A Utility-Preserving Obfuscation Approach for YouTube Recommendations

Jiang Zhang[1] Hadi Askari[2] Konstantinos Psounis[1] Zubair Shafiq[2]
[1]University of Southern California [2]University of California, Davis
{jiangzha,kpsounis}@usc.edu,{haskari,zubair}@ucdavis.edu

ABSTRACT

Online content platforms optimize engagement by providing personalized recommendations to their users. These recommendation systems track and profile users to predict relevant content a user is likely interested in. While the personalized recommendations provide utility to users, the tracking and profiling that enables them poses a privacy issue. There is increasing interest in building privacy-enhancing obfuscation approaches that do not rely on cooperation from online content platforms. However, existing obfuscation approaches primarily focus on enhancing privacy but at the same time they degrade the utility because obfuscation introduces unrelated recommendations. We design and implement De-Harpo, an obfuscation approach for YouTube's recommendation system that not only *obfuscates* a user's video watch history to protect privacy but then also *denoises* the video recommendations by YouTube to preserve their utility. In contrast to prior obfuscation approaches, De-Harpo adds a *denoiser* that makes use of a "secret" input (i.e., a user's actual watch history) as well as information that is also available to the adversarial recommendation system (i.e., obfuscated watch history and corresponding "noisy" recommendations). Our large-scale evaluation of De-Harpo shows that it outperforms the state-of-the-art by a factor of 2× in terms of preserving utility for the same level of privacy, while maintaining stealthiness and robustness to de-obfuscation.

KEYWORDS

privacy, utility, obfuscator, denoiser

1 INTRODUCTION

Online content platforms, such as YouTube, heavily rely on recommendation systems to optimize user engagement on their platforms. For instance, 70% of the content watched on YouTube is recommended by its algorithm [54]. These recommendation systems provide personalized content recommendations by tracking and profiling user activity. For instance, YouTube tracks and profiles activities of its users on YouTube as well as off of YouTube to this end [28]. This tracking and profiling enables these platforms to predict relevant content that a user is likely to be interested in. On one hand, this tracking and profiling enables desirable utility to users by providing relevant content recommendations. On the other hand, this tracking and profiling poses a privacy issue because the platform might infer potentially sensitive user interests.

Some platforms, including YouTube, allow users to remove a subset of the tracked activity (e.g., remove a specific video from YouTube watch history) or even disable the use of certain profiled user interests (e.g., gambling) to influence the recommendations. However, these controls do not necessarily stop the platform from tracking and profiling user activities in the first place. Thus, they may not provide much, if any, privacy benefit to users. Moreover, the exercising of these controls would hurt the quality of personalized recommendations. For example, if users employ these controls to curtail tracking or profiling then they will likely not receive personalized recommendations they are actually interested in.

The research community is increasingly interested in developing privacy-enhancing obfuscation approaches that do not rely on cooperation from online content platforms [13, 30, 48, 74]. At a high level, these privacy-enhancing approaches work by adding fake activity to real user activity to lessen the ability of the recommendation system to infer sensitive information. However, the addition of fake activity for the sake of obfuscation also ends up impacting the utility users might derive from the recommendation system in terms of relevance of personalized recommendations. Prior obfuscation approaches attempt to navigate the trade-off between privacy and utility, for example [74], by carefully adding fake activity so as to obfuscate "private" interests but allow "non-private" interests.

In this work, we are interested in designing a privacy-enhancing *and* utility-preserving obfuscation approach for recommendation systems. In contrast to prior approaches that are typically limited to only obfuscating inputs to the recommendation system, our key idea is to design an obfuscation approach that can obfuscate inputs to preserve user privacy but at the same time remove "noise" from outputs to preserve the utility of recommendations. Since an adversarial recommendation system might also attempt to remove "noise", it is crucial that the denoiser can only be used by the user and not by the recommendation system. To this end, our insight is that the denoiser uses a "secret" input (specifically, a user's actual browsing history), which is only available to the user and not the recommendation system. The recommendation system instead only has access to the obfuscated browsing history of the user. Therefore, by leveraging the knowledge of a user's actual browsing history, the denoiser allows the user to preserve the recommendations related to the users' actual interests while discarding the unrelated recommendations caused by obfuscation.

We design and implement De-Harpo, an obfuscation approach for YouTube's recommendation system that not only obfuscates a user's video watch history to protect privacy but then also denoises the video recommendations by YouTube to preserve their utility. De-Harpo uses an *obfuscator* to inject obfuscation videos into a user's video watch history and a *denoiser* to remove recommended videos that are unrelated to the user's actual interests.

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
Proceedings on Privacy Enhancing Technologies 2023(4), 522–539
© 2023 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2023-0123

The *obfuscator* is a RL model trained to insert YouTube videos in a users' watch history that will maximize the distortion in their interests being inferred by YouTube. We address three key issues in designing De-Harpo's *obfuscator*, which is a non-trivial adaptation of Harpo [74] to YouTube. First, we build a surrogate of YouTube's recommendation system to efficiently train the RL model in a virtual environment. Second, we design the surrogate model to predict the distribution of hundreds of different classes of YouTube recommendation videos (we use the 154 affinity segments used by Google [26] as our video classes) rather than the sheer number (order of hundreds of millions) of individual YouTube videos. Lastly, the *obfuscator* selects obfuscation videos based on embedding similarity, which is scalable to millions of obfuscation videos.

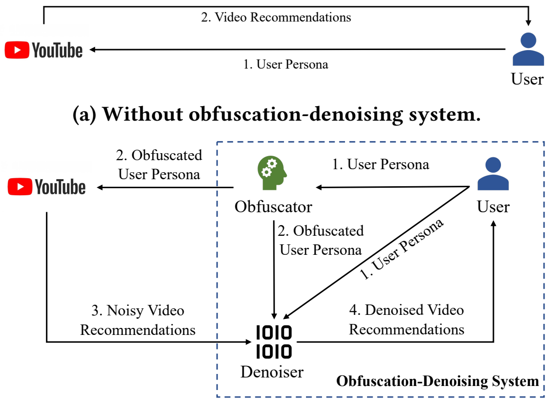
The *denoiser* is a ML model that is trained to reproduce the original recommendations that would have been received in the absence of the *obfuscator*. We address two key issues in designing De-Harpo's *denoiser*. First, *denoiser* makes use of a "secret" input (i.e., a user's actual watch history) as well as information that is also available to the adversarial recommendation system (i.e., obfuscated watch history and corresponding "noisy" recommendations). As we show later, this design ensures that only De-Harpo is able to remove "noise' while the adversary is unable to de-obfusacte without prohibitive collateral damage. Second, we define new divergence-based metrics to measure privacy and utility in training *obfuscator* and *denoiser*.

We deploy and evaluate De-Harpo's effectiveness on YouTube using 10,000 sock puppet based personas, 10,000 Reddit user personas, and 936 real-world YouTube users [7]. Our evaluation shows that De-Harpo's *obfuscator* is able to degrade the quality of YouTube's recommendations by up to 87.23% (privacy) and its *denoiser* is able to recover up to 90.40% of the actual recommendations (utility). We show that De-Harpo outperforms the state-of-the-art by a factor of 2× in terms of improving utility for the same level of privacy. Crucially, we also demonstrate that De-Harpo is stealthy and robust to de-obfuscation by an adversarial system. Our evaluation shows that the adversary incurs a prohibitively large number of false positives (order of tens/hundreds of millions) in attempting to undermine stealthiness and achieving de-obfuscating.

2 PRELIMINARIES

2.1 Problem Statement

Recommendation systems track users' browsing activity to provide personalized recommendations. YouTube, for example, tracks users' browsing activity on YouTube (e.g., videos watched, channel subscriptions) as well as off of YouTube (e.g., activity on other Google services such as Google Search and Google Analytics, or web pages opened in Chrome browser) to personalize homepage and up-next video recommendations [28]. Users can selectively remove certain videos from their YouTube watch history or clear their browsing activity altogether to influence personalized video recommendations. However, doing so does not necessarily mean that their browsing activity is not tracked in the first place, and thus there is no material privacy benefit to users. It will also hurt the quality of personalized recommendations because users will likely not receive recommendations for videos they are interested in. In summary, users are unable to exert meaningful control over recommendation systems to protect their privacy while preserving the utility of personalized recommendations.


2. Video Recommendations
YouTube
1. User Persona
User
2. Obfuscated User Persona
YouTube
1. User Persona
Obfuscator
User
2. Obfuscated User Persona
1. User Persona
3. Noisy Video Recommendations
4. Denoised Video Recommendations
1010 1010
Denoiser
Obfuscation-Denoising System

(a) Without obfuscation-denoising system.

(b) With obfuscation-denoising system.

Figure 1: Problem Overview.

Prior work has proposed obfuscation approaches to protect user privacy in personalized recommendation systems without relying on cooperation from online content platforms. Existing approaches obfuscate a user's browsing history by injecting fake activity (e.g., webpage visits) to manipulate a user's interest segments and targeted ads in online behavioral advertising [68, 74]. These obfuscation approaches are designed for recommendation systems (e.g., online behavioral advertising) where users are not necessarily interested in consuming the output of the recommendation system, rather users are mainly interested in subverting it. While these approaches aim to protect user privacy (e.g., inferred interest segments), they do not consider the utility of recommendations (e.g., whether targeted ads are of interest to the user). In contrast, in recommendation systems such as YouTube, these obfuscation tools would render the utility of YouTube's video recommendations useless to the user.

Can we design privacy-enhancing obfuscation approaches that can enhance privacy of users and at the same time preserve utility for users in recommendation systems? With this goal in mind, we propose to build a denoiser to remove the "noisy" videos injected as part of obfuscation. It is crucial that the denoiser can only be used by the user and not by the recommendation system. To this end, our insight is that the denoiser uses a "secret" (specifically, the user's actual browsing history), which is only available to the user and not the recommendation system. Therefore, by leveraging the knowledge of a user's actual browsing history, the denoiser may preserve the recommendations related to the users' actual interests while discarding the unrelated recommendations caused by obfuscation. Figure 1 illustrates this idea that we next operationalize in De-Harpo.

2.2 Threat Model

User. The user's goal is to routinely browse YouTube videos and get high-quality recommendation videos fitting their interests, while misleading the YouTube recommendation system such that it can not accurately infer the user's interests. To achieve this goal, users install a local obfuscation-denoising system, which consists of an *obfuscator* and a *denoiser*. The *obfuscator* will obfuscate their video watching history by injecting fake video watches into the user's real video watches, and the *denoiser* will automatically remove "noisy" recommended videos from YouTube (i.e. caused by obfuscation) that do not fit user's interests. The obfuscation-denoising system is designed to satisfy the following properties:

- it is **privacy-preserving** in that the user's interests are protected from being inferred by YouTube.
- it is **utility-preserving** in that the user can receive high-quality videos fitting their interests.
- t has **low overhead** in that the amount of obfuscation videos inject will not affect the user experience.
- it is **stealthy** in that it is impractical for YouTube to detect the usage of obfuscation-denoising system.
- it is **robust to deobfuscation** in that it is impossible for YouTube to distinguish fake video watches from real video watches.
- it can be **personalized** in that it can treat video classes differently based on user preferences.

Recommendation system. The goal of the recommendation system is to track user activity for personalized recommendations to maximize user engagement (e.g., click rate and watch time). We assume that the recommendation system has full access to the user's video watching history (including both fake and real video watches though it does not know which is which) and it recommends videos based on the user's video watching history, which is true for YouTube [22] (unless the user deletes their watching history). We further assume that the recommendation system does not have access to the user's off-platform browsing history (e.g., the user is not simultaneously signed-in to YouTube and other services by YouTube's parent company Google, the user employs Google account controls to prevent off-YouTube information linking (if the user is signed-in to YouTube and other services by YouTube's parent company Google) [27], or the user uses a browser such as Safari [66] or Firefox [46] – or privacy-enhancing browser extension [64] – that prevents cross-site tracking). We also assume that the recommendation system has substantial computation resources to train a machine learning model for its recommendations. This assumption also holds for YouTube [11]. Moreover, we assume that the recommendation system has access to De-Harpo once it is public, such that it can use it to analyze the obfuscation approach and possibly train adversarial detectors to detect and filter the usage of De-Harpo. More specifically, we assume that the recommendation system has a two-step detection workflow. In the first step, the adversary will train a classifier to detect whether or not a user uses De-Harpo. Then, in the second step, if De-Harpo usage is detected, the adversary further attempts to achieve deobfuscation by filtering out obfuscation videos and keeping the remaining videos.

