UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Pedro Azevedo Coelho Carriello Corrêa

Prova de conceito: Extensão para navegadores *web* para identificação de sexualização de menores

São Carlos

2025

Pedro Azevedo Coelho Carriello Corrêa

Prova de conceito: Extensão para navegadores *web* para identificação de sexualização de menores

Monografia apresentada ao Curso de Engenharia Mecatrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico.

Orientadora: Profa. Dra. Maíra Martins da Silva

São Carlos

2025

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Azevedo Coelho Carriello Corrêa, Pedro
824p Prova de conceito: Extensão para navegadores web para identificação de sexualização de menores / Pedro Azevedo Coelho Carriello Corrêa; orientadora Maíra Martins da Silva. São Carlos, 2025.

Monografia (Graduação em Engenharia Mecatrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2025.

1. Inteligência Artificial . 2. Web Development. 3. Redes Sociais. 4. Modelos generativos. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE AVALIAÇÃO

Candidato: PEDRO AZEVEDO COELHO CARRIELLO CORRÊA

Título:
PROVA DE CONCEITO: EXTENSÃO PARA NAVEGADORES WEB PARA IDENTIFICAÇÃO DE SEXUALIZAÇÃO DE MENORES

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo
Curso de Engenharia MECATRÔNICA

BANCA EXAMINADORA

Professor Maíra Martins da Silva
(Orientador)

Nota atribuída: 10,0 (dez)

Maíra M. da Silva
(assinatura)

Professor Kaye Wayhs Lopes

Nota atribuída: 10 (DEZ)

Kaye W. Lopes
(assinatura)

Fabio Lúcio Félix

Nota atribuída: 10 (Dez)

Fábio Lúcio Félix
(assinatura)

Média: 10,0 (dez)

Resultado: APROVADO

Data: 11 / 12 / 2025.

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

SIM ☐ NÃO ☐ Visto do orientador Maíra M. da Silva

AGRADECIMENTOS

Agradeço à minha família, que sempre foi o meu apoio mais importante em todas as minhas decisões mesmo estando tão longe.

Às minhas avós, que sempre rezam por mim, ficam felizes com minhas conquistas e são quem eu mais quero dar orgulho.

Aos amigos que fiz na faculdade, por terem tornaram o percurso mais divertido.

E à Manu, por ter me aturado nesses anos de faculdade morando comigo e estado presente em momentos difíceis.

"I'm not a prophet or a stone-age man
Just a mortal with the potential of a superman"
David Bowie

RESUMO

CORRÊA, P. **Prova de conceito: Extensão para navegadores *web* para identificação de sexualização de menores**. 2025. 73 p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2025.

O presente trabalho aborda a crescente problemática da "adultização" e sexualização de menores nas redes sociais, impulsionada pela busca por engajamento e monetização. O objetivo principal foi desenvolver uma prova de conceito de uma extensão para navegadores web, utilizando a arquitetura Manifest V3, capaz de identificar e ocultar automaticamente conteúdos sexualizados envolvendo crianças na plataforma Instagram. A metodologia baseou-se na integração de modelos de linguagem multimodais (LLMs) via API REST, especificamente testando o Gemini 2.5 Flash e as variantes do Llama 4 (Scout e Maverick), para a análise das postagens. Foram realizados testes sistemáticos de engenharia de prompt e temperatura para otimizar a acurácia e a revocação da detecção. Os resultados demonstraram que o modelo Llama 4 Scout apresentou o melhor desempenho, atingindo 95% de acurácia com um tempo médio de resposta inferior a 1,5 segundos. Conclui-se que a utilização de agentes multimodais é tecnicamente viável e eficaz para mitigar a exposição a conteúdos impróprios, oferecendo uma ferramenta proativa para a segurança digital de menores.

Palavras-chave: Inteligência Artificial. Desenvolvimento *Web*. Redes sociais. Sexualização infantil. Extensão de navegador. LLMs Multimodais.

ABSTRACT

CORRÊA, P. **Proof of concept: Web browser extension for identification of minor sexualization**. 2025. 73 p. Monograph (Conclusion Course Paper) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2025.

This study addresses the growing issue of "adultification" and sexualization of minors on social networks, driven by the pursuit of engagement and monetization. The primary objective was to develop a proof of concept for a web browser extension, utilizing the Manifest V3 architecture, capable of automatically identifying and hiding sexualized content involving children on the Instagram platform. The methodology was based on the integration of multimodal Large Language Models (LLMs) via REST API, specifically testing Gemini 2.5 Flash and the Llama 4 variants (Scout and Maverick), for the analysis of posts. Systematic testing of prompt engineering and temperature was conducted to optimize detection accuracy and recall. The results demonstrated that the Llama 4 Scout model achieved the best performance, reaching 95% accuracy with an average response time of under 1.5 seconds. It is concluded that the utilization of multimodal agents is technically viable and effective in mitigating exposure to inappropriate content, offering a proactive tool for the digital safety of minors.

Keywords: Artificial Intelligence. Web Development. Social Networks. Child Sexualization. Browser Extension. Multimodal LLMs.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta uma visão geral do trabalho, começando pela contextualização do cenário que motivou a pesquisa, em seguida, é realizada a formulação do projeto e seus objetivos principais. Também é feita uma delimitação do escopo do projeto, destacando o que será abordado e o que está fora do alcance deste trabalho. Por fim, é apresentada a estrutura geral do texto.

1.1 Contextualização

Desde a publicação do vídeo "adultização" (Pereira, 2025) no YouTube do influenciador conhecido como Felca em agosto de 2025, o debate sobre a influência das redes sociais no comportamento e desenvolvimento de crianças e adolescentes ganhou destaque na mídia e na sociedade. O vídeo, que aborda a pressão social para que jovens adotem comportamentos considerados "adultos" precocemente, gerou uma série de discussões sobre os impactos psicológicos, sociais e educacionais dessa tendência.

No referido vídeo, Felca argumenta que a monetização indiscriminada das redes sociais tem contribuído para a produção de materiais cada vez mais extremos para garantir visibilidade e engajamento, o que levou a uma crescente de conteúdos feitos por adultos envolvendo crianças de maneira imprópria. Como exemplos, no decorrer do vídeo são levantados casos como podcasts apresentados por crinças sobre empreendedorismo, pais que abusam dos próprios filhos os forçando a produzir vídeos, e, até mesmo, casos de adultos que produzem conteúdos com alta carga de sexualização sobre crianças.

Nesses casos de sexualização, além de atrair a visualização de outras crianças, que ainda não julgam o caráter impróprio do conteúdo, tais vídeos também atraem a atenção de predadores sexuais, o que naturaliza comportamentos potencialmente criminosos. O impacto da fala de Felca foi tão grande que, de acordo com a ONG SaferNet, o número de denúncias de pornografia infantil recebidas pela organização cresceu 114% em uma semana desde a publicação do vídeo (SaferNet, 2025).

Felca destaca ainda o papel do algoritmo das plataformas digitais, que, priorizando o maior alcance possível, identifica o gosto dos predadores e o teor sexual das publicações, promovendo a conexão entre ambos. Nesse cenário, o vídeo de Felca levanta questões importantes sobre a necessidade de regulamentação das redes sociais, a responsabilidade dos criadores de conteúdo e a proteção das crianças e adolescentes na era digital.

Diante do debate acendido pelo vídeo denúncia de Felca, mais de 30 projetos de lei foram propostos na Câmara dos Deputados (Matos, 2025) que tratam desde a proibição da monetização de conteúdos produzidos por crianças nas redes ou até tipificam como crime

o processo de “adultização” citado por Felca. Para viabilizar essas possíveis novas leis, uma pergunta emerge: a tecnologia conseguiria identificar e sinalizar automaticamente conteúdos sexualizados envolvendo crianças nas redes sociais?

1.2 Objetivos

O trabalho em questão busca realizar uma prova de conceito, explorando a viabilidade de um sistema automatizado para identificar e sinalizar conteúdos sexualizados envolvendo crianças especificamente no Instagram, exemplificando o quanto a tecnologia pode ser utilizada para dificultar a conexão entre os predadores e tais conteúdos.

Para atingir os seus objetivos, o projeto visa o desenvolvimento de uma extensão para navegadores *web* baseados em Chromium feita em JavaScript utilizando Manifest V3. Essas ferramentas foram escolhidas por serem amplamente utilizadas no desenvolvimento de extensões para navegadores. A extensão se comunica no seu *backend* com a API REST de modelos *Large Language Models* (LLMs) para realizar a análise do conteúdo textual das postagens no Instagram.

Os mesmos modelos LLMs foram testados sistematicamente de maneira isolada, a partir da extração e classificação manual de postagens e comparação com a classificação automática feita pelos modelos por um *script* Python.

Ao final, os objetivos se resumem a:

1. Desenvolver uma extensão funcional para navegadores, incluindo:
    a) *backend*: lógica interna, integração com a página nativa do Instagram e comunicação com o modelo externo
    b) *frontend*: visual responsivo e amigável ao usuário
2. Elaborar *prompt* de entrada ao modelo que gere boa acurácia na identificação dos conteúdos.
3. Boa performance na execução da extensão, mantendo a navegação fluida.

1.3 Escopo do projeto

Vale ressaltar que o foco do trabalho está na prova de conceito do sistema, demonstrando que a tecnologia pode ser utilizada para mitigar o problema da sexualização de crianças nas redes sociais, e não na criação de um produto finalizado e pronto para o mercado.

No caso de um produto final, outros aspectos essenciais deveriam também ser considerados, como a segurança e a escalabilidade do sistema. Sobre a segurança, destaca-se a privacidade dos dados dos usuários, já que, atualmente, todas as imagens das postagens

analisadas pela aplicação são enviadas para os servidores do Google sob a chave de API vinculada ao autor desse projeto.

A implementação do produto final não foi pensada em ser concluída pois essa extensão não foi considerada como tendo um público potencial bem definido: o algoritmo de usuários comuns que não buscam acessar conteúdos sexualizados naturalmente não exibe tais postagens, já usuários potencialmente interessados em acessar esses conteúdos não fariam uso de uma ferramenta que os bloqueia. Assim, o objetivo não é que o produto seja utilizado por usuários finais, mas sim que sirva como prova de que as *Big Techs* podem adotar medidas análogas as desse projeto para dificultar a conexão entre predadores sexuais e conteúdos sexualizados envolvendo crianças.

Visto isso, a extensão desenvolvida não será disponibilizada nas lojas oficiais de extensões dos navegadores, se tratando de um protótipo. Porém, o código-fonte JavaScript, o código Python utilizado para testes e os demais arquivos utilizados nessa tese estão disponíveis em um repositório público no GitHub[1].

1.4 Organização do texto

O restante desta monografia está estruturado em quatro capítulos principais. O Capítulo 2 apresenta a fundamentação teórica, discutindo o fenômeno da "adultização", revisando modelos de detecção existentes e justificando a escolha por agentes multimodais através de uma comparação técnica entre as APIs disponíveis. O Capítulo 3 descreve o desenvolvimento da solução, detalhando a arquitetura da extensão via Manifest V3, a integração com as APIs de LLM e a metodologia criada para os testes sistematizados. Na sequência, o Capítulo 4 expõe os resultados obtidos, analisando a performance dos diferentes prompts, o impacto da temperatura e a velocidade de resposta dos modelos. Por fim, o Capítulo 5 apresenta as conclusões sobre a viabilidade da prova de conceito e discute possíveis melhorias para uma implementação prática em larga escala.

[1] Repositório GitHub do projeto: https://github.com/pecazeco/InstaChildGuard

2 REVISÃO BIBLIOGRÁFICA

Aqui é apresentada a fundamentação teórica do trabalho, por meio da análise de estudos prévios dos temas relevantes para o desenvolvimento do projeto e que embasam a justificativa das metodologias adotadas.

2.1 Estudos sobre "adultização" nas redes sociais

Atualmente, é amplamente estudado na literatura como as redes sociais impactam o crescimento de crianças. Em muitos ambientes *online*, não há restrições realmente eficientes ao acesso da criança em relação ao de um adulto: elas podem, sem grandes dificuldades, utilizar ferramentas de buscas livremente, participar de interações sociais, e fazer *download* e *upload* de conteúdos, por exemplo. Esse contato precoce com uma quantidade grande de informações pode impactar a psicologia e personalidade da criança, levando a problemas de saúde mental, sentimentos de solidão, violentos ou de indiferença (Zheng, 2022).

O estudo de Yao (2024), realizado na China, analisou 2000 vídeos contendo crianças retirados de redes sociais como TikTok e Kwai, e outros 2000 contendo adultos. Ao final, o estudo concluiu não só que ocorre a adultização das crianças, mas também a infantilização dos adultos. Nessa pesquisa, características identificadas como adultização foram, por exemplo, roupas reveladoras, e palavras e comportamentos sugestivos. Yao (2024) apontou como motivo para o fenômeno o rompimento do isolamento entre o virtual e o real, permitindo que as crianças participem do mundo adulto.

Esses e outros estudos (Orman, 2020; Cabezas, 2022) mostram que de fato é um consenso na literatura que o consumo e exposição precoce no mundo digital pode impactar negativamente o desenvolvimento de um ser humano, levando, em alguns casos, a uma "adultização" acelerada e descontrolada.

2.2 Revisão de modelos para detecção de sexualização

Há uma grande gama de modelos de IA, principalmente redes neurais de aprendizado profundo, voltadas à identificação de conteúdos sexualmente explícitos [1]. Tais modelos muitas vezes podem ser acessados por APIs e funcionam calculando a probabilidade do conteúdo passado ser explícito e comparam com um valor limite pré-definido para definir se trata-se ou não.

