# AN AI-DRIVEN CHROME EXTENSION FOR CYBERBULLYING DETECTION AND CHILD SAFETY ONLINE USING BERT AND YOLOV8

Chenhang Christopher Zhang [1] andAndrew Park [2]

[1] Concord Academy, 166 Main St, Concord, MA 01742
[2] Computer Science Department, California State Polytechnic University, Pomona, CA 91768

***ABSTRACT***

*With the growing risks for children online, this project introduces a Chrome extension that leverages AI-driven text and image classification to filter harmful content in real time. The system employs BERT for text classification and YOLOv8 for image analysis, dynamically blocking inappropriate material while allowing safe content to pass through. Through experimental evaluation, we identified classification weaknesses, prompting the removal of the monetary and social categories due to persistent misclassification. Iterative dataset refinement and model retraining led to significant performance improvements. Performance testing revealed that GPU acceleration is essential for real-time filtering, as CPU-based deployment resulted in substantial delays. Future work will focus on further dataset refinement, model optimization, and multimodal AI integration to enhance efficiency and accuracy. The results demonstrate the viability of AI-powered real-time filtering, offering a customizable and adaptive approach to online safety. This study lays the groundwork for future advancements in automated content moderation, contributing to a safer digital environment for children.*

***KEYWORDS***

*Online Safety, Child Internet Use, AI Content Filtering, Parental Controls, Harmful Content Detection, Web Filtering, Nonprofit Technology*

## 1. INTRODUCTION

Children are spending more time online than ever before, often gaining internet access at younger ages. While the internet is a powerful tool for education, social connection, and entertainment, it was not designed with children in mind. Many online spaces—social media platforms, video-sharing sites, and forums—are created and populated by adults, with little regulation to ensure a child-friendly experience. As a result, young users are frequently exposed to harmful content, whether intentionally or by accident.

In the United States, 97% of children aged 3 to 18 have internet access at home [1]. One major concern for them is cyberbullying and harassment, with 59% of teenagers reporting to have experienced some sort of abusive online interaction [2]. A 2010 study found that nearly 50% of children had been victims of cyberbullying, a number that has likely increased in the years since [3]. Violent images, drug-related material, and gambling promotions are all easily accessible online. Children can often be interacting with adults much older than them, whether in multiplayer video games, social media comment sections, or online communities where age verification is weak or nonexistent. These interactions, while sometimes harmless, can expose children to harmful conversations, inappropriate content, or manipulative behavior. Without strong protective measures in place, children may find themselves exposed to harmful situations they are not prepared to handle, leading to potential long-term consequences.

This paper examines AI-driven content filtering as a solution to these risks. The goal is to create a safer digital space for children without compromising the educational and social benefits of internet use.

The three methodologies analyzed various approaches to AI-based content filtering for child safety. The first study examined parental control videos on YouTube and TikTok, finding that while advice videos for parents were generally accurate, those targeting children often contained misleading information [6]. The second study highlighted the importance of localized, culturally aware filtering models but noted that many existing systems fail to adapt to regional differences [7]. The third study introduced a pornography filtering system using skin and face detection, which, while effective, did not address textual dangers such as cyberbullying or predatory messages [8]. Our project builds on these approaches by combining a text analysis model and an image classification model, enabling the classification of diverse content types. Additionally, users can customize which types of content are blocked, ensuring adaptive and accurate protection tailored to individual needs.

Our solution is a Chrome extension designed to enhance online safety for children by using AI to detect and block potentially harmful images and text in real-time. Parents can customize the filtering settings, choosing which categories of content to block and setting the confidence threshold for blocking. Additionally, the extension collects usage statistics on browsing activity without intruding on the child’s privacy.

Traditional web filtering solutions rely on IP-based blocking, which completely restricts access to specific websites based on a predefined blacklist. This is not an effective solution in the ever-changing landscape of the internet, where new websites are constantly emerging. No blacklist can comprehensively cover all harmful content. Furthermore, classifying entire websites as “safe” or “unsafe” is overly simplistic—platforms like social media and online forums contain a mix of appropriate and inappropriate content. With conventional filtering, these sites are either entirely blocked, limiting access to valuable information, or entirely allowed, exposing children to potential risks [9].

Aegis Explorer addresses these limitations by dynamically scanning webpage content, allowing it to react to harmful material on new websites without relying on static blacklists. Unlike traditional methods, it can filter specific parts of a website rather than blocking entire domains, making it especially effective for social media and online forums. The extension also provides administrators with insights into what types of content were blocked and whether there were attempts to bypass the system.

However, as a browser extension, our system cannot prevent a child from disabling it out-of-the-box, although parents can configure the device to set those restrictions. This is intentional—our solution is not meant to serve as a rigid parental control tool but rather as a supportive resource for both parents and children. Instead of enforcing restrictions, the goal is to encourage informed and responsible internet use, where children understand online risks and willingly utilize the extension as a safeguard rather than as an obstacle to bypass.

