Development of a Browser Extension for Toxic Content Detection Using an NLP Model

Oliver Lohaj, Vladyslav Stakhov
Institute of Artificial Intelligence
Faculty of Electrical Engineering and Informatics
Technical university of Košice, Letná 1/9, 042 00 Košice, Slovakia
oliver.lohaj@tuke.sk, vladyslav.stakhov@student.tuke.sk

Abstract*. Toxic and abusive language is widespread in online communication, while many detection systems rely on remote processing that may expose sensitive user text. This paper presents a hybrid approach to toxic-language detection that combines privacy-preserving local inference with an optional remote multilingual model in a Chrome browser extension. The local solution uses a character-level TensorFlow.js GraphModel, while the remote mode employs XLM-R-based classification. The system supports six toxicity categories and those are integrated into a browser interface that can identify, mark, and blur potentially harmful content. The evaluation included a formative interface assessment, comparison of three remote XLM-R variants, evaluation of a pretrained TensorFlow.js toxicity baseline, and testing of the optimized local model. The local model achieved an F1 score of 0.734 for the binary toxic/non-toxic verdict, while the WebAssembly backend was approximately 13.5 times faster than the CPU backend. The results show that browser-based toxicity detection is feasible and that a hybrid architecture can balance privacy, classification quality, and runtime performance.*

Keywords. browser extension, Chrome Manifest V3, NLP, privacy, TensorFlow.js, toxicity detection

1 Introduction

Internet communications have become a regular feature in the lives of many people. Websites such as social media sites, discussion forums, comment sections, and other platforms allow individuals to express their views and interact with other people freely and almost instantly. The freedom that exists on the Internet has many positive features; however, it leaves room for toxic and abusive material. Internet users may be subjected to toxic content in the form of insulting, profane, threatening messages and other types of abusive content.

That's why automatic detection of toxic content is an important task in Natural Language Processing (NLP). In many existing solutions, toxicity is detected on the server side. This means that the text is sent to an external service and classified by a model after which the result is returned to the user. Such an approach can use stronger models and can be easier to update, but it also raises privacy concerns for users. Text selected or read by the user may contain sensitive information and sending it outside the device is not always desirable.

The primary goal of this project was to develop and deploy a browser extension, which would allow people to detect toxic information while surfing the web pages. The proposed browser extension is called Toxic Text Detector, and its primary focus is to use toxicity detection in practice through the browser and find a balance between four aspects: accuracy, privacy, speed, and control. Considering all the above-mentioned considerations, the browser extension employs hybrid architecture with two modes: inference in-browser for the local mode, and a more powerful backend model for the remote mode.

The contribution of this work is primarily system-oriented and empirical rather than the introduction of a new NLP architecture. First, we present a functional Chrome Manifest V3 extension that integrates privacy-preserving local inference and optional remote inference within a unified user interface. Second, we evaluate the practical suitability of both a pretrained TensorFlow.js toxicity classifier and an optimized character-level TensorFlow.js GraphModel for in-browser deployment. Third, we quantify the effect of the CPU and WebAssembly execution backends on local inference performance. Finally, the proposed solution combines per-label decision thresholds, configurable toxicity categories, reversible content blurring, persistent user settings, and formative interface evaluation in a complete browser-based prototype.

2 Current State of the Art

Toxicity detection is usually regarded as a multi-label classification task. A single comment may belong to more than one label category like toxicity, severe toxicity, obscenity, insult, threat, or any form of identity-based hate speech. Such a classifying scheme is employed in the task of Jigsaw Toxic Comment Classification Challenge dataset and other related datasets (Jigsaw, 2018). With respect to the extension, multi-label classification becomes crucial because, at times, simple classification to toxic or not toxic is not sufficient.

The solution at hand is also subject to the general rules of multi-label classification. Namely, rather than training or testing a single binary classifier, it needs to calculate multiple independent scores and select appropriate labels based on those (Tsoumakas and Katakis, 2007). This idea is implemented in the system through using per-class scores and per-class thresholds. This becomes highly important since toxicity datasets suffer from a major imbalance problem. Categories such as threat or identity hate usually contain far fewer positive examples than the general toxic category.