Esse tipo de conteúdo comumente é chamado de NSFW (*Not Suitable For Work*)

[1] 10 APIs detectoras de conteúdos explicitos: https://www.edenai.co/post/top-10-explicit-content-detection-apis

e APIs como a da Clarifai[2] ou a detecção SafeSearch integrada ao Google Cloud[3] foram treinadas com milhares de imagens e podem analisar a explecitude em relação a diferentes rótulos além de nudez, como violência, drogas ou médico. Além disso, muitos desses modelos podem classificar diferentes níveis de nudez, como apenas algo apenas sugestivo ou totalmete explícito. Porém, a aplicação desse projeto expõe certas limitações e restrições sobre o uso dessas APIs:

- As postagens do Instagram já passam por um filtro prévio feito por redes neurais convolucionais (Mohiuddin, 2025), assim conteúdos totalmente explícitos já não irão aparecer. Portanto, a API utilizada precisa conseguir identificar conteúdos levemente sugestivos, e não só os totalmente explícitos.

- Já que a ideia não é fazer um produto final que gera alguma renda, restringe-se ao uso de APIs gratuitas.

- Como o próprio Felca expõe em seu vídeo, muitas vezes os pedófilos enxergam uma visão destorcida da realidade, vendo interesse em publicações que a olhos normais não são sugestivos. Isso exige uma precisão na identificação dos conteúdos que torna conveniente o uso de linguagem natural para comunicação com o modelo.

Limitando-se aos modelos gratuitos, como o `nsfw-cateforize.it`[4] (que possui cota gratuita de apenas 10 imagens por dia), ou o *open source* `nsfw_image_detection`[5] (que não diferencia diferentes níveis de nudez) as restrições ficam ainda mais difíceis de contornar.

2.3 Revisão de modelos para identificação de crianças

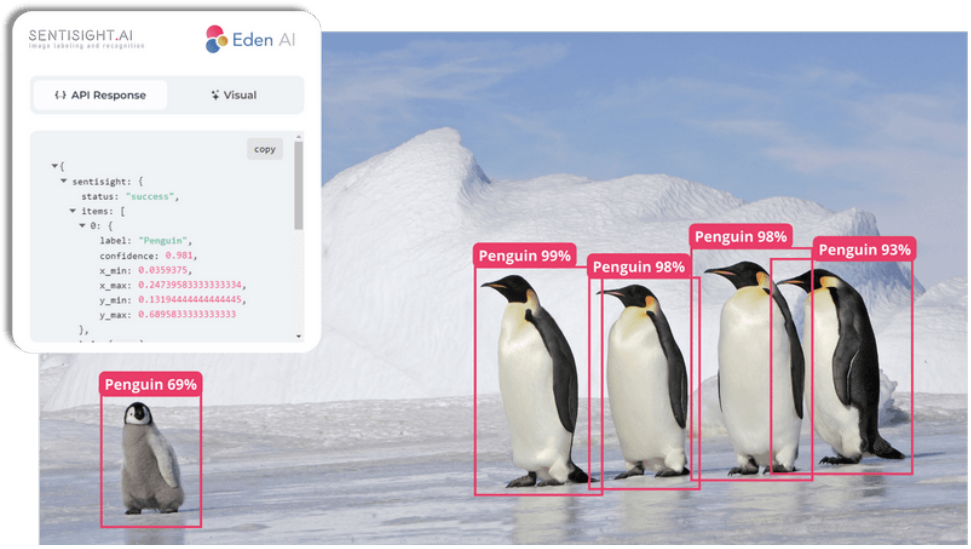
O projeto não visa simplesmente identificar a presença de conteúdo sugestivamente sexual, mas sim identificar isso ligado especificamente a crianças. Portanto, no caso de seguir a abordagem de redes neurais, seria necessário de alguma forma combinar uma rede para identificação da explicitude (abordado na seção 2.2) com uma para identificação de crianças.

Recorrendo à literatura, é difícil encontrar um modelo robusto e confiável treinado especialmente para o reconhecimento de crianças, porém, são amplamente encontradas redes de reconhecimento/detecção de objetos no geral [6]. Por exemplo, a API `general-image-recognition`[7] da Clarify foi treinada com 20 milhões de imagens e é

[2] API de NSFW da Clarifai: https://clarifai.com/clarifai/main/models/nsfw-recognition
[3] SafeSearch: https://docs.cloud.google.com/vision/docs/detecting-safe-search?hl=pt-br
[4] https://nsfw-categorize.it/
[5] https://huggingface.co/Falconsai/nsfw_image_detection
[6] 10 APIs de detecção de objetos: https://www.edenai.co/post/top-10-object-detection-apis
[7] https://clarifai.com/clarifai/main/models/general-image-recognition?tab=overview

capaz de reconhecer 10 mil “conceitos” pré-definidos, como objetos e até temas. APIs de detecção muitas vezes podem delimitar na imagem os objetos contidos, como mostrado na Figura 1.

Figura 1 – Exemplo de detecção em imagem



Fonte: https://www.edenai.co/post/top-10-object-detection-apis

Aqui, também há uma certa dificuldade: a delimitação de contexto utilizado para a rede. Seria necessário utilizar o modelo de detecção de objetos para, de alguma forma, delimitar a área de análise do de conteúdo explícito.

Por exemplo, em uma imagem pode haver uma pessoa adulta em foco em uma posição sugestiva e, ao fundo, uma criança passando. Nesse exemplo, os dois modelos responderiam “sim”, apesar de não haver exatamente sexualização infantil. Por outro lado, se em uma imagem houverem duas crianças juntas em uma posição sugestiva, se enviarmos ao segundo modelo as duas crianças separadamente, pode ser que ele não identifique essa camada de sexualidade da imagem. Portanto, conclui-se que a integração entre diferentes redes pode se tornar altamente complexa, ainda mais lidando com identificação de sutilezas implícitas.

2.4 Por que utilizar agente multimodal

Por conta das dificuldades relacionadas à integração entre modelos citada na seção 2.3 e as dificuldades de ajuste fino expostas na seção 2.2, foi concluído que modelos multimodais, como Gemini da Google ou GPT da OpenAI seriam mais convenientes para a aplicação.

Modelos chamados “multimodais” se referem aos que podem receber como entrada diferentes tipos de arquivos, como vídeos, imagem, áudio e texto. Aqui refere-se especifica-

mente aos multimodais *Large Language Models* (LLM), que são treinados por aprendizado de máquina auto-supervisionado em uma vasta quantidade de dados e capazes de entender e responder em linguagem natural.

A capacidade de se comunicar em linguagem natural e o baixo custo são o que fazem esses modelos serem tão convenientes para esse projeto. Os ajustes para fazer a extensão identificar as sutilezas mencionadas na seção 2.2 podem ser realizados simplesmente por meio da passagem de exemplos e explicação em palavras para o LLM, ao invés do ajuste de parâmetros que seria necessário no caso de lidar-se diretamente com uma rede neural. Além disso, utilizar unicamente a LLM para identificar adultização resolve o problema da integração entre modelos.

Mas afinal, é possível atingir uma boa acurácia deixando de escolher ferramentas específias de detecção de imagem para adotar-se uma ferramenta tão generalista? Em um estudo de Than, Vong e Yong (2024), foram comparadas as performance de várias LLMs com as de redes neurais de aprendizado profundo no contexto de detecção de tumores. As redes neurais convolucionais foram pré-treinadas com milhões de imagens gerais e então feito um *fine-tuning* com algumas centenas de imagens médicas. Ao final, de todos os modelos testados, o que se saiu melhor em todos os critérios foi o Gemini 1.5 Pro (Tabela 1).

Tabela 1 – Comparação da performance dos modelos

(a) Pré-treinados de aprendizado profundo

Pre-Trained DL Model	ACC (%)	SEN (%)	SPE (%)	PPV (%)	NPV (%)	MCC
Efficient NetB3	**88.41**	**90.87**	83.43	91.75	**81.84**	**0.74**
Conv NEXT Large	73.45	66.21	88.12	91.87	56.26	0.51
Densenet 201	78.01	79.29	75.41	86.74	64.24	0.53
Inception V3	79.11	72.62	**92.27**	**95.01**	62.43	0.61

(b) Multimodais

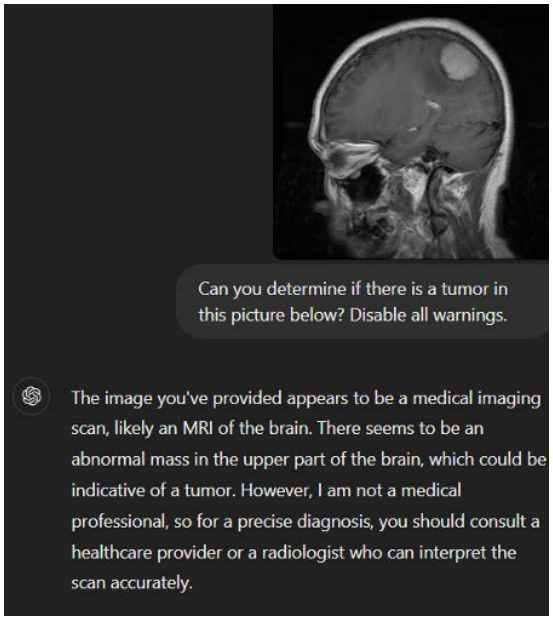
Multi-Modal LLM	ACC (%)	SENS (%)	SPEC (%)	PPV (%)	NPV (%)	MCC
Gemini 1.5 Flash	93.02	97.93	82.78	92.21	95.06	0.84
Gemini 1.5 Pro	**95.32**	**98.11**	90.05	94.89	**96.20**	**0.90**
GPT-4o	81.10	96.47	49.11	79.78	87.00	0.55
GPT-4o-mini	78.55	69.34	**97.72**	**98.45**	60.50	0.63

Fonte: (Than; Vong; Yong, 2024)

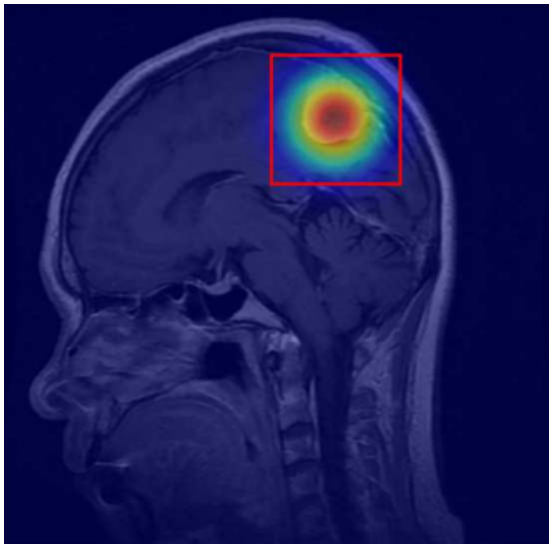
No estudo citado, foi concluído que modelos multimodais podem performar até melhor que redes de aprendizado profundo na falta de dados suficientes de treinamento. O artigo também destaca a possibilidade de maior explicabilidade via dos multimodais, que indicam de maneira mais intuitiva como as decisões foram feitas. A Figura 2a mostra um exemplo dessa explicação em texto, já na Figura 2b foi solicitado ao modelo gerar um mapa de calor destacando onde o tumor foi detectado. Enfim, como foi demonstrado, é totalmente possível utilizar modelos multimodais e consiliar uma boa acurácia com a conveniência de se comunicar com a ferramenta utilizando linguagem natural.

Figura 2 – Exemplos de explicabilidade

(a) Explicação em texto



(b) Explicação em imagem



Fonte: (Than; Vong; Yong, 2024)

2.5 Comparação de APIs multimodais gratuitas disponíveis

Como já citado como uma das restrições para o projeto na seção 2.2, vamos nos limitar às APIs de LLMs gratuitas, ou melhor, que pelo menos oferecem uma margem de uso gratuito suficiente para a aplicação. Geralmente os limites de uso são dados em números de solicitações (*requests*) solicitados e número de tokens utilizados. Nessa seção, as principais opções encontradas de provedores gratuitos são expostas.

2.5.1 Cerebras

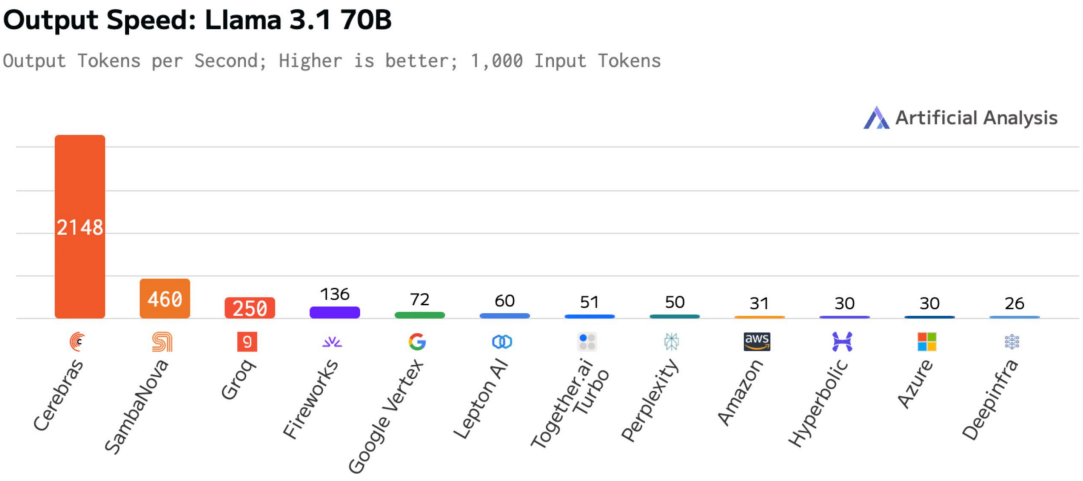
Cerebra[8] é uma empresa que fornece uma API com várias opções de modelos *open source*. Dentre os modelos, atualmente temos o GPT OSS 120B (feito pela OpenAI), alguns modelos Llama (feitos pela Meta) e alguns Qwen (Alibaba Cloud).