3 PROPOSED APPROACH

In this section, we present the proposed utility-preserving obfuscation approach De-Harpo.

3.1 Overview

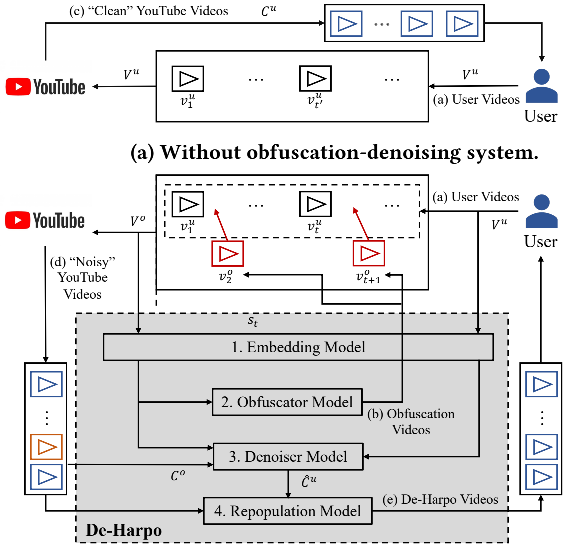
As already discussed, at a high-level De-Harpo consists of an *obfuscator* designed for enhancing user privacy and a *denoiser* designed for preserving user utility, as demonstrated in Figures 1 and 2 (in more detail). The De-Harpo *obfuscator* is a non-trivial adaptation of Harpo's obfuscator [74] in the context of YouTube's recommendation system. The *obfuscator* injects fake video playing records into a user's video playing history at random times. We refer to videos played by the user as *user videos* and to videos played by the *obfuscator* as *obfuscation videos*. Note that without any *obfuscation videos* in the user's video playing history (which is denoted by V^u in this case), YouTube will recommend a set of videos desired by the user. We refer to this set of videos as "clean" YouTube videos. However, with *obfuscation videos* in the user's video playing history (which is denoted by V^o in this case), YouTube will recommend a set of videos which include videos undesired by the user. We refer to this set of videos as "noisy" YouTube videos. The *denoiser* is designed to predict the class distribution of "clean" YouTube videos from the class distribution of "noisy" YouTube videos, such that De-Harpo can repopulate a new set of videos with the same class distribution as the "clean" YouTube videos. We refer to the repopulated videos as De-Harpo videos. Note that each video class represents a video topic, and we use the 154 affinity segments used by Google [26] as our video classes.

In more detail, see Figure 2, De-Harpo starts by generating video embeddings of past played videos via an embedding model. It then uses an *obfuscator* model to select obfuscation videos based on the generated video embeddings. Note that we follow a similar methodology with that in [74] to formulate the process of inserting *obfuscation videos* as a Markov Decision Process (MDP), and use reinforcement learning (RL) to train the *obfuscator* model to maximize the divergence between the class distribution of "noisy" YouTube videos (denoted by C^o) and the class distribution of "clean" YouTube videos (denoted by C^u). After receiving the "noisy" YouTube videos, the *denoiser* outputs an estimate of the class distribution of "clean" YouTube videos (denoted by $\hat{C}^u$), by taking as inputs V^u, V^o, and C^o. Finally, De-Harpo will use a repopulation model to generate the set of De-Harpo videos with class distribution $\hat{C}^u$.

3.2 System Preliminaries

User persona. We define a user persona as a sequence of YouTube videos. Formally, we denote the non-obfuscated user persona as $V^u = [v_1^u, ..., v_n^u]$, where v_i^u represents the ith video played by the user, and n is the total number of videos played by the user. We denote the obfuscated user persona as $V^o = [v_1^j, ..., v_{n'}^j]$, where $j \in \{u, o\}$, v_i^u and v_i^o represent that the ith video is played by the user and *obfuscator* respectively, and n' is the total number of videos played by the user and *obfuscator* combined.

Recommended video class distribution. We define the recommended video class distribution of a non-obfuscated user persona V^u (i.e. the class distribution of "clean" YouTube videos) as $C^u = [c_1^u, ..., c_K^u]$, where $\sum_{k=1}^{k=K} c_k^u = 1$, c_k^u is the percentile of videos from the kth class among recommended videos for V^u, and K is the total number of classes. Similarly, we define the recommended video class distribution of an obfuscated user persona V^o (i.e. the class distribution of "noisy" YouTube videos) as $C^o = [c_1^o, ..., c_K^o]$, where $\sum_{k=1}^{k=K} c_k^o = 1$ and c_k^o is the percentile of videos from the kth class among the recommended videos for V^o. We use the recommended video class distribution as a representation of the user interest profile built by YouTube instead of directly using the recommended videos. This design choice is made to (i) mitigate the impact



(a) Without obfuscation-denoising system.

(b) With obfuscation-denoising system.

Figure 2: Overview of De-Harpo. Note that V^u denotes the non-obfuscated user persona, V^o denotes the obfuscated user persona generated by the *obfuscator*, C^u is the recommended video class distribution based on V^u, C^o is the recommended video class distribution based on V^o, $\hat{C}^u$ is the *denoiser*'s estimate of C^u, and v_i^u and v_i^o represent user video and obfuscation video respectively.

of non-determinism in YouTube's recommendations and (ii) alleviate the difficulty of making video-level recommendations given an incomplete set of available videos while still making reasonably fine-grained recommendations (among 154 different classes).

Privacy metric. At a high level, we want to distort the user interest profile built by YouTube for user personas to enhance user privacy. Motivated by the use of the recommended video class distribution as a representation of YouTube's user interest profile, we first define the following privacy metric:

$$P = E[D_{KL}(C^o||C^u)] = E[\sum_{k=1}^{k=K} c_k^o \log \frac{c_k^o}{c_k^u}], \tag{1}$$

which measures the expected KL divergence between the two probability distributions (C^o and C^u)[1].

It is worth noting that we use KL divergence since it is a well-established measure of the discrepancy between two distributions, and, together with the closely related mutual information measure they have been used as on-average privacy metrics in myriad of applications including recommendation systems [10, 12, 19, 51, 52, 73]. We do not use stricter privacy metrics which provide worst-case privacy guarantees (e.g. differential privacy (DP) [18]), since in the context of our application one would need to inject an enormous number of obfuscation videos to satisfy such guarantees (see Section 3.3 and Appendix A for a detailed, formal discussion on DP in our context).

During real-world experimentation on YouTube, we observe that the recommended video class distribution of the same persona may differ a bit due to an inherent randomness of the system. Since we are interested to measure the divergence thanks to obfuscation only, we define D^{Min} as the expected KL divergence between a random sample of C^u and its mean $\bar{C}^u$ (i.e., $D^{Min} = E[D_{KL}(\bar{C}^u, C^u)]$), and subtract from P the divergence caused by randomness, that is, we work with $P - D^{Min}$. Furthermore, since P is unbounded, we normalize the privacy metric as follows. Denote the user persona set as $\mathcal{V}$, which consists of all user personas. Let V^u and $V^{u'}$ be two user personas uniformly and randomly sampled from $\mathcal{V}$, and let their associated recommended video class distributions be C^u and $C^{u'}$ respectively. Then, we define the normalized privacy metric P^{Norm} by:

$$P^{Norm} = \frac{P - D^{Min}}{D^{Max} - D^{Min}}, \tag{2}$$

where $D^{Max} = E[D_{KL}(C^u, C^{u'})]$ is the expectation of the KL divergence between C^u and $C^{u'}$ and thus corresponds to the average "distance" between two video class distributions of two randomly selected users. Hence, P^{Norm} measures the fraction of the maximum possible divergence that obfuscation achieves, on average. Note that for both P and P^{Norm}, the higher their value is, the more effective the *obfuscator* is in enhancing user privacy (see Figure 3).

Utility metric. In our threat model, the user sends the obfuscated persona to YouTube and then receives a "noisy" recommended video list with class distribution C^o. However, the user desires the "clean" recommended video list with class distribution C^u. Our *denoiser* is designed to predict C^u from C^o, such that De-Harpo can repopulate the "clean" recommended video list from C^u. With the above in mind, we define our utility loss metric as follows:

$$U_{Loss} = E[D_{KL}(\hat{C}^u||C^u)] = E[\sum_{k=1}^{k=K} \hat{c}_k^u \log \frac{\hat{c}_k^u}{c_k^u}], \tag{3}$$

where $\hat{C}^u$ is the output of the *denoiser*, representing its estimation of C^u. Smaller U_{Loss} means smaller divergence between the non-obfuscated recommended video class distribution C^u and the *denoiser*'s estimate of such distribution $\hat{C}^u$ and thus a better estimate. The theoretical minimum that this value can take is 0, representing two identical distributions i.e. the noise is perfectly removed. Note that without applying the *denoiser*, the utility loss equals the value of privacy P (since $\hat{C}^u = C^o$). The *denoiser* can reduce the utility loss caused by the *obfuscator* by $P - U_{Loss}$ which represents the *denoiser* utility gain. Similarly to above, because P is unbounded and YouTube's randomness causes, on average, a divergence of D^{Min}, we define the normalized utility gain metric as follows:

$$U_{Gain}^{Norm} = \frac{P - U_{Loss}}{P - D^{Min}}, \tag{4}$$

which represents the fraction of obfuscation noise reduced by the *denoiser*, on average. Higher U_{Gain}^{Norm} implies that the *denoiser* can reduce the utility loss caused by the *obfuscator* more effectively and a value of 100% indicates a complete removal of noise (see Figure 3).

3.3 Performance Goals and Guarantees

Performance goals. As discussed already, our goal is to obfuscate the actual user profile, that is, the inferred user's interests

[1] Note that if $c_k^i = 0$ ($i \in \{u, o\}$), we assign a small value to it to avoid getting ∞ in KL divergence calculation.

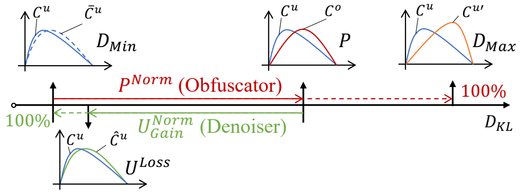


Figure 3: Privacy and utility metrics.

by YouTube from the user's video watch history. (We do not consider other sub-channels via which YouTube may infer user interests, see threat model details in Section 2.2.) In view of obfuscation, YouTube's goal is to reconstruct the actual user profile (what YouTube would have inferred by the user's video watch history in the absence of obfuscation) as accurately as possible from the obfuscated user profile (what YouTube infers by the user's video watch history in the presence of obfuscation). Since YouTube's user profiles are not public, we infer them from YouTube's recommended videos to the user, and, more specifically, from the the recommended video class distribution (where we use the 154 affinity segments used by Google as our video classes).

Motivated by the above, our privacy metric maximizes the distance (normalized KL divergence) between the recommended video class distribution before and after obfuscation. If the distance between the recommended video class distribution before and after obfuscation is almost the same with the distance between the recommended video class distribution before obfuscation and the recommended video class distribution of another random user, then YouTube's recommendations for the user under study are essentially random implying that YouTube is not able to learn the user's actual interests from the obfuscated user's video watch history. Tellingly, in Section 6.3 we do show that with merely 70% of videos in a user persona being obfuscation videos, the distance between the recommended video class distribution before and after obfuscation is already 93% of the distance between random distributions.

