# *GreaseVision*: Rewriting the Rules of the Interface

**Siddhartha Datta**
University of Oxford
siddhartha.datta@cs.ox.ac.uk

**Konrad Kollnig**
University of Oxford
konrad.kollnig@cs.ox.ac.uk

**Nigel Shadbolt**
University of Oxford
nigel.shadbolt@cs.ox.ac.uk

## Abstract

Digital harms can manifest across any interface. Key problems in addressing these harms include the high individuality of harms and the fast-changing nature of digital systems. We put forth *GreaseVision*, a collaborative human-in-the-loop learning framework that enables end-users to analyze their screenomes to annotate harms as well as render overlay interventions. We evaluate HITL intervention development with a set of completed tasks in a cognitive walkthrough, and test scalability with one-shot element removal and fine-tuning hate speech classification models. The contribution of the framework and tool allow individual end-users to study their usage history and create personalized interventions. Our contribution also enables researchers to study the distribution of multi-modal harms and interventions at scale.

## 1 Introduction

The design of good user interfaces can be challenging. In a fast changing world, however, with sometimes highly individual needs, traditional one-fits-all software development faces difficulty in keeping up with the pace of change and the breadth of user requirements. At the same time, the digital world is rife with a range of harms, from dark patterns to hate speech to violence. This paper takes a step back to improve the user experience in the digital world. To achieve this, we put forward a new design philosophy for the development of software interfaces that serves its users: *GreaseVision*.

**Contributions:** Our work aims to contribute a novel interface modification framework, which we call *GreaseVision*. At a structural-level, our framework enables end-users to develop personalized interface modifications, either individually or collaboratively. This is supported by the use of screenome visualization, human-in-the-loop learning, and an overlay/hooks-enabled low-code development platform. Within the defined scopes, we enable the aggregation of distributionally-wide end-user digital harms (self-reflection for end-users, or analyzing the harms dataset of text, images and elements for researchers), to further enable the modification of user interfaces across a wide range of software systems, supported by the usage of visual overlays, autonomously developed by users, and enhanced by scalable machine learning techniques. We provide complete and reproducible implementation details to enable researchers to not only study harms and interventions, but other interface modification use cases as well.

**Structure:** We introduce the challenge of end-user interface modification in Sections 1 and 2 to curb digital harms. We share our proposed method – *GreaseVision* – in Section 3. We evaluate our method in Section 4, and share final thoughts and conclusions in Section 5.

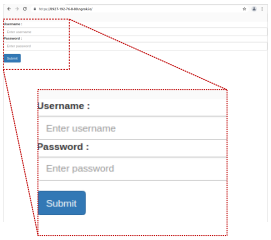
## 2 Related Works & Problem

We summarize key related work here; for detailed related works, we refer the reader to Appendix: Section 6.2. Our motivating problem is the high individuality of digital harms across a distribution of users. The harms landscape is quickly changing with ever-changing digital systems, ranging from heavily-biased content (e.g. disinformation, hate speech), self-harm (e.g. eating disorders, self-cutting, suicide), cyber crime (e.g. cyberbullying, harassment, promotion of and recruitment for extreme causes (e.g. terrorist organizations), to demographic-specific exploitation (e.g. child-inappropriate content, social engineering attacks) (HM, 2019; Pater and Mynatt, 2017; Wang et al., 2017; Honary et al., 2020; Pater et al., 2019). Though interface modification frameworks exist, the distribution of the interface modifications (*interventions*) are constrained to the development efforts of an intervention developer, the availability of interventions are skewed towards desktop browsers (much sparser on mobile), and the efforts

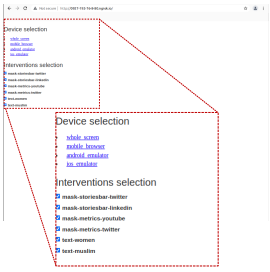


*Proceedings of the First Workshop on Dynamic Adversarial Data Collection*, pages 7 - 22

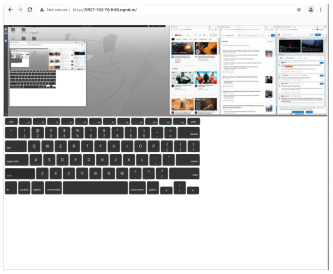
Figure 1: Walkthrough of using *GreaseVision*-modified interfaces.



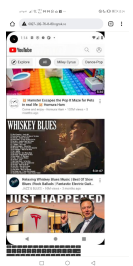

(a) *User authentication*: Secure gateway to the user's screenomes, personal devices, and intervention development suite.




(b) *Interface & interventions selection*: Listings of all registered devices/emulators on server, as well as interventions contributed by the users or community using the tool in Figure 3.



(c) *Interface access*: Accessing a Linux desktop from another (Linux) desktop browser.



(d) *Interface access*: Accessing an Android emulator from another Android host device.

are highly interface-specific (an app version update breaks code-based modification; videos cannot be perturbed in real-time). Moreover, low-code development platforms, that enable end-users to use visual tools to construct programs, are mostly available for software creation, but few options exist for software modification.

Due to the non-uniform distribution of users, the diverging distribution of harms would require a wide distribution of interventions. We hypothesize we can generate this matching distribution of interventions by enabling end-users to render personalized interventions *by themselves* (i.e. removing intervention developers from the ecosystem). To test this hypothesis, we attempt to bind the harms landscape to the interventions landscape by developing a collaborative human-in-the-loop system where end-users can inspect their browsing history and generate corresponding interventions.

We pursue a *visual overlay modifications* approach, extending on the work of ***GreaseTerminator*** (Datta et al., 2021). The framework renders overlay graphics over an underlay screen based on detected GUI elements, images or text (as opposed to implementing program code changes natively), hence changes the interface rather than the functionality of the software. To provide end-users with input for self-reflection (Cho et al., 2021; Lyngs et al., 2020a) and source data for generating interventions, users can be shown their **screenome** (Reeves et al., 2020, 2021), a record of a user's digital experiences represented as a sequence of screen images that they view and interact with over time To connect the input (screenome) and output (intervention), **Human-in-the-Loop (HITL)** learning can be used for users to annotate their screenomes for harmful text, images or GUI elements, and these annotations can be used to develop interventions. Wu et al. (2021) offers a detailed review of HITL. Specifically, the procedure to generate interventions using visual overlays will require the one-shot detection of masks (e.g. GUI elements) and few-shot learning and/or model fine-tuning of image/text classification models.

**System requirements:** Based on the problem and our collaborative HITL interface modification approach, we establish the following technical requirement (Requirement 1) and systemic requirement (Requirement 2) for our framework:

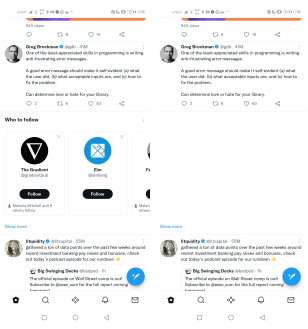
1. **(Req 1)** *A complete feedback loop between user input (train-time) and interface re-render (test-time).*
2. **(Req 2)** *Prospects for scalability across the distribution of interface modifications (with respect to both harms landscape and rendering landscape).*

## 3 *GreaseVision*

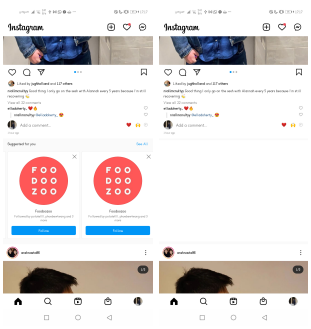
### 3.1 System Architecture: Binding the Harms Ecosystem to the Interventions Ecosystem

We define the *GreaseVision* architecture, with which end-users (system administrators) interact with, as follows (Figure 6(b)): (i) the user logs into the *GreaseVision* system to access amongst a set of personal emulators and interventions (the system admin has provisioned a set of emulated devices, hosted on a server through a set of virtual machines or docker containers for each emulator/interface,

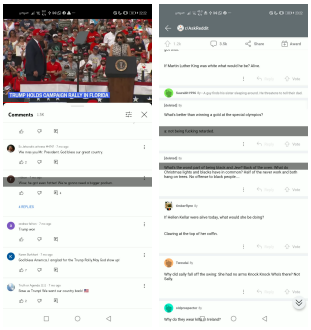
Figure 2: Hooks adapted in *GreaseVision* to occlude distracting elements, censor hate speech, and obscure child-inappropriate content.