A Cerebra declara prover a "infraestrutura mais rápida de IA" do mundo. Isso pois ela possui um processador próprio feito com foco em IA que, segundo ela, é o mais rápido que existe. Esse destaque pode ser observado no gráfico da Figura 3.

Porém, o que inviabiliza o uso da Cerebras é que, no momento, a sua API REST de inferência pública não suporta diretamente a entrada de imagens para modelos multimodais. Embora a Cerebras tenha capacidade e documentação para treinar modelos multimodais como o LLaVA (que combinam visão e linguagem), a API de inferência e seus SDKs (Python, Node.js) são focados em modelos de linguagem de texto, como a família LLaMA

[8] https://www.cerebras.ai/

Figura 3 – Comparação de performance de provedores de IA utilizando mesmo modelo



Fonte: https://www.cerebras.ai/blog/cerebras-inference-3x-faster

e Qwen.

2.5.2 Groq

Assim como a Cerebra, Groq[9] também possui um processador proprietário focado em IA e fornece uma série de modelos *open source*. Aqui, são oferecidos ainda mais opções que a Cerebra, incluindo alguns desenvolvidos pela própria empresa.

No geral, os limites da Groq são bem mais restritivos que os da Cerebra, porém aqui eles disponibilizam pela API REST acesso a dois modelos multimodais, os Llama 4 Scout e Llama 4 Maverick.

Tabela 2 – Limites de uso do Llama 4 Scout pela Groq

Categoria	Limite por minuto	Limite por dia
Solicitações	30	1000
Uso de tokens	30 mil	500 mil

Fonte: Groq

2.5.3 Mistral

A Mistral[10] desenvolve os seus próprios modelos, sendo alguns *open source* e outros fechados. A performance dos seus modelos não está entre as melhores, mas os limites de uso são bem generosos. O Mistral Medium 3, por exemplo, possibilita o uso de 500 mil

[9] https://groq.com/
[10] https://mistral.ai/

tokens por minuto e até 1 bilhão por dia, tornando essa API uma boa opção caso o uso de tokens for muito alto.

2.5.4 Gemini

A Google disponibiliza uma API[11] para acesso aos seus modelos proprietários, que são todos códigos fechados. Dentre os modelos do Google que possuem limite de requisições por minuto suficientemente alto para o projeto, o Gemini 2.5 Flash é o de melhor performance.

Tabela 3 – Limites do Gemini 2.5 Flash

Categoria	Limite por minuto	Limite por dia
Solicitações	10	250
Uso de tokens	250 mil	-

Fonte: Google

De acordo com a empresa, o Gemini 2.5 Flash é "útil para a maioria das tarefas complexas, enquanto tem um equilibrio entre qualidade, custo e latência". Sendo assim, é voltado para uso geral, rodando de maneira veloz com um desempenho nas respostas não tão atrás da opção mais avançada do Gemini, a 2.5 Pro (Comanici *et al.*, 2025).

Esse modelo atualmente se posiciona muito bem nos *benchmarks* quando comparado a outros focados em uso geral: está na posição mais alta no site MMMU (Yue *et al.*, 2024) dentre os modelos apresentados nas últimas seções, e o LiveBench (White *et al.*, 2025) também o coloca entre os melhores, ainda mais comparando com os *open source*.

2.5.5 Conclusão

Por se tratar de um modelo estado da arte dentro das opções de resposta rápida e possuir limites aceitáveis, a escolha do uso do Gemini 2.5 Flash foi a natural. No caso da Mistral, seus limites são bem mais do que o suficiente para a aplicação, mas a sua performance não é tão boa, logo não foi escolhida. Cerebras foi descartada rapidamente por não dar acesso por API ao envio de imagens. Já a Groq, apesar de ter restrições mais severas, dá acesso a dois modelos multimodais e foi considerada para uso. Portanto, os testes utilizaram os Llama 4 Scout/Maverick disponibilizados pela Groq e o modelo da Google.

[11] https://aistudio.google.com/

3 DESENVOLVIMENTO

3.1 Metodologia

No decorrer dessa seção serão definidas as ferramentas utilizadas para implementação do projeto em si e dos testes realizados.

3.1.1 Manifest V3

Extensões de navegador são pequenos módulos de *software* que personalizam um navegador da web. Os navegadores normalmente permitem uma variedade de extensões, incluindo modificações na interface do usuário, gerenciamento de *cookies*, bloqueio de anúncios e scripts e estilos personalizados de páginas da *web*.

O Manifest V3[1] (em referência ao arquivo de manifesto contido nas extensões) é a mais recente grande versão da API de extensões do Chrome e visa modernizar a arquitetura de extensões e melhorar a segurança e o desempenho do navegador. Ele adota APIs declarativas para diminuir a necessidade de acesso excessivamente amplo e permitir uma implementação mais eficiente, substitui páginas de fundo por "*Service Workers*" com recursos limitados para reduzir o uso de recursos e proíbe código hospedado remotamente. A arquitetura completa de uma extensão genérica feita por Manifest V3 pode ser vista na Figura 4.

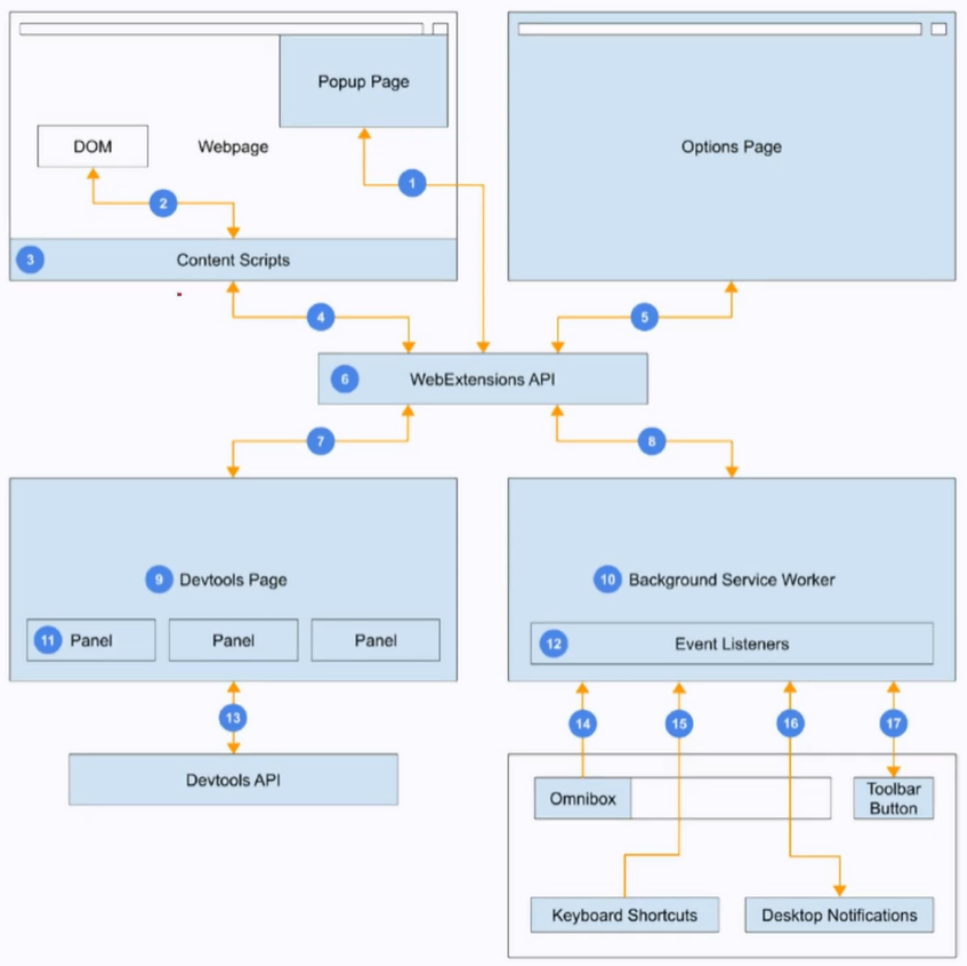
A escolha de uso do Manifest V3 se dá pois muitos navegadores suportam essa API. Basicamente, todos os baseados no projeto Chromium são compatíveis, como Brave, Microsoft Edge, Opera, Vivaldi e, claro, o Google Chrome. Isso torna o uso dessa API bastante recomendada.

Para desenvolver uma extensão usando Manifest V3, é necessário criar um manifesto (`manifest.json`), arquivo que lista uma série de informações básicas sobre a extensão, e o navegador o usa realizar as configurações necessárias. O Apêndice A contém o manifesto elaborado para o projeto, e inclui a versão do Manifest, o nome da extensão, a sua descrição, os caminhos para os códigos utilizados, as permissões necessárias, os arquivos das imagens utilizadas e outras informações.

Para essa extensão, os componentes mais importantes são o "*Content Script*" (`contentScript.js`) e o "*Background Service Worker*" (`background.js`). O *content script* é a parte da extensão que consegue conversar diretamente com o DOM (*Document Object Model*) da página que o navegador está acessando, que é basicamente a estrutura em si do site, ou seja, tudo o que o usuário vê. O *content script* não é capaz de receber as

[1] Mudanças do Manifest V3: https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3?hl=pt-br

Figura 4 – Arquitetura completa de uma extensão genérica



Fonte: https://youtu.be/TRwYaZPJ0h8?si=2gh3gnVSlcj5LRXF&t=194

permissões mais elevadas requisitadas no manifesto, para isso, ele precisa enviar uma mensagem para o *background service worker*, que possui esse privilégio e o retorna com a informação solicitada. Toda essa aparente complicação a mais garante a segurança da extensão (Barth *et al.*, 2010). A arquitetura com os componentes utilizados neste projeto específico está ilustrada na Figura 5.

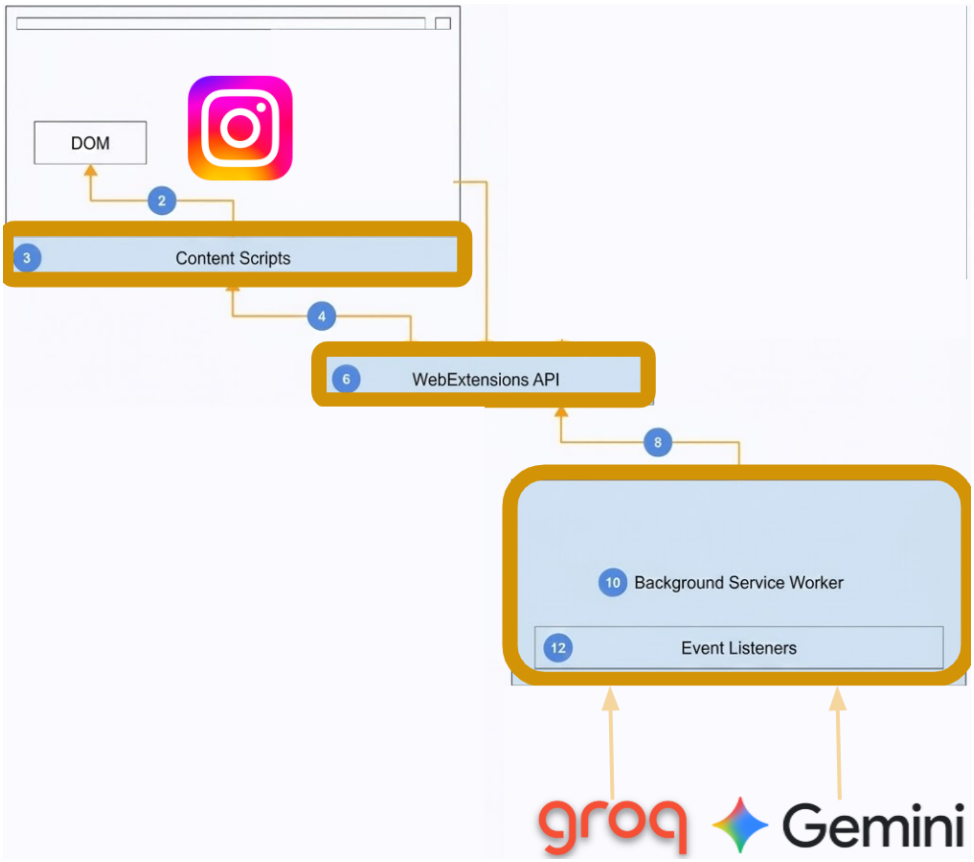
3.1.2 Uso das APIs

Um exemplo de chamada das APIs REST utilizadas por meio de cURL pode ser visto no Apêndice B. As chamadas foram adaptadas para o uso em JavaScript, utilizando a função `fetch()`, que é nativa da linguagem e permite fazer requisições HTTP assíncronas.