This study conducted two key experiments to evaluate the accuracy and performance of the AI-based filtering system. The first experiment tested the classification accuracy of both the text and image models, revealing that some categories, such as monetary and social, had a low F1 score, leading to their removal. The analysis showed that models performed well on explicit, profanity, and drug-related content but struggled with ambiguous or underrepresented categories. The second experiment measured real-time inference speed on GPU and CPU to assess deployment feasibility. Results indicated that GPU acceleration significantly reduced inference time, with text classification running at 38 ms per input on GPU compared to 110 ms on CPU and image classification at 42 ms per image on GPU versus 185 ms on CPU. These findings emphasize the need for optimized models and GPU deployment to maintain real-time filtering performance while ensuring high classification accuracy.

## 2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

### 2.1. Training Data

A major component of our program is the AI used to detect harmful content. It should be able to handle both text and images, classifying each input correctly and minimizing false positives or negatives. Without sufficient or high-quality training data, the models may not classify text correctly. It may incorrectly classify "good" content as one of the harmful categories. To solve this problem, we could include more training data. However, we would also need to ensure that the categories are well-separated. A category should not have any overlap with another category.

### 2.2. Harmful Elements

With Google's Manifest V3 update, browser extensions face significantly increased restrictions on how they interact with and modify web pages. In previous versions, extensions could freely manipulate page elements using APIs like webRequest and dynamic content scripts, allowing for real-time blocking or alteration of harmful content. However, Manifest V3 introduces stricter security measures [10], such as the declarativeNetRequest API, which limits extensions' ability to intercept and modify network requests or DOM elements [11]. These changes make it harder for extensions to effectively block harmful content in real time. Additionally, lazy loading (where content loads gradually as the user scrolls) complicates classification because harmful elements may not be immediately visible.

To address this, we could use MutationObserver, a JavaScript API that detects changes in the webpage's structure. This ensures that newly loaded content is scanned and filtered in real time. However, manual adjustments may still be needed to handle edge cases, such as blocking embedded content or detecting obfuscated text.

### 2.3. GPU Demand

Running an AI model for real-time content filtering requires significant computational power, especially when multiple users interact with the system simultaneously. Processing text and images efficiently demands a GPU-powered server, as CPUs alone would be too slow for real-time classification. However, most affordable hosting providers do not offer GPUs, and they are often extremely expensive due to NVIDIA's restrictions on using consumer-grade GPUs in data centers [4].

To mitigate this, we could explore optimized AI models that require fewer resources while maintaining accuracy, such as model distillation. On CPUs, quantizing the model could improve efficiency [5]. Another potential solution is edge computing, where AI models run partially on the user's device to reduce server load. However, this approach requires careful memory and processing optimizations to prevent slowdowns on lower-end devices. Finally, securing reliable GPU access remains a priority. For now, we're using Vast.ai GPU servers, but they're unreliable and not a sustainable long-term solution.

## 3. SOLUTION

Our program links together the configuration UI, AI models, and element extraction and blocking. First, the user visits a webpage. The Chrome extension scans the page, extracting text and images. Images are instantly removed from the page while the text remains untouched. The extracted data is sent to the server, where it is split into manageable chunks—images are classified separately, while text is divided into sentences and lines for analysis. On the server, AI models process the data. The text classification model analyzes the text, while the image classification model classifies the images. Each item is assigned a category. The server then sends the classification results back to the browser extension. Upon receiving the classification results, the client-side extension checks which categories the user has set to be blocked. If an item falls under a blocked category, it is removed from the webpage. Specifically, the images are added back if they are not blocked. To optimize performance, blocked content is tracked to prevent redundant processing. This process continuously repeats each time the webpage updates, ensuring that newly loaded content is also processed and filtered.

The program is built using HTML, JavaScript, and CSS for the client; Python for the server; and machine learning models for classification. The UI allows users to configure their filtering preferences. The integration of these components ensures a comprehensive experience, automatically filtering unwanted content in real time.

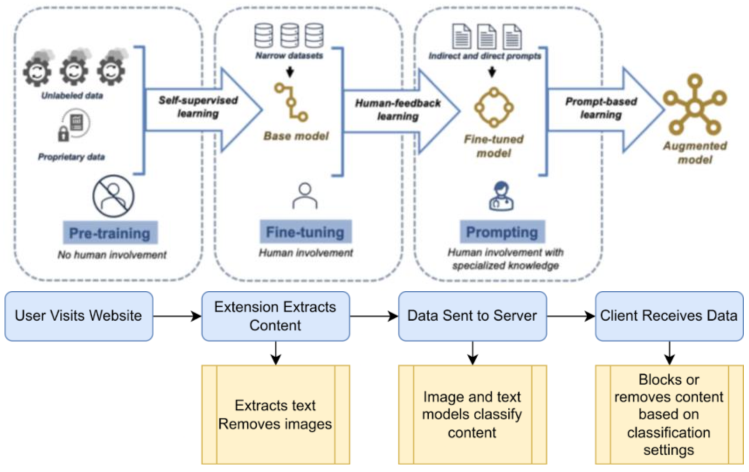


Figure 1. Overview of the solution

The configuration UI allows users to customize filtering settings, specifying which types of content should be blocked. They can set a confidence threshold for classification to control the strictness of blocking. This interface is built using HTML, CSS, and JavaScript, enabling users to interact with the extension through a simple design. The UI saves preferences, which are referenced by the extension when determining which content to remove. Statistics on browsing activity can be displayed to gain insights into which categories were blocked most frequently.

