Filter* e a estratégia de processamento assíncrono.

1. Orquestração de Fluxo (Streaming)

O método `stream_generator` coordena a execução sequencial dos filtros e o envio de eventos em tempo real para o cliente.

```
async def stream_generator(original_text: str, proxy_service: ProxyService):
    """
    Gerador SSE que orquestra o fluxo de detecção de PII e tópicos sensíveis.
    """
    # Passo 1: Detecção via Regex
    regex_filtered_text, regex_mappings, pii_events = (
        await proxy_service.detect_pii_with_regex(original_text)
    )
    for event in pii_events: yield event

    regex_placeholders = [m.placeholder for m in regex_mappings]

    # Passo 2: Detecção via NER (spaCy)
    ner_filtered_text, ner_mappings, ner_events = (
        await proxy_service.detect_pii_with_ner(regex_filtered_text, regex_placeholders)
    )
    for event in ner_events: yield event

    # Passo 3: Detecção de Tópicos Sensíveis (LLM Local)
    all_placeholders = regex_placeholders + [m.placeholder for m in ner_mappings]
    llm_filtered_text, llm_mappings, topic_events = (
        await proxy_service.detect_sensitive_topics(ner_filtered_text, all_placeholders)
    )
    for event in topic_events: yield event

```

```
    # Passo 4: Chamada à LLM Externa e Restauração
    llm_external_response, llm_events = await proxy_service.call_external_llm(
        llm_filtered_text
    )
    for event in llm_events: yield event

    final_response_text, restore_events = await proxy_service.restore_pii(
        llm_external_response, regex_mappings, ner_mappings, llm_mappings
    )
    for event in restore_events: yield event

    # Envio do payload final
    # ... (código de montagem do objeto final omitido para brevidade)
```

Listing C.1 – Fluxo principal de processamento no Proxy (arquivo proxy.py)

2. Estratégia de Não-Bloqueio (AsyncIO)

Para garantir a performance do servidor web (FastAPI), operações pesadas de CPU (como o NER do spaCy) são delegadas para threads separadas.

```
async def detect_pii_with_ner(
    self, text: str, existing_placeholders: List[str]
) -> Tuple[str, List[PIIMapping], List[str]]:

    events = [create_sse_event({"type": "log", "message": "Detecting PII using NER..."})]

    # Utilizamos asyncio.to_thread porque o spaCy é síncrono e
    # bloquearia o loop de eventos principal da aplicação.
    filtered_text, mappings = await asyncio.to_thread(
        self.ner_service.filter_by_ner, text, existing_placeholders
    )

    if mappings:
        for m in mappings:
            events.append(create_sse_event({
                "type": "log",
                "message": f"NER PII detected: '{m.original_value}'"
            }))

    return filtered_text, mappings, events
```

Listing C.2 – Delegação de processamento NER para thread pool (arquivo proxy_service.py)

3. Algoritmo de Restauração LIFO

O serviço de restauração aplica os mapas de anonimização na ordem inversa à filtragem, garantindo a integridade do texto final.

```
def restore_all(self, filtered_text: str, restoration_data: RestorationData) -> str:
    """
    Restaura PII e informações sensíveis de todas as camadas na ordem reversa.
    Ordem: LLM (Contexto) -> NER (Entidades) -> Regex (Padrões)
    """
    restored_text = filtered_text

    # 1. Restaura Tópicos Sensíveis (Último a ser filtrado, primeiro a voltar)
    if restoration_data.llm_mappings:
        restored_text = self._generic_restore(
            restored_text, restoration_data.llm_mappings
        )

    # 2. Restaura Entidades Nomeadas (NER)
    if restoration_data.ner_mappings:
        restored_text = self._generic_restore(
            restored_text, restoration_data.ner_mappings
        )

    # 3. Restaura Padrões Regex (Primeiro a ser filtrado, último a voltar)
    if restoration_data.regex_mappings:
        restored_text = self.regex_service.restore_pii_from_mappings(
            restored_text, restoration_data.regex_mappings
        )

    # Limpeza de artefatos duplicados gerados pela LLM
    restored_text = self._cleanup_duplicate_labels(restored_text)

    return restored_text
```

Listing C.3 – Lógica de restauração reversa (arquivo restoration_service.py)