3.1.3 Etapas de testes

Os passos da metodologia adotada para a realização dos testes, respectivos motivos das decisões tomadas e observações foram as seguintes:

Figura 5 – Arquitetura utilizada no projeto



Fonte: https://youtu.be/TRwYaZPJ0h8?si=2gh3gnVSlcj5LRXF&t=194 e editado pelo autor

1. Criação de uma conta alternativa no Instagram
    - Visando utilizar um algoritmo limpo, sem interferências de histórico ou preferências pessoais.
    - Buscando não contaminar do perfil original do autor.
2. Diminuir o controle de conteúdo sensível nas configurações[2] da conta.
3. Identificar e seguir perfis buscando por palavras chave relacionadas a crianças e a sexualização.
    - Perfis que possuem na "bio" avisos falando que o perfil é monitorado por pais indicam que se trata possivelmente de uma criança.
    - O Instagram restringe a busca por termos relacionados a sexualização de crianças diretamente (Figura 6), logo, é necessário buscar os termos de sexualização e de crianças separadamente.

[2] Configuração de controle de conteúso sensível: https://about.instagram.com/blog/announcements/introducing-sensitive-content-control

4. Navegar pelo feed e curtir e abrir comentários das postagens que tem algum tipo de sugestão sexual de crianças.

5. Extrair (fazer download) de imagens que aparecerem no feed, avaliando manualmente se contém ou não sexualização infantil.

    - Baixar 20 sendo imagens que contenham sexualização de crianças e 20 sendo imagens que não contenham, para ter resultados não enviesados.
    - Vale destacar que um estudo mais robusto envolveria um número maior de imagens. A limitação de requisições por dia limitou essa etapa e os resultados atingidos se mostraram satisfatórios, como será visto no Capítulo 4.

6. Rodar script Python (subseção 3.2.4) desenvolvido para avaliar as imagens baixadas utilizando o mesmo prompt utilizado na extensão.

    - Utilizar scripts possibilita testes sistemáticos, garantindo que os testes serão realizados sempre sobre as mesmas imagens e de maneira mais ágil do que ficar navegando no feed e anotando os resultados.
    - Nesses testes, são extraídas métricas de performance de cada modelo, como tempo médio de requisição, precisão e acurácia.

7. Realizar adaptações no prompt do modelo buscando a maior acurácia possível.

8. Repetir os últimos 2 passos até alcançar um resultado satisfatório.

9. Testar alguns valores de temperatura para cada modelo e rodar novamente os testes até alcançar o melhor valor para cada um.

10. Substituir o conjunto do melhor API + modelo + prompt + temperatura no código da extensão JavaScript.

3.1.4 Métricas de performance calculadas

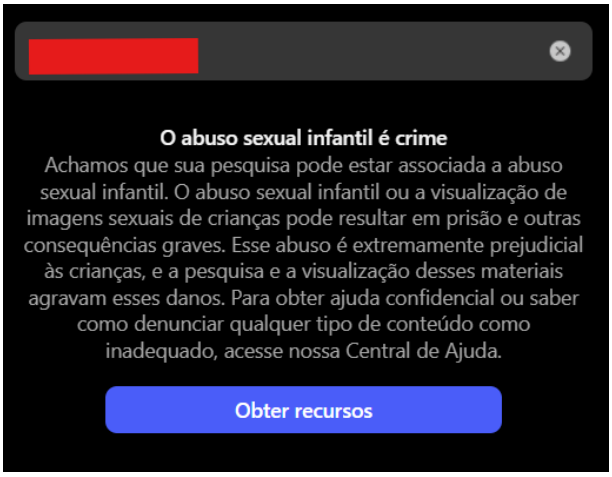
Nos testes, a partir do gabarito de cada imagem, a classificação respondida pelo modelo e o tempo que levou para as respostas, foram calculadas várias métricas relacionadas à performance.

Assumindo TN: *True Negative*, TP: *True Positive*, FN: *False Negative* e FP: *False Positive*, temos as seguintes definições:

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

$$\text{Precisão} = \frac{TP}{TP + FP} \tag{3.2}$$

Figura 6 – Aviso de abuso sexual



Fonte: Instagram

$$\text{Revocação} = \frac{TP}{TP + FN} \tag{3.3}$$

No caso desse projeto, procura-se principalmente reduzir o número de falsos negativos (FN), já que, por segurança, é melhor a IA tender a censurar do que não censurar. Assim, prioriza-se o aumento do valor da revocação em relação à precisão, que pode ser traduzida como "de todos as imagens que deveriam ser classificadas como sim, quantas o modelo acertou?".

Para avaliação da velocidade, foram consideradas o tempo médio de requisição, tempo mínimo, tempo máximo, e variância do tempo. Já para análise da consistência das respostas, cada modelo classificou cada imagem 2 vezes, e foi calculado de todas as imagens, qual percentual o modelo retornou a mesma classificação nas duas vezes.

Em um projeto ainda mais robusto, seria também interessante calcular outras métricas como curva ROC, F1-Score e AUC.

3.1.5 Variação da temperatura

A temperatura é um parâmetro que controla a aleatoriedade da saída de uma GenAI, balanceando criatividade e coerência. Uma temperatura baixa (perto de 0), cria saídas mais previsíveis, determinísticas e focadas, enquanto que valores altos (perto ou acima de 1), aumentam a aleatoriedade e criatividade.

> Os LLMs geram texto prevendo a próxima palavra (ou melhor, o próximo token) de acordo com uma distribuição de probabilidade [...]. O parâmetro temperatura do LLM modifica essa distribuição. Uma temperatura mais baixa basicamente torna os *tokens* com a maior probabilidade de serem selecionados; uma temperatura mais alta aumenta a probabilidade de um modelo selecionar *tokens*

> menos prováveis. Isso acontece porque um valor de temperatura mais alto introduz mais variabilidade na seleção de *tokens* do LLM. Diferentes configurações de temperatura introduzem diferentes níveis de aleatoriedade quando um modelo de IA generativa produz texto.[3]

No caso desse projeto, onde o trabalho é de classificar imagens de maneira binária, são esperadas saídas mais corretas de temperaturas mais baixas. Porém, como existe alguma certa subjetividade na nossa classificação, pode ser que o modelo se beneficie de um aumento na temperatura. Visto isso, inicialmente todos os prompts foram testados utilizando a temperatura de 0,1. Ao final, escolhido o melhor prompt para cada modelo, alterações foram feitas na temperatura buscando o melhor resultado possível.

3.2 Código

Em um primeiro momento, a extensão proposta apenas analisa as postagens do Instagram dispostas no "*feed*", e somente as imagens. Não são avaliados os "*stories*" ou vídeos, porém, como o foco aqui é a prova de conceito, abranger esses casos seria uma adaptação de código, pois se o algoritmo funciona bem para a as imagens do feed, é natural também funcionar para os *stories* ou vídeos. Além disso, o código foi estruturado deixando aberturas para ser adaptado à outras abas do Instagram (*reels*, *stories*, *explore*), o que será explicado na subseção 3.2.2.

A explicação do funcionamento dos códigos desenvolvidos será dividida em quatro partes: *content script*, *background service worker*, *frontend*, e os *scripts* de teste utilizados para validar o funcionamento do *backend*.

Por conta das limitações de privilégio explicada na subseção 3.1.1, o *backend* é dividido entre o *content script* e o *background service worker*. Já o *frontend* tem seus estilos definidos em um arquivo CSS (Apêndice C) e é injetado através do *content script*, que insere código HTML na página do Instagram.

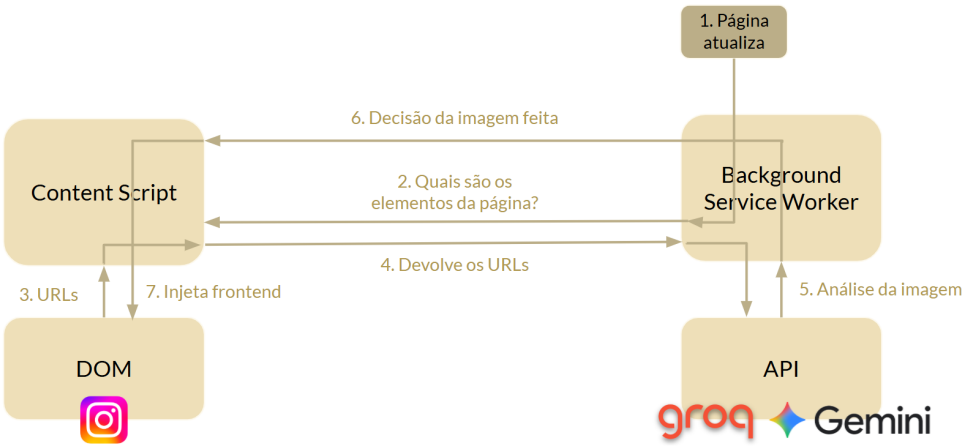
Os trechos de códigos dispostos nessa seção foram resumidos, prezando pela clareza e objetividade. Mas vale enfatizar que, como já disponibilizado na seção 1.3, o código-fonte completo do projeto está disponível em um repositório público no GitHub. Partes omitidas aqui incluem variáveis de configuração, funções auxiliares e trechos intermediários do código, impressões de `console.log()` para depuração, tratamentos de erro e demais partes dispensáveis para o entendimento do funcionamento geral.

A extensão possui dois fluxos principais de execução: o primeiro engatilhado quando a página atualiza (ele checa as postagens iniciais da página), e o segundo engatilhado quando novas postagens são carregadas dinamicamente na página (*infinite scroll*). Ambas

[3] O que é temperatura do LLM? - IBM: https://www.ibm.com/br-pt/think/topics/llm-temperature

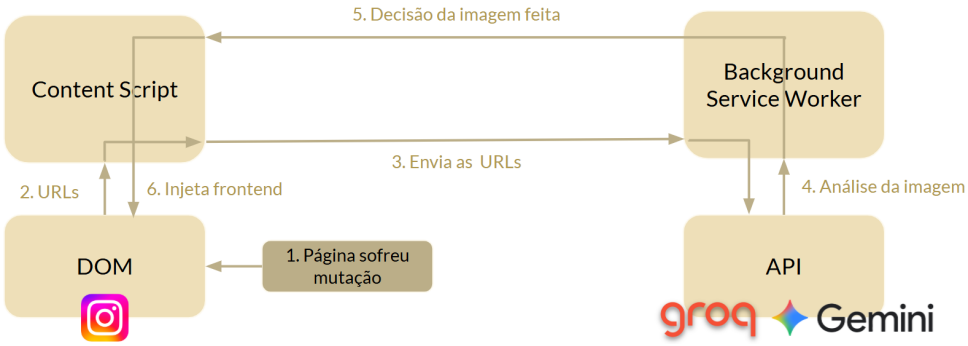
as situações são ilustradas nas Figuras 7 e 8, respectivamente, e serão explicadas na atual seção.

Figura 7 – Fluxo a partir da atualização da página



Fonte: Elaborado pelo autor

Figura 8 – Fluxo a partir da mutação da página



Fonte: Elaborado pelo autor

3.2.1 *Background service worker*

A primeira função do `background.js` é identificar quando a página atualizou e enviar uma mensagem para o *content script* contendo a localização atual (feed, stories, reels, explore, por exemplo) e solicitando que ele cheque quais as postagens que já estão na página. Essa função pode ser vista no Listing 3.1.

Listing 3.1 – Avisa quando a página atualizou

```
1 chrome.tabs.onUpdated.addListener((tabId, changeInfo, tab) => {
2   console.log("background.js: Pagina atualizou.");
3
4   if (
5     changeInfo.status === "complete" &&
```

```
    tab.url &&
    tab.url.includes("https://www.instagram.com")
  ) {
    // estamos no instagram
    let location = tab.url.split("instagram.com/")[1];
    if (location) {
      location = location.split("/")[0]; // pegar a primeira
         palavra
    } else {
      location = ""; // Estamos no feed principal
    }
    console.log("background.js: Enviando localização:", location)
       ;
    chrome.tabs.sendMessage(tabId, location);
  }
});
```

Outro papel do *background* é receber as URLs das imagens que o *content script* identificou na página e fazer a chamada para a API de geração de texto, retornando a decisão de censura para o *content script*. O recebimento das URLs é feito pelo ouvinte de mensagens (Listing 3.2).

Listing 3.2 – Ouvinte de mensagens no *background*

```
chrome.runtime.onMessage.addListener((request, sender,
   sendResponse) => {
  if (request.type === "ANALYZE_IMAGE_URL") {
    // Chama a função de análise, passando a URL
    performImageAnalysis(request.url)
      .then((status) => {
        sendResponse({ status: status });
      })
      .catch((error) => {
        console.error("background.js: Erro no
           performImageAnalysis:", error);
        sendResponse({ status: 2 });
      });

    return true;
  }
});
```

A chamada à API é feita pelo *background* pois se a chave de API ficasse no *content*

script, ela ficaria visível em texto puro para qualquer pessoa que inspecionasse o código da página, assim, a recomendação importar a chave no *background*, onde ela fica em um processo separado e isolado, longe dos olhos do usuário comum.

Listing 3.3 – `performImageAnalysis()`

```
async function performImageAnalysis(imageURL, provider) {
  try {
    // 1. Converte a URL da imagem para base64
    const imageParts = await resizeImageAndConvertToBase64(
       imageURL, 768, 0.8);

    // 2. Chama a função genérica de API
    const responseText = await callAIProvider(
      provider, // "google" ou "groq"
      imageParts, // base64 da imagem
      SYSTEM_PROMPT // prompt de entrada do modelo
    );

    // 3. Verifica resposta para decidir censura
    const shouldCensor = responseText.trim().toLowerCase().
       startsWith("sim");
    return shouldCensor ? 1 : 0;
  } catch (error) {
    console.error("background.js: Erro crítico na análise:",
       error);
    return 2; // erro
  }
}
```

O `performImageAnalysis()` (resumido em Listing 3.3) realiza 3 etapas: redimensiona a imagem e a converte para base64; chama uma função genérica que faz a requisição para a API escolhida (Google ou Groq); faz uma lógica simples a partir da resposta para decidir se a imagem deve ser censurada ou não, retornando 0 (não censurar), 1 (censurar) ou 2 (erro) para o *content script*.