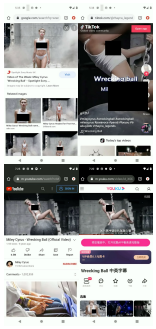
(a) Occlusion of recommended items on Twitter (before *left*, after *right*)



(b) Occlusion of recommended items on Instagram (before *left*, after *right*)



(c) Text censoring (YouTube *left*, Reddit *right*)



(d) Content moderation (Google Images, TikTok, YouTube, YouKu)

and handling streaming of the emulators, handling pre-requisites for the emulators, handling data migrations, etc); (ii) the user selects their desired interventions and continues browsing on their interfaces; (iii) after a time period, the user accesses their screenome and annotates interface elements, graphics, or text that they would like to generate interventions off of, which then re-populate the list of interventions available to members in a network.

In addition to the contributions stated in Section 1, *GreaseVision* is an improved visual overlay modification approach with respect to interface-agnosticity and ease of use. We discuss the specific aspects of *GreaseTerminator* we adopt in *GreaseVision* (hooks and overlays), and the technical improvements upon *GreaseTerminator* in Appendix: Section 6.1, specifically latency, device support, and interface-agnosticity.

In our current implementation, the user accesses a web application (compatible with both desktop and mobile browsers). With their login credentials, the database loads the corresponding mapping of the user's virtual machines/containers that are shown in the interface selection page. The central server carries information on accessing a set of emulated devices (devices loaded locally on the central server in our setup). Each emulator is rendered in docker containers or virtual machines where input commands can be redirected. The database also loads the corresponding mapping of available interventions (generated by the user, or by the network of users) in the interventions selection page. The database also loads the screenomes (images of all timestamped, browsed interfaces) in the screenome visualization page. Primary input commands for both desktop and mobile are encoded, including keystroke entry (hardware keyboard, on-screen keyboard), mouse/touch input (scrolling, swiping, pinching, etc); input is locked to the coordinates of the displayed screen image on the web app (to avoid stray/accidental input commands), and the coordinates correspond to each virtual machine/container's display coordinates. Screen images are captured at a configurable framerate (we set it to 60FPS), and the images are stored under a directory mapped to the user. Generated masks and fine-tuned models are stored under an interventions directory and their intervention/file access is also locked by mapped users. Interventions are applied sequentially upon a screen image to return a perturbed/new image, which then updates the screen image shown on the client web app.

### 3.2 Low-code Development: Binding Screenomes to Interface Modifications

We make use of the three hooks from *GreaseTerminator* (*text*, *mask*, and *model* hooks), and link it with the screenome visualization tool. While in *GreaseTerminator* the hooks ease the intervention development process for intervention developers with previous programming knowledge, we further generalize the intervention development process for intervention developers to the extent that even an end-user can craft their own interventions without developer support nor expert knowledge. *GreaseTerminator* enables intervention generation (via hooks) and interface re-rendering (via overlays). The added *GreaseVision* contribution of con-

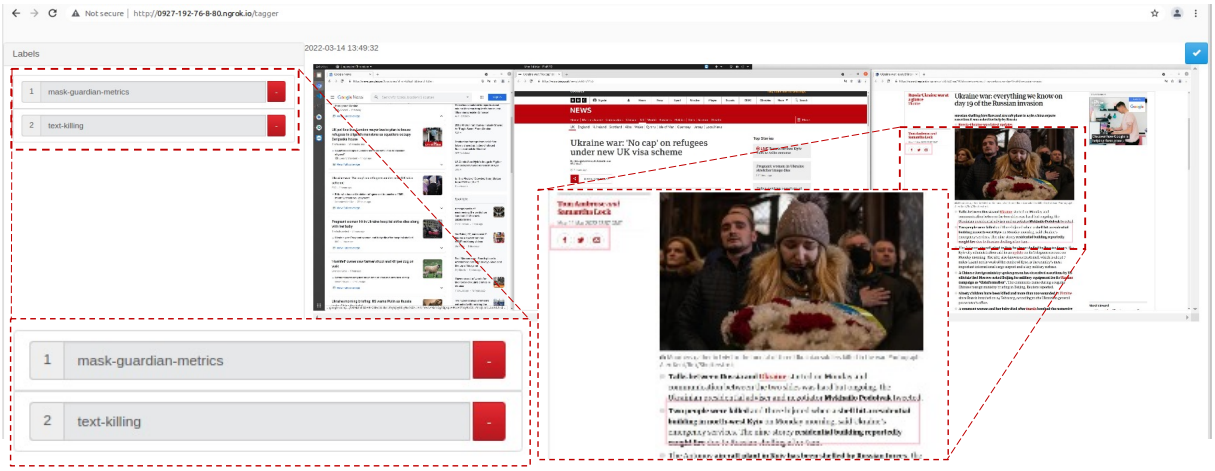



Figure 3: *Screenome visualization page*: The page offers the end-user the ability to traverse through the sequence of timestamped screen images which compose their screenome. They can use bounding boxes to highlight GUI elements, images or text. They can label these elements with specific encodings, such as `mask-` or `text-`.

necting these components with HITL learning and screenome visualization to replace developers is what exemplifies end-user autonomy and scalability in personalized interventions.

An intersecting data source that enables both end-user self-reflection (Cho et al., 2021; Lyngs et al., 2020a) and interface re-rendering via overlay (Datta et al., 2021) is the screenome. Specifically, we can orchestrate a loop that receives input from users and generates outputs for users. Through *GreaseVision*, end-users can browse through their own screen history, and beyond self-analysis, they can constructively build interface modifications to tackle specific needs. Extending on the interface rendering approach of overlays and hook-based intervention development, a generalizable design pattern for *GreaseTerminator*-based interventions is observed, where current few-shot/fine-tuning techniques can reasonably approach many digital harms, given appropriate extensions to the end-user development suite. In the current development suite (Figure 3), an end-user can inspect their screenomes across all *GreaseVision*-enabled interfaces (ranging from iOS, Android to desktops), and make use of image segment highlighting techniques to annotate interface patterns to detect (typically UI elements or image/text) and subsequently intervene against these interface patterns. Specifically, the interface images being stored and mapped to a user is shown in time-series sequence to the user. The user can go through the sequence of images to reflect on their browsing behavior. The current implementation focuses on one-shot detection of masks and fine-tuning of image and text classification models. When the user identifies a GUI element they do not wish to see across interfaces and apps, they highlight the region of the image, and annotate it as `mask-<name-of-intervention>`, and the mask hook will store a mask of intervention `<name-of-intervention>`, which will then populate a list of available interventions with this option, and the user can choose to activate it during a browsing session. When a user identifies text (images) that they do not wish to see of similar variations, they can highlight the text (image) region, and annotate it as `text-<name-of-intervention>` (`image-<name-of-intervention>`). The text hook will extract the text via OCR, and fine-tune a pretrained text classification model specifically for this type of text `<name-of-intervention>`. For images, the highlighted region will be cropped as input to fine-tune a pretrained image classification model. The corresponding text (image) occlusion intervention will censor similar text (images) during the user's browsing sessions if activated.

Extending on model few-shot training and fine-tuning, we can scale the accuracy of the models, not just through improvements to these training methods, but also by improving the data collection dynamics. More specifically, based on the spectrum of personalized and overlapping intervention needs for a distribution of users, we can leverage model-human and human-human collaboration to scale the generation of mask and model interventions. In the case of mask hooks, end-users who encounter certain harmful GUI elements (perhaps due to exposure to specific apps or features prior to other users) can tag and share the mask intervention

with other users collaboratively.