Performance guarantees. A discussion about performance guarantees is in order. First, can De-Harpo effectively de-noise the noisy recommendations such that their utility is high, despite that recommendations are as if they were random? Section 6.2 answers affirmatively. Related to this, if De-Harpo can de-noise recommendations, can't YouTube de-noise them as well? Sections 6.5, 6.6 show that it cannot in practice, and Section 3.5 offers a formal explanation why it can't. Note that even though YouTube unavoidably learns the interests of a user corresponding to the user videos that the user actually watches, it also learns interests corresponding to the obfuscation videos, the relative importance of each interest is altered, and YouTube has no practical way of telling which interest is real and which is not. [2]

Second, both our privacy and utility metrics are based on expectations, see Eq. (1)-(4). Hence, De-Harpo guarantees performance goals "on-average". But what about "worst-case" privacy guarantees? In our context this would require that no matter how unique the original video watch history of a specific user may be, YouTube should not be able to learn any unique interests of this user regardless of how unsuccessful it may be across all users on average. There is a large line of prior work on both "on-average" [10, 19, 51, 52, 73] and "worst-case" [12, 17–19, 73] privacy guarantees. It is intuitive that strict definitions of privacy like differential privacy (DP) [18], which guarantee privacy in the worst-case, cannot be satisfied for recommendations systems actively used by users. For a matter of completeness, we provide a formal proof about why differential privacy can not be achieved in Appendix A. A summary of the argument follows: Assume that there is one video V in user persona P_1 (i.e. video watch history) which is not in user persona P_2, and the obfuscator O (the randomized function in the DP definition) can not remove it from P_1. Let P be a user persona without video V that we observe. Then, the probability of $O(P_1)$ being P is zero while the probability of $O(P_2)$ being P is non-zero. Thus, per the DP definition, the ϵ for this worst-case scenario will be infinite and DP is violated.

3.4 System Model

Obfuscator. The obfuscation video selection process of *obfuscator* is formulated as a Markov Decision Process (MDP). At the beginning of each time step, a video will be played. If the played video is an obfuscation video injected by the *obfuscator*, we refer to this time step as an obfuscation step. Let $\alpha \in [0, 1)$ be the obfuscation budget which we use as a system parameter to control the percentile of obfuscation videos. At each time step, with probability α an obfuscation video will be injected by *obfuscator* into the user persona. Let s_t denote the state of the MDP at obfuscation step t, defined as all the played videos until now, and a_t denote the action of the MDP at obfuscation step t, which represents the obfuscation video sampled based on the MDP policy. The MDP policy is a probability distribution which outputs the probability of selecting obfuscation video i given state s_t, and we denote this probability by $\pi(a_t = i|s_t)$. We associate a reward r_t for the action a_t at obfuscation step t. We set the reward r_t to $P_t - P_{t-1}$, where P_t is the privacy metric value at obfuscation step t. The goal of solving this MDP is to find the optimal policy, such that the accumulative rewards $\sum_{t=1}^{t=T} r_t$ can be maximized. Note that T denotes the total number of obfuscation steps. (We consider a finite-horizon MDP) Appendix B.1 discusses the MDP in more detail.

Denoiser. At a high level, we model the *denoiser* as a mapping from the recommended video class distribution of the obfuscated user persona $C^o \in \mathbb{R}^K$ to the recommended video class distribution of the non-obfuscated user persona $C^u \in \mathbb{R}^K$ (K is the total number of video categories).

Estimating directly C^u from C^o can be challenging. In the extreme case, where the mutual information between C^u and C^o is zero [41], it is impossible for the *denoiser* to estimate C^u from C^o. To estimate C^u, the *denoiser* may leverage side information indicating how the obfuscation videos are injected into the user personas, as in this case it may be able to undo the effect of obfuscation videos in the recommendations list. In our application, such side information is explicitly available to users (V^u portion of V^o), since the *obfuscator* is installed locally and users know exactly how the obfuscation videos are injected into user personas. Therefore, our *denoiser* is modeled to be a functional mapping from (V^u, V^o, C^o) to C^u.

[2] If a user wishes YouTube to not learn about the user's real interests at all, the user should not use YouTube: Even though YouTube in theory offers a method to remove a video from the watch history, (i) even if the video is deleted the corresponding interest categories are not [3] and (ii) there is no "unlearning" at the ML level and hence the recommendation algorithm will still recommend videos based on the total watch history.

3.5 The “Secret" of the Denoiser

We use the information theory concept of mutual information (MI) to explain why the *denoiser* works. Recall that the recommendation system cannot distinguish user from obfuscation videos thus does not know the user's video playing history V^u. In our system, both V^u and V^o are modelled as random vectors, and V^o is generated from V^u by the *obfuscator*, which is a random function. Additionally, both C^u and C^o are random vectors, which are generated from V^u and V^o respectively by the YouTube recommendation system. By applying the chain rule of MI, we can derive the following equation:

$$I(C^o, V^o, V^u; C^u) = I(C^o, V^o; C^u) + I(V^u; C^u | C^o, V^o), \qquad (5)$$

where $I(C^o, V^o, V^u; C^u)$ is the MI between (C^o, V^o, V^u) and C^u, $I(C^o, V^o; C^u)$ is the MI between (C^o, V^o) and C^u, and $I(V^u; C^u | C^o, V^o)$ is the MI between V^u and C^u conditioning on (C^o, V^o).

First, we show that the non-obfuscated user persona V^u can be leveraged by the *denoiser* to better estimate C^u. Since C^u is generated by YouTube recommendation system given V^u, V^u is correlated with C^u, thus $I(V^u; C^u | C^o, V^o) > 0$. Hence,

$$I(\underbrace{C^o, V^o, V^u}_{\text{with secret}}; C^u) > I(\underbrace{\ C^o, V^o\ }_{\text{without secret}}; C^u). \qquad (6)$$

Since the MI between (V^u, V^o, C^o) and C^u is larger than the MI between (C^o, V^o) and C^u, (C^o, V^o, V^u) can reveal more information about C^u than (C^o, V^o), leading to a more accurate estimate of C^u. As an aside, note that YouTube may attempt to de-obfuscate V^u from V^o. We evaluate the robustness of the *obfuscator* against de-obfuscation in Section 6.6.

Second, we show that including C^o and V^o may help to further enhance the effectiveness of the *denoiser*, compared with using V^u only. Based on the chain rule of MI, we can rewrite Eq. (5) as follows:

$$\begin{aligned} &I(V^u, V^o, C^o; C^u) \\ &= I(V^u; C^u) + I(C^o; C^u | V^u) + I(V^o; C^u | C^o, V^u). \end{aligned} \qquad (7)$$

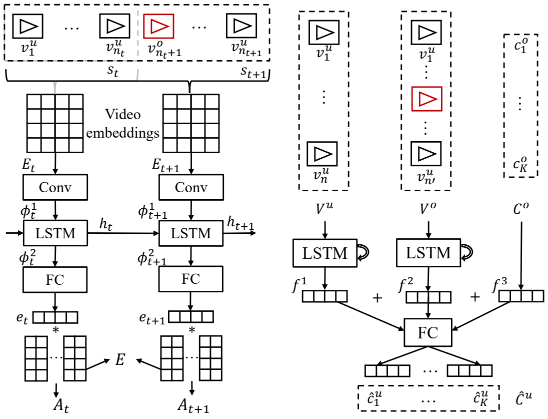
Consider the term $I(C^o; C^u | V^u)$. C^o depends on V^u and the obfuscation videos, and C^u depends on V^u. Crucially, they both also depend on the (non deterministic) YouTube recommendation system. Hence, even when V^u is given, there is non-zero MI between C^o and C^u, that is, $I(C^o; C^u | V^u) > 0$, leading to the following inequality:

$$I(V^u, V^o, C^o; C^u) > I(V^u; C^u), \qquad (8)$$

which means the MI between (V^u, V^o, C^o) and C^u is larger than the MI between V^u and C^u only. Intuitively, knowing the pair V^o, C^o reveals information about how the YouTube recommendation system selects videos to recommend given a user video watching history. Therefore, the *denoiser* taking C^o and V^o as additional inputs can learn more information about C^u, as compared to the *denoiser* taking only V^u as input. Our evaluation results in Section 6.2 empirically support the above analysis.

4 SYSTEM DESIGN AND IMPLEMENTATION

In this section, we describe the detailed design of De-Harpo and how we implement De-Harpo as a browser extension. De-Harpo consists of five modules: (1) a video embedding model that maps videos into embeddings; (2) a *obfuscator* model that selects obfuscation videos based on the video embeddings of played videos; (3) a *denoiser* model that estimates the class distribution of “clean" YouTube videos from the class distribution of “noisy" YouTube videos; (4) a repopulation model that outputs De-Harpo videos with the estimated class distribution of “clean" YouTube videos; (5) a surrogate model used to train the *obfuscator* model offline efficiently (see Figure 2b for the workflow of modules (1)-(4)).



(a) ***obfuscator*** (b) ***denoiser***

Figure 4: Details of system design.

4.1 Video Embedding

To make our system scalable to millions of YouTube videos without being restricted to a fixed set, we represent each video by an embedding vector. A YouTube video typically consists of metadata (e.g. title, description, view count, rating, thumbnail, etc), a sequence of image frames (i.e. the video), and the transcript for the video. Since a video's transcript is a good representation of its content and it is more computationally and spatially efficient to process the transcript compared to processing the original video stream, we use video metadata and transcript to generate the video embedding, where the video embedding for video v_i is denoted by $e_i \in \mathbb{R}^{404}$ (see Figure 9a in Appendix B.2 for details) [3].

4.2 Obfuscator Model

As discussed before, we model the process of injecting obfuscation videos as an MDP. Due to the prohibitively large state space of this MDP, we use RL, parameterized by a deep neural network, to learn the optimal policy for obfuscation video selection.

The *obfuscator* takes as input the state at each obfuscation step, and outputs a video embedding. By measuring the cosine similarity between the output video embedding and each obfuscation video embedding, the *obfuscator* derives the probability distribution of the obfuscation video selection, where an obfuscation video with more similar embedding as the output video embedding is assigned a higher probability. Specifically, as shown in Figure 4a, the *obfuscator* consists of a convolutional layer (Conv), a LSTM layer, and a

[3] Note that the YouTube recommendation system will use the image frames and some other private features to generate the video embedding (see [11]). We acknowledge that by including these features, our video embeddings may be closer to the actual embeddings used by YouTube. However, since our video embeddings can already yield a surrogate model (see Section 4.5 with reasonable performance and it is more computationally efficient, we choose the current design of our video embeddings.

fully-connected layer (FC). At step t, the convolutional layer takes the embeddings of the past n_t videos as input ($E_t \in \mathbb{R}^{n_t \times 404}$) and outputs a real vector with m_1 elements ($\phi_t^1 \in \mathbb{R}^{m_1}$). Next, the LSTM layer takes ϕ_t^1 and the hidden vector at obfuscation step $t-1$ with m_3 elements $h_{t-1} \in \mathbb{R}^{m_3}$ as input, and outputs a real vector with m_2 elements ($\phi_t^2 \in \mathbb{R}^{m_2}$) and the hidden vector $h_t \in \mathbb{R}^{m_3}$ for obfuscation step t ($m_1 = m_2 = m_3 = 128$ in our experiments). Finally, a linear layer converts ϕ_t^2 into a real vector with the same dimension as the video embedding. We denote this vector by $e_t \in \mathbb{R}^{404}$ as it represents the target embedding for the obfuscation video. Let $E = [e_1, ..., e_M]$ denote the embedding vectors of the M obfuscation videos at our disposal. Then, the probability of selecting the i-th obfuscation video, $i = 1 \dots M$, is calculated proportionally to the similarity between its embedding and the target embedding after normalizing using a softmax function:

$$\pi(a_t = i | s_t) = \frac{e^{\langle e_t, e_i \rangle}}{\sum_{i=1}^{i=M} e^{\langle e_t, e_i \rangle}}, \tag{9}$$

where $\langle x, y \rangle$ denotes the inner product between x and y. Note that we use the on-policy RL algorithm A2C (Advantage Actor and Critic)[50] to train the *obfuscator* (see Appendix D).

Recall that the De-Harpo *obfuscator* is a non-trivial adaptation of Harpo [74] to YouTube. An important technical difference is that by calculating a target embedding and then selecting an obfuscation item (video in case of YouTube) based on the similarity between its embedding and the target embedding, the De-Harpo *obfuscator* can handle an unlimited and varying number of possible obfuscation videos without requiring re-training when the set of obfuscation videos changes.