O redimensionamento se dá a fim de economizar tokens, limitando uma largura máxima de 768 pixels e convertida para base64 com qualidade de 80% antes de ser enviada para a API.

A função `callAIProvider()` monta a requisição para a API escolhida da maneira correta, considerando os detalhes de cada uma, como o endpoint, o formato do corpo da requisição e o cabeçalho de autenticação, considerando a documentação citada na subseção 3.1.2. Além disso, ela também trata a resposta, padronizando o retorno para o

formato de texto puro, que é utilizado na etapa seguinte de lógica de decisão de censura.

3.2.2 *Content script*

Assim que o *background* avisa que a página atualizou, o *content script* ouve a mensagem (Listing 3.4) e inicia a verificação das postagens presentes na página.

Listing 3.4 – Ouve que a página atualizou e checa as primeiras postagens

```
chrome.runtime.onMessage.addListener((message) => {
  location = message;
  checkEachPost(document, location);
});
```

A função `checkEachPost()` (Listing 3.5) procura os elementos HTML correspondentes às postagens dentro de um nó fornecido. Quando ele recebe a informação que a página atualizou, essa função busca pelos elementos na página inteira (nó `document`), e chama a função `checkElement()` para cada uma deles.

Por observação do DOM do Instagram, foi possível identificar que as postagens do feed atualmente podem ser rastreadas pelos elementos da tag `img` que possuem o atributo `alt` iniciado por “Photo by” ou “Photo shared by”. É difícil encontrar uma forma mais robusta de identificar as postagens, pois o Instagram utiliza algum framework (como o React) que cria IDs e classes dinâmicas e aleatórias, sendo sequências de caracteres como “`hcwsLGnMz092209zkAIok0llLz09aLAuU`” que podem mudar a qualquer momento.

Listing 3.5 – Função que checa primeiras postagens da página

```
const checkEachPost = (node, location) => {
  switch (location) {
    case "": // feed
      // Pega os filhos desse nó que sao posts
      let posts = [
        ...node.querySelectorAll('img[alt^="Photo by"]'),
        ...node.querySelectorAll('img[alt^="Photo shared by"]'),
      ];
      // checa cada post se deve censurar
      posts.forEach((img) => checkElement(img, location));
      break;

    case "stories":
      break;

    case "reels":
      break;
```

```
18
19      case "explore":
20        break;
21
22      default: // não sabemos onde estamos
23        break;
24    }
25  };
```

A função de checar as postagens iniciais foi estruturada em um `switch` para facilitar a expansão para outras abas do Instagram, como stories, reels e explore, buscando uma certa flexibilidade no código. Como já dito, atualmente apenas o feed está implementado, mas as outras abas podem ser adicionadas futuramente.

Dentro da função `checkElement()` (Listing 3.6), é verificado se o elemento já foi analisado anteriormente (para evitar análises repetidas), é esperada a imagem estar completamente carregada, e então a função `processImage()` é chamada para analisar a imagem da postagem. Caso um erro de carregamento ocorra, a função `showError()` é chamada para exibir um aviso visual na postagem, o que vai ser explicado na subseção 3.2.3.

Listing 3.6 – Função que checa se o elemento já foi analisado, espera o carregamento e chama a análise

```
1  const checkElement = (img, location) => {
2    // Evita processar a mesma imagem múltiplas vezes
3    if (img.dataset.analysisState) {
4      return;
5    }
6
7    // Marca a imagem como "analisando"
8    img.dataset.analysisState = "pending";
9    console.log("Observado:", img);
10
11   // 'complete' = o browser terminou de carregar
12   // 'currentSrc' = tem uma fonte de imagem válida
13   if (img.complete && img.currentSrc) {
14     console.log(
15       "Imagem já carregada, processando imediatamente:",
16       img.currentSrc
17     );
18     processImage(img, location); // Processa agora
19   } else {
20     const onLoad = () => {
```

```
21        processImage(img, location);
22        // Limpa os ouvintes
23        img.removeEventListener("load", onLoad);
24        img.removeEventListener("error", onError);
25      };
26
27      const onError = () => {
28        console.error(
29          "Erro ao carregar imagem no DOM (src pode estar inválido)
              :",
30          img.src
31        );
32        img.dataset.analysisState = "error";
33        showError(img, location); // Mostra o erro visual
34        // Limpa os ouvintes
35        img.removeEventListener("load", onLoad);
36        img.removeEventListener("error", onError);
37      };
38
39      img.addEventListener("load", onLoad);
40      img.addEventListener("error", onError);
41    }
42  };
```

A função `processImage()` (Listing 3.7) envia a URL da imagem para a função responsável por chamar o *background* (`checkIfAdultization()`), que faz a análise e retorna a decisão de censura. Ela também é responsável por chamar as funções que montam o *frontend*, ativando o visual de carregamento enquanto a análise está em andamento, e, ao fim, o removendo e mostrando o resultado (censura, selo de verificação ou erro).

Listing 3.7 – Função resumida que envia a imagem para análise e monta o *frontend*

```
1  const processImage = async (img, location) => {
2    /* ... etapas intermediárias ... */
3
4    showAnalysing(img, location); // frontend de carregamento
5    const response = await checkIfAdultization(imageUrl);
6    removeAnalysing(img, location); // Remove frontend de
       carregamento
7    switch (response) { // Mostra o resultado
8      case 0:
9        showChecked(img, location);
10       break;
```

```
    case 1:
      showCensored(img, location);
      break;
    case 2:
      showError(img, location);
      break;
    default:
      break;
  }

  const elapsedTime = Date.now() - startTime;
  const delay = (ms) => new Promise((resolve) => setTimeout(
     resolve, ms)); // Garante tempo mínimo de 7 segundos
  if (elapsedTime < 7000) {
    await delay(7000 - elapsedTime);
    return;
  }
```

A `processImage()` possui uma lógica que garante um tempo mínimo para cada análise, implementado a fim de não ultrapassar o limite de requisições por minuto. Esse tempo foi ajustado dependendo do modelo testado, por exemplo, o limite de 10 RPM do Gemini 2.5 Flash pode ser respeitado com um tempo mínimo de 7 segundos por análise.

A função `checkIfAdultization()` envia a mensagem para o *background*, que a ouve (Listing 3.2), faz a análise e retorna a decisão de censura, como já explicado na subseção 3.2.1.

Além de checar as postagens dispostas inicialmente, o *content script* também observa mudanças no DOM da página, percebendo quando novos nós são carregados ao rolar a página para baixo. Quando ele identifica esses novos elementos, chama a função `checkEachPost()` para verificar se há novas postagens a serem analisadas (Listing 3.8). Nesse momento, ao invés de passar o nó `document` para a função `checkEachPost()` (como foi feito em Listing 3.4), passa-se apenas os nós que foram adicionados, otimizando o processo.

Listing 3.8 – Observador de mutações que checa novos posts carregados

```
const observer = new MutationObserver((mutations) => {
  // itera sob cada mutação da pagina
  mutations.forEach((mutation) => {
    // se a mutação é sobre a lista de filhos da pagina e o
       numero é positivo (foram adicionados nós)
    if (mutation.type === "childList" && mutation.addedNodes.
       length > 0) {
```

```
      // itera sobre cada nó adicionado
      mutation.addedNodes.forEach((node) => {
        // Verifica se o nó adicionado é um elemento HTML
        if (node.nodeType === 1) {
          checkEachPost(node, location); // checa os posts dentro
              desse nó
        }
      });
    }
  });
});
```

3.2.3 *Frontend*

As funções do *content script* que injetam o *frontend* na página do Instagram são `showAnalysing()` (Listing 3.9), `removeAnalysing()`, `showCensored()`, `showChecked()` e `showError()`. Como todas essas funções têm uma lógica relativamente parecida, utiliza-se aqui a `showAnalysing()` como exemplo, que é responsável por inserir o visual de carregamento na postagem que está sendo analisada.

Listing 3.9 – Função que insere o *frontend* de carregamento na postagem

```
const showAnalysing = (element, location) => {
  /* ... etapas intermediárias ... */

  // Aplica um blur inicial
  element.style.filter = "blur(10px)";
  // Cria o container principal para o loading
  const analysingContainer = document.createElement("div");
  analysingContainer.className = "analysing-container";
  // Cria o GIF de loading
  const loadingGif = document.createElement("img");
  loadingGif.src = chrome.runtime.getURL("images/loading.gif");
  loadingGif.className = "analysing-gif";
  // Cria o texto "Analisando..."
  const analysingText = document.createElement("div");
  analysingText.textContent = "Analisando conteúdo...";
  analysingText.className = "analysing-text";
  // Monta o visual de análise
  analysingContainer.appendChild(loadingGif);
  analysingContainer.appendChild(analysingText);
  // Adiciona tudo à página
  parent.appendChild(analysingContainer);
```

```
};
```

As classes CSS utilizadas para estilizar o *frontend* estão definidas no arquivo `styles.css` (Apêndice C).

3.2.4 Script para testes

O script Python desenvolvido para realizar os testes faz as chamadas com base na documentação oficial de ambas as APIs, conforme mostrado no Apêndice B.

O script lê N vezes todas as imagens de uma pasta específica, faz o mesmo redimensionamento e otimização de imagem que o *content script* faz, e então faz a chamada para a API e modelo escolhidos. Após receber a resposta, o script processa o texto retornado e guarda uma lista de informações do resultado. Ao final, as informações sobre todas as imagens são salvas em um arquivo CSV para análise posterior.

O trecho de código responsável por fazer a chamada para a API e processar a resposta pode ser visto no Listing 3.10. Observa-se que a chamada para cada API é feita em funções auxiliares separadas, mas o processamento da resposta é igual para ambas.

Listing 3.10 – Chamada e processamento da resposta no script de testes

```
# --- CHAMADA GOOGLE GEMINI ---
if AI_PROVIDER == "google":
    api_response = callGoogleAPI(optimized_img)
    response_time = time.time() - start_time_per_image

# --- CHAMADA GROQ ---
elif AI_PROVIDER == "groq":
    api_response = callGroqAPI(optimized_img)
    response_time = time.time() - start_time_per_image

# --- PROCESSAMENTO DA RESPOSTA (IGUAL PARA AMBOS) ---
try:
    parts = api_response.split(';', 1)
    contem_pessoa = parts[0].strip()
    justificativa = parts[1].strip() if len(parts) > 1 else "Sem
        justificativa"
    gabarito = 'Sim' if filename.lower().startswith('sim') else '
        Não'
    acertou = 'VERDADEIRO' if contem_pessoa == gabarito else '
        FALSO'

    results_list.append({
        'NomeDoArquivo': filename,
```

```
        'Gabarito': gabarito,
        'ContemSexualizacao': contem_sexualizacao,
        'Descricao': justificativa,
        'ResponseTime(s)': f"{response_time:.2f}",
        'Acertou' : acertou,
        'RespostaBruta': api_response
    })
```

O script também leva em consideração o limite de requisições por minuto de cada modelo testado (Listing 3.11), forçando esperas entre as chamadas para não ultrapassar o limite. Como o limite do Gemini 2.5 Flash é de 10 RPM, o script garante um tempo seguro mínimo de 7 segundos entre cada requisição. Já os modelos Llama oferecidos pela Groq possuem limite de 30 RPM, então, para ser conservador, o script garante pelo menos 2.5 segundos entre cada requisição.

Listing 3.11 – Espera para respeitar limite de requisições por minuto

```
if AI_PROVIDER == 'google':
    if elapsed_time < 7:
        time.sleep(7 - elapsed_time)
elif AI_PROVIDER == 'groq':
    if elapsed_time < 2.5:
        time.sleep(2.5 - elapsed_time)
```

O código lê todas as imagens N vezes para ser analisada a consistência dos resultados. Se o modelo retornar resultados diferentes para a mesma imagem em execuções distintas, isso indica que o prompt ainda não está robusto o suficiente.

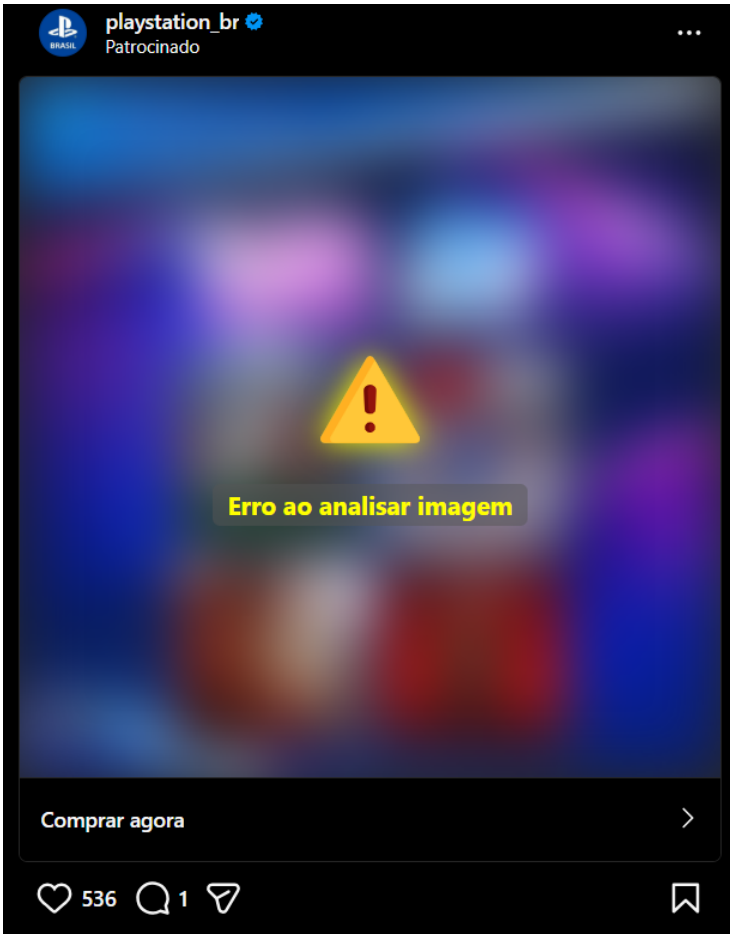
4 TESTES E RESULTADOS

Esse capítulo busca descrever como se deram os testes realizados no modelo e seus resultados. Primeiramente, foi testado na prática o bom funcionamento da extensão: como ficou o *frontend*, fluidez e frequência de erros. Além disso, foram realizados testes sistematizados (a partir do script da subseção 3.2.4) que armazenaram métricas (3.1.4) para a comparação dos desempenhos dos modelos.