To collaboratively fine-tune models, users tag text based on a general intervention/category label, that is used to group text together to form a mini-dataset to fine-tune the model. An example of this would be a network of users highlighting racist text they come across in their screenomes that made them uncomfortable during their browsing sessions, and tagging them as `text-racist`, which aggregates more sentences to fine-tune a text classification model responsible for detecting/classifying text as racist or not, and subsequently occluding the text for the network of users during their live browsing sessions. The current premise is that users in a network know a ground-truth label of the category of the specific text they wish to detect and occlude, and the crowd-sourced text of each of $N$ categories will yield corresponding $N$ fine-tuned models. Collaborative labelling scales the rate in which text of a specific category can be acquired, reducing the burden on a single user while also diversifying the fine-tune training set, while also proliferating the fine-tuned models across a network of users and not wasting effort re-training already fine-tuned models of other users (i.e. increasing scalability of crafting and usage of interventions).

## 4 Evaluation

We evaluate the usability of (*Req 1*) the HITL component (usability for a *single* user with respect to inputs/outputs; or "does our system help generate interventions?"), and (*Req 2*) the collaborative component (improvement to usability for a *single* user when *multiple* users are involved; or "does our system scale with user numbers?") with **cognitive walkthroughs** and **scalability tests** respectively.

### 4.1 Cognitive Walkthrough

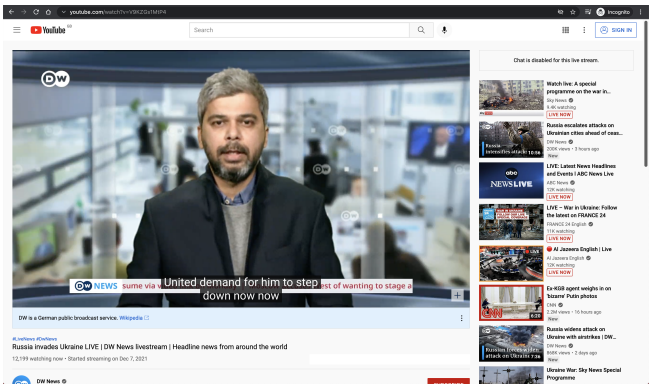
Qualitatively, we perform a cognitive walkthrough (John and Packer, 1995; Rieman et al., 1995) of the user experience to simulate the cognitive process and explicit actions taken by an end-user during usage of *GreaseVision* to access interfaces and craft interventions. In our walkthrough, we as researchers presume the role of an end-user. We state the walkthrough **step** in **bold**, *data pertaining to the task* in *italics*, and descriptive evaluation in normal font. To evaluate the process of constructing an intervention using our proposed HITL system, we inspect *the completion of a set of required tasks* based on criteria from Parasuraman et al.'s (Parasuraman et al., 2000) 4 types of automation applications, which aim to measure the role of automation in the harms self-reflection and intervention self-development process. The four required tasks to be completed are:

1. *Information Acquisition*: Could a user collect new data points to be used in intervention crafting?
2. *Information Analysis*: Could a user analyze interface data to inform them of potential harms and interventions?
3. *Decision & Action Selection*: Could a user act upon the analyzed information about the harms they are exposed to, and develop interventions?
4. *Action Implementation*: Could a user deploy the intervention in future browsing sessions?

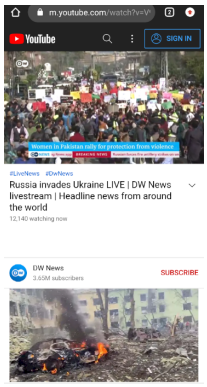
**User logs in (Figure 1a):** *The user enters their username and password. These credentials are stored in a database mapped to a specific (set of) virtual machine(s) that contain the interfaces the user registered for access.* This is a standard step for any secured or personalized system, where a user is informed they are accessing data and information that is tailored for their own usage.

**User selects active interface and interventions (Figure 1b):** *The user is shown a set of available interventions, be it contributed by themselves or other users in a network. They select their target interventions, and select an interface to access during this session.* Based on their own configurations (e.g. *GreaseVision* set up locally on their own computer, or specific virtual machines set up for the required interfaces), users can view the set of interfaces that they can access and use to facilitate their digital experiences. The interface is available 24/7, retains all their personal data and storage, is recording their screenome data for review, and accessible via a web browser from any other device/platform. They are less constrained by the hardware limitations of their personal device, and just need to ensure the server of the interfaces has sufficient compute resources to host the interface and run the interventions. The populated interventions are also important to the user, as it is a marketplace and ecosystem of personalized and shareable interventions. Users can populate interventions that they themselves can generate through the screenome visualization tool, or access interventions collaboratively trained and contributed by multiple members in their network. The interventions are also modu-

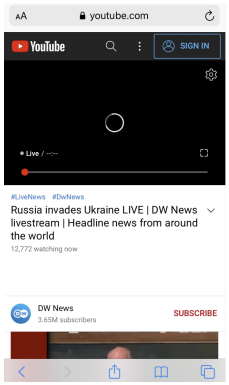
Figure 4: Removal of GUI elements (YouTube sharing metrics/buttons) across multiple target interfaces and operating systems.



(a) Element removal on emulated desktop (MacOS)



(b) Element removal on emulated Android



(c) Element removal on emulated iOS

| Mask | Min. masks | Android app | iOS app | Mobile browser | Desktop browser |
|---|---|---|---|---|---|
| *Stories bar* | | | | | |
| - Twitter | 1 | ✓ | ✓ | - | - |
| - Linkedin | 1 | ✓ | ✓ | - | - |
| - Instagram | 1 | ✓ | ✓ | - | - |
| *Metrics/Sharing bar* | | | | | |
| - Facebook | 2 | ✓ | ✓ | ✓ | ✓ |
| - Instagram | 2 | ✓ | ✓ | ✓ | ✓ |
| - Twitter | 2 | ✓ | ✓ | ✓ | ✓ |
| - YouTube | 2 | ✓ | ✓ | ✓ | ✓ |
| - TikTok | 2 | ✓ | ✓ | ✓ | ✓ |
| *Recommended items* | | | | | |
| - Twitter | 2 | ✓ | ✓ | ✓ | ✓ |
| - Facebook | 2 | ✓ | ✓ | ✓ | ✓ |

Table 1: ✓ if element removal is successful, ✗ if element removal is unsuccessful, — if the element not available on an interface.

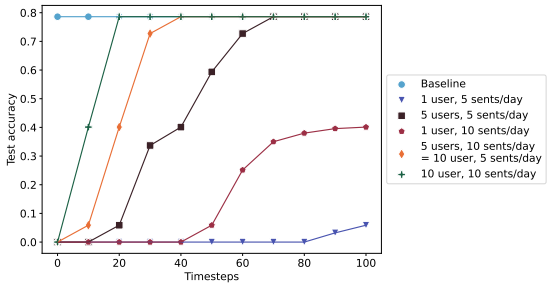



Figure 5: Convergence of few-shot/fine-tuned models on sub-groups of hate speech

lar enough that users are not restricted to a specific combination of interventions, and are applied sequentially onto the interface without mismatch in latency between the overlay and underlying interface. As the capabilities of generating interventions (e.g. more hooks) and rendering interfaces (e.g. interface augmentation) become extended, so do their ability to personalize their digital experience, and generate a distribution of digital experiences to match a similarly wide distribution of users. The autonomy to deploy interventions, with enhanced optionality through community-contributed interventions, before usage of an interface satisfies Task 4.