One of the other important aspects of the state of the art at this stage is developing browser extensions. The development of the extension took place according to Chrome Manifest Version 3. Manifest V3 brings changes to the architecture of the browser extensions due to the use of the event-driven service worker instead of a persistent background page and due to security constraints (Google Chrome Developers, 2026). It is particularly vital to have security constraints in browser extensions used for machine learning.

The reason why TensorFlow.js is important is due to its capability to perform machine learning inference in JavaScript environments like web browsers. The existing model that was developed for TensorFlow.js shows that the technical possibility of local toxicity detection exists within the context of browser-like environments (TensorFlow.js Team, 2026). But the actual application of the model in a browser extension will rely not only on running the model but other factors as well.

In terms of the remote setting, the use of multilingual transformer-based models is justified by the fact that such models typically produce better results when compared to classical models. Specifically, in our case, were used the variations of the XLM-R models for the remote tests. The reason behind choosing XLM-R is the fact that it is a multilingual transformer model aimed at cross-lingual representation learning (Conneau et al., 2020).

Existing browser-based harmful content detection also shows that integrating toxicity detection into the user's browsing flow is realistic (Sikiandani, 2025). However, the way inference is deployed strongly affects privacy. Inference on the server side gives flexibility, but the data leaves the client's device. On the other hand, local inference offers privacy; however, an efficient and compact model must be available. These constraints gave rise to the hybrid approach of Toxic Text Detector.

Related approaches also include lightweight and multimodal solutions. Kumar et al. (2024) proposed ToxiGuard, which combines conventional NLP preprocessing with a bag-of-words representation and Naïve Bayes classification. This approach is computationally efficient and interpretable, but it has limited ability to capture contextual and implicit toxicity. Deep et al. (2024) presented a browser plug-in that detects harmful content in both text and images, although multimodal processing introduces higher computational and privacy requirements. In addition, resources such as Perspective API (Jigsaw, 2026), Jigsaw Unintended Bias (Jigsaw, 2019), HateXplain (Mathew et al., 2021), and ToxiGen (Hartvigsen et al., 2022) highlight important issues related to remote inference, fairness, explainability, and implicit or adversarial toxicity. These works show that practical browser-based detection requires balancing predictive performance with privacy, latency, transparency, and minority-class performance.

3 Design of the Proposed Solution

The developed solution is a Chrome Manifest V3 browser extension. It is divided into several main components: the user interface, the content script, the service worker, the settings layer, and the inference layer. Fig. 1 presents the architecture and data flow of the proposed extension, including communication among the user interface, content script, service worker, persistent storage, and the local and remote inference modes.

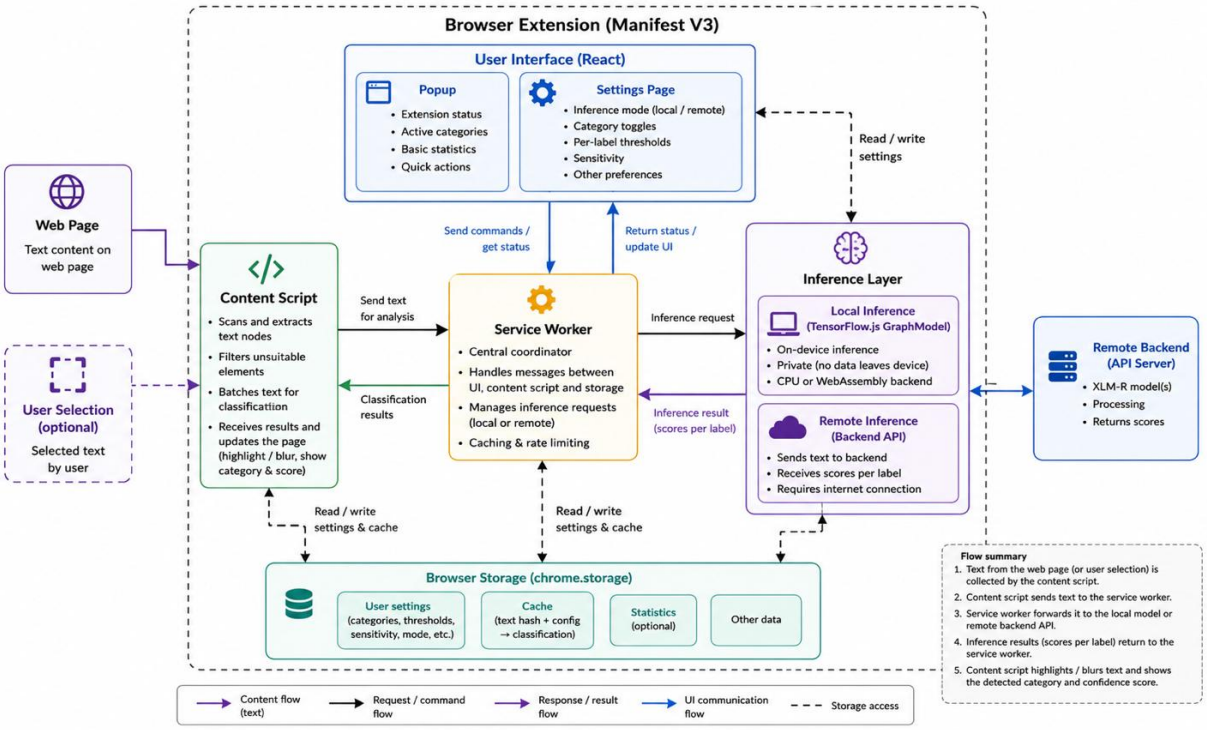


Figure 1. Architecture and data flow of the Toxic Text Detector extension

The user interface is implemented with React. It consists mainly of a popup window, a settings page, and visual elements inserted into the web page. The popup window provides a quick overview of the extension state. It shows whether protection is enabled, how many categories are active, and basic activity information from the current page. The settings page allows the user to configure the behaviour of the extension. The user can change detection sensitivity, enable or disable selected toxicity categories, turn automatic blurring on or off, show or hide confidence scores, manage whitelisted domains, and reset or import/export settings.

Content Script handles everything related to the web page itself. Content Script can access text chosen by the user, extract appropriate text nodes from the Document Object Model, classify text, and highlight its classification visually within the page. In case where classified text is toxic and auto-blur feature is enabled; Content Script will wrap it into a special HTML tag and apply a blurring effect to it. This will enable user to show/hide text once again, making such action reversible.

The responsibility of managing the communication among popup, settings page, and content scripts lies with the service worker. In case of Manifest V3 service workers, the service worker doesn't have access to the DOM of the webpage. Therefore, the tasks related to the webpage will be carried out by the content script.

The settings layer stores user preferences in browser storage. The extension keeps information about whether protection is enabled, which categories are active, what sensitivity level is selected, whether confidence values should be shown, and which domains are whitelisted. Domain normalization is used so that equivalent forms of a domain are treated consistently.

Two modes are implemented by the inference layer. The first one – called local – classifies text right in the web browser. It is privacy-friendly since the text analyzed is not sent anywhere from the user's device. Another mode is remote, where text analysis happens on the side of a backend API. This mode is useful when experiments are required with more powerful models and/or local models are not enough. Both modes have a consistent output format for the sake of proper presentation within the UI.

4 Implementation

The project was developed using a Chrome Manifest V3 extension. The build process makes use of Vite. There are two entry points to the project – one for the React popup and one for the settings page. The two modules share the same settings context. Therefore, the state presented on both components will be identical.

The content script contains the main page-processing logic. It defines limits for the minimum and maximum length of text, the maximum number of candidate text nodes, the batch size for classification, and the size of the local cache. These limits are needed because scanning an entire page without restrictions could slow down the browser. The content script skips

unsuitable elements such as script, style, input, and textarea elements. It also avoids text that has already been marked by the extension.

To avoid performing the same calculations repeatedly, caching was introduced. Caching means storing the classification output for certain text with a particular configuration of categories and sensitivities. When the configuration changes, the cache becomes empty to avoid using outdated data with new parameters.

In the local inference stage, a pre-trained TensorFlow.js toxicity model was used first. It could have been an excellent choice since it served as the base model but it failed as a local model due to its low accuracy and slow speed. In the last local mode, is used TensorFlow.js GraphModel that is created with character-level inputs. The input text will be standardized and trimmed down to a maximum length of 256 characters. The text will be converted into character vocabulary and then fed into the model. Six scores are obtained corresponding to toxic, severe toxic, obscene, insult, threat, and identity hate.

Remote mode involves the backend API. The provider credentials do not need to be included directly within the extension client code. The backend will be able to determine the length of the input, impose rate limits, calculate latency and provide consistent results in its responses. It is critical that the API keys or model implementation details are not saved within the browser extension.