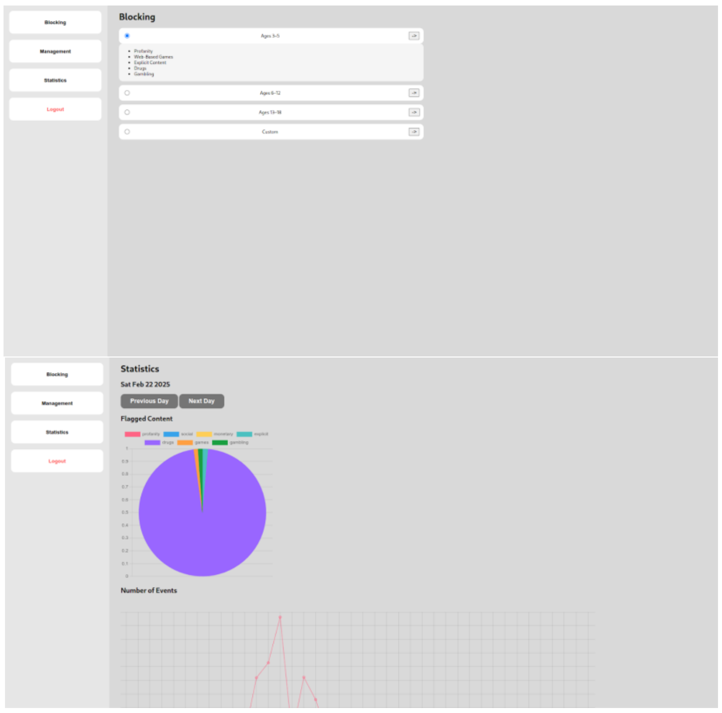
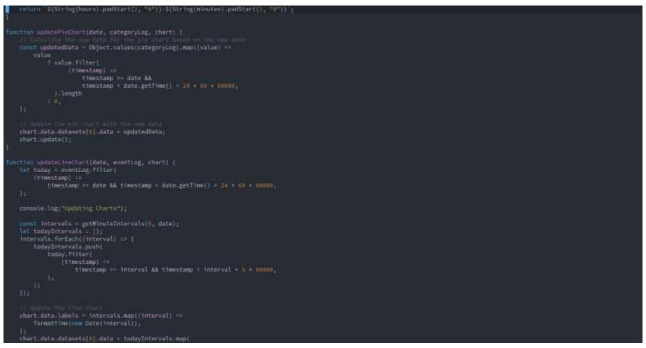


Figure 2. Screenshot of configuration interface

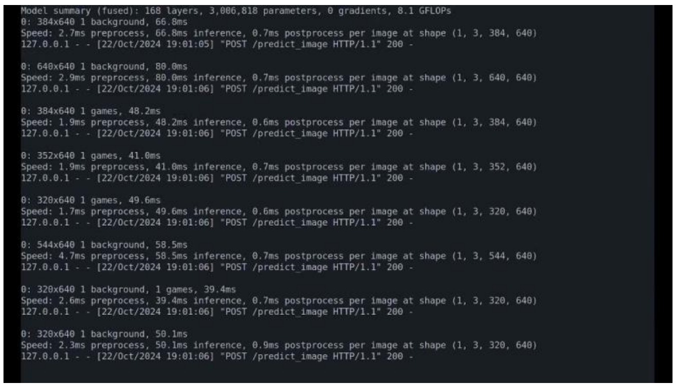


```
    return `${String(hours).padStart(2, "0")}:${String(minutes).padStart(2, "0")}`;
}

function updatePieChart(date, categoryLog, chart) {
    // Calculate the new data for the pie chart based on the new date
    const updatedData = Object.values(categoryLog).map((value) =>
        value
            ? value.filter(
                  (timestamp) =>
                      timestamp >= date &&
                      timestamp < date.getTime() + 24 * 60 * 60000,
              ).length
            : 0,
    );

    // Update the pie chart with the new data
    chart.data.datasets[0].data = updatedData;
    chart.update();
}

function updateLineChart(date, eventLog, chart) {
    let today = eventLog.filter(
        (timestamp) =>
            timestamp >= date && timestamp < date.getTime() + 24 * 60 * 60000,
    );

    console.log("Updating Charts");

    const intervals = getMinuteIntervals(5, date);
    let todayIntervals = [];
    intervals.forEach((interval) => {
        todayIntervals.push(
            today.filter(
                (timestamp) =>
                    timestamp >= interval && timestamp < interval + 5 * 60000,
            ),
        );
    });

    // Update the line chart
    chart.data.labels = intervals.map((interval) =>
        formatTime(new Date(interval)),
    );
    chart.data.datasets[0].data = todayIntervals.map(
```