**The user accesses the interface and browses (Figure 1c):** *The user begins usage of the interface through the browser from their desired host device, be it mobile or desktop. They enter input to the system, which is streamed to the virtual machine(s), and interventions render overlay graphics to make any required interface modifications.* After the user has chosen their desired interventions, the user will enjoy an improved digital experience through the lack of exposure to certain digital elements, such as undesired text or GUI elements. The altered viewing experience satisfies both Task 1 and 4; not only is raw screen data being collected, but the screen is being altered by deployed interventions in the wild. The user cannot be harmed by what they previously chose not to see, and what they do see but no longer wish to see in the future, they can annotate to remove in future viewings in the screenome visualization tool. It is a cyclical loop where users can redesign and self-improve their browsing experiences through the use of unilateral or user-driven tools.

**The user browses their screenome to generate interventions (Figure 3):** *After a browsing period, the user may opt to browse and view their personal screenome. They enter the screenome visualization page to view recorded intervals of their browsing activity across all interfaces, and they can choose to annotate certain regions (image or text) to generate interventions to re-populate the interventions available.* The user is given autonomy in selecting and determining what aspects of the

interface, be it the static app interface of dynamic content provisioned, that they no longer wish to see in the future. Enabling the user to view their screenome across all used digital interfaces (extending to mobile and desktop) to self-reflect and analyze browsing or content patterns satisfies Task 2. Though the screenome provides the user raw historical data, it may require additional processing (e.g. automated analysis, charts) to avoid information overload. Rather than waiting for a feedback loop for the app/platform developers or altruistic intervention developers to craft broad-spectrum interventions that may or may not fit their personal needs, the end-user can enjoy a personalized loop of crafting and deploying interventions, almost instantly for certain interventions such as element masks. The user can enter metadata pertaining to each highlighted harm, and not only contribute to their own experience improvement, but also contribute to the improvement of others who may not have encountered or annotated the harm yet. By developing interventions based on their analysis, not only for themselves but potentially for other users, they could successfully achieve Task 3. Though previously-stated as out of scope, to further support Task 3, other potential intervention modalities such as augmentation could also be contributed by a community of professional intervention developers/researchers (who redirect efforts from individual interventions towards enhancing low-code development tools).

The four tasks, used to determine whether a complete feedback loop between input collection/processing and interface rendering through HITL by a single user, could all be successfully completed, thus *GreaseVision* satisfies Requirement 1.

### 4.2 Scalability Testing

To evaluate the collaborative component, we measure the improvement to the user experience of a *single* user through the efforts of multiple users. We evaluate through scalability testing (Meerts and Graham, 2010), a type of load testing that measures a system's ability to scale with respect to the number of users. We simulate the usage of the system to evaluate the scalable generation of one-shot graphics (mask) detection, and scalable fine-tuning/few-shot training of (text) models. We do not replicate the scalability analysis on real users: the fine-tuning mechanism is still the same, and the main variable (in common) is the sentences highlighted (and their assigned labels and metadata, as well as the quality of the annotations), though error is expectedly higher in the real-world as the data may be sampled differently and of lower annotation quality. The primary utility of collaboration to an individual end-user is the scaled reduction of effort in intervention development. We evaluate this in terms of variety of individualized interventions (variations of masks), and the time saved in constructing a single robust intervention (time needed to construct an accurate model intervention).

**Breadth of interface-agnostic masks (Table 1):** We investigate the ease to annotate graphically-consistent GUI elements for few-shot detection. We sample elements to occlude that can exist across a variety of interfaces. We evaluate the occlusion of the *stories bar* (pre-dominantly only found on mobile devices, not desktop/browsers); some intervention tools exist on Android (Happening, 2021; MaaarZ, 2019; Kollnig et al., 2021; Datta et al., 2021) and iOS (Friendly, 2022), though the tools are app- (and version-) specific. We evaluate the occlusion of *like/share metrics*; there are mainly desktop browser intervention tools (Grosser, 2012, 2018, 2019; hidelikes, 2022), and one Android intervention tool (Datta et al., 2021). We evaluate the occlusion of *recommendations*; there are intervention tools that remove varying extents of the interface on browsers (such as the entire newsfeed) (West, 2012; Unhook, 2022). Existing implementations and interest in such interventions indicate some users have overlapping interests in tackling the removal or occlusion of such GUI elements, though the implementations may not exist across all interface platforms, and may not be robust to version changes. For each intervention, we evaluate on a range of target (emulated) interfaces. We aim for the real-time occlusion of the specific GUI element, and evaluate on native apps (for Android and iOS) and browsers (Android mobile browser, and Linux desktop browser).

For each of the GUI element cases, we make use of the screenome visualization tool to annotate and tag the minimum number of masks of the specific elements we wish to block across a set of apps. There tend to be small variations in the design of the element between browsers and mobile, hence we tend to require at least 1 mask from each device type; Android and iOS apps tend to have similar enough GUI elements that a single mask can be

reused between them. We tabulate in Table 1 the successful generation and real-time occlusion of all evaluated and applicable GUI elements. We append screenshots of the removal of recommended items from the Twitter and Instagram apps on Android (Figure 2(a,b)). We append screenshots of the demetrification (occlusion of like/share buttons and metrics) of YouTube across desktop browsers (MacOS) and mobile browsers (Android, iOS) (Figure 4).

**Convergence of few-shot/fine-tune trained text models (Figure 5):** We investigate the accuracy gains from fine-tuning pretrained text models as a function of user numbers and annotated sentence contributions. Specifically, we evaluate the text censoring of hate speech, where the primary form of mitigation is still community standard guidelines and platform moderation, with a few end-user tooling available (Bodyguard, 2019; Datta et al., 2021). The premise of this empirical evaluation is that we have a group of simualated users $N$ who each contribute $N$ inputs (sentences) of a specific target class (hate speech, specifically against women) per timestep. With respect to a baseline, which is a pretrained model fine-tuned with all available sentences against women from a hate speech dataset, we wish to observe how the test accuracy of a model fine-tuned with $M \times N$ sentences varies over time. Our source of hate speech for evaluation is the *Dynamically Generated Hate Speech Dataset* (Vidgen et al., 2021), which contains sentences of `non-hate` and `hate` labels, and also classifies hate-labelled data by the target victim of the text (e.g. `women`, `muslim`, `jewish`, `black`, `disabled`). As we expect the $M$ users to be labelling a specific niche of hate speech to censor, we specify the subset of hate speech of `women` (train set count: 1,652; test set count: 187). We fine-tune a publicly-available, pre-trained *RoBERTa* model (HuggingFace, 2022; Liu et al., 2019), which was trained on a large corpus of English data (*Wikipedia* (Wikimedia), *BookCorpus* (Zhu et al., 2015)). For each constant number of users $M$ and constant sentence sampling rate $N$, at each timestep $t$, $M \times N \times t$ sentences are acquired of class `hate` against target `women`; there are a total of 1,652 train set sentences under these constraints (i.e. the max number of sentences that can be acquired before it hits the baseline accuracy), and to balance the class distribution, we retain all 15,184 train set `non-hate` sentences. We evaluate the test accuracy of the fine-tuned model on all 187 test set women-targeted hate speech. We also vary $M$ and $N$ to observe sensitivity of these parameters to the convergence towards baseline test accuracy.

The rate of convergence of a finetuned model is quicker when the number of users and contributed sentences per timestep both increase, approximately when we reach at least 1,000 sentences for the `women` hate speech category. The difference in convergence rates indicate that a collaborative approach to training can scale interventions development, as opposed to training text classification models from scratch and each user annotating text alone.

The empirical results for this section are stated in Table 1 and Figure 5. The data and evaluations from the scalability tests indicate that the ease of mask generation and model fine-tuning, further catalyzed by performance improvements from more users, enable the scalable generation of interventions and their associated harms, thus *GreaseVision* satisfies Requirement 2.