4.3 Denoiser Model

As mentioned in Section 3.4, the *denoiser* has three inputs: the non-obfuscated user persona V^u, the obfuscated user persona V^o, and the recommended video class distribution of obfuscated user persona C^o. The *denoiser* uses two LSTM layers and an FC layer to encode inputs, as shown in Figure 4b. Specifically, the first LSTM layer takes as input the embeddings of videos in the non-obfuscated user persona V^u recurrently and outputs its final hidden vector $f^1 \in \mathbb{R}^n$ (we use $n = 128$ in our experiments). Similarly, the inputs of the second LSTM layer are the embeddings of videos in the obfuscated user persona V^o and its output is its last hidden vector $f^2 \in \mathbb{R}^n$. Last, the FC layer converts the class distribution $C^o \in \mathbb{R}^K$ (where K represents the number of categories) into a real vector $f^3 \in \mathbb{R}^n$. By concatenating vectors f^1, f^2, and f^3 into a single vector with dimension $3n$, a final FC layer is used to map it into the estimated recommended video class distribution $\hat{C}^u \in \mathbb{R}^K$. Note that we train the *denoiser* based on supervised learning with stochastic gradient descent (see Appendix D).

4.4 Repopulating Recommended Videos

Recall that the *denoiser* in De-Harpo outputs a target video class distribution $\hat{C}^u$. In order to go from a target video class distribution back to actual videos on the user's screen, we repopulate the recommendations using a browser extension.

For efficiency, we maintain a "bank" of videos per class and use it to repopulate the recommendations. This leads to the question of how often should we refresh this bank in order to get a suitable trade-off between the recency of the videos and the overhead required to collect the videos. To ascertain what the optimal time period would be to refresh this bank we run a 24-hour experiment where we query the name of a class in the YouTube search bar as a proxy for the explicit class and collect statistics for each class's most popular recommended videos. Specifically, we run the same query each hour, collect the top 20 search results per query, and compute the percentile of top queries that remain the same. The results indicate that for most classes about 70-80% of the top search results remain the same. Motivated by this, we periodically – or on an on-demand basis – crawl a sufficiently large number of videos for each class to re-populate our bank. Note that the "noisy" recommended videos removed during the repopulation process will be included into our obfuscation video sets such that they can be played later to augment the obfuscation effect.

4.5 YouTube Surrogate Model

The training of the *obfuscator* requires frequent interactions with the YouTube recommendation system. However, directly interacting with YouTube is time-consuming, since it takes more than 30 minutes to construct a single persona (as described in Section 5.2). To train the *obfuscator* efficiently, we build a surrogate model as a replication of the actual YouTube recommendation system.

The architecture of our surrogate model consists of a LSTM layer and a FC layer. The LSTM layer takes as input the embeddings of videos in a user persona recurrently and outputs its last hidden vector, which will be used as the input of the FC layer. Then, the FC layer will output the recommended video class distribution $C^i \in \mathbb{R}^K$, where $i \in \{u, o\}$ (see Figure 9b in Appendix B.3 for details). Note that we train the surrogate model via supervised learning with stochastic gradient descent (see Appendix D). We also provide detailed discussion about the rationale of designing such surrogate model and the differences between our surrogate model and prior works in Appendix B.3.

4.6 De-Harpo Implementation

We implement De-Harpo as a browser extension, which consists of two components: *obfuscator* and *denoiser*.

Obfuscator. The *obfuscator* is a lightly modified version of Harpo's browser extension [74]. The browser extension plays the selected obfuscation videos in a background tab that is hidden from users. In order to determine the timing of playing obfuscation videos, the *obfuscator* component uses a background script to keep monitoring the URLs visited by the user and estimating the arrival rate of YouTube videos watched by user as λ^u. Then, given obfuscation budget α, the *obfuscator* component will use a Poisson Process with rate $\lambda^o = \frac{\lambda^u \alpha}{1-\alpha}$ to inject randomly select obfuscation videos. To mimic a typical user who watches one video at a time, the selected obfuscation videos can be played only when the user is not already using YouTube. However, if a user continues to watch YouTube videos for an extended time period, we can simultaneously play the selected obfuscation videos (in the background as explained above) to prevent YouTube from getting unfettered user watch history.[4]

[4] It is not entirely uncommon for YouTube users to play videos in multiple browser tabs.

Denoiser. The *denoiser* has two modules: HTML modification and the denoising. The HTML modification module is implemented in the background script. Whenever the user visits YouTube homepage, the HTML modification module sends the “noisy" homepage recommendation video list requested from the content script to the denoising module. Once HTML modification module receives the “clean" homepage recommended video list from the denoising module, it will modify the HTML of YouTube homepage to show “clean" homepage recommended videos. The denoising module is implemented in the back-end, which is responsible for accessing the metadata of the received “noisy” homepage recommended videos, running the *denoiser* model to convert the “noisy” homepage recommended video list into a “clean" one, and then sends the “clean” video list back to the HTML modification module. We evaluate the implementation overhead of the *obfuscator* and *denoiser* components in Section 6.4.

5 EXPERIMENTAL SETUP

5.1 User Personas

To train and evaluate De-Harpo, we need to construct realistic user personas. However, it is challenging to have access to real-world YouTube users’ video watch history in a large scale as our training data. To address this concern, we design two approaches that can generate a large number of synthetic user personas to simulate real-world users: 1) the first approach creates sock puppet based personas by following the “up next” videos recommended by YouTube; 2) the second approach leverages the YouTube videos publicly posted by Reddit users as an approximation of their YouTube user personas. We use these synthetic user persona datasets to train De-Harpo. Then, we evaluate it on both synthetic user persona datasets and a real user persona dataset that contains YouTube video watch history collected from real-world users. We describe these three datasets in detail below.

Sock Puppet Based Personas. According to YouTube, about 70% of the videos viewed on the platform are sourced from its recommendation system [57]. Accordingly, given the current video, the “up next” videos recommended by YouTube are good representations of the potential subsequent videos watched by real-world YouTube users. Based on this insight, we build a sock puppet user persona model that generates random *recommendation trails* from a single *seed video* to model realistic YouTube user personas, by keeping playing one of the “up next” videos recommended by YouTube randomly with uniform probability (see Appendix C.1 for details). Since these personas are synthetically built, we are able to exercise more control over the distribution of watched videos.

In total, we generate 10,000 sock puppet based personas with 40 videos each. Note that we set the length of each user persona as 40, since we empirically observe that 40 videos can trigger enough personalized recommended videos on the YouTube homepage and the average time it takes to watch them is close to the average daily time spent by each YouTube user (35 min) [22].

Reddit User Personas. As a second way of simulating real-world user personas in a large scale, we gather YouTube links publicly posted by social media users as an approximation of their YouTube personas. While there are various social media platforms where users can share YouTube videos, we choose to collect data from Reddit, since it is one of the largest and most popular communities where users post links related to their interests, and millions of Reddit’s user submissions[5] are publicly available.

Specifically, we download Reddit user submissions from 2017 to 2021 using APIs provided by `pushshift.io` [60]. For each user submission, we first extract the username and all YouTube links posted by this Reddit account. Next, we filter out any duplicate or broken links. Then, we extract the YouTube video ids from these remaining links in order. Finally, we remove users with less than 40 YouTube video posts, since a small number of videos is not a fair approximation of the user’s actual YouTube persona. In total, we collect 10,000 Reddit user personas with length 40.

Real-world YouTube Users. To conduct a more realistic evaluation of De-Harpo, we use a real-world user dataset from [7]. This dataset contains the web browsing histories of 936 real users collected through Web Historian [29] for three months. It is a good representative of real YouTube users, since: 1) the demographic distribution of these users, including their gender, age (18-65+), and education level (from less than high school to Doctoral degree), are relatively uniform; 2) on average 650 YouTube video URLs are watched by each user in three months; 3) the first 40 videos watched by these users have different video class distribution, indicating diverse user interests. Considering that the dataset is collected over a long period, we select the first 40 YouTube videos watched by each of these 936 users as our real user personas, to evaluate De-Harpo.

5.2 Data Collection

User Persona Construction. We use a fresh Firefox browser based on Selenium to construct each user persona. For each sock puppet based persona, we start with a seed video and then follow the “up next" video recommendations to generate a *recommendation trail*. We play each video in a user persona for 30 seconds before playing the next video. Note that we clear any pop-up windows and skip the ads before playing the video. For each Reddit user and real user persona, since we already known the video ids in each persona, we visit these videos sequentially[6]. Similar to constructing synthetic user personas, if there are any pop-up windows or ads, we clear them and then play the video for 30 seconds.

Recommended Video Collection. After we complete the construction of each user persona, we go back to the YouTube homepage and refresh it for 50 times to collect all the recommended videos into a list. Note that we refresh the homepage multiple times since we want to collect enough homepage recommended videos to estimate the recommended video class distribution. We choose the number of refresh times as 50 since we empirically observe that it is a good tradeoff between collecting enough samples and minimizing the quantity of crawls to be performed. Because extremely popular videos are common across many users regardless of their profile, we remove them to underscore personalized recommendations. With this in mind, we filter out videos which appear in more than 1% of personas’ homepage recommended video lists. We also exclude YouTube videos showing in the homepage of a fresh browser. Then,

[5] A Reddit user submission is a json file storing metadata of a Reddit user’s posts, including the username, the timestamp, the URL of post, the text, etc.

[6] Note that directly visiting the URL of each video doesn’t trigger cookies from YouTube and hence no personal recommendation can happen. To address this, we first search the video id at YouTube and then click the first search result.

for each recommended video, we extract the associated tags (i.e. a list of keywords) from its metadata, and map each of them into one of the 154 topic-level video classes we have (note that a video may belong to multiple video classes). Last, for each persona, we count the number of recommended videos in each class and divide it by the sum of videos in all classes to derive the recommended video class distribution of each persona.

5.3 Training and Testing

We discuss details about how we prepare the training and testing datasets, and use them to train and test surrogate model, *obfuscator* and *denoiser* in Appendix C.2-D.

5.4 Baselines

Obfuscator. We compare the privacy-enhancing performance of De-Harpo *obfuscator* with three baselines:

1) Rand-Obf: At each obfuscation step, we randomly select one obfuscation video from the obfuscation video set, and the probability of selecting each obfuscation video is equal to $\frac{1}{M}$ (M is the total number of obfuscation videos in the set).

2) Bias-Obf: At each obfuscation step, we randomly select one obfuscation video from the obfuscation video set. However, the probability of selecting each obfuscation video is proportional to the reward triggered by each obfuscation video. To create such non-uniform distribution, we first use Rand-Obf to randomly select obfuscation videos and then record the reward after injecting them into non-obfuscated user personas. We repeat this experiment for 50 epochs and count the accumulative reward of each obfuscation video, normalize it by the sum of the accumulative rewards of all obfuscation videos, and use the normalized rewards as the non-uniform probability distribution.

3) PBooster-Obf: At each obfuscation step, we select one obfuscation video from the obfuscation video set which can maximize the reward for the current step based on the greedy algorithm PBooster proposed in [4].

Denoiser. We compare the utility-preserving performance of the *denoiser* in De-Harpo with a baseline that uses the same architecture as the surrogate model to predict C^u directly from a non-obfuscated user persona V^u, without taking the obfuscated persona V^o and the associated recommended video class distribution C^o as inputs. We refer to this baseline as *Surro-Den.* Ideally, if the surrogate model is a perfect replication of YouTube's recommendation system, then users could directly use it to get recommended videos based on their non-obfuscated user personas. Clearly this is unrealistic in practice since the surrogate model does not have access to the complete universe of YouTube videos which are updated constantly, and the model is merely an approximation of the actual YouTube recommendation system.

For convenience, we denote the De-Harpo *obfuscator* and De-Harpo *denoiser* by De-Harpo-Obf and De-Harpo-Den respectively in the rest of the paper.

6 EVALUATION

In this section, we evaluate the effectiveness of De-Harpo from six perspectives: privacy, utility, overhead, stealthiness, robustness to de-obfuscation, and personalization.

Table 1: Privacy evaluation results against YouTube w.r.t. P and P^{Norm}.

Obfuscator	Rand-Obf	Bias-Obf	PBooster-Obf	**De-Harpo-Obf**
P	0.71	0.70	0.81	**0.91**
P^{Norm}	21.55%	20.76%	31.24%	**41.63%**

(a) Using sock puppet based personas (D^{Min} : 0.49, D^{Max} : 1.51).