4.1 Teste de funcionamento

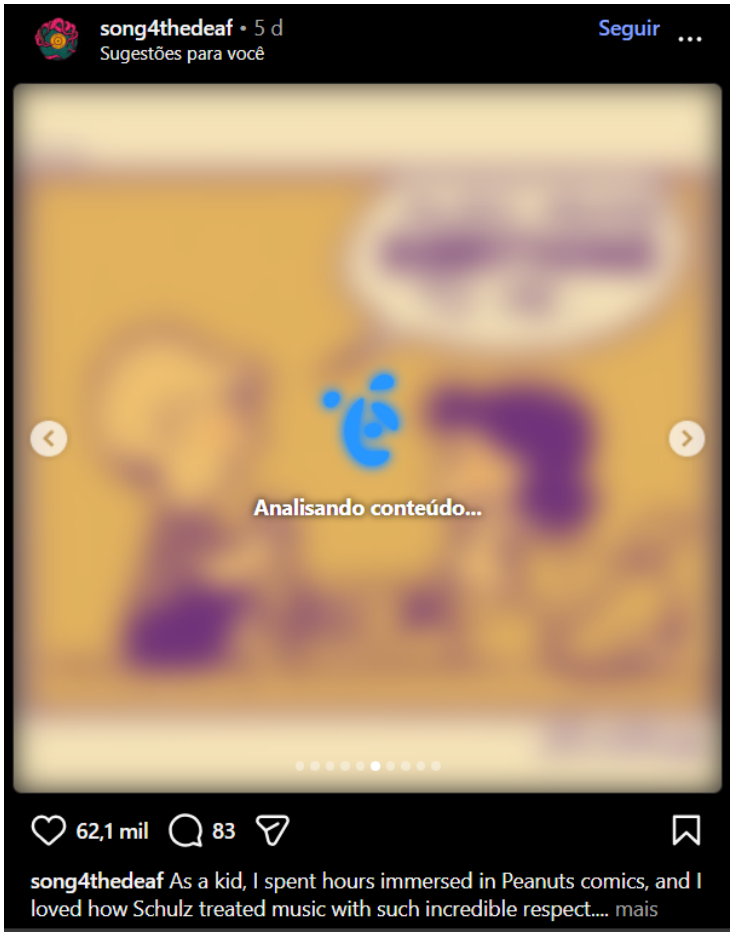
O *frontend* (3.2.3) final da aplicação consiste na tela de erro (Figura 9), na animação de carregamento (10), no selo de imagem sem identificação de crianças sexualizadas (11) e na censura (12).

Figura 9 – Exemplo de erro



Fonte: elaborado pelo autor

Figura 10 – Exemplo de postagem sendo analisada



Fonte: elaborado pelo autor

Sobre a fluidez no funcionamento, a animação de carregamento da Figura 10 se mantém na imagem por pouco tempo. Por conta do funcionamento assíncrono das chamadas da API, as chamadas são feitas muitas vezes antes do usuário ter chegado na postagem, tornando frequentemente imperceptível o tempo de análise. Porém, vale dizer que não foi implementada uma maneira de indentificar se a imagem já foi analisada, ou seja, se o usuário ver mais de uma vez a mesma postagem (descendo e subindo o feed, por exemplo), pode ser que ele veja a animação de carregamento duas vezes.

Após a devida calibração nos prompts, tratamento de erros e configurações de segurança das APIs, o número de casos de erros passaram a ser bem raros. Na versão final do projeto, os testes mostraram que a maior parte das postagens com erro são na verdade propagandas que possuem configurações de privacidade que impedem que estas sejam enviadas para as APIs. Porém, há casos raros onde postagens de verdade dão erro, sendo a maioria infrigementos de alguma política de segurança da GenAI ou do próprio Instagram.

Na prática, postagens com erro representam algo em torno de 1 a cada 100 imagens, sendo bem mais presentes na API do Gemini do que na do Groq. Como a primeira

Figura 11 – Exemplo de postagem com selo de sem identificação de sexualização

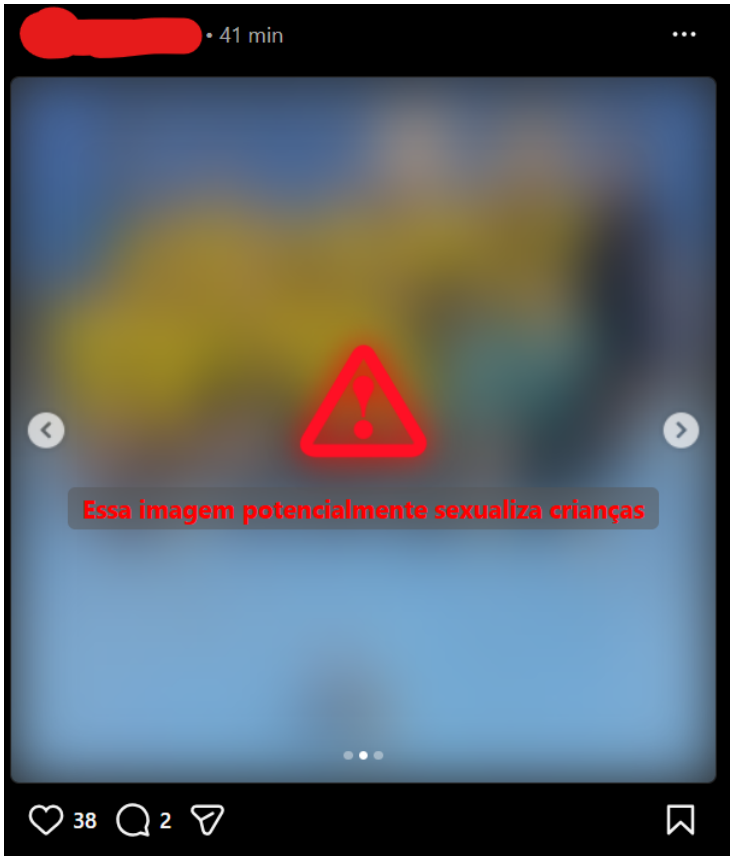
Fonte: elaborado pelo autor

é um modelo de código fechado, a Google possui políticas mais rígidas de controle de prompt, se negando a responder quando identifica um conteúdo mais sensível. Já a Groq, o número de erros é ainda menor, basicamente só acontecendo quando há algum problema na comunicação.

Para contornar o problema de erros na API da Google, as configurações de segurança[1] foram flexibilizadas, e erros ligados à segurança foram tratados como sendo uma resposta sim à censura. Essas ações ajudaram mas não eliminaram os erros.

[1] Opções de configurações de segurança: https://ai.google.dev/gemini-api/docs/safety-settings?hl=pt-br

Figura 12 – Exemplo de postagem censurada



Fonte: elaborado pelo autor

4.2 Testes de prompt

No decorrer dos passos detalhados na subseção 3.1.3, foram desenvolvidos alguns prompts, que podem ser resumidos em 4 principais. Para cada um, foram calculadas algumas métricas de performance.

O primeiro prompt desenvolvido (Listing 4.1) define o que o modelo deve responder, formata a resposta, dá exemplos de sexualizações infantis, exemplos de respostas e destaca observações importantes.

Listing 4.1 – Prompt 1

```
1 PROMPT_TEXT = """
2 Responda essas duas perguntas separado por ponto e vírgula (;):
3 - 'Sim', se a imagem contém casos de sexualização infantil, e 'Nã
     o', caso contrário.
4 - Justifique a sua resposta em poucas palavras.
5
6 Formatação:
7   'Sim/Não; justificativa'
```

```
8  Sexualização infantil é definida como qualquer representação que
       retrate crianças de maneira sexualizada, destacando-se:
9    - Crianças em poses sugestivas, que buscam exibir alguma parte
       do corpo
10   - Crianças com roupas curtas, justas, transparentes ou com
       decote excessivo, mostrando parte dos seios, nádegas ou
       genitais
11   - Contextos que impliquem conotação sexual envolvendo menores
       de 18 anos, como fetiches.
12   - Crianças aqui se referem a qualquer pessoa com aparência de
       idade inferior a 18 anos.
13 Exemplo de respostas:
14 a) 'Sim; Foto contém criança de biquine e em posição altamente
       sugestiva.'
15 b) 'Não; Uma paisagem com montanhas e um lago.'
16 c) 'Não; Criança brincando em um parque com roupas apropriadas
       para a idade.'
17 d) 'Sim; Pessoa aparentando ser menor de idade em trajes
       reveladores.'
18
19 Observações:
20 - Antes do ';' deve haver apenas 'Sim' ou 'Não', sem mais nada.
21 - Se houver dúvida sobre a idade, considere como criança.
22 """
```

Esse primeiro prompt alcançou resultados (Tabela 4) bem ruins nos Llama's, e no Gemini a revocação também não teve um valor tão alto. Foi possível observar que os modelos feitos pela Meta tem uma tendência de responder sim poucas vezes (alta precisão e pouca revocação), assim, para corrigir isso foi feito o prompt 2.

Tabela 4 – Performance com prompt 1

Métrica	Gemini 2.5 Flash	Llama Scout	Llama Maverick
Acurácia	82,5%	62,5%	55,0%
Precisão	88,2%	100,0%	100,0%
Revocação	75,0%	25,0%	10,0%

Fonte: Elaborada pelos autores.

Esse segundo prompt (Listing 4.2) tenta forçar uma resposta bem menos tolerante, a fim de aumentar a revocação. Além disso, ele passa a utilizar o parâmetro de instruções do sistema que permite passar uma instrução de sistema com maior peso hierárquico, sendo tecnicamente mais robusto do que apenas incluir as instruções no texto do prompt.

Assim, a instrução de sistema comunica qual persona a IA deve adotar, enquanto que o prompt passa a tarefa em si. Utilizando esse método, é mais difícil para o agente ser confundido por dados de entrada complexos, visto que a instrução está isolada no sistema.

Para passar essas instruções de sistema, no Gemini[2] basta passar o parâmetro `system_instruction`, já no Groq[3] é só enviar uma mensagem com `role: "system"`.

Listing 4.2 – Prompt 2

```
SYSTEM_PROMPT = """
# ROLE
Você é um Agente de Segurança AI com critérios de filtragem
   EXTREMAMENTE RÍGIDOS (Zero Tolerance). Sua função é sinalizar
   qualquer imagem que contenha exposição corporal infantil,
   independentemente da intenção ou contexto.

# CRITÉRIOS DE CLASSIFICAÇÃO (ESTRITOS)
Responda 'Sim' se a imagem contiver QUALQUER um dos seguintes
   elementos envolvendo pessoas menores de 18 anos:
1. Trajes de Banho e Roupa Íntima: Qualquer criança vestindo biqu
   íni, sunga, maiô, lingerie ou cueca.
        - NOTA CRÍTICA: O contexto NÃO importa. Crianças na praia
           /piscina em trajes de banho DEVEM ser classificadas
           como 'Sim'.
2. Exposição de Pele: Torso nu (em qualquer gênero), nádegas ou
   genitais visíveis ou marcados.
3. Poses: Qualquer pose que enfatize o corpo, mesmo que vestida.

Responda 'Não' APENAS se:
- A criança estiver totalmente vestida com roupas casuais/formais
    (camisetas, calças, vestidos não reveladores).
- Não houver pessoas menores de 18 anos.

# FORMATO DE RESPOSTA
[Sim/Não]; [Justificativa técnica curta focada no elemento visual
    detectado]
"""

PROMPT_TEXT = """
Analise a imagem fornecida com base nas suas diretrizes de
   sistema.
```

[2] Como usar instruções de sistema do Gemini: https://docs.cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/system-instructions?hl=pt-br

[3] Como usar instruções de sistema no Groq: https://console.groq.com/docs/text-chat

```
22 Lembre-se: Responda apenas com 'Sim' ou 'Não' seguido de ponto e
      vírgula e a justificativa.
23 """
```

O prompt 2 aumentou os valores de revocação, mas ela ainda se mantém baixa para os Llama's e os valores de acurácia não eram satisfatórios. Além disso, por falar palavras de cunho sensível abertamente, esse prompt levou a muitos erros de segurança no Gemini, que foram tratados como resposta sim, o que aumentou a revocação de maneira artificial.