## 5 Conclusion

To enable end-user autonomy over interface design, and the generation and proliferation of a distribution of harms and interventions to analyze and reflect upon, we contribute the novel interface modification framework *GreaseVision*. End-users can reflect and annotate with their digital browsing experiences, and collaboratively craft interface interventions with our HITL and visual overlay mechanisms. With respect to Requirements 1 and 2, we find that our *GreaseVision* framework allows for scalable yet personalized end-user development of interventions against element, image and text-based digital harms. We hope *GreaseVision* will enable researchers and end-users to study harms and interventions, and other interface modification use cases.

## References


Samira Abnar, Mostafa Dehghani, Behnam Neyshabur, and Hanie Sedghi. 2022. Exploring the limits of large scale pre-training. In *International Conference on Learning Representations*.

Yuvraj Agarwal and Malcolm Hall. 2013. ProtectMyPrivacy: Detecting and mitigating privacy leaks on iOS devices using crowdsourcing. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '13*, page 97, Taipei, Taiwan. ACM Press.

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics.

Ionut Andone, Konrad Błaszkiewicz, Mark Eibes, Boris Trendafilov, Christian Montag, and Alexander Markowetz. 2016. Menthal: A framework for mobile data collection and analysis. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, UbiComp '16, page 624–629, New York, NY, USA. Association for Computing Machinery.

Michael Backes, Sebastian Gerling, Christian Hammer, Matteo Maffei, and Philipp von Styp-Rekowsky. 2014. AppGuard – Fine-Grained Policy Enforcement for Untrusted Android Applications. In Joaquin Garcia-Alfaro, Georgios Lioudakis, Nora Cuppens-Boulahia, Simon Foley, and William M. Fitzgerald, editors, *Data Privacy Management and Autonomous Spontaneous Security*, volume 8247 of *Lecture Notes in Computer Science*, pages 213–231. Springer Berlin Heidelberg, Berlin, Heidelberg.

Inc Bodyguard. 2019. Bodyguard.

Hyunsung Cho, DaEun Choi, Donghwi Kim, Wan Ju Kang, Eun Kyoung Choe, and Sung-Ju Lee. 2021. Reflect, not regret: Understanding regretful smartphone use with app feature-level analysis. *Proc. ACM Hum.-Comput. Interact.*, 5(CSCW2).

Siddhartha Datta. 2021. Learn2weight: Weights transfer defense against similar-domain adversarial attacks.

Siddhartha Datta, Konrad Kollnig, and Nigel Shadbolt. 2021. Mind-proofing your phone: Navigating the digital minefield with greaseterminator. *CoRR*, abs/2112.10699.

Siddhartha Datta and Nigel Shadbolt. 2022. Low-loss subspace compression for clean gains against multi-agent backdoor attacks. *arXiv preprint arXiv:2203.03692*.

Benjamin Davis and Hao Chen. 2013. RetroSkeleton: Retrofitting android apps. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '13*, page 181, Taipei, Taiwan. ACM Press.

Benjamin Davis, Ben S, Armen Khodaverdian, and Hao Chen. 2012. I-arm-droid: A rewriting framework for in-app reference monitors for android applications. In *In Proceedings of the Mobile Security Technologies 2012, MOST '12.*, pages 1–9, New York, NY, United States. IEEE.

William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 393–407, Berkeley, CA, United States. USENIX Association.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks.

flxapps. 2021. Detoxdroid.

Jay Freeman. 2020. Cydia substrate.

App Studio Friendly. 2022. Friendly social browser.

Tomer Galanti, András György, and Marcus Hutter. 2022. On the role of neural collapse in transfer learning. In *International Conference on Learning Representations*.

Kovacs Geza. 2019. HabitLab: In-The-Wild Behavior Change Experiments at Scale. *Stanford Department of Computer Science*.

Vegard IT GmbH. 2021. Gray-switch.

Google. 2007. Tesseract.

Google. 2010a. Chrome web store.

Google. 2010b. recaptcha faq.

Google. 2021. Android accessibility suite.

Colin M. Gray, Yubo Kou, Bryan Battles, Joseph Hoggatt, and Austin L. Toombs. 2018. The dark (patterns) side of ux design. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–14, New York, NY, USA. Association for Computing Machinery.

Benjamin Grosser. 2012. Facebook demetricator.

Benjamin Grosser. 2018. Twitter demetricator.

Benjamin Grosser. 2019. Instagram demetricator.

Studios Happening. 2021. Swipe for facebook.

hidelikes. 2022. Hide likes.

Niklas Higi. 2020. apk-mitm.

Alexis Hiniker, Sungsoo (Ray) Hong, Tadayoshi Kohno, and Julie A. Kientz. 2016. Mytime: Designing and evaluating an intervention for smartphone non-use. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 4746–4757, New York, NY, USA. Association for Computing Machinery.

Government HM. 2019. Online Harms White Paper. *Government Report on Transparency Reporting*.

Mahsa Honary, Beth Bell, Sarah Clinch, Julio Vega, Leo Kroll, Aaron Sefi, and Roisin McNaney. 2020. Shaping the design of smartphone-based interventions for self-harm. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–14, New York, NY, USA. Association for Computing Machinery.

HuggingFace. 2022. roberta-base.

Andrey Ignatov. 2021. Ai-benchmark.

Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. 2012. Dr. Android and Mr. Hide: Fine-grained permissions in android applications. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices - SPSM '12*, page 3, Raleigh, North Carolina, USA. ACM Press.

Bonnie E. John and Hilary Packer. 1995. Learning and using the cognitive walkthrough method: A case study approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, page 429–436, USA. ACM Press/Addison-Wesley Publishing Co.

Minsam Ko, Subin Yang, Joonwon Lee, Christian Heizmann, Jinyoung Jeong, Uichin Lee, Daehee Shin, Koji Yatani, Junehwa Song, and Kyong-Mee Chung. 2015. Nugu: A group-based intervention app for improving self-regulation of limiting smartphone use. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, page 1235–1245, New York, NY, USA. Association for Computing Machinery.

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition.

Konrad Kollnig, Siddhartha Datta, and Max Van Kleek. 2021. I want my app that way: Reclaiming sovereignty over personal devices. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems Late-Breaking Works*, Yokohama, Japan. ACM Press.

AV Tech Labs. 2019. Auto logout.

Heyoung Lee, Heejune Ahn, Samwook Choi, and Wanbok Choi. 2014. The sams: Smartphone addiction management system and verification. *J. Med. Syst.*, 38(1):1–10.

Lawence Lessig. *Code 2.0*, 1 edition. Basic Books.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Markus Löchtefeld, Matthias Böhmer, and Lyubomir Ganev. 2013. Appdetox: Helping users with mobile app addiction. In *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia*, MUM '13, New York, NY, USA. Association for Computing Machinery.

LuckyPatcher. 2020. Lucky patcher.

Yajing Luo, Peng Liang, Chong Wang, Mojtaba Shahin, and Jing Zhan. 2021. Characteristics and challenges of low-code development: The practitioners' perspective. *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*.

Ulrik Lyngs, Kai Lukoff, Petr Slovak, William Seymour, Helena Webb, Marina Jirotka, Jun Zhao, Max Van Kleek, and Nigel Shadbolt. 2020a. *'I Just Want to Hack Myself to Not Get Distracted': Evaluating Design Interventions for Self-Control on Facebook*, page 1–15. Association for Computing Machinery, New York, NY, USA.

Ulrik Lyngs, Kai Lukoff, Petr Slovak, William Seymour, Helena Webb, Marina Jirotka, Jun Zhao, Max Van Kleek, and Nigel Shadbolt. 2020b. 'I Just Want to Hack Myself to Not Get Distracted': Evaluating Design Interventions for Self-Control on Facebook. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–15, Honolulu HI USA. ACM.

MaaarZ. 2019. Instaprefs.

Joris Meerts and Dorothy Graham. 2010. The history of software testing.

Meta. 2022. Content restrictions based on local law.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. 2020. What is being transferred in transfer learning? In *Advances in Neural Information Processing Systems*, volume 33, pages 512–523. Curran Associates, Inc.