Obfuscator	Rand-Obf	Bias-Obf	PBooster-Obf	**De-Harpo-Obf**
P	1.05	1.07	1.13	**1.30**
P^{Norm}	48.79%	50.99%	57.84%	**76.49%**

(b) Using Reddit user personas (D^{Min} : 0.60, D^{Max} : 1.51).

Obfuscator	Rand-Obf	Bias-Obf	PBooster-Obf	**De-Harpo-Obf**
P	0.98	1.00	1.05	**1.39**
P^{Norm}	45.45%	48.01%	55.34%	**87.23%**

(c) Using real-world user personas (D^{Min} : 0.53, D^{Max} : 1.51).

6.1 Privacy

We first evaluate the effectiveness of De-Harpo in enhancing privacy using three user persona datasets, and report the results in TABLE 1. Note that we test De-Harpo-Obf and other obfuscator baselines against the real-world YouTube recommendation system.

As shown in TABLE 1a, De-Harpo-Obf can trigger 0.91 KL divergence in the recommended video class distribution after obfuscation (P) on sock puppet based personas, which translates into triggering 41.63% of the maximum possible KL divergence in the recommended video class distribution (P^{Norm}). Compared with other baselines, De-Harpo-Obf can increase P^{Norm} by up to 2.01× and at least 1.33×. Similarly, on Reddit user personas, De-Harpo-Obf outperforms all baselines by up to 1.57× and at least 1.32×, as reported in TABLE 1b.

Moreover, we evaluate whether the effectiveness of De-Harpo in enhancing privacy can be transferred to real-world user personas. Specifically, we use the same obfuscator trained on sock puppet based personas to inject obfuscated videos into real-world user's video watch history, and then test it against YouTube. As reported in TABLE 1c, De-Harpo-Obf can trigger 87.23% of the maximum possible KL divergence in the recommended video class distribution (P^{Norm}), which outperforms all baselines against YouTube by up to 1.92× and at least 1.58× in terms of P^{Norm}.

6.2 Utility

Next, we evaluate the effectiveness of De-Harpo in preserving user utility. TABLE 2a reports our evaluation results in terms of U_{Loss} and U^{Norm}_{Gain} using sock puppet based personas. Compared with Surro-Den, De-Harpo-Den achieves on average 26% better performance in terms of decreasing U_{Loss} (i.e. increasing U^{Norm}_{Gain}). Recall that different from Surro-Den, De-Harpo-Den also takes as inputs the obfuscated user persona V^o, and the associated recommended video class distribution C^o which comes directly from the actual YouTube system. In contrast, the surrogate model is merely a "first-order" model of the actual, quite complex YouTube system. We also evaluate the effectiveness of De-Harpo-Den in preserving user utility using both Reddit user personas and real-world users. As reported in TABLE 2b-2c, De-Harpo-Den can consistently preserve the utility well, reducing the utility loss by 93.80% and 90.40% respectively.

It is worth noting that the effectiveness of the denoiser in preserving utility does not depend on the effectiveness of the obfuscator in

Table 2: Utility evaluation results w.r.t. U_{Loss} and U_{Gain}^{Norm}. Note that each cell in the table reports U_{Loss}/U_{Gain}^{Norm}.

Denoiser Obfuscator	Surro-Den	**De-Harpo-Den**
Rand-Obf	0.60 / 50.91%	0.54 / 79.09%
Bias-Obf	0.60 / 49.06%	0.53 / 82.08%
PBooster-Obf	0.60 / 66.14%	0.53 / 86.83%
De-Harpo-Obf	0.60 / 74.59%	**0.53 / 90.35%**

(a) Using sock puppet based personas (D^{Min} : 0.49).

Denoiser Obfuscator	Surro-Den	**De-Harpo-Den**
Rand-Obf	0.68 / 83.26%	0.64 / 91.18%
Bias-Obf	0.68 / 83.98%	0.66 / 88.96%
PBooster-Obf	0.68 / 85.88%	0.65 / 90.46%
De-Harpo-Obf	0.68 / 89.32%	**0.65 / 93.80%**

(b) Using Reddit user personas (D^{Min} : 0.6).

Denoiser Obfuscator	Surro-Den	**De-Harpo-Den**
Rand-Obf	0.66 / 70.79%	0.62 / 81.12%
Bias-Obf	0.66 / 72.34%	0.61 / 82.34%
PBooster-Obf	0.66 / 76.99%	0.61 / 85.31%
De-Harpo-Obf	0.66 / 84.78%	**0.61 / 90.40%**

(c) Using real-world user personas (D^{Min} : 0.53).

enhancing privacy. As shown in Table 2a-2c, the same denoiser can achieve almost the same utility loss U_{Loss} under different obfuscators, which implies the denoiser does not need to sacrifice privacy in order to preserve utility. We discuss the privacy-utility tradeoff in the next subsection.

6.3 Varying the Obfuscation Budget

So far, the obfuscation budget α is set to 0.2 in our evaluation. To evaluate how the obfuscation budget (i.e. the percentile of obfuscation videos in a user persona) can affect the performance of De-Harpo, we increase the value of α and evaluate how the performance of De-Harpo changes w.r.t. both privacy (P^{Norm}) and utility (U_{Loss}). We use sock puppet based persona dataset and consider three baselines: Rand-Obf/De-Harpo-Den (i.e. the combination of Rand-Obf and the De-Harpo *denoiser*), Bias-Obf/De-Harpo-Den (i.e. the combination of Bias-Obf and the De-Harpo *denoiser*), and PBooster-Obf/De-Harpo-Den (i.e. the combination of PBooster-Obf and the De-Harpo *denoiser*).

Privacy-utility tradeoff. Figure 5 shows the privacy-utility tradeoff between P^{Norm} and U_{Loss} with varying α from $\{0.2, 0.3, 0.5\}$, where the top left region corresponds to both high privacy and utility. We observe that, with De-Harpo-Den, the utility loss caused by different obfuscators can be significantly reduced without sacrificing privacy. Note that since our denoiser is designed to work after obfuscation, it does not hurt the performance of the obfuscator. Moreover, with De-Harpo-Den, the utility loss remains almost the same as we keep increasing the obfuscation budget to get higher privacy. For example, compared with baselines without using De-Harpo-Den, De-Harpo can reduce the utility loss by 2.12× when $\alpha = 0.5$. Note that without De-Harpo-Den, the obfuscator needs to sacrifice utility (higher utility loss) to achieve higher privacy. This is a key difference between De-Harpo and prior works that consider the privacy-utility tradeoff (see Section 8).

Obfuscation budget and privacy level. Recall that we use the recommended video class distribution as a proxy to a user profile, see Section 3.3. To evaluate whether De-Harpo can privatize a user profile to look almost random, we increase the obfuscation budget beyond 0.5 aiming to achieve a P^{Norm} value as close to 100% as possible. As shown in Figure 6, for α equal to 0.7 (i.e. 70% of the videos in a user persona are obfuscation videos), P^{Norm} will reach 92.95%, which means the on-average (averaged over all users) divergence between the recommended video class distribution before and after obfuscation is 93% of the on-average divergence between the recommended video class distribution of two random users. It is also worth noting that for real-world user personas, P^{Norm} can get very close to 100% with α merely equal to 0.5. Hence, we conclude that De-Harpo can achieve meaningful privacy for practical obfuscation budgets α.

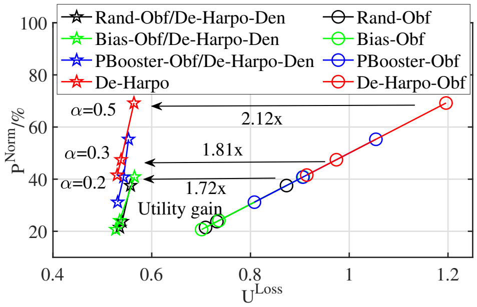


Figure 5: Privacy-utility tradeoff w.r.t. P^{Norm} and U_{Loss} under different obfuscation budget α. Note that Rand-Obf/De-Harpo-Den represents the combination of Rand-Obf obfuscator and the De-Harpo *denoiser*, Bias-Obf/De-Harpo-Den represents the combination of Bias-Obf obfuscator and the De-Harpo *denoiser*, and PBooster-Obf/De-Harpo-Den represents the combination of PBooster-Obf obfuscator and the De-Harpo *denoiser*. Top left of figure represent both high privacy and high utility.

6.4 Overhead

Obfuscation budget and overhead. The larger the obfuscation budget the larger the overhead as more obfuscation videos need to be injected in the video watch history. Not surprisingly, as shown in Figure 5, with increasing obfuscation budget α, the privacy (P^{Norm}) will increase for all obfuscators. That said, De-Harpo can increase privacy with less obfuscation budget than the rest. Specifically, with $\alpha = 0.2$, De-Harpo can achieve the same level of privacy as other baselines achieve with $\alpha = 0.5$. That is, De-Harpo can be as effective as baseline obfuscator in terms of enhancing privacy with 2.5× less obfuscation budget.

System overhead. We evaluate the system overhead of De-Harpo in terms of CPU and memory usage and the video page load time using a an Intel i7 workstation with 64GB RAM on a campus WiFi network. We report that for the *obfuscator* component, the increased CPU and memory usage are less than 5% and 2% respectively, and the increased video page load time is less than 2% even when $\alpha = 0.5$. For the *denoiser* component, the increased CPU and memory usage are less than 28% and 3% respectively, and the YouTube's homepage load time is only increased by 38 millisecond. Overall, we conclude that De-Harpo has a negligible impact on the user experience. (See Appendix E for more detailed analysis).

Table 3: Stealthiness evaluation results under different obfuscation budget α with 5% DeHarpo users. Note that we choose α from $\{0.2, 0.3, 0.5\}$ and report (Precision, Recall) of the adversarial detector for different obfuscators.

Obfuscator	(Precision, Recall)		
	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.5$
Rand-Obf	(4%, 99%)	(5%, 92%)	(5%, 92%)
Bias-Obf	(7%, 72%)	(5%, 81%)	(36%, 94%)
PBooster-Obf	(19%, 86%)	(16%, 93%)	(49%, 88%)
De-Harpo-Obf	(67%, 98%)	(73%, 99%)	(74%, 99%)

Table 4: De-obfuscation robustness evaluation results under different obfuscation budget. Note that we set $\alpha \in \{0.2, 0.3, 0.5\}$ and report (Precision, Recall) of adversarial detector under different obfuscation approaches.

Obfuscator	(Precision, Recall)		
	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.5$
Rand-Obf	(62%, 97%)	(67%, 91%)	(69%, 99%)
Bias-Obf	(67%, 89%)	(71%, 89%)	(77%, 92%)
PBooster-Obf	(68%, 93%)	(71%, 90%)	(77%, 94%)
De-Harpo-Obf	(79%, 93%)	(83%, 97%)	(84%, 97%)

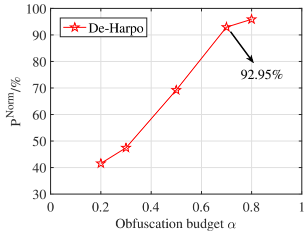


Figure 6: Privacy level P^{Norm} vs obfuscation budget α.

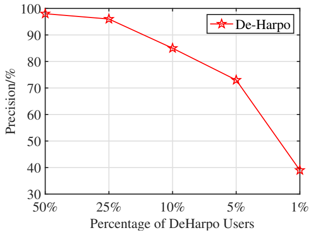


Figure 7: Precision of the adversarial detector vs the percentage of De-Harpo users under $\alpha = 0.5$.

6.5 Stealthiness

In this subsection, we evaluate whether an adversary can train an ML classifier to accurately detect the usage of obfuscators. We use the precision and recall of this adversarial detector to measure stealthiness of obfuscation. If the detector achieves high precision and recall, then it means that an obfuscator is less stealthy. Specifically, the input of the adversary is a user persona consisting of a sequence of videos and the binary output indicates whether or not the user persona contains at least one obfuscation video.