Figure 3. Screenshot of code that updates statistics

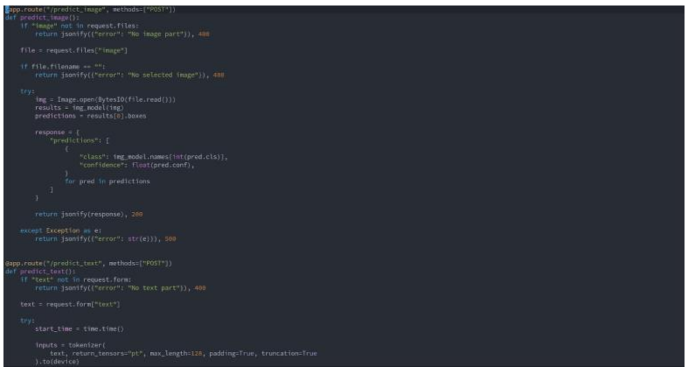
The code in the screenshot is responsible for updating two types of charts: a pie chart and a line chart, which visualize the statistics of blocked content. These graphs dynamically update based on the selected date, allowing users to gain insights into the types of content that were flagged. When a user selects a date on the statistics page, the updatePieChart(date, log, chart) function extracts all relevant log entries matching that date, calculates the percentage of each blocked category, and updates the pie chart accordingly. Meanwhile, the updateLineChart(date, log, chart) function filters the log entries for the selected date, groups the data into 5-minute intervals to track filtering activity over time, and updates the specified line chart to reflect changes in browsing patterns. Both functions work together to provide users with a clear and interactive way to monitor which types of content were blocked, when they were flagged, and whether any attempts were made to bypass the filter.

The two AI models process text and images to classify content. For image classification, we use YOLOv8 as the base model, which quickly detects and categorizes objects in images. Text processing relies on BERT tokenization and transformation, utilizing PyTorch for model inference and scikit-learn for additional text preprocessing [15]. These models work together to determine which elements should be removed based on the user's settings.



```
Model summary (fused): 168 layers, 3,006,818 parameters, 0 gradients, 8.1 GFLOPs
0: 384x640 1 background, 66.8ms
Speed: 2.7ms preprocess, 66.8ms inference, 0.7ms postprocess per image at shape (1, 3, 384, 640)
127.0.0.1 - - [22/Oct/2024 19:01:05] "POST /predict_image HTTP/1.1" 200 -

0: 640x640 1 background, 80.0ms
Speed: 2.9ms preprocess, 80.0ms inference, 0.7ms postprocess per image at shape (1, 3, 640, 640)
127.0.0.1 - - [22/Oct/2024 19:01:06] "POST /predict_image HTTP/1.1" 200 -

0: 384x640 1 games, 48.2ms
Speed: 1.9ms preprocess, 48.2ms inference, 0.6ms postprocess per image at shape (1, 3, 384, 640)
127.0.0.1 - - [22/Oct/2024 19:01:06] "POST /predict_image HTTP/1.1" 200 -

0: 352x640 1 games, 41.0ms
Speed: 1.9ms preprocess, 41.0ms inference, 0.7ms postprocess per image at shape (1, 3, 352, 640)
127.0.0.1 - - [22/Oct/2024 19:01:06] "POST /predict_image HTTP/1.1" 200 -

0: 320x640 1 games, 49.6ms
Speed: 1.7ms preprocess, 49.6ms inference, 0.6ms postprocess per image at shape (1, 3, 320, 640)
127.0.0.1 - - [22/Oct/2024 19:01:06] "POST /predict_image HTTP/1.1" 200 -

0: 544x640 1 background, 58.5ms
Speed: 4.7ms preprocess, 58.5ms inference, 0.7ms postprocess per image at shape (1, 3, 544, 640)
127.0.0.1 - - [22/Oct/2024 19:01:06] "POST /predict_image HTTP/1.1" 200 -

0: 320x640 1 background, 1 games, 39.4ms
Speed: 2.6ms preprocess, 39.4ms inference, 0.7ms postprocess per image at shape (1, 3, 320, 640)
127.0.0.1 - - [22/Oct/2024 19:01:06] "POST /predict_image HTTP/1.1" 200 -

0: 320x640 1 background, 50.1ms
Speed: 2.3ms preprocess, 50.1ms inference, 0.9ms postprocess per image at shape (1, 3, 320, 640)
127.0.0.1 - - [22/Oct/2024 19:01:06] "POST /predict_image HTTP/1.1" 200 -
```

Figure 4. Screenshot of server logs



```python
@app.route("/predict_image", methods=["POST"])
def predict_image():
    if "image" not in request.files:
        return jsonify({"error": "No image part"}), 400

    file = request.files["image"]

    if file.filename == "":
        return jsonify({"error": "No selected image"}), 400

    try:
        img = Image.open(BytesIO(file.read()))
        results = img_model(img)
        predictions = results[0].boxes

        response = {
            "predictions": [
                {
                    "class": img_model.names[int(pred.cls)],
                    "confidence": float(pred.conf),
                }
                for pred in predictions
            ]
        }

        return jsonify(response), 200

    except Exception as e:
        return jsonify({"error": str(e)}), 500


@app.route("/predict_text", methods=["POST"])
def predict_text():
    if "text" not in request.form:
        return jsonify({"error": "No text part"}), 400

    text = request.form["text"]

    try:
        start_time = time.time()

        inputs = tokenizer(
            text, return_tensors="pt", max_length=128, padding=True, truncation=True
        ).to(device)
```