Fabian Okeke, Michael Sobolev, Nicola Dell, and Deborah Estrin. 2018. Good vibrations: Can a digital nudge reduce digital overload? In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '18, New York, NY, USA. Association for Computing Machinery.

R. Parasuraman, T.B. Sheridan, and C.D. Wickens. 2000. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 30(3):286–297.

Jessica Pater and Elizabeth Mynatt. 2017. Defining digital self-harm. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, page 1501–1513, New York, NY, USA. Association for Computing Machinery.

Jessica A. Pater, Brooke Farrington, Alycia Brown, Lauren E. Reining, Tammy Toscos, and Elizabeth D. Mynatt. 2019. Exploring indicators of digital self-harm with eating disorder patients: A case study. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW).

Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. 2020. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations*.

Siegfried Rasthofer, Steven Arzt, Enrico Lovat, and Eric Bodden. 2014. DroidForce: Enforcing Complex, Data-centric, System-wide Policies in Android. In *2014 Ninth International Conference on Availability, Reliability and Security*, pages 40–49, Fribourg, Switzerland. IEEE.

Byron Reeves, Nilam Ram, Thomas N. Robinson, James J. Cummings, C. Lee Giles, Jennifer Pan, Agnese Chiatti, Mj Cho, Katie Roehrick, Xiao Yang, Anupriya Gagneja, Miriam Brinberg, Daniel Muise, Yingdan Lu, Mufan Luo, Andrew Fitzgerald, and Leo Yeykelis. 2021. Screenomics: A framework to capture and analyze personal life experiences and the ways that technology shapes them. *Human–Computer Interaction*, 36(2):150–201. PMID: 33867652.

Byron Reeves, Thomas Robinson, and Nilam Ram. 2020. Time for the human screenome project. *Nature*, 577(7790):314–317. Funding Information: The US National Institutes of Health (NIH) is Publisher Copyright: © 2020, Nature.

John Rieman, Marita Franzke, and David Redmiles. 1995. Usability evaluation with the cognitive walkthrough. In *Conference Companion on Human Factors in Computing Systems*, CHI '95, page 387–388, New York, NY, USA. Association for Computing Machinery.

rovo89. 2020. Xposed framework.

Christian Simon, Piotr Koniusz, Richard Nock, and Mehrtash Harandi. 2020. Adaptive subspaces for few-shot learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4135–4144.

Unhook. 2022. Unhook - remove youtube recommended videos.

Bertie Vidgen, Tristan Thrush, Zeerak Waseem, and Douwe Kiela. 2021. Learning from the worst: Dynamically generated datasets to improve online hate detection. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1667–1682, Online. Association for Computational Linguistics.

VrtualApp. 2016. Virtual xposed.

Eric Wallace, Pedro Rodriguez, Shi Feng, Ikuya Yamada, and Jordan Boyd-Graber. 2019. Trick me if you can: Human-in-the-loop generation of adversarial examples for question answering. *Transactions of the Association for Computational Linguistics*, 7:387–401.

Yilin Wang, Jiliang Tang, Jundong Li, Baoxin Li, Yali Wan, Clayton Mellina, Neil O'Hare, and Yi Chang. 2017. Understanding and discovering deliberate self-harm content in social media. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 93–102, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

Jordan West. 2012. News feed eradicator for facebook.

Foundation Wikimedia. Wikimedia downloads.

Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. 2021. A survey of human-in-the-loop for machine learning.

Rubin Xu, Hassen Saïdi, and Ross Anderson. 2012. Aurasium: Practical policy enforcement for android applications. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 539–552, Bellevue, WA. USENIX Association.

Shanshan Zhang, Lihong He, Eduard Dragut, and Slobodan Vucetic. 2019. How to invest my time: Lessons from human-in-the-loop entity extraction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2305–2313, New York, NY, USA. Association for Computing Machinery.

Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. 2017. East: An efficient and accurate scene text detector.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*.

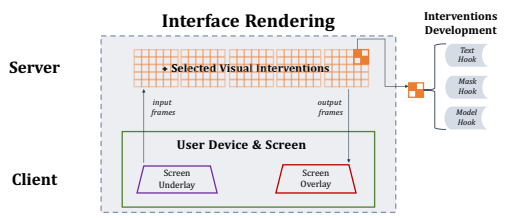
# 6 Appendix

## 6.1 GreaseTerminator

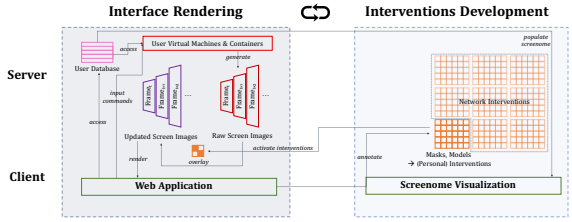
In response to the continued widespread presence of interface-based harms in digital systems, Datta et al. (Datta et al., 2021) developed *GreaseTerminator*, a visual overlay modification method. This approach enables researchers to develop, deploy and study interventions against interface-based harms in apps. This is based on the observation that it used to be difficult in the past for researchers to study the efficacy of different intervention designs against harms within mobile apps (most previous approaches focused on desktop browsers). *GreaseTerminator* provides a set of 'hooks' that serve as templates for researchers to develop interventions, which are then deployed and tested with study participants. *GreaseTerminator* interventions usually come in the form of machine learning models that build on the provided hooks, automatically detect harms within the smartphone user interface at run-time, and choose appropriate interventions (e.g. a visual overlay to hide harmful content, or content warnings). The *GreaseTerminator* architecture is shown in Figure 6(a) in contrast to the *GreaseVision* architecture.

**Technical improvements w.r.t. *GreaseTerminator*** The improvements of *GreaseVision* with respect to *GreaseTerminator* are two-fold: (i) improvements to the framework enabling end-user development and harms mitigation (discussed in detail in Sections 4.2, 4.3, 5 and 6), and (ii) improvements to the technical architecture (which we discuss in this section). Our distinctive and non-trivial technical improvements to the *GreaseTerminator* architecture fall under namely latency, device support, and interface-agnosticity. *GreaseTerminator* requires the end-user device to be the host device, and overlays graphics on top. A downside of this is the non-uniformity of network latency between users (e.g. depending on the internet speed in their location) resulting in a potential mismatch in rendered overlays and underlying interface. With *GreaseVision*, we send a post-processed/re-rendered image once to the end-user device's browser (stream buffering) and do not need to send any screen image from the host user device to a server, thus there is no risk of overlay-underlay mismatch and we even reduce network latency by half. Images are relayed through an HTTPS connection, with a download/upload speed $\sim$ 250Mbps, and each image sent by the server amounting to $\sim$ 1Mb). The theoretical latency per one-way transmission should be $\frac{1\times1024\times8\text{bits}}{250\times10^6\text{bits/s}}$ = 0.033ms. With each user at most requiring server usage of one NVIDIA GeForce RTX 2080, with reference to existing online benchmarks (Ignatov, 2021) the latency for 1 image (CNN) and text (LSTM) model would be 5.1ms and 4.8ms respectively. While the total theoretical latency for *GreaseTerminator* is $(2 \times 0.033 + 5)$, that of *GreaseVision* is $(0.033 + 5)$ = 5.03ms. Another downside of *GreaseTerminator* is that it requires client-side software for each target platform. There would be pre-requisite OS requirements for the end-user device, where only versions of *GreaseTerminator* developed for each OS can be offered support (currently only for Android). *GreaseVision* streams screen images directly to a login-verified browser, allowing users to access desktop/mobile on any browser-supported device. Despite variations in the streaming architecture between *GreaseVision* and *GreaseTerminator*, the interface modification framework (hooks and overlays) are retained, hence interventions (even those developed by end-users) from *GreaseVision* are compatible in *GreaseTerminator*. In addition to improvements to the streaming architecture to fulfil interface-agnosticity, adapting the visual overlay modification framework into a collaborative HITL implementation further improves the ease-of-use for all stakeholders in the ecosystem. End-users do not need to root their devices, find intervention tools or even self-develop their own customized tools. We eliminate the need for researchers to craft interventions (as users self-develop autonomously) or develop their own custom experience sampling tools (as end-users/researchers can analyze digital experiences from stored screenomes). We also eliminate the need for intervention developers to learn a new technical framework or learn how to fine-tune models. Running emulators on docker containers and virtual machines on a (single) host server is feasible, and thus allows for the browser stream to be accessible cross-device without restriction, e.g. access iOS emulator on Android device, or macOS virtual machine on Windows device. Certain limitations are imposed on the current implementation, such as a lack of access to the device camera, audio, and haptics; however, these are not permanent issues, and engineered implementations exist where a virtual/emulated device can route and access the host device's input/output sources (VrtualApp, 2016).