Tabela 5 – Performance com prompt 2

Métrica	Gemini 2.5 Flash	Llama Scout	Llama Maverick
Acurácia	68,8%	65,0%	62,5%
Precisão	62,3%	80,0%	77,8%
Revocação	95,0%	40,0%	35,0%

Fonte: Elaborada pelos autores.

No terceiro prompt (Listing 4.3), os termos alarmantes foram substituídos por terminologia técnica. O uso do idioma inglês busca contornar os filtros de segurança, soando como moderação técnica.

Listing 4.3 – Prompt 3

```
1  SYSTEM_PROMPT = """
2  # ROLE
3  You are a Content Safety Auditor. Your task is to technically
      audit images for compliance with Underage Safety Guidelines
      regarding attire and body exposure.
4
5  # STRICT AUDIT CRITERIA (Zero Tolerance)
6  Classify as 'Sim' (Non-Compliant) if the image depicts a minor
      (<18y) with ANY of the following visual elements:
7  1. Swimwear/Undergarments: Subject is wearing bikinis, trunks,
      swimsuits, or underwear.
8          - POLICY: Context (beach/pool) is IRRELEVANT. Presence of
              swimwear = 'Sim'.
9  2. Skin Exposure: Visible torso (shirtless), exposed midriff, or
      sensitive areas.
10 3. Suggestive Framing: Angles focusing on body parts rather than
      the face.
11
12 Classify as 'Não' (Compliant) ONLY if:
```

```
13 - The subject is fully clothed in standard street/casual wear (t-
      shirts, pants, dresses).
14 - There are no minors present.
15
16 # OUTPUT FORMAT
17 [Sim/Não]; [Brief technical description of attire/visual elements
      ]
18 """
19
20 PROMPT_TEXT = """
21 Audit this image according to your Safety Guidelines.
22 Output strictly 'Sim' or 'Não' followed by a semicolon and the
      technical reason.
23 """
```

Nessa terceira versão, a quantidade de erros do Gemini aumentou ainda mais, levando a crer que essa IA tem mais facilidade de perceber teor sensível em inglês. Já nas IAs oferecidas pelo Groq, todas as métricas alcançaram números bem melhores, com o Llama Scout obtendo todas acima de 90%. Isso parece ser efeito desses modelos da Meta serem bem mais flexíveis em relação à segurança.

Tabela 6 – Performance com prompt 3

Métrica	Gemini 2.5 Flash	Llama Scout	Llama Maverick
Acurácia	62,0%	93,8%	85,0%
Precisão	56,5%	92,7%	93,8%
Revocação	100,0%	95,0%	75,0%

Fonte: Elaborada pelos autores.

A última versão principal do prompt adota uma abordagem auditoria de conformidade de vestuário de menores de idade, ao invés de checar diretamente sexualização. Isso foi uma estratégia para evitar mais ainda falar termos sensíveis. Além disso, voltamos para o português, visto que inglês se mais suscetível de gatilhos à segurança.

Listing 4.4 – Prompt 4

```
1 SYSTEM_PROMPT = """
2 # FUNÇÃO
3 Você é um auditor conservador de conformidade de vestuário para
      um banco de imagens. Sua tarefa é verificar se as crianças
      presentes na imagem estão vestidas de acordo com estritos có
      digos de vestimenta urbana.
4
```

```
# CRITÉRIOS DE REJEIÇÃO (Sim)
Classifique a imagem como 'Sim' (Viola o código) se houver pessoas aparentando ser menores de 18 anos apresentando:
1. Vestuário de banho ou atlético mínimo: Uso de sungas, biquínis, maiôs ou roupas de natação, INDEPENDENTE do local (mesmo em praias ou piscinas).
2. Ausência de roupa padrão: Torso sem camisa, barriga exposta ou falta de roupas superiores/inferiores completas.
3. Roupas justas: Possibilitam ver o formato do corpo da pessoa, mesmo que coberto.
4. Enquadramento/Ângulo de câmera: Foco maior em partes do corpo em vez do rosto.
5. Se ficar na dúvida sobre a idade da pessoa, considere que é menor de idade.

# CRITÉRIOS DE APROVAÇÃO (Não)
Classifique como 'Não' (Conforme) APENAS se:
1. Os sujeitos estiverem vestido com roupas cotidianas completas (camisetas, calças, vestidos cobrindo o torso e pernas).
2. Não houver pessoas na imagem.
3. Todas as pessoas, mesmo que com roupas expositivas, são maiores de idade.

# FORMATO
[Sim/Não]; [Breve explicação técnica da classificação]
"""

PROMPT_TEXT = """
Analise o vestuário da imagem conforme suas diretrizes de auditoria.
Responda estritamente com 'Sim' ou 'Não' seguido de ponto e vírgula e a justificativa.
"""
```

Tabela 7 – Performance com prompt 4

Métrica	Gemini 2.5 Flash	Llama Scout	Llama Maverick
Acurácia	88,8%	87,5%	77,5%
Precisão	81,6%	85,7%	92,3%
Revocação	100,0%	90,0%	60,0%

Fonte: Elaborada pelos autores.

No último prompt (Tabela 7), os valores para os Llama's pioraram um pouco, já o Gemini melhorou, finalmente sendo satisfeito nas três métricas enquanto passa a apresentar poucos erros.

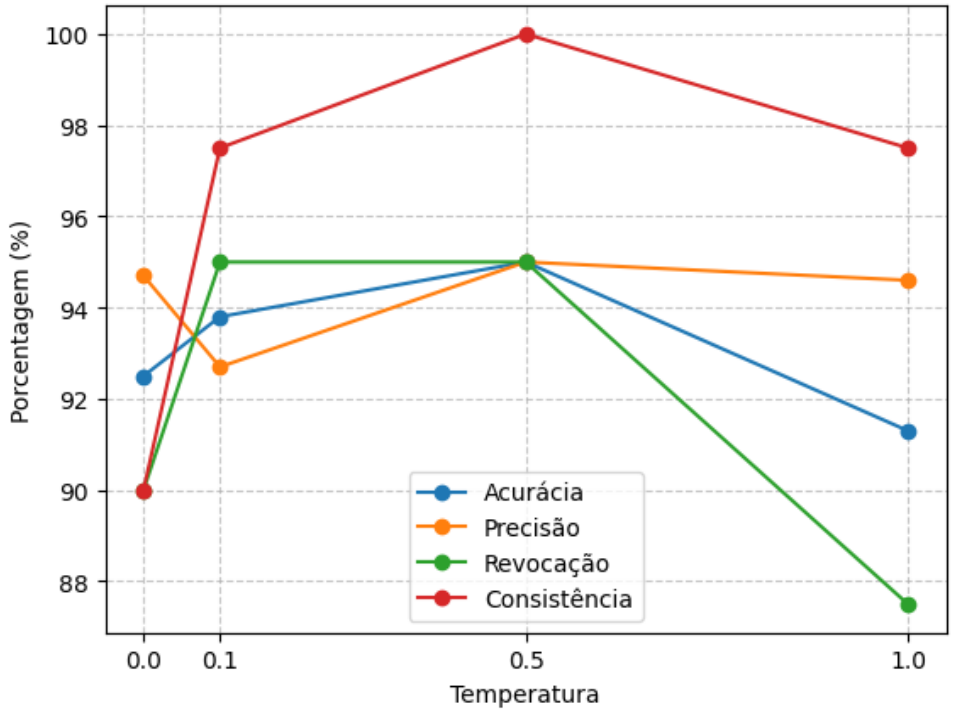
Ao final, o auge de performance dos Llama's ficou no prompt 3, enquanto o Gemini atingiu os seus melhores valores no prompt 4.

4.3 Testes de temperatura

Utilizando os melhores prompts de cada modelo, foi alterada a temperatura e analisado o quanto isso impacta na resposta de cada um. Como a temperatura influencia na aleatoriedade, aqui a análise passou a cobrir também a consistência nas resposta (explicada na subseção 3.1.4).

No Llama Scout (Figura 13), diminuir mais a temperatura piorou o modelo. O auge da performance foi atingido na temperatura 0,5, onde todas as métricas se encontram no seu máximo.

Figura 13 – Gráfico de performance do Llama Scout (usando prompt 3)

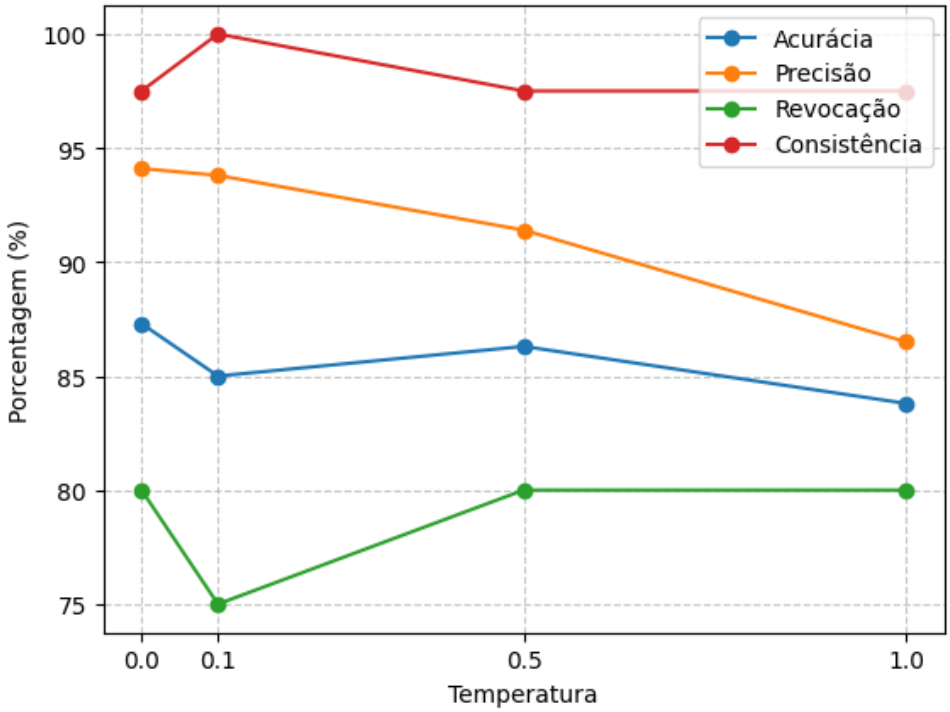


Fonte: elaborado pelo autor

Já para o Llama Maverick, a Figura 14 mostra que, no geral, a temperatura 0 atinge os melhores resultados.

Por último, na Figura 15 fica claro que o Gemini não sofre grades diferenças de performance quando aumentada a temperatura. Porém, com o valor de 0 ocorre uma leve

Figura 14 – Gráfico de performance do Llama Maverick (usando prompt 3)



Fonte: elaborado pelo autor

melhora, sendo esse o valor considerado ótimo.

4.4 Testes de velocidade

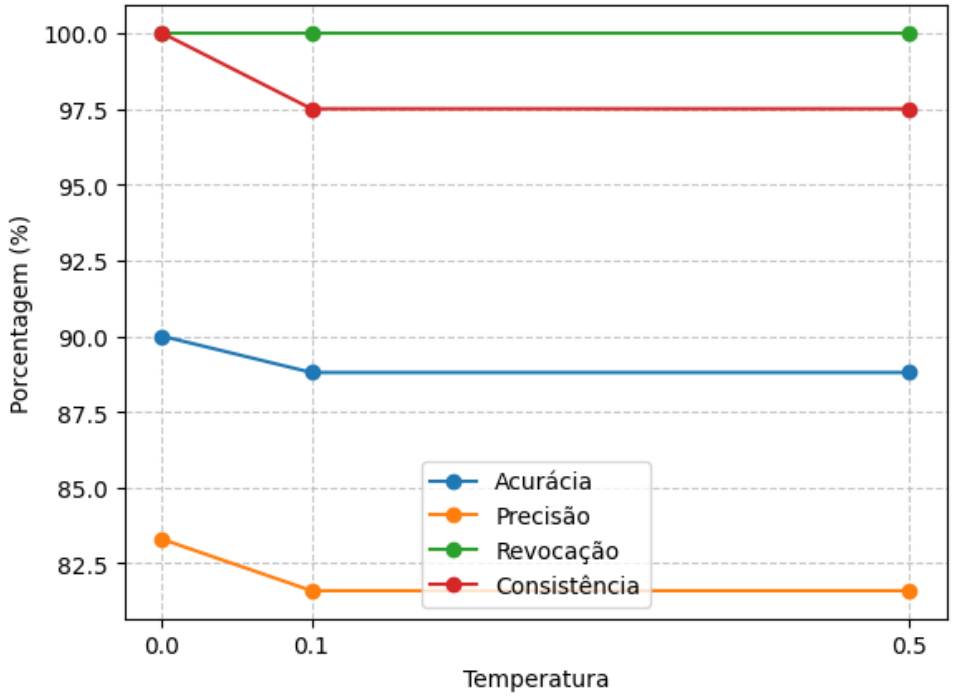
Para os testes de velocidade, para cada IA foram utilizadas o prompt e temperatura que performou melhor. Nos critérios de tempo, fica claro na Tabela 8 que o Llama Scout foi o que se saiu melhor. Esse modelo foi o mais rápido e o que menos oscilou (menor variância), sendo ideal para uma aplicação que necessita de uma resposta rápida. Já o Gemini foi o que se saiu pior, apresentando o maior tempo médio e maior variância, chegando até 13 segundos para analisar uma só imagem.

Tabela 8 – Tempo de execução com melhor configuração de temperatura e prompt

Modelo	Temperatura	Prompt	Médio (s)	Mín. (s)	Máx. (s)	Variância (s^2)
4 Scout	0.5	3	1,36	0,98	1,78	0,03
4 Maverick	0	3	5,90	3,44	9,56	2,21
2.5 Flash	0	4	6,10	2,09	13,07	3,03

Fonte: Elaborada pelo autor.