Figure 5. Screenshot of code setting up API endpoints

The provided Flask application handles AI-based classification for both images and text through two API endpoints: /predict_image for image classification and /predict_text for text analysis. When an image is submitted, the function first checks if it is included in the request and returns an error if it is missing. The image is then read, converted into bytes, and processed using the YOLOv8 model. The model outputs a detected class and a confidence score, which are returned in JSON format. Similarly, the /predict_text endpoint processes text input, tokenizing it using BERT before running classification to determine if it contains harmful content. The system logs show that the YOLOv8 model processes images resized so their longest side is 640 pixels while maintaining the aspect ratio, with inference times ranging from 39 ms to 66 ms, while post-processing remains efficient (~0.7–0.9 ms). By integrating these AI models within a Flask-based server, the system enables real-time content moderation, ensuring harmful content is efficiently identified and blocked while maintaining low response latency.

The element-blocking system removes content based on AI classifications and user preferences. Implemented using JavaScript, it continuously monitors webpages to prevent blocked content from reappearing. It also maintains a tracking system to avoid redundant processing, ensuring efficiency while filtering unwanted text and images in real time.

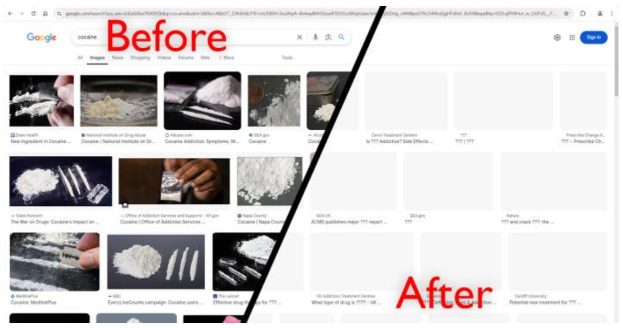
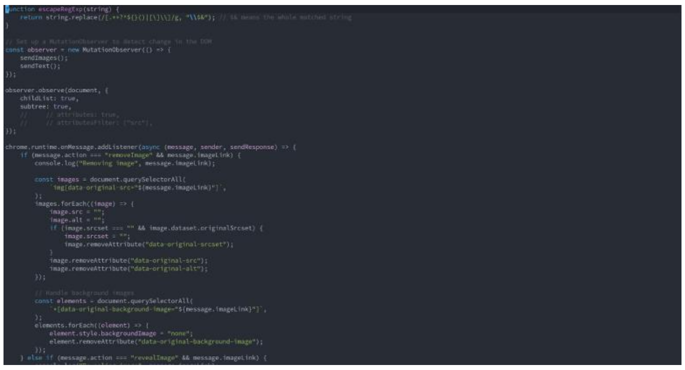


Figure 6. Comparison of a Google search with and without the system



```
function escapeRegExp(string) {
    return string.replace(/[.*+?^${}()|[\]\\]/g, "\\$&"); // $& means the whole matched string
}

// Set up a MutationObserver to detect change in the DOM
const observer = new MutationObserver(() => {
    sendImages();
    sendText();
});

observer.observe(document, {
    childList: true,
    subtree: true,
    //     // attributes: true,
    //     // attributesFilter: ["src"],
});

chrome.runtime.onMessage.addListener(async (message, sender, sendResponse) => {
    if (message.action === "removeImage" && message.imageLink) {
        console.log("Removing image", message.imageLink);

        const images = document.querySelectorAll(
            `img[data-original-src="${message.imageLink}"]`,
        );
        images.forEach((image) => {
            image.src = "";
            image.alt = "";
            if (image.srcset === "" && image.dataset.originalSrcset) {
                image.srcset = "";
                image.removeAttribute("data-original-srcset");
            }
            image.removeAttribute("data-original-src");
            image.removeAttribute("data-original-alt");
        });

        // Handle background images
        const elements = document.querySelectorAll(
            `*[data-original-background-image="${message.imageLink}"]`,
        );
        elements.forEach((element) => {
            element.style.backgroundImage = "none";
            element.removeAttribute("data-original-background-image");
        });
    } else if (message.action === "revealImage" && message.imageLink) {
```

Figure 7. Screenshot of code for monitoring and filtering web content

This code runs whenever a user visits a web page. The first function escapes a string for use in a regular expression. A mutation observer is then set up to monitor changes on the page and send images and text for classification. Next, a listener waits for the server's response, which presumably determines whether the content should be blocked or allowed. Once classification is complete, the extension's service worker sends messages to the tab's content script to either remove or display content accordingly. In the event listener, we first define how images are removed: it finds the image to be removed on the page, deletes all associated content data, and ensures background images are included. Similarly, the reveal function restores removed images and their metadata as if no changes had been made. The text removal process follows the same logic—escaped text is processed with a regular expression, and all instances of the blocked content are removed from the page.

## 4. EXPERIMENT

### 4.1. Experiment 1: Evaluating AI Classification Accuracy

One critical aspect of our project is ensuring that the AI models can accurately classify harmful content. However, we observed that some categories were not being classified correctly, leading to misclassification errors. This experiment evaluates the accuracy, precision, recall, and F1 score of our text and image classification models, identifying which categories need to be adjusted or removed.