We train the adversarial classifier via supervised learning. To create the labeled dataset, we use the same set of non-obfuscated and obfuscated sock puppet based user personas used for evaluation in Section 6.4 as inputs, and assign the corresponding labels to the personas (0: non-obfuscated, 1: obfuscated). For each obfuscator and obfuscation budget α, we get a balanced training dataset with 1,440 obfuscated personas and the corresponding 1,440 non-obfuscated personas. After training, we use an unbalanced dataset with 5% obfuscation personas (a total of 360 obfuscated personas and 6,840 non-obfuscated personas) to test the detector, since only a small fraction of YouTube users are expected to employ De-Harpo.

Table 3 reports the testing precision and recall of the adversarial detector under different α values. We observe that as α increases, both the precision and recall of the detector also increase. This is expected as larger α represents more obfuscation videos, which makes it easier for the adversarial detector to distinguish obfuscated personas from non-obfuscated personas.

Not surprisingly, Rand-Obf is the most stealthy obfuscator since it injects obfuscation videos randomly. De-Harpo-Obf, which injects obfuscation videos that introduce new user interests to confuse YouTube, can still achieve reasonable stealthiness even when $\alpha = 0.5$. Specifically, it leads to 74% precision (36% false positive rate) even with $\alpha = 0.5$. Note the the high false positive rate presents a major obstacle in deployment of the adversarial detector due to base-rate fallacy [2]. We further vary the percentage of De-Harpo users over all YouTube users to show how the precision of the adversarial detector changes as we go from a very unbalanced dataset to a perfectly balanced one. As shown in Figure 7, as the percentage of De-Harpo users varies from 1% to 50%, the adversarial detector's precision will increase, as expected. However, it is unlikely in practice that a large fraction of YouTube users will use obfuscation measures. And, even in the case of a balanced dataset, a 2% false positive rate still corresponds to tens of millions of users making it prohibitively expensive to deploy it. Essentially, the adversarial detector will have to achieve exceptionally high precision to be useful in practice.

Note that such a binary detector may be used as a first step of the detection workflow. Once the adversary detects the usage of De-Harpo, it may further attempt to de-obfuscate the obfuscated user personas. That is, the adversary may attempt to identify obfuscation videos in the obfuscated user persona such that it may remove them to retrieve the non-obfuscated user personas. We evaluate this de-obfuscation performance of an adversary next.

6.6 De-obfuscation Robustness

Once the adversary detects the usage of De-Harpo in a user persona, it can conduct de-obfuscation. To evaluate whether an obfuscator is robust to de-obfuscation, we train a second adversarial detector to distinguish the obfuscation videos from the actual user videos. Specifically, we build a second ML classifier to detect the type of each video (user versus obfuscation video) in each sock puppet based user persona, and use its precision and recall to measure the de-obfuscation robustness. Smaller precision and recall represents higher de-obfuscation robustness.

We use the same set of obfuscated personas as in Section 6.4 as inputs. For each video in an obfuscated user persona, we assign a binary label, where 0 represents it is watched by the user while 1 represents that it is injected by the obfuscator. The detector model takes as input the obfuscated user persona, and predicts a label for each video in the user persona. We use a recurrent neural network (LSTM layer) to model this adversarial detector.

As shown in Table 4, the precision of this adversarial detector is lower than 85%, which means more than 15% of the obfuscated videos identified by the adversary are false positives (they are actual user videos). Similar to stealthiness, false positives present a bigger challenge to the adversary in deploying this detector in practice. Hence, we conclude that De-Harpo is robust to de-obfuscation by an adversary.

Note that while the adversary has lower precision against Rand-Obf and Bias-Obf than against De-Harpo, this is because De-Harpo is 2.5× more effective in preserving privacy (see Section 6.4), thus, overall, it is more privacy-preserving.

Table 5: Personalization results. $D_{KL}^{NonSens}$ and D_{KL}^{Sens} denote the divergence in *non-sensitive* classes and *sensitive* classes respectively.

	D_{KL}^{NonSen}	D_{KL}^{Sen}
De-Harpo	1.18	0.26
Personalized De-Harpo	0.81 (↓ 31.36%)	0.05 (↓ 80.77%)

6.7 Personalization

De-Harpo so far is trained to maximize the KL divergence in the recommended video class distribution after obfuscation, by either increasing or reducing the probability of each video class. However, a YouTube user may have a list of *sensitive* video classes (e.g. health or wellness related), where they do not want the YouTube recommendations containing videos from these classes after obfuscation (i.e. reducing their probability to zero).

Motivated by this, we design a mechanism that can treat *sensitive* video classes and *non-sensitive* video classes differently based on user preferences. Without loss of generality, suppose the first L classes of the recommended video class distribution are *non-sensitive* and the remaining $K - L$ classes are *sensitive*. We then train De-Harpo to maximize the following privacy metric, which aims to treat non-sensitive classes like before (maximize divergence before and after obfuscation) and eliminate sensitive class videos:

$$P^{Personalized} = E[\underbrace{D_{KL}(C^o_{1:L}, C^u_{1:L})}_{D_{KL}^{NonSens}} - \lambda \underbrace{D_{KL}(C^o_{L+1:K}, [\epsilon]_{L+1:K})}_{D_{KL}^{Sens}}], \quad (10)$$

where $[\epsilon]_{L+1:K} \in \mathbb{R}^{K-L}$ indicates a close-to-zero vector filled with a small positive number ϵ (e.g. 0.0001), and $\lambda > 0$ is an adjustable parameter for controlling the relative importance of $D_{KL}^{NonSens}$ versus D_{KL}^{Sens}. Specifically, the term $D_{KL}^{NonSens}$ aims to maximize the distance between the distribution of *non-sensitive* classes before and after obfuscation, like we did before for all classes. The term $-\lambda D_{KL}^{Sens}$ aims to minimize the distance between the distribution of the sensitive classes and a distribution of very small probabilities.[7]

Table 5 reports our evaluation results of personalized De-Harpo against surrogate models, where we select 27 out of 154 video classes related to Beauty & Wellness and Sports & Fitness as *sensitive* classes. Compared with non-personalized De-Harpo, personalized De-Harpo can reduce the divergence between *sensitive* video class distribution and a zero vector (D_{KL}^{Sens}) by more than 80%, while still triggering high divergence in *non-sensitive* class distribution ($D_{KL}^{NonSens}$).

7 DISCUSSION

7.1 Ethical Considerations

We outline the potential benefits and harms to the user and the recommendation system. We argue that the potential benefits of De-Harpo outweigh its potential harms.

Users. De-Harpo provides a clear privacy benefit to its users, especially when platforms such as YouTube do not provide any meaningful control over its tracking and profiling of users. Crucially, De-Harpo is able to enhance privacy while mostly preserving the utility of personalized recommendations. Thus, De-Harpo does not degrade user experience on YouTube. However, users of De-Harpo potentially violate YouTube's Terms of Service (TOS) [71] because YouTube might interpret obfuscation as "fake engagement". Therefore, if a user is signed-in to YouTube, their YouTube account might be suspended if YouTube is able to detect De-Harpo's usage (though we showed that YouTube would be unable to do so without risking significant collateral damage). More seriously, the violation of TOS might be considered possible violation of the Computer Fraud and Abuse Act (CFAA, 18 U.S. Code § 1030) [1]. However, given that De-Harpo users only watch videos that they are authorized to (i.e., publicly available videos), we argue that the videos injected to the watch history for obfuscation nor the videos injected to the recommendations for repopulation exceed authorized access that could be a violation of CFAA [42].

YouTube. Since De-Harpo aims to preserve utility of recommendations to YouTube users, we argue that it will not directly hurt user engagement on YouTube. De-Harpo's *obfuscator* and *denoiser* would, however, contribute to additional traffic to YouTube servers and may have some indirect impact on the effectiveness of YouTube's recommendation system, if a large enough portion of the users adopt De-Harpo. We note that De-Harpo can be applied with satisfactory trade-off privacy vs. utility as long as only a minority of YouTube users employ obfuscation tools, which is arguably a realistic expectation. Otherwise, if a significant fraction of users adopts De-Harpo, the obfuscation may lead to data poisoning, which will indirectly affect the quality of recommendations for all users. In this case, and in the absence of legal regulation of tracking and user profiling by YouTube, future research will need to explore an alternative scalable solution for privacy preservation that is complementary to obfuscation. Overall, as compared to extant privacy-enhancing obfuscation tools, we conclude that De-Harpo is more favorable since it specifically aims to preserve utility and user engagement on YouTube.

7.2 Limitations & Future Work

Side channels. De-Harpo's stealthiness can be undermined by exploiting various implementation side channels. For example, YouTube could use Page Visibility API [47] or the Performance API [6] to detect whether obfuscation videos are unusually not being played in the foreground. However, there are patches such as wpears [67] to avoid detection. Additionally, the *obfuscator* plays the obfuscation videos in full in a background tab while disabling background throttling (or other such optimizations [6, 61]) to prevent detection by such side channels. As another example, the repopulation of recommendations on the homepage after denoising would entail manipulation of the HTML DOM [16], which might be detectable. However, such an attach would be infeasible in practice, because the detection approaches would add an overhead of up to several seconds [34, 59].

Deployment on mobile devices. De-Harpo is currently implemented as a browser extension for desktops. Since browser extensions are not supported on iOS or Android, the only option for users

[7] Notice that we are somewhat abusing the "distribution" term above, because we do not re-normalize the corresponding probabilities to sum up to 1, as this would (i) de-emphasize the contrast between the patterns of interest and (ii) is not required to meaningfully use the KL divergence formula.

to benefit from De-Harpo on their mobile phone is to use other Chromium based browsers that allow extensions [37, 69]. Another option for mobile users is to use a remote desktop utility [15] to access YouTube with De-Harpo on a desktop. Finally, users might still be able to reap the obfuscation benefits of De-Harpo if they deploy the extension on their desktop and be logged-in to the same Google account [23] on both their mobile app and desktop with De-Harpo.

8 RELATED WORK

In this section, we discuss prior work on privacy-enhancing obfuscation in recommendation systems.

One line of prior research focuses on developing privacy-enhancing obfuscation approaches in online behavioral advertising. These efforts are relevant to our work because online behavioral advertising is essentially a recommendation system where the advertiser's goal is to "recommend" personalized ads to users based on their online activity. However, most of these privacy-enhancing obfuscation approaches are not designed to preserve the utility (i.e., relevance of personalized ads) [13, 30, 32, 36, 44], as they generally randomly insert a curated set of obfuscation inputs to manipulate online behavioral advertising.

TrackThis [32] by Mozilla injects a curated list of 100 URLs to obfuscate a user's browsing profile. AdNauseam [30] clicks a random set of ads to "confuse" advertisers. One subset of these efforts propose "pollution attacks" against online behavioral advertising that also serve a dual role as privacy-enhancing obfuscation [13, 36, 44]. Meng et al. [44] propose a pollution attack that can be launched by publishers to increase their advertising revenue by manipulating advertisers into targeting higher paying ads. The attack involves the addition of curated URLs into a user's browsing profile. Degeling et al. [13] and Kim et al. [36] propose similar attacks but focus on two distinct stages of the online behavioral advertising pipeline: user profiling and ad targeting. Degeling et al. [13] propose an obfuscation approach that involves adding URLs posted on Reddit into a user's browsing profile. Kim et al. [36] propose "AdbudgetKiller" that involves adding a sequence of URLs into a user's browsing profile to trigger retargeted ads, which are costly for advertisers and waste their advertising budget.

Moving beyond online behavioral advertising, Xing et al [68] propose pollution attacks against more general personalized recommendation systems such as YouTube, Amazon, and Google Search. The authors show that personalized recommendations could easily be manipulated by injecting random or curated obfuscation inputs. Since the attack's victim is the user, the work does not take into account the utility of recommendations to the user. In contrast, De-Harpo is a privacy-enhancing obfuscation system that also takes into account the utility of the recommendations.