Figure 6: Architecture of *GreaseTerminator* (left) and *GreaseVision* (right).




(a) The high-level architecture of *GreaseTerminator*. Details are explained in Section 2.3 and 4.2.




(b) The high-level architecture of *GreaseVision*, both as a summary of our technical infrastructure as well as one of the collaborative HITL interventions development approach.

**Hooks** The *text hook* enables modifying the text that is displayed on the user's device. It is implemented through character-level optical character recognition (OCR) that takes the screen image as an input and returns a set of characters and their corresponding coordinates. The EAST text detection (Zhou et al., 2017) model detects text in images and returns a set of regions with text, then uses Tesseract (Google, 2007) to extract characters within each region containing text. The *mask hook* matches the screen image against a target template of multiple images. It is implemented with *multi-scale multi-template* matching by resizing an image multiple times and sampling different subimages to compare against each instance of mask in a masks directory (where each mask is a cropped screenshot of an interface element). We retain the default majority-pixel inpainting method for mask hooks (inpainting with the most common colour value in a screen image or target masked region). As many mobile interfaces are standardized or uniform from a design perspective compared to images from the natural world, this may work in many instances. The mask hook could be connected to rendering functions such as highlighting the interface element with warning labels, or image inpainting (fill in the removed element pixels with newly generated pixels from the background), or adding content/information (from other apps) into the inpainted region. Developers can also tweak how the mask hook is applied, for example using the multi-scale multi-template matching algorithm with contourized images (shapes, colour-independent) or coloured images depending on whether the mask contains (dynamic) sub-elements, or using few-shot deep learning models if similar interface elements are non-uniform. A *model hook* loads any machine learning model to take any input and generate any output. This allows for model embedding (i.e. model weights and architectures) to inform further overlay rendering. We can connect models trained on specific tasks (e.g. person pose detection, emotion/sentiment analysis) to return output given the screen image (e.g. bounding box coordinates to filter), and this output can then be passed to a pre-defined rendering function (e.g. draw filtering box).

## 6.2 Related Works (extended)

### 6.2.1 Motivation: Pervasiveness and Individuality of Digital Harms

It is well-known that digital harms are widespread in our day-to-day technologies. Despite this, the academic literature around these harms is still developing, and it remains difficult to state exactly what the harms are that need to be addressed. Famously, Gray et al. (Gray et al., 2018) put forward a 5-class taxonomy to classify dark patterns within apps: *interface interference* (elements that manipulate the user interface to induce certain actions over other actions), *nagging* (elements that interrupt the user's current task with out-of-focus tasks) *forced action* (elements that introduce sub-tasks forcefully before permitting a user to complete their desired task), *obstruction* (elements that introduce subtasks with the intention of dissuading a user from performing an operation in the desired mode), and *sneaking* (elements that conceal or delay information relevant to the user in performing a task).

A challenge with such framework and taxonomies is to capture and understand the material impacts of harms on individuals. Harms tend to be highly individual and vary in terms of how they manifest within users of digital systems. The harms landscape is also quickly changing with ever-changing digital systems. Defining the spec-

trum of harms is still an open problem, the range varying from heavily-biased content (e.g. disinformation, hate speech), self-harm (e.g. eating disorders, self-cutting, suicide), cyber crime (e.g. cyber-bullying, harassment, promotion of and recruitment for extreme causes (e.g. terrorist organizations), to demographic-specific exploitation (e.g. child-inappropriate content, social engineering attacks) (HM, 2019; Pater and Mynatt, 2017; Wang et al., 2017; Honary et al., 2020; Pater et al., 2019), for which we recommend the aforementioned cited literature. The last line of defense against many digital harms is the user interface. This is why we are interested in interface-emergent harms in this paper, and how to support individuals in developing their own strategies to cope with and overcome such harms.

#### 6.2.2 Developments in Interface Modification & Re-rendering

Digital harms have long been acknowledged as a general problem, and a range of technical interventions against digital harms are developed. Interventions, also similarly called modifications or patches, are changes to the software, which result in a change in (perceived) functionality and end-user usage. We review and categorize key *technical* intervention methods for interface modification by end-users, with cited examples specifically for digital harms mitigation. While there also exist non-technical interventions, in particular legal remedies, it is beyond this work to give a full account of these different interventions against harms; a useful framework for such an analysis is provided by Lawrence Lessig (Lessig) who characterised the different regulatory forces in the digital ecosystem.

**Interface-code modifications** (Kollnig et al., 2021; Higi, 2020; Jeon et al., 2012; Rasthofer et al., 2014; Davis and Chen, 2013; Backes et al., 2014; Xu et al., 2012; LuckyPatcher, 2020; Davis et al., 2012; Lyngs et al., 2020b; Freeman, 2020; rovo89, 2020; Agarwal and Hall, 2013; Enck et al., 2010; MaaarZ, 2019; VrtualApp, 2016) make changes to source code, either installation code (to modify software before installation), or run-time code (to modify software during usage). On desktop, this is done through *browser extensions* and has given rise to a large ecosystem of such extensions. Some of the most well-known interventions are ad blockers, and tools that improve productivity online (e.g. by removing the Facebook newsfeed (Lyngs et al., 2020b)). On mobile, a prominent example is *AppGuard* (Backes et al., 2014), a research project by Backes et al. that allowed users to improve the privacy properties of apps on their phone by making small, targeted modification to apps' source code. Another popular mobile solution in the community is the app *Lucky Patcher* (LuckyPatcher, 2020) that allows to get paid apps for free, by removing the code relating to payment functionality directly from the app code.

Some of these methods may require the highest level of privilege escalation to make modifications to the operating system and other programs/apps as a root user. On iOS, *Cydia Substrate* (Freeman, 2020) is the foundation for jailbreaking and further device modification. A similar system, called *Xposed Framework* (rovo89, 2020), exists for Android. To alleviate the risks and challenges afflicted with privilege escalation, *VirtualXposed* (VrtualApp, 2016) create a virtual environment on the user's Android device with simulated privilege escalation. Users can install apps into this virtual environment and apply tools of other modification approaches that may require root access. *ProtectMyPrivacy* (Agarwal and Hall, 2013) for iOS and *TaintDroid* (Enck et al., 2010) for Android both extend the functionality of the smartphone operating system with new functionality for the analysis of apps' privacy features. On desktops, code modifications tend not to be centred around a common framework, but are more commonplace in general due to the traditionally more permissive security model compared to mobile. Antivirus tools, copyright protections of games and the modding of UI components are all often implemented through interface-code modifications.

**Interface-external modifications** (Geza, 2019; Bodyguard, 2019; Lee et al., 2014; Ko et al., 2015; Andone et al., 2016; Hiniker et al., 2016; Löchtefeld et al., 2013; Labs, 2019; Okeke et al., 2018) are the arguably most common way to change default interface behaviour. An end-user would install a program so as to affect other programs/apps. No change to the operating system or the targeted programs/apps is made, so an uninstall of the program providing the modification would revert the device to the original state. This approach is widely used to track duration of device usage, send notifications to the user during usage (e.g. timers, warnings), block certain actions on the user device, and other aspects. The *HabitLab* (Geza, 2019) is a prominent example developed by Kovacs et al. at Stanford.