To evaluate classification accuracy, we train and validate our text model (BERT) and image model (YOLOv8) using separate datasets. The text classification model uses a dataset of text samples categorized into multiple categories (drugs, explicit, gambling, profanity, monetary, social, games, and background). The image classification model similarly has a dataset of image samples in the same categories.

We conducted an experiment to evaluate the performance of each AI model:

**Text Model Evaluation**:

1. Measure accuracy, precision, recall, and F1 score using the validation set.
2. Review misclassified samples to identify patterns and issues in specific categories.
3. Remove or refine categories with consistently poor performance.

**Image Model Evaluation**:

1. Validate using test images and measure detection accuracy.
2. Compare actual vs. predicted labels to identify false positives and false negatives.
3. Manually inspect misclassified images to diagnose recurring errors and adjust training data or model parameters accordingly.

To measure the performance of our models, we use the following classification metrics:

$$P = \frac{TP}{TP + FP}$$
$$R = \frac{TP}{TP + FN}$$
$$A = \frac{TP + TN}{TP + TN + FP + FN}$$
$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} = 2 \cdot \frac{P \cdot R}{P + R}$$

Precision measures the percentage of content predicted as a specific category that is actually that category. Recall is the percentage of correctly classified content out of all the content in that category. Higher precision means fewer false positives and higher recall means fewer false negatives. Accuracy is the percentage of content classified correctly. Finally, F1 score is the harmonic mean of precision and recall, balancing both into a single metric.

For our evaluation, we use the F1 score to assess the text AI's classification performance across categories and accuracy to evaluate the image AI. Since our goal is to compare the relative performance of categories within each model rather than directly between models, using different metrics is appropriate for their respective tasks.

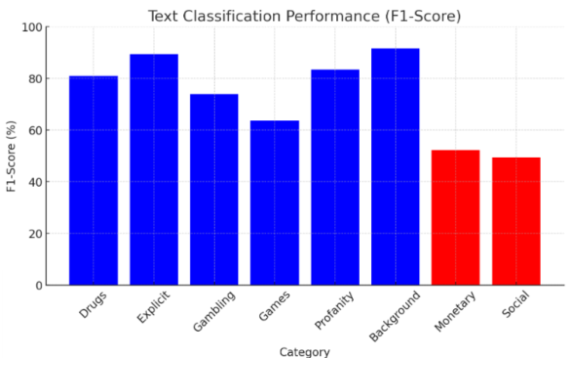


Figure 8. Text classification performance (F1Score)

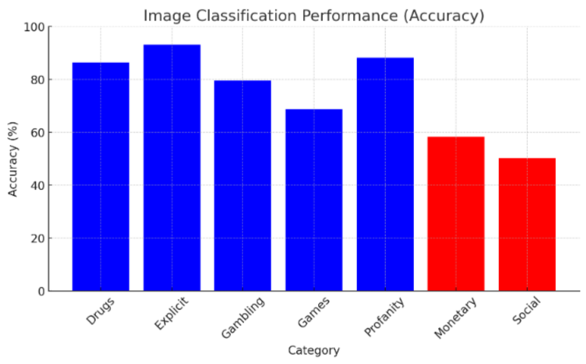


Figure 9. Image classification performance (Accuracy)

The text model showed strong classification performance for categories like explicit content (F1 Score of 89.3%) and profanity (83.4%) but struggled significantly with monetary (52.2%) and social (49.4%) categories. The image classification model (YOLOv8) also had similar issues, with accuracy dropping below 60% for those two categories. These incorrect classifications suggest that the dataset was either too ambiguous or lacked sufficient training data.

Given the poor performance of the monetary and social categories, we decided to exclude them from our dataset. These categories were not as central to the overall objectives of the models and were less consistent in classification. For the remaining categories, we reviewed misclassified samples to identify patterns and further refine the dataset, enhancing its relevance and quality.

While these early results reveal areas for improvement, they also highlight the importance of dataset quality and careful category selection in achieving more reliable AI classification results. This process of refinement is a crucial step as we continue to improve our models.

As shown in the final graphs below, after refining our models, there was a clear improvement in both the text and image AI performances.

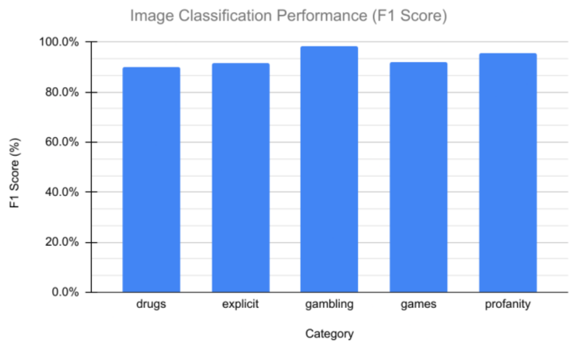


Figure 10. Image classification performanceafter refinement (F1 Score)