Follow up privacy-enhancing obfuscation systems do attempt to take into account the utility-privacy trade-off. Beigi et al, [4] propose PBooster, a greedy search approach to obfuscate a user's browsing profile while also keeping utility in consideration. PBooster employs topic modeling to select a subset of target topics and corresponding obfuscation URLs to add in a user's browsing history. Zhang et al. [74] propose Harpo, a reinforcement learning approach to obfuscate a user's browsing profile such that a subset of interest segments are kept while others are modified. Different from Harpo, De-Harpo pairs the obfuscator with a denoiser to preserve the recommended videos related to the users' actual interests while removing the unrelated recommended videos caused by obfuscation. Moreover, the De-Harpo obfuscator non-trivially adapts Harpo to YouTube, by building 1) a surrogate model with a different embedding model and loss function for replicating the YouTube recommendation system and 2) an obfuscator model which selects obfuscation videos based on the similarity between its embedding and the output embedding, such that it can handle an unlimited and varying number of possible obfuscation videos without requiring retraining. Huang et al. [31] propose a context-aware generative adversarial privacy (GAP) approach to train a "privatizer" for privacy-enhancing obfuscation against an adversary who attempts to infer sensitive information from input data. This approach is used to obfuscate mobile sensor data while navigating the privacy-utility tradeoff [38, 43, 53]. While in theory we can apply GAP to jointly train the obfuscator and denoiser, in practice training them against YouTube in the wild which is prohibitively time consuming due to the iterative nature of GAP, and training them against the surrogate model is ineffective because the denoiser is able to trivially replicate the surrogate model (see Appendix F). Beiga et al. [5] propose a crowd-based obfuscation approach that allows individual users to preserve privacy by scrambling their browsing profiles via mediator accounts, which are selected such that the personalized recommendations to these mediator accounts are still coherent and utility-preserving to the users behind each mediator account. However, this approach requires a collaboration across multiple users of a recommendation system, and cannot be used by standalone users.

While recent work on privacy-enhancing obfuscation has attempted to balance the privacy-utility tradeoff, they are limited to obfuscating the input to the recommendation system to achieve this balance. These approaches are fundamentally limited as to how much utility can be preserved without undermining privacy by obfuscating the input to the recommendation system (see Fig. 9). In contrast, De-Harpo employs a two-step approach to this end. It first obfuscates the input to the recommendation system to preserve user privacy and then attempts to de-obfuscate the output recommendations to preserve utility.

9 CONCLUSION

In this paper, we proposed De-Harpo, a privacy-enhancing and utility-preserving obfuscation approach for YouTube's recommendation system that does not rely on cooperation from YouTube.

De-Harpo used an *obfuscator* to inject obfuscation videos into a user's video watching history and a *denoiser* to remove the "noisy" recommended videos thus recovering the initial, unobfuscated recommendations. Our evaluation results demonstrated that De-Harpo can reduce the utility loss by 2× for the same level of privacy compared to existing state-of-the-art obfuscation approaches. Our work provides a template for implementing such utility-preserving obfuscation approaches on other similar online platforms, such as TikTok [58] and Facebook [49]. We will publicly release our code in conjunction with this paper to facilitate follow-up research.

ACKNOWLEDGMENTS

The authors would like to thank Sean Hackett for his help with the discussion of the denoiser idea, Muhammad Haroon for his help with the data collection and browser extension implementation, and Magdalena Wojcieszak for sharing the web browsing histories of real-world users. This work is supported in part by the National Science Foundation under grant numbers 1956435, 1901488, and 2103439.

REFERENCES

[1] 18 US Code § 1030. 2022. Fraud and related activity in connection with computers. https://www.law.cornell.edu/uscode/text/18/1030.

[2] Stefan Axelsson. 2000. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)* 3, 3 (2000), 186–205.

[3] Ricks Becca and McCrosky Jesse. 2022. Does This Button Work? Investigating YouTube's ineffective user controls. https://foundation.mozilla.org/en/research/library/user-controls/report/.

[4] Ghazaleh Beigi, Ruocheng Guo, Alexander Nou, Yanchao Zhang, and Huan Liu. 2019. Protecting user privacy: An approach for untraceable web browsing history and unambiguous user profiles. In *Proceedings of the twelfth ACM international conference on web search and data mining*. 213–221.

[5] Asia J Biega, Rishiraj Saha Roy, and Gerhard Weikum. 2017. Privacy through solidarity: A user-utility-preserving framework to counter profiling. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*. 675–684.

[6] Chromium Blog. 2022. Tab throttling and more performance improvements in Chrome M87. https://blog.chromium.org/2020/11/tab-throttling-and-more-performance.html.

[7] Andreu Casas, Ericka Menchen-Trevino, and Magdalena Wojcieszak. 2022. Exposure to extremely partisan news from the other political side shows scarce boomerang effects. *Political Behavior* (2022), 1–40.

[8] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. 2020. Efficient neural matrix factorization without sampling for recommendation. *ACM Transactions on Information Systems (TOIS)* 38, 2 (2020), 1–28.

[9] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 456–464.

[10] Matthew Clark and Konstantinos Psounis. 2020. Optimizing primary user privacy in spectrum sharing systems. *IEEE/ACM Transactions on Networking* 28, 2 (2020), 533–546.

[11] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[12] Paul Cuff and Lanqing Yu. 2016. Differential privacy as a mutual information constraint. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 43–54.

[13] Martin Degeling and Jan Nierhoff. 2018. Tracking and Tricking a Profiler: Automated Measuring and Influencing of Bluekai's Interest Profiling. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*. 1–13.

[14] Shuiguang Deng, Longtao Huang, Guandong Xu, Xindong Wu, and Zhaohui Wu. 2016. On deep learning for trust-aware recommendations in social networks. *IEEE transactions on neural networks and learning systems* 28, 5 (2016), 1164–1177.

[15] Chrome Remote Desktop. 2022. https://remotedesktop.google.com/access.

[16] MDN Web Docs. 2022. HTML DOM API. https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API.

[17] Cynthia Dwork. 2006. Differential privacy. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33*. Springer, 1–12.

[18] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[19] Ahmed Roushdy Elkordy, Jiang Zhang, Yahya H Ezzeldin, Konstantinos Psounis, and Salman Avestimehr. 2023. How Much Privacy Does Federated Learning with Secure Aggregation Guarantee? *Proceedings on Privacy Enhancing Technologies* (2023), 510–526.

[20] Wenqi Fan, Qing Li, and Min Cheng. 2018. Deep modeling of social relations for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[21] Wenqi Fan, Yao Ma, Dawei Yin, Jianping Wang, Jiliang Tang, and Qing Li. 2019. Deep social collaborative filtering. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 305–313.

[22] Cristos Goodrow. 2021. On YouTube's recommendation system. https://blog.youtube/inside-youtube/on-youtubes-recommendation-system/.

[23] Google Account. 2022. https://www.google.com/account/about.

[24] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. 2018. Outer product-based neural collaborative filtering. *arXiv preprint arXiv:1808.03912* (2018).

[25] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[26] Google Ads Help. 2022. About audience targeting. https://support.google.com/google-ads/answer/2497941?hl=en#zippy=%2Cin-market-segments%2Caffinity-segments.

[27] Google Account Help. 2023. Web & App Activity Controls. https://support.google.com/accounts/answer/54068?hl=en.

[28] YouTube Help. 2019. Manage your recommendations and search results. https://support.google.com/youtube/answer/6342839?hl=en.

[29] Web Historian. 2022. Visualize your web use to understand your habits. https://webhistorian.github.io/.

[30] Daniel C Howe and Helen Nissenbaum. 2017. Engineering Privacy and Protest: A Case Study of AdNauseam.. In *IWPE@ SP*. 57–64.

[31] Chong Huang, Peter Kairouz, Xiao Chen, Lalitha Sankar, and Ram Rajagopal. 2017. Context-aware generative adversarial privacy. *Entropy* 19, 12 (2017), 656.

[32] Liz Hull. 2019. Hey advertisers, track THIS. https://blog.mozilla.org/firefox/hey-advertisers-track-this.

[33] Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*. 135–142.

[34] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the privacy threats of browser extension fingerprinting. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.

[35] Erin Kenneally and David Dittrich. 2012. The menlo report: Ethical principles guiding information and communication technology research. *Available at SSRN 2445102* (2012).

[36] I Luk Kim, Weihang Wang, Yonghwi Kwon, Yunhui Zheng, Yousra Aafer, Weijie Meng, and Xiangyu Zhang. 2018. Adbudgetkiller: Online advertising budget draining attack. In *Proceedings of the 2018 World Wide Web Conference*. 297–307.

[37] KiwiBrowser. 2022. https://kiwibrowser.com.

[38] Sicong Liu, Junzhao Du, Anshumali Shrivastava, and Lin Zhong. 2019. Privacy adversarial network: representation learning for mobile data privacy. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 4 (2019), 1–18.

[39] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. 2008. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 931–940.

[40] Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. 2011. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*. 287–296.

[41] David JC MacKay, David JC Mac Kay, et al. 2003. *Information theory, inference and learning algorithms*. Cambridge university press.

[42] Aaron Mackey and Kurt Opsahl. 2021. Van Buren is a Victory Against Overbroad Interpretations of the CFAA, and Protects Security Researchers. EFF. https://www.eff.org/deeplinks/2021/06/van-buren-victory-against-overbroad-interpretations-cfaa-protects-security.

[43] Mohammad Malekzadeh, Richard G Clegg, Andrea Cavallaro, and Hamed Haddadi. 2019. Mobile sensor data anonymization. In *Proceedings of the international conference on internet of things design and implementation*. 49–58.

[44] Wei Meng, Xinyu Xing, Anmol Sheth, Udi Weinsberg, and Wenke Lee. 2014. Your online interests: Pwned! a pollution attack against targeted advertising. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 129–140.

[45] Andriy Mnih and Russ R Salakhutdinov. 2007. Probabilistic matrix factorization. *Advances in neural information processing systems* 20 (2007), 1257–1264.

[46] Mozilla. 2022. Firefox rolls out Total Cookie Protection by default to all users worldwide . https://blog.mozilla.org/en/mozilla/firefox-rolls-out-total-cookie-protection-by-default-to-all-users-worldwide/.

[47] Mozilla page visibility API. 2022. https://developer.mozilla.org/en-US/docs/Web/API/Page_Visibility_API.

[48] Helen Nissenbaum and Howe Daniel. 2009. TrackMeNot: Resisting surveillance in web search. *Oxford: Oxford University Press* (2009).

[49] O'Flaherty. 2021. The 1 Facebook Setting You Should Change Now. https://www.forbes.com/sites/kateoflahertyuk/2021/11/20/facebook-has-hijacked-your-news-feed-heres-how-to-get-it-back/?sh=4c942aa62e79.

[50] OpenAI. 2022. OpenAI Baselines: ACKTR & A2C. https://openai.com/blog/baselines-acktr-a2c/.

[51] Javier Parra-Arnau, Jagdish Prasad Achara, and Claude Castelluccia. 2017. Myadchoices: Bringing transparency and control to online advertising. *ACM Transactions on the Web (TWEB)* 11, 1 (2017), 1–47.

[52] Javier Parra-Arnau, David Rebollo-Monedero, and Jordi Forné. 2014. Measuring the privacy of user profiles in personalized information systems. *Future Generation Computer Systems* 33 (2014), 53–63.

[53] Nisarg Raval, Ashwin Machanavajjhala, and Jerry Pan. 2019. Olympus: Sensor Privacy through Utility Aware Obfuscation. *Proc. Priv. Enhancing Technol.* 2019, 1 (2019), 5–25.

[54] Ashley Rodriguez. 2018. YouTube's recommendations drive 70% of what we watch. Quartz. https://qz.com/1178125/youtubes-recommendations-drive-70-of-what-we-watch.

[55] SentenceTransformers Documentation. 2022. https://www.sbert.net/.

[56] Hanhuai Shan and Arindam Banerjee. 2010. Generalized probabilistic matrix factorizations for collaborative filtering. In *2010 IEEE International Conference on Data Mining*. IEEE, 1025–1030.

[57] Joan E. Solsman. 2021. YouTube's AI is the puppet master over most of what you watch. https://www.cnet.com/news/youtube-ces-2018-neal-mohan/. (2021).

[58] WSJ Staff. 2021. Inside Tiktok's Highly Secretive Algorithm. https://www.wsj.com/video/series/inside-tiktoks-highly-secretive-algorithm/investigation-how-tiktok-algorithm-figures-out-your-deepest-desires/6C0C2040-FF25-4827-8528-2BD6612E3796.

[59] Oleksii Starov and Nick Nikiforakis. 2017. Xhound: Quantifying the fingerprintability of browser extensions. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 941–956.