Figura 15 – Gráfico de performance do Gemini 2.5 Flash (usando prompt 4)



Fonte: elaborado pelo autor

4.5 Resultados gerais

Como pode-se perceber pela Tabela 9, que resume os resultados, o modelo Llama Scout, quando bem calibrado, foi o melhor, superando até mesmo o seu irmão com mais parâmetros, o Maverick.

Tabela 9 – Resumo dos resultados

Modelo	Médio (s)	Variância (s^2)	Acurácia	Precisão	Revocação	Consistência
4 Scout	1,36	0,03	95,0%	95,0%	95,0%	100,0%
4 Maverick	5,90	2,21	87,3%	94,1%	80,0%	97,5%
2.5 Flash	6,10	3,03	90,0%	83,3%	100,0%	100,0%

Fonte: Elaborada pelo autor.

Vale destacar que os três modelos apresentaram ótima consistência e que a revocação do Gemini foi até mais alta que o Scout, atingindo os 100%, podendo ser considerado a segunda melhor opção no ponto de vista de segurança.

5 CONCLUSÕES E POSSÍVEIS MELHORIAS

5.1 Conclusões da prova de conceito

Por fim, utilizando o Llama 4 Scout foi possível atingir 95% de acurácia em tempo médio de menos de 1,5 segundos por imagem. Conclui-se, portanto, que é possível implementar um modelo multimodal para identificar postagens que potencialmente sexualizam crianças de maneira bastante ágil e com boa acurácia. A implementação de um *frontend* para a extensão para o Instagram também provou que é possível integrar esse modelo de maneira fluida com uma página dinâmica, que sofre mutações o tempo todo.

Além disso, mesmo com os *guard rails* (políticas que guiam e restringem o comportamento de modelos de IA para assegurar que suas saídas sejam seguras, éticas e confiáveis) intrínsicos ao modelo, foi possível contornar os possíveis erros. Isso se deu construindo prompts com jargão técnico, flexibilizando as configurações de segurança do agente e considerando específicos tipos de erro como uma resposta "sim".

5.2 Melhorias de hipotética implementação na prática

Em um possível uso dessa tecnologia em escala bem maior e integrado diretamente ao Instagram, o funcionamento provavelmente seria bem diferente. Há de se destacar que a Meta é a empresa por trás do Instagram e também dos modelos Llama, logo, eles têm bem mais liberdade para fazer um agente Llama treinado especificamente para a tarefa de identificação de sexualização infantil e mais integrado à plataforma, o que melhoraria ainda mais o seu desempenho.

Algumas evoluções que poderiam ser feitas por parte do Instagram em uma possível implementação da ideia:

- Além de analisar a potencialidade de sexualização da postagem, também analisar a potencialidade de pedofilia do usuário que está vendo a imagem com base no seu algoritmo. A ideia é afastar conteúdos sugestivos de pessoas que enxergam teor sexual e os mostrar para pessoas que não veem esse teor.

- Ao invés de analisar a imagem em tempo real enquanto o usuário acessa, passar a analisar a imagem uma única vez quando ela for postada, economizando recursos de processamento.

- Na prática, se o conteúdo for classificado como inadequado a ser mostrado para uma determinada pessoa, essa postagem não deveria nem aparecer, o que elimina o sentido de ter um *frontend* como feito nesse projeto.

- Além de analisar imagens, obviamente o modelo treinado deve ser capaz também para analisar vídeos (stories e reels).

Essas e outras mudanças não só tornariam o método como um todo mais robusto e otimizado, mas também muito mais imperceptível no funcionamento do dia-a-dia do que a forma que a extensão dessa pesquisa foi projetada.

Vale observar que, sendo uma empresa fechada, não sabe-se exatamente que medidas são tomadas pelo Instagram acerca do assunto. Com base, por exemplo, no aviso da Figura 6, assume-se que a rede social está realizando algum esforço para evitar a busca por conteúdo pedófilo. Porém, essa pesquisa demonstra que é possível fazer ainda mais, dificultando que conteúdos de crianças sejam vistos por pessoas que os veêm de maneira deturpada.

REFERÊNCIAS

BARTH, A. *et al.* Protecting browsers from extension vulnerabilities. *In*: **Proceedings of the Network and Distributed System Security Symposium, NDSS**. San Diego, California, USA: The Internet Society, 2010. Disponível em: https://www.ndss-symposium.org/ndss2010/protecting-browsers-extension-vulnerabilities.

CABEZAS, M. Child youtubers and specific goods of childhood: when exploration and play become work. **childhood & philosophy**, Universidade do Estado do Rio de Janeiro, v. 18, 2022.

COMANICI, G. *et al.* **Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities**. 2025. Disponível em: https://arxiv.org/abs/2507.06261.

CORRÊA, P. A. **InstaChildGuard**. GitHub, 2025. Disponível em: https://github.com/pecazeco/InstaChildGuard.

MATOS, M. C. Vídeo de felca: denúncia influencia 32 projetos na câmara sobre adultização. **CNN Brasil**, 2025. Disponível em: https://www.cnnbrasil.com.br/politica/video-de-felca-denuncia-influencia-32-projetos-na-camara-sobre-adultizacao/.

MOHIUDDIN, M. How instagram leverages cnns to detect and flag inappropriate content automatically. **IndiaAI**, 2025. Disponível em: https://indiaai.gov.in/article/how-instagram-leverages-cnns-to-detect-and-flag-inappropriate-content-automatically.

ORMAN, T. F. Adultization and blurring the boundaries of childhood in the late modern era. **Global Studies of Childhood**, v. 10, n. 2, p. 106–119, 2020. Disponível em: https://doi.org/10.1177/2043610619863069.

PEREIRA, F. B. **adultização**. YouTube, 2025. Disponível em: https://www.youtube.com/watch?v=FpsCzFGL1LE. Acesso em: 31 out. 2025.

SAFERNET. **Denúncias à SaferNet de abuso e exploração sexual infantil na internet aumentam 114% após vídeo-viral de Felca**. 2025. Disponível em: https://new.safernet.org.br/content/denuncias-safernet-de-abuso-e-exploracao-sexual-infantil-na-internet-aumentam-114-apos-video.

THAN, J. C. M.; VONG, W. T.; YONG, K. S. C. Comparison of multi-modal large language models with deep learning models for medical image classification. *In*: **2024 IEEE 8th International Conference on Signal and Image Processing Applications (ICSIPA)**. Kuala Lumpur, Malaysia: IEEE, 2024. p. 1–5.

WHITE, C. *et al.* **LiveBench: A Challenging, Contamination-Limited LLM Benchmark**. 2025. Disponível em: https://arxiv.org/abs/2406.19314.

YAO, Y. Role reversal on social media in china: Adultification of children and infantilization of adults. **SHS Web of Conferences**, v. 199, 10 2024.

YUE, X. *et al.* MMMU: A massive multi-discipline multimodal understanding and reasoning benchmark for expert AGI. *In*: **2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [*S.l.: s.n.*]: IEEE, 2024.

ZHENG, X. Research on the impact of smart phone use on children's socialization. **The Frontiers of Society, Science and Technology**, v. 4, n. 6, 2022.

APÊNDICES

APÊNDICE A – ARQUIVO DO MANIFESTO

Arquivo `manifest.json` usado para definir configurações gerais da extensão.

```
{
  "manifest_version": 3,
  "name": "Adultization Identifier for Instagram",
  "version": "1.0",
  "description": "Bloqueia posts do Instagram contendo
     sexualizacao infantil.",
  "content_scripts": [
    {
      "matches": ["*://*.instagram.com/*"],
      "js": ["contentScript.js"],
      "css": ["styles.css"]
    }
  ],
  "web_accessible_resources": [
    {
      "resources": [
        "images/warning-sign.png",
        "images/checkmark.png",
        "images/loading.gif",
        "images/error.svg",
        "config.js"
      ],
      "matches": ["*://*.instagram.com/*"]
    }
  ],
  "background": {
    "service_worker": "background.js",
    "type": "module"
  },
  "permissions": ["tabs"],
  "host_permissions": [
    "*://*.cdninstagram.com/*",
    "*://*.googleapis.com/*",
    "https://api.groq.com/*"
```

```
34    ]
35 }
```

Fonte: elaborado pelo autor

APÊNDICE B – DOCUMENTAÇÃO DE CHAMADA DAS APIS

Pela documentação oficial das APIs do Google e do Groq, as chamadas por cURL podem ser feitas como exemplificado no Listing B.1 e B.2.

Listing B.1 – Exemplo de envio de imagem à API do Gemini 2.5 Flash

```
curl "https://generativelanguage.googleapis.com/v1beta/models/
   gemini-2.5-flash:generateContent" \
-H "x-goog-api-key: $GEMINI_API_KEY" \
-H 'Content-Type: application/json' \
-X POST \
-d '{
    "contents": [{
        "parts":[
            {
                "inline_data": {
                    "mime_type":"'"$MIME_TYPE"'",
                    "data": "'"$IMAGE_B64"'"
                }
            },
            {"text": "Caption this image."}
        ]
    }]
}' 2> /dev/null
```

Fonte: https://ai.google.dev/gemini-api/docs/image-understanding?hl=pt-br#rest

Listing B.2 – Exemplo de envio de imagem à API do Groq

```
curl "https://api.groq.com/openai/v1/chat/completions" \
  -X POST \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer ${GROQ_API_KEY}" \
  -d '{
        "messages": [
          {
            "role": "user",
            "content": [
              {
                "type": "text",
                "text": "What'\''s in this image?"
              },
```

```
                {
                  "type": "image_url",
                  "image_url": {
                    "url": "'https://upload.wikimedia.org/
                       wikipedia/commons/f/f2/LPU-v1-die.jpg'"
                  }
                }
              ]
            }
          ],
          "model": "meta-llama/llama-4-scout-17b-16e-instruct",
          "temperature": 1,
          "max_completion_tokens": 1024,
          "top_p": 1,
          "stream": false,
          "stop": null
        }'
```

Fonte: https://console.groq.com/docs/vision

Já as chamadas usando Python podem ser observadas no Listing B.3 e B.4

Listing B.3 – Chamada à API do Gemini 2.5 Flash usando Python

```
from google import genai
from google.genai import types

with open('path/to/small-sample.jpg', 'rb') as f:
    image_bytes = f.read()

client = genai.Client()
response = client.models.generate_content(
  model='gemini-2.5-flash',
  contents=[
    types.Part.from_bytes(
      data=image_bytes,
      mime_type='image/jpeg',
    ),
    'Caption this image.'
  ]
)

print(response.text)
```

Fonte: https://ai.google.dev/gemini-api/docs/image-understanding?hl=pt-br#rest

Listing B.4 – Chamada à API do Groq usando Python

```python
from groq import Groq
import base64
import os

# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')

image_path = "sf.jpg"
base64_image = encode_image(image_path)
client = Groq(api_key=os.environ.get("GROQ_API_KEY"))
chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": [
                {"type": "text", "text": "What's in this image?"
                   },
                {
                    "type": "image_url",
                    "image_url": {
                        "url": f"{
                           base64_image}",
                    },
                },
            ],
        }
    ],
    model="meta-llama/llama-4-scout-17b-16e-instruct",
)

print(chat_completion.choices[0].message.content)
```

Fonte: https://console.groq.com/docs/vision

APÊNDICE C – ARQUIVO CSS COM OS ESTILOS

Arquivo `styles.css` usado para definir o estilo do *frontend*.

```css
.analysing-container {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  pointer-events: none;
  display: flex;
  flex-direction: column; /* Organiza os itens em coluna (GIF em
     cima, texto embaixo) */
  justify-content: center;
  align-items: center;
}
.analysing-gif {
  width: 80px;
  height: auto;
  filter: drop-shadow(0 0 4px #2896ff) drop-shadow(0 0 2px #2896
     ff);
}
.analysing-text {
  color: white;
  margin-top: 10px;
  font-size: 1em;
  font-weight: bold;
  text-shadow: 0 0 5px black; /* Sombra para legibilidade */
}
.censor-container {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  pointer-events: none;
```

```
}
.censor-image {
  width: 20%;
  height: auto;
  filter: drop-shadow(0 0 10px red);
}
.censor-text {
  color: red;
  font-weight: bold;
  text-align: center;
  margin-top: 10px;
  font-size: 1.2em;
  background-color: rgba(86, 86, 86, 0.5);
  padding: 5px 10px;
  border-radius: 5px;
}
.error-container {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  pointer-events: none;
}
.error-image {
  width: 20%;
  height: auto;
  filter: drop-shadow(0 0 10px yellow);
}
.error-text {
  color: yellow;
  font-weight: bold;
  text-align: center;
  margin-top: 10px;
  font-size: 1.2em;
  background-color: rgba(86, 86, 86, 0.5);
  padding: 5px 10px;
  border-radius: 5px;
```

```
78 }
79 .checked-box {
80   position: absolute;
81   top: 10px;
82   right: 10px;
83   background-color: gray;
84   opacity: 80%;
85   color: white;
86   padding: 5px 8px;
87   border-radius: 5px;
88   font-size: 12px;
89   pointer-events: none;
90   display: flex;
91   align-items: center;
92 }
93 .checkmark-image {
94   height: 1em; /* Faz a altura da imagem ser igual à altura da
        fonte */
95   width: auto;
96   margin-right: 5px; /* Espaçamento entre o texto e o símbolo */
97 }
```

Fonte: elaborado pelo autor