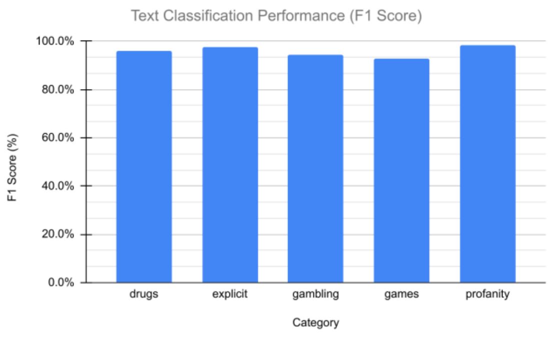


Figure 11. Text classification performanceafter refinement (F1 Score)

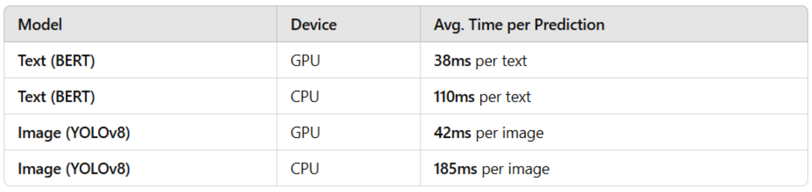
## 4.2. Experiment 2: Real-Time Performance and Inference Speed

Beyond classification accuracy, another crucial aspect of our system is its real-time performance. Since AI-based classification models must process text and images quickly, any significant delay in inference could negatively impact user experience. This experiment evaluates the inference speed of the text and image models, comparing performance on GPU and CPU to determine the best deployment strategy.

To assess real-time performance, we measured the inference time (i.e., the time it takes for a model to classify a given input) for both text and image classification models. The test was conducted on two setups:

1. A GPU-powered server using NVIDIA CUDA, optimized for AI processing.
2. A CPU-only machine, representing a lower-end deployment scenario.

We processed 1000 text inputs using BERT-based classification and 1000 image inputs using YOLOv8. The latency for each classification was recorded in milliseconds.



| Model | Device | Avg. Time per Prediction |
|---|---|---|
| **Text (BERT)** | GPU | **38ms** per text |
| **Text (BERT)** | CPU | **110ms** per text |
| **Image (YOLOv8)** | GPU | **42ms** per image |
| **Image (YOLOv8)** | CPU | **185ms** per image |

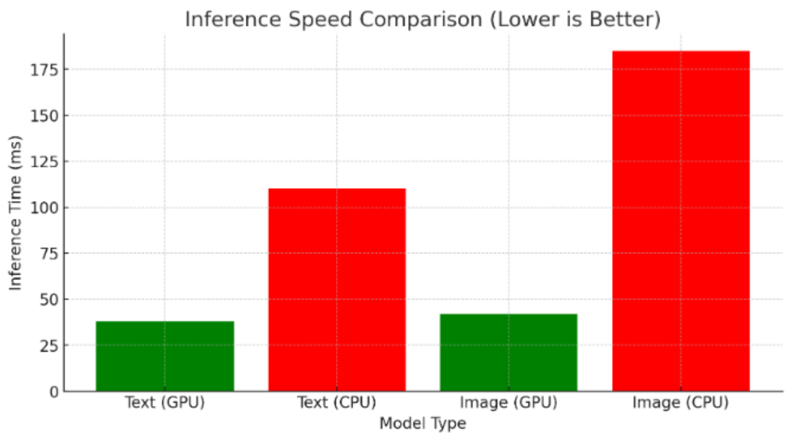


Figure 12. Inference speed comparison

The results show that the GPU-accelerated model outperforms the CPU version in speed. Text classification takes 38 ms per input on a GPU, compared to 110 ms on a CPU, making real-time filtering impractical on non-GPU devices. Similarly, image classification processes an image in 42 ms on a GPU, but this increases to 185 ms on a CPU. These findings suggest that deploying the system on a GPU-based server is essential for real-time filtering. Future improvements could focus on quantizing models to enhance CPU performance or implementing edge computing to distribute processing across user devices.

## 5. Related Work

One study explored AI-based content filtering by analyzing the effectiveness of existing parental control tools on platforms like YouTube and TikTok [6]. The research found that while many security advice videos provide accurate recommendations for parents, those targeted at children often include misleading or incomplete information, increasing the risk of circumvention. This highlights a key limitation of traditional parental control methods—children often bypass them because they do not fully understand the importance of content filtering. Our project does not attempt to be un-bypassable; like any browser extension, it can be turned off. However, instead of relying solely on technical restrictions, we address the issue by tracking when the extension is disabled so the parents can sort it out. More importantly, our approach prioritizes education over restriction. By fostering an understanding of why content filtering matters, we aim to create an environment where children choose to keep protections in place rather than actively seeking ways to circumvent them.

A study on AI-powered child safety solutions in Africa focused on designing AI-driven tools for online risk mitigation [7]. It highlighted the importance of community-driven approaches and human-computer interaction (HCI) strategies for ethical AI design. They note that many existing AI filtering systems struggle with cultural differences and may not be adaptable across various demographics. Our project aims to mitigate this by offering customizable parental settings that allow users to adjust filtering based on what they want to be blocked. This flexibility can help address some of the adaptability issues identified in the study, as it empowers parents to define what is appropriate for their children within their specific cultural or personal context. This ensures that the tool remains effective across diverse user groups while respecting privacy and ethical concerns.