[60] Reddit User Submissions. 2022. https://files.pushshift.io/reddit/submissions/.

[61] Mozilla Support. 2018. Non-Active-Tabs. https://support.mozilla.org/si/questions/1228604.

[62] Jiliang Tang, Charu Aggarwal, and Huan Liu. 2016. Recommendations in signed social networks. In *Proceedings of the 25th International Conference on World Wide Web*. 31–40.

[63] Jiliang Tang, Xia Hu, Huiji Gao, and Huan Liu. 2013. Exploiting local and global social context for recommendation.. In *IJCAI*, Vol. 13. Citeseer, 2712–2718.

[64] uBlock Origin. 2022. uBlock-Origin. https://ublockorigin.com/.

[65] Xiang Wang, Xiangnan He, Liqiang Nie, and Tat-Seng Chua. 2017. Item silk road: Recommending items from information domains to social users. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 185–194.

[66] John Wilander. 2020. Full Third-Party Cookie Blocking and More. https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/ .

[67] Wpears. 2018. Don't Make Me Watch, Page Visibility API Blocker.

[68] Xingyu Xing, Wei Meng, Dan Doozan, Alex C Snoeren, Nick Feamster, and Wenke Lee. 2013. Take this personally: Pollution attacks on personalized services. In *22nd {USENIX} Security Symposium ({USENIX} Security 13)*. 671–686.

[69] Yandex. 2022. https://yandex.com.

[70] Bo Yang, Yu Lei, Jiming Liu, and Wenjie Li. 2016. Social collaborative filtering by trust. *IEEE transactions on pattern analysis and machine intelligence* 39, 8 (2016), 1633–1647.

[71] YouTube. 2022. Terms of Service. https://www.youtube.com/static?template=terms.

[72] youtube-dl downloads. 2022. https://youtube-dl.org/.

[73] Jiang Zhang, Lillian Clark, Matthew Clark, Konstantinos Psounis, and Peter Kairouz. 2022. Privacy-utility trades in crowdsourced signal map obfuscation. *Computer Networks* 215 (2022), 109187.

[74] Jiang Zhang, Konstantinos Psounis, Muhammad Haroon, and Zubair Shafiq. 2022. HARPO: Learning to Subvert Online Behavioral Advertising. *NDSS* (2022).

[75] Zhou Zhao, Qifan Yang, Hanqing Lu, Tim Weninger, Deng Cai, Xiaofei He, and Yueting Zhuang. 2017. Social-aware movie recommendation via multimodal network learning. *IEEE Transactions on Multimedia* 20, 2 (2017), 430–440.

A WHY DP CAN NOT BE GUARANTEED

Theorem A.1. *Assume that there is one video, which the obfuscator (O, a randomized function) can not delete from a user persona (P), then we can not achieve ϵ-DP or (ϵ, δ)-DP in terms of protecting the user persona.*

Proof. First, to achieve ϵ-DP, for any two user personas P_1 and P_2 differing from one video, and for any user persona set $\mathcal{P}$ belonging to the output space of the obfuscator, $\frac{Pr(O(P_1) \in \mathcal{P})}{Pr(O(P_2) \in \mathcal{P})} \le e^{\epsilon}$ should be satisfied. Now, assume that that there is one video V which only exists in P_2 but not in P_1, and the obfuscator O can not remove it from P_2 after obfuscation, which means $O(P_2)$ will always contain video V. Then, there exists an user persona set $\mathcal{P}$ which contains user personas without video V, where $Pr(O(P_1) \in \mathcal{P}) = 1$ but $Pr(O(P_2) \in \mathcal{P}) = 0$. Therefore, $\frac{Pr(O(P_1) \in \mathcal{P})}{Pr(O(P_2) \in \mathcal{P})} = +\infty$ and hence ϵ will be infinite in order to bound this worst case.

Second, to achieve (ϵ, δ)-DP, for any two user personas P_1 and P_2 differing from one video, and for any user persona set $\mathcal{P}$ belonging to the output space of the obfuscator, $|Pr(O(P_1) \in \mathcal{P}) - e^{\epsilon} Pr(O(P_2) \in \mathcal{P})| \le \delta$ should be satisfied. Moreover, for δ to be meaningful, it has to be inversely proportional to the size of the dataset, which in our case is enormous (all possible user personas). However, since there exists an user persona set $\mathcal{P}$ without video V, where $Pr(O(P_1) \in \mathcal{P}) = 1$ but $Pr(O(P_2) \in \mathcal{P}) = 0$, the value of δ equals 1, which is meaningless in terms of (ϵ, δ)-DP. □

Theorem A.2. *Assume that there is one interest category, which the obfuscator (O, a randomized function) can not remove from a user profile (i.e. a list of interest categories) created by YouTube (R), then we can not achieve ϵ-DP or (ϵ, δ)-DP in terms of protecting the user profiles.*

Proof. Define the YouTube recommendation system as R. First, to achieve ϵ-DP, for any two user profiles $R(P_1)$ and $R(P_2)$ differing from one interest category, and for any user profile set $\mathcal{R}$ in the output space of recommendation system, $\frac{Pr(R(O(P_1) \in \mathcal{R})}{Pr(R(O(P_2) \in \mathcal{R})} \le e^{\epsilon}$ should be satisfied. Now, assume that there is one interest category I which is only in user profile $R(P_2)$ but not in user profile $R(P_1)$, and the obfuscator O can not remove it from user profile $R(O(P_2))$, which means user profile $R(O(P_2))$ will always contain interest category I. Then, there exists a user profile set $\mathcal{R}$ containing user profiles without interest category I, where $Pr(R(O(P_1)) \in \mathcal{R}) = 1$ but $Pr(R(O(P_2) \in \mathcal{R}) = 0$. Therefore, $\frac{Pr(R(O(P_1) \in \mathcal{R})}{Pr(R(O(P_2) \in \mathcal{R})} = +\infty$ and hence ϵ will be infinite in order to bound this worst case.

Second, to achieve (ϵ, δ)-DP, for any two user profiles $R(P_1)$ and $R(P_2)$ differing from one interest category, and for any user profile $R(P)$ in the output space of recommendation system, $|Pr(R(O(P_1)) = R(P)) - e^{\epsilon} Pr(R(O(P_2)) = R(P))| \le \delta$ should be satisfied. However, since there exists a user profile set $\mathcal{R}$ containing user profiles without interest category I, where $Pr(R(O(P_1) \in \mathcal{R}) = 1$ but $Pr(R(O(P_2) \in \mathcal{R}) = 0$, the value of δ equals 1, which is meaningless in terms of (ϵ, δ)-DP. □

B SYSTEM DESIGN DETAILS

B.1 MDP

The obfuscation video selection process of *obfuscator* can be formulated as a Markov Decision Process (MDP) defined as follows:

1) Obfuscation step: As shown in Figure 8, at the beginning of each time step, a video will be played. If the played video is an obfuscation video injected by the *obfuscator*, we refer to this time step as an obfuscation step. We denote the number of videos that have been played up to obfuscation step t by n_t. Note that we use the obfuscation budget α as a system parameter to control the percentile of obfuscation videos. At each time step, with probability α, an obfuscation video will be injected by *obfuscator* into the user persona.

2) State: We define state $s_t \in \mathcal{S}$ at obfuscation step t as $s_t = [v_1, ..., v_{n_t}]$, where n_t is the total number of videos played until the beginning of obfuscation step t, and $\mathcal{S}$ is the state space of the MDP.

3) Action: At obfuscation step t, an action a_t will be taken by the MDP. We define action $a_t \in \mathcal{A}$ as the obfuscation video selected by the MDP policy, where $\mathcal{A}$ is the action space of the MDP, i.e. the obfuscation videos set in our application.

4) State Transition: We define the state transition function as $\mathcal{T}(\cdot|\mathcal{S},\mathcal{A}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, which outputs the probability of $s_{t+1} = s'$ given $s_t = s$ and $a_t = a$ as $\mathcal{T}(s_{t+1} = s'|s_t = s, a_t = a)$. In our system, state s_{t+1} contains all videos played until state s_t, the action a_t (i.e. the obfuscation videos selected at obfuscation step t), and all the videos played by users between obfuscation step t and obfuscation step $t+1$. Note that the randomness of this MDP comes from the random injection of obfuscation videos.

5) Reward: We associate a reward r_t for the action a_t at obfuscation step t. Specifically, we define r_t as the difference of the privacy metric P (see Eq. (1)) between this obfuscation step and the previous one, i.e., $r_t = P_t - P_{t-1}$, where P_t represents the privacy metric value at obfuscation step t, calculated based on the recommended video class distributions of a non-obfuscated user persona and the corresponding obfuscated user persona at the end of obfuscation step t.

6) Policy: The policy of the MDP can be defined as $\pi(\cdot|\mathcal{S}) : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$, which outputs the probability of $a_t = a$ given $s_t = s$ as $\pi(a_t = a|s_t = s)$. In our system, the *obfuscator* is modeled as the policy in MDP, which outputs the probability distribution of obfuscation video selection. Suppose we have M obfuscation videos in the obfuscation video set ($\mathcal{A}$), then we have $\sum_{i=1}^{i=M} \pi(a_t = i|s_t) = 1$, where $a_t = i$ represents the selection of i-th obfuscation video. At each obfuscation step t, we randomly choose one obfuscation video based on a multinomial distribution parameterized by $A_t = [\pi(a_t = 1|s_t), \cdots, \pi(a_t = M|s_t)]$, conditioning on the current state s_t. The goal of solving this MDP is to find the optimal policy, such that the accumulative rewards $\sum_{t=1}^{t=T} r_t$ can be maximized. Note that T is the total number of obfuscation steps since we consider a finite-horizon MDP.

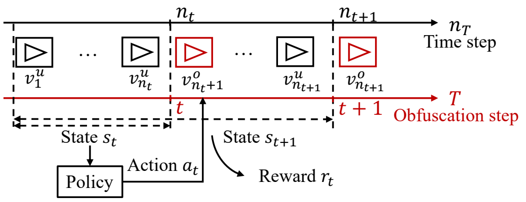
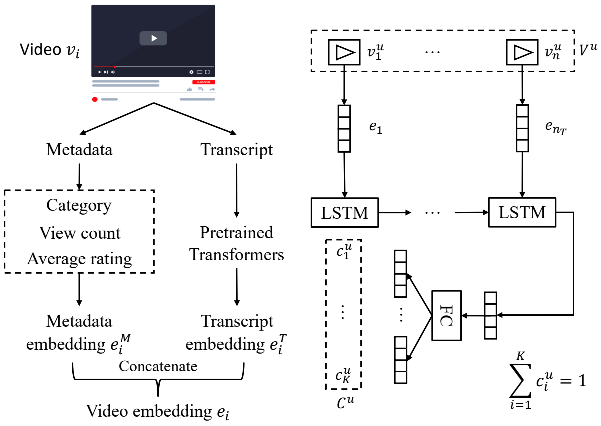


Figure 8: MDP for the *obfuscator*.

Note that the state s_t (i.e. video sequence) will be continuously updated by appending new videos and is only growing unless users manually delete the history. De-Harpo is designed to take the whole state s_t as input of its *obfuscator* to select an obfuscation video, and then run denoising at each step. Hence, the calculation made by De-Harpo at each step will depend on the calculation made by De-Harpo in the previous step, which is consistent with how YouTube works.

Moreover, we clarify that Harpo [74] and De-Harpo use a similar MDP formulation but with a different state (video sequence instead of webpage sequence) and reward function (privacy metric). They apply the same RL algorithm (A2C) to train the obfuscator, though the implementation differs due to MDP differences.



(a) Video embedding **(b) Surrogate model**

Figure 9: Details of system design.

B.2 Video Embedding

As demonstrated in Figure 9a, we start by extracting the category, view count and average rating of each video from its metadata. We then use an one-hot embedding to represent the category of each video (with dimension 18)[8], and use two real numbers to represent the standardized view count and average rating of each video. By combining them together, we derive the metadata embedding with 20 elements. We denote the metadata embedding for video v_i as $e_i^M \in \mathbb{R}^{20}$.

Next, we use a pretrained natural language processing (NLP) Transformer from [55] to generate the transcript embedding for the video transcript. Since