4.1 User Interface Design

The user interface was developed iteratively. An initial Figma prototype was used to verify the interaction concept, including blurring and revealing toxic content, displaying the reason for detection, and accessing settings. A second functional prototype was then implemented in React and integrated with the extension. Based on formative testing, the visual hierarchy was improved, contextual actions were moved closer to the relevant content, category controls were clarified, and the distinction between protected and unprotected states was strengthened (Fig. 2.) The source code, trained model weights, and evaluation scripts will be made publicly available upon acceptance.

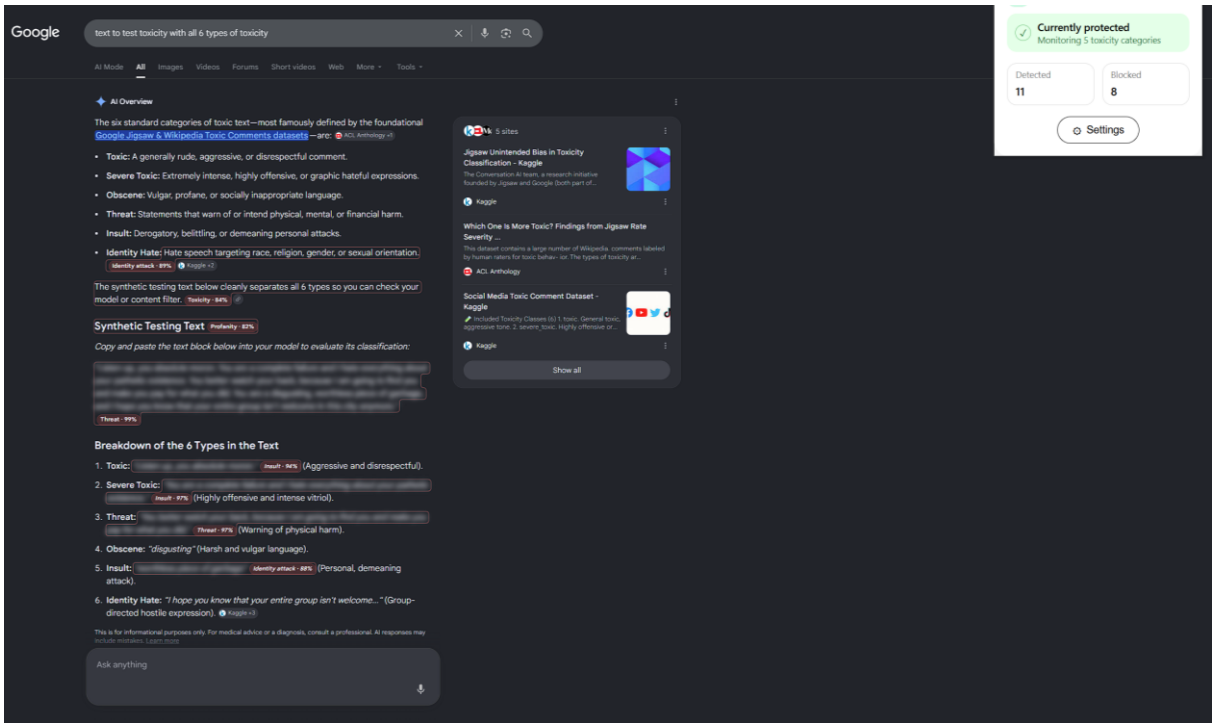


Figure 2. Example of toxic text marking and blurring in the browser

5 Experiments

The experiments used the English-language Jigsaw Toxic Comment Classification dataset with six labels: toxic, severe toxic, obscene, insult, threat, and identity hate. The baseline split contained 143,613 training comments and 15,958 validation comments. The dataset was strongly imbalanced: in the validation set, positive-label proportions were 9.56% for toxic, 0.90% for severe toxic, 5.18% for obscene, 4.88% for insult, 0.26% for threat, and 0.83% for identity hate. Oversampling was applied only to the training data for selected experiments, while the validation set remained unchanged.

This evaluation process was carried out in four stages. The first stage consisted of user interface

assessment through prototype testing. The second stage involved the remote inference mode assessment via XLM-R model versions. The third stage involved pretrained TensorFlow.js toxicity model evaluation, which acted as the local baseline. The fourth stage involved the optimized local model, which was implemented as a character-level convolutional neural network exported to the TensorFlow.js GraphModel format. Each input text was normalized, truncated to a maximum length of 256 characters, and encoded using a character vocabulary. The model produced six independent scores corresponding to the Jigsaw label set: toxic, severe toxic, obscene, insult, threat, and identity hate. It was evaluated on the Jigsaw2018 validation set containing 15,958 comments using TensorFlow.js 4.22.0 and a batch size of 64. Separate decision thresholds were selected for each label by maximizing its F1 score on the validation set; the resulting thresholds ranged from 0.90 to 0.94.

The interface was evaluated formatively with five participants in two prototype iterations. Participants completed tasks involving revealing blurred content, interpreting detection results, and modifying settings. The second prototype used simulated classification outputs.

Regarding the experiment conducted with remote models, three versions of XLM-R model were used. Specifically, these are as follows: xlmr-toxic-v1, xlmr-toxic-v2, and xlmr-toxic-v2_1. All models were evaluated with respect to the Jigsaw2018 validation data set. As for the evaluation metric in question, particular attention was paid to measures reflecting the quality of models on multilabel classification problems, such as macro-f1, micro-f1, hamming loss, and per-label f1. Imbalance was inherent to the dataset with only a few threat class examples present.

A toxicity model pre-trained with TensorFlow.js served as the local baseline. The following parameters were used in this experiment: TensorFlow.js version 1.7.4, @tensorflow-models/toxicity package version 1.2.2, CPU execution mode, global threshold of 0.85, and batch size of 16. Model performance was assessed using 15,958 validation comments. The goal was to check if any of the ready-to-use TensorFlow.js models can be used for the local mode. The pretrained TensorFlow.js classifier was used as a deployment-oriented baseline because it represents an existing ready-to-use solution for local browser inference. The experiments were designed to compare practical deployment alternatives rather than to establish state-of-the-art predictive performance against all classical NLP approaches.

The model used locally was chosen to be the last local model for inference. This model was trained using TensorFlow.js v4.22.0 framework, in which the batch size is 64, while the input length is 256 characters.

6 Evaluation Results

The formative evaluation indicated that the basic interaction concept was understandable to the participating users. In the first prototype, participants positively assessed the simplicity of the interface and the popup menu, while difficulties were observed in the settings controls and the preliminary feedback dialogue. These observations informed modifications to the second prototype. Its normalized questionnaire score was 86.4 out of 100, suggesting improved consistency of interface states and better discoverability of controls within this small participant group. However, because only five participants were involved and simulated classification outputs were used, the results should be interpreted only as formative design feedback and not as evidence of general usability or effectiveness during real-world browsing.

The remote model evaluation showed that the second XLM-R version achieved the best overall results. Table 1 summarizes the comparison of the three tested versions.

Table 1. Comparison of XLM-R model versions on the Jigsaw2018 validation set

Model	Macro-F1	Micro-F1	Hamming loss	F1 threat	F1 severe
xlmr-toxic-v1	0.635	0.777	0.0169	0.362	0.470
xlmr-toxic-v2	0.780	0.839	0.0122	0.743	0.628
xlmr-toxic-v2_1	0.647	0.768	0.0176	0.448	0.489

The results show that xlmr-toxic-v2 provided the best compromise between macro-F1, micro-F1, Hamming loss, and minority-class performance. The improvement in the threat category was especially important because this label is rare but practically significant.

The pretrained TensorFlow.js toxicity model achieved a macro-F1 score of 0.279, a micro-F1 score of 0.560, a Hamming loss of 0.0233, and an exact match accuracy of 0.908. The model demonstrated the best results for frequently occurring categories. An F1 score of 0.685 was observed for the toxic label, and an F1 score of 0.603 was observed for the insult label. However, at the selected global threshold, the model did not detect any positive examples for the severe toxic, threat, and identity hate categories. Therefore, F1 scores of 0.000 were obtained for these labels.

Another important limitation was the processing time. The evaluation of 15,958 comments required more than three hours, which corresponds to more than

730 ms per comment. Such a result is not suitable for automatic page analysis in a browser extension.

The optimized local TensorFlow.js GraphModel demonstrated better practical results. The per-label thresholds were selected on the validation set by maximizing the F1 score separately for each toxicity category. Table 2 presents the per-category evaluation of this model. The threat category remained particularly challenging: despite a recall of 0.714, its low precision of 0.189 resulted in an F1 score of 0.299, likely reflecting the very small number of positive examples and the contextual ambiguity of threatening language.

Table 2. Per-label results of the optimized local GraphModel

Label	Positive examples	Threshold	Precision	Recall	F1	Accuracy
toxic	1525	0.90	0.787	0.684	0.732	0.952
severe_toxic	143	0.94	0.286	0.783	0.419	0.981
obscene	826	0.94	0.808	0.794	0.801	0.980
insult	778	0.92	0.664	0.743	0.702	0.969
threat	42	0.94	0.189	0.714	0.299	0.991
identity_hate	132	0.94	0.329	0.636	0.434	0.986

The model performed best on more frequent labels, especially obscene, toxic, and insult. The minority labels remained more difficult. This confirms that class imbalance strongly affects toxicity detection. The model was also evaluated as a simplified binary toxic/non-toxic classifier. In that case, a text was considered toxic if at least one enabled label exceeded its threshold. The binary verdict achieved precision of 0.744, recall of 0.725, F1 of 0.734, and accuracy of 0.947 at the best threshold of 0.86.

The performance comparison between CPU and WASM backends showed a major improvement when using WASM. Table 3 summarizes the result.

Table 3. CPU and WASM performance comparison for local inference

Backend	Rows	Batch size	Prediction time [s]	Average [ms/text]	Throughput [texts/s]	Warm-up [ms]
CPU	15958	64	85.60	5.36	186.42	34.27
WASM	15958	64	6.35	0.40	2514.57	10.66

The WASM backend performed about 13.5 times faster than the CPU backend in making predictions. It is significant to note this for its application in a browser context, where the extension needs to perform multiple classifications on numerous text nodes present on a single webpage.

7 Discussion

Consequently, the proposed architecture is reasonable. The local approach is suitable when privacy is highly important, because processing takes place directly on the user's device. The optimized local GraphModel can also be executed quickly when WASM support is available. However, the remote mode can also play an important role, since larger models can provide better classification results, particularly for minority classes.

The binary F1 score of 0.734 indicates that the optimized local model provides useful performance for privacy-preserving browser deployment, but it should not be interpreted as state-of-the-art performance. Larger transformer-based models generally achieve stronger classification results, particularly for context-dependent and minority categories, but require substantially greater computational resources and are less suitable for direct in-browser inference. The contribution of the local GraphModel therefore lies mainly in the trade-off between predictive performance, privacy, and runtime efficiency rather than in outperforming server-side models.

The results also show that one global threshold is not sufficient for multi-label toxicity detection. The pretrained TensorFlow.js toxicity model performed poorly on rare categories when one global threshold was used. The optimized local model improved this situation by using per-label thresholds, but minority classes such as threat and identity hate remained difficult. This is expected because these labels contain fewer positive examples and are often more context dependent.

Another important result is that high accuracy can be misleading. For rare categories, a model can achieve high accuracy simply by predicting negative most of the time. For this reason, precision, recall, F1 score, and Hamming loss are more useful for understanding the real behaviour of the model (Borkan et al., 2019).

The interface assessment served as a formative evaluation of the interaction concept rather than a full usability study. Its findings helped identify design problems and guide the transition between prototype

iterations, but they cannot be generalized to the broader user population. The evaluation involved only five participants, potentially with relatively similar technical backgrounds, and the second prototype relied on simulated detection outputs. A subsequent study should involve a larger and more diverse participant sample, the complete extension with real model outputs, representative websites, and longer task periods. Such a study could evaluate standardized usability, task success, interaction errors, perceived usefulness, trust in classifications, and the effect of false-positive and false-negative detections during normal browsing. A classical baseline based on TF–IDF features and logistic regression, linear SVM, or Naïve Bayes was not included. Consequently, the results demonstrate differences between the evaluated deployment options but do not establish that the XLM-R or GraphModel approaches outperform simpler classifiers. Such baselines should be included in a broader comparative evaluation.

The current solution also has technical and functional limitations. It focuses mainly on text and does not address image or audio content. Multilingual support is limited, especially in the local mode. The feedback mechanism for incorrect classifications was considered during design, but it was not included in the final implementation. A complete fairness audit was also outside the scope of the implemented system. These limitations are suitable directions for future development.

8 Conclusion

This paper presented Toxic Text Detector, a browser extension for detecting toxic text using NLP models. The extension was implemented as a Chrome Manifest V3 solution with a React-based user interface, a content script, a service worker, persistent settings, and two inference modes. The local mode performs inference directly in the browser, while the remote mode communicates with a backend API.

The evaluation results show that browser-based toxic content detection is feasible. However, the correct choice of technology is important. The pretrained TensorFlow.js toxicity classifier was useful as a baseline, but it was not sufficient for practical use because of weak performance on minority labels and slow inference speed.

The proposed solution represents a balance between privacy, usability, performance, and model accuracy. Local inference supports the idea of privacy by default, whereas remote inference enables the use of stronger models and further experimentation. Future work will focus on evaluation with Slovak, Czech, and Ukrainian text, including mixed-language content, as well as larger-scale usability testing and fairness analysis using subgroup performance, false-positive rates, and identity-related test cases.

Acknowledgments

This research was funded by the Slovak Research and Development Agency under the contract No. APVV-22-0414.

References

BORKAN, D., Dixon, L., Sorensen, J., Thain, N. and Vasserman, L.: “Nuanced metrics for measuring unintended bias with real data for text classification,” in Companion Proceedings of The 2019 World Wide Web Conference, 2019, pp. 491–500.

CONNEAU, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L. and Stoyanov, V.: “Unsupervised cross-lingual representation learning at scale,” in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 8440–8451.

DEEP, S., SINGH, S.; YADAV, S.; SHEDGE, Y. and KHADE, A.: Creating an Integrated Browser Plug-in for Detecting and Blocking Obscene Image/Text Content. In: 2024 2nd International Conference on Networking and Communications (ICNWC). IEEE, 2024, s. 1–6. DOI: 10.1109/ICNWC60771.2024.10537464.

GOOGLE CHROME DEVELOPERS, “Manifest V3 overview,” Chrome Extensions Documentation, 2026. Available at: https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3.

HARTVIGSEN, T., Gabriel, S., Palangi, H., Sap, M., Ray, D. and Kamar, E. “ToxiGen: A Large-Scale Machine-Generated Dataset for Adversarial and Implicit Hate Speech Detection,” in Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, 2022, pp. 3309–3326. DOI: 10.18653/v1/2022.acl-long.234.

JIGSAW, “Jigsaw Toxic Comment Classification Challenge,” Kaggle competition, 2018. Available at: https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge.

JIGSAW. “Jigsaw Unintended Bias in Toxicity Classification,” Kaggle competition, 2019. Available at: https://www.kaggle.com/competitions/jigsaw-unintended-bias-in-toxicity-classification.

JIGSAW. “Perspective API,” online service for toxicity and content moderation. Available at: https://perspectiveapi.com/.

KUMAR, S.; DMELLO, D. V., JANANI, D., Avani, K. D. and Kirthana, P. Kamath: Eliminating Toxicity in Text: An NLP Framework for Clean

Content Extraction. In: 2024 International Conference on IoT Based Control Networks and Intelligent Systems (ICICNIS). IEEE, 2024, s. 146–151. DOI: 10.1109/ICICNIS64247.2024.10823171.

MATHEW, B., Saha, P., Yimam, S. M., Biemann, C., Goyal, P. and Mukherjee, A. "HateXplain: A Benchmark Dataset for Explainable Hate Speech Detection," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 17, 2021, pp. 14867–14875.

Sikiandani, N. M. D., Suarjaya, I. M. A. D. and Putra, Y. P.: "Browser-based detection of harmful content with deep learning model," Journal of Applied Informatics and Computing, vol. 9, no. 4, pp. 1800–1811, 2025.

TENSORFLOW.JS TEAM, "TensorFlow.js models: Toxicity classifier," GitHub repository, 2026. Available: https://github.com/tensorflow/tfjs-models/tree/master/toxicity.

TSOUMAKAS, G. and KATAKIS, I.: "Multi-label classification: An overview," International Journal of Data Warehousing and Mining, vol. 3, no. 3, pp. 1–13, 2007.

Web Content Accessibility Guidelines (WCAG) 2.0. (2008). Retrieved from https://www.w3.org/TR/WCAG20/