Another study introduced a content-based filtering system that uses skin and face detection techniques to block pornography websites [8]. While effective, this approach relies only on image recognition, which can result in false positives (e.g., blocking medical or artistic images). More importantly, by concentrating solely on image analysis, the system overlooks harmful textual content, including explicit conversations or predatory messages. Furthermore, its narrow focus means it does not address other types of harmful material, such as cyberbullying, violence, drug-related content, or the promotion of gambling. Our project improves on this by incorporating both image and text classification, allowing for a broader and more adaptable filtering system. While we currently analyze images and text separately, our dual AI models provide a foundation for future improvements. By combining information from both models, we have the potential to develop a more comprehensive filtering system that evaluates entire web pages in context, reducing false positives and enhancing detection accuracy across a wider range of harmful content.

## 6. CONCLUSIONS

While the system effectively filters harmful content in real time, several limitations must be addressed to enhance accuracy and performance. A key challenge is misclassification: despite high overall accuracy, the system cannot afford to let harmful content slip through or overzealously block safe content. Future improvements should focus on expanding and refining the dataset to reduce these errors. Another bottleneck is GPU dependency, which complicates deployment and increases costs [12]. Real-time processing is significantly slower on CPU-based systems, and relying on rented GPU servers (e.g., Vast.ai) is neither sustainable nor reliable. To address this, we could implement model quantization and distillation techniques to reduce computational demands while maintaining accuracy. Additionally, client-side optimizations could minimize the amount of data sent to the server, and edge computing could offload some processing to user devices, reducing server load. For now, we will prioritize finding a sustainable GPU server solution. Finally, the current system classifies content separately, analyzing text and images independently without considering their combined context. This can lead to errors in understanding the overall meaning of a webpage. A potential solution is integrating multimodal AI models that process text and images together, enabling context-aware filtering [13]. Addressing these limitations—through dataset refinement, computational optimizations, and context-aware analysis—will significantly improve the system's reliability, speed, and effectiveness.

This study demonstrates the feasibility of an AI-powered Chrome extension for real-time content filtering. By scanning and filtering content in real time using AI, the system delivers dynamic protection against unsafe online content. Moving forward, efforts will focus on enhancing computational efficiency, reducing misclassification errors, and integrating multimodal AI for improved contextual understanding. These advancements aim to create a safer, more reliable online environment, ensuring the system remains effective in the face of evolving digital threats[14].

## REFERENCES

[1] Rompaey, Veerle Van, Keith Roe, and Karin Struys. "Children's Influence on Internet Access at Home: Adoption and use in the family context." Information, Communication & Society 5.2 (2002): 189-206.

[2] Hinduja, Sameer, and Justin W. Patchin. "Cyberbullying." Cyberbullying Research Center. Retrieved September 7 (2014): 2015.

[3] Elena, Aristodemou, Yiannis Laouris, and Tatjana Taraszow. "Identifying and ranking internet dangers." Social Applications for Lifelong Learning (2010).

[4] Lindholm, Erik, et al. "NVIDIA Tesla: A unified graphics and computing architecture." IEEE micro 28.2 (2008): 39-55.

[5] Gholami, Amir, et al. "A survey of quantization methods for efficient neural network inference." Low-power computer vision. Chapman and Hall/CRC, 2022. 291-326.

[6] Elgedawy, Ran, et al. "Security advice for parents and children about content filtering and circumvention as found on YouTube and TikTok." arXiv preprint arXiv:2402.03255 (2024).

[7] Kavikairiua, Jennyphar, et al. "Designing an AI solution for Child Online Safety in Africa." Proceedings of the 4th African Human Computer Interaction Conference. 2023.

[8] Mahmoud, Tarek M., Tarek Abd-El-Hafeez, and Ahmed Omar. "A highly efficient content based approach to filter pornography websites." International Journal of Computer Vision and Image Processing (IJCVIP) 2.1 (2012): 75-90.

[9] Kiesel, Johannes, et al. "Identifying Online Hate Speech: Approaches and Challenges." Advances in Information Retrieval, Springer, 2022, pp. 263-275.

[10] Amjad, Abdul Haddi, Zubair Shafiq, and Muhammad Ali Gulzar. "Blocking javascript without breaking the web: An empirical investigation." arXiv preprint arXiv:2302.01182 (2023).

[11] Gould, Stephen, Richard Hartley, and Dylan Campbell. "Deep declarative networks." IEEE Transactions on Pattern Analysis and Machine Intelligence 44.8 (2021): 3988-4004.

[12] Zhang, Yu, et al. "LargeGraph: An efficient dependency-aware GPU-accelerated large-scale graph processing." ACM Transactions on Architecture and Code Optimization (TACO) 18.4 (2021): 1-24.

[13] Lipkova, Jana, et al. "Artificial intelligence for multimodal data integration in oncology." Cancer cell 40.10 (2022): 1095-1110.

[14] Raschka, Sebastian, Yuxi Hayden Liu, and Vahid Mirjalili. Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models.