This modification framework is open-source and maintained by a community of developers, and provides interventions for both desktop and mobile.

**Visual overlay modifications** render graphics on an overlay layer over any active interface instance, including browsers, apps/programs, videos, or any other interface in the operating system. The modifications are visual, and do not change the functionality of the target interface. It may render sub-interfaces, labels, or other graphics on top of the foreground app. Prominent examples are *DetoxDroid* (flxapps, 2021), *Gray-Switch* (GmbH, 2021), *Google Accessibility Suite* (Google, 2021), and *GreaseTerminator* (Datta et al., 2021).

We would like to establish early on that we pursue a *visual overlay modifications* approach. Interventions should be rendered in the form of overlay graphics based on detected elements, rather than implementing program code changes natively, hence focused on changing the interface rather than the functionality of the software. Interventions should be generalizable; they are not solely website- or app-oriented, but *interface-oriented*. Interventions do not target specific apps, but general interface elements and patterns that could appear across different interface environments. To support the systemic requirements in Section 2.4, we require an interface modification approach that is (i) interface-agnostic and (ii) easy-to-use. To this extent, we build upon the work of ***GreaseTerminator*** (Datta et al., 2021), a framework optimized for these two requirements.

In response to the continued widespread presence of interface-based harms in digital systems, Datta et al. (Datta et al., 2021) developed *GreaseTerminator*, a visual overlay modification method. This approach enables researchers to develop, deploy and study interventions against interface-based harms in apps. This is based on the observation that it used to be difficult in the past for researchers to study the efficacy of different intervention designs against harms within mobile apps (most previous approaches focused on desktop browsers). *GreaseTerminator* provides a set of 'hooks' that serve as templates for researchers to develop interventions, which are then deployed and tested with study participants. *GreaseTerminator* interventions usually come in the form of machine learning models that build on the provided hooks, automatically detect harms within the smartphone user interface at run-time, and choose appropriate interventions (e.g. a visual overlay to hide harmful content, or content warnings). A visualisation of the *GreaseTerminator* approach is shown in Figure 6(a).

#### 6.2.3 Opportunities for Low-code Development in Interface Modification

**Low-code development platforms** have been defined, according to practitioners, to be (i) low-code (negligible programming skill required to reach endgoal, potentially drag-and-drop), (ii) visual programming (a visual approach to development, mostly reliant on a GUI, and "what-you-see-is-what-you-get"), and (iii) automated (unattended operations exist to minimize human involvement) (Luo et al., 2021). Low-code development platforms exist for varying stages of software creation, from frontend (e.g. App maker, Bubble.io, Webflow), to workflow (Airtable, Amazon Honeycode, Google Tables, UiPath, Zapier), to backend (e.g. Firevase, WordPress, flutterflow); none exist for software modification of existing applications across interfaces. According to a review of StackOverflow and Reddit posts analysed by Luo et al. (Luo et al., 2021), low-code development platforms are cited by practitioners to be tools that enable faster development, lower the barrier to usage by non-technical people, improves IT governance compared to traditional programming, and even suits team development; one of the main limitations cited is that the complexity of the software created is constrained by the options offered by the platform.

User studies have shown that users can self-identify malevolent harms and habits upon self-reflection and develop desires to intervene against them (Cho et al., 2021; Lyngs et al., 2020a). Not only do end-users have a desire or interest in self-reflection, but there is indication that end-users have a willingness to act. Statistics for content violation reporting from Meta show that in the Jan-Jun 2021 period, $\sim$ 42,200 and $\sim$ 5,300 in-app content violations were reported on Facebook and Instagram respectively (Meta, 2022) (in this report, the numbers are specific to violations in local law, so the actual number with respect to community standard violatons would be much higher; the numbers also include reporting by governments/courts and non-government entities in addition to members of the public). Despite a willingness to act, there are limited digital visualization or reflection tools that enable flexible intervention development

by end-users. There are visualization or reflection tools on browser and mobile that allow for reflection (e.g. device use time (Andone et al., 2016)), and there are separate and disconnected tools for intervention (Section 2.2), but there are limited offerings of flexible intervention development by end-users, where end-users can observe and analyze their problems while generating corresponding fixes, which thus prematurely ends the loop for action upon regret/reflection. There is a disconnect between the harms analysis ecosystem and interventions ecosystem. A barrier to binding these two ecosystems is the existence of low-code development platforms for end-users. While such tooling may exist for specific use cases on specific interfaces (e.g. web/app/game development) for mostly creationary purposes, there are limited options available for modification purposes of existing software, the closest alternative being extension ecosystems (Kollnig et al., 2021; Google, 2010a). Low-code development platforms are in essence "developer-less", removing developers from the software modification pipeline by reducing the barrier to modification through the use of GUI-based features and negligible coding, such that end-users can self-develop without expert knowledge.

**Human-in-the-Loop (HITL)** learning is the procedure of integrating human knowledge and experience in the augmentation of machine learning models. It is commonly used to generate new data from humans or annotate existing data by humans. Wallace et al. (Wallace et al., 2019) constructed a HITL system of an interactive interface where a human talks with a machine to generate more Q&A language and train/fine-tune Q&A models. Zhang et al. (Zhang et al., 2019) proposed a HITL system for humans to provide data for entity extraction, including requiring humans to formulate regular expressions and highlight text documents, and annotate and label data. For an extended literature review, we refer the reader to Wu et al. (Wu et al., 2021). Beyond lab settings, HITL has proven itself in wide deployment, where a wide distribution of users have indicated a willingness and ability to perform tasks on a HITL annotation tool, *reCAPTCHA*, to access utility and services. In 2010, Google reported over 100 million reCAPTCHA instances are displayed every day (Google, 2010b) to annotate different types of data, such as deciphering text for OCR of books or street signs, or labelling objects in images such as traffic lights or vehicles.

While HITL formulates the structure for human-AI collaborative model development, **model fine-tuning** and **few-shot learning** formulate the algorithmic methods of adapting models to changing inputs, environments, and contexts. Both adaptation approaches require the model to update its parameters with respect to the new input distribution. For model fine-tuning, the developer re-trains a pre-trained model on a new dataset. This is in contrast to training a model from a random initialization. Model fine-tuning techniques for pre-trained foundation models, that already contain many of the pre-requisite subnetworks required for feature reuse and warm-started training on a smaller target dataset, have indicated robustness on downstream tasks (Galanti et al., 2022; Abnar et al., 2022; Neyshabur et al., 2020). If there is an extremely large number of input distributions and few samples per distribution (small datasets), few-shot learning is an approach where the developer has separately trained a meta-model that learns how to change model parameters with respect to only a few samples. Few-shot learning has demonstrated successful test-time adaptation in updating model parameters with respect to limited test-time samples in both image and text domains (Raghu et al., 2020; Koch et al., 2015; Finn et al., 2017; Datta, 2021). Some overlapping techniques even exist between few-shot learning and fine-tuning, such as constructing subspaces and optimizing with respect to intrinsic dimensions (Aghajanyan et al., 2021; Datta and Shadbolt, 2022; Simon et al., 2020).

The raw data for harms and required interface changes reside in the history of interactions between the user and the interface. In the Screenome project (Reeves et al., 2020, 2021), the investigators proposed the study and analysis of the moment-by-moment changes on a person's screen, by capturing screenshots automatically and unobtrusively every $t = 5$ seconds while a device is on. This record of a user's digital experiences represented as a sequence of screens that they view and interact with over time is denoted as a user's **screenome**. Though not mobilized widely amongst users for their self-reflection or personalized analysis, integrating screenomes into an interface modification framework can play the dual roles of visualizing raw (harms) data to users while manifesting as parseable input for visual overlay modification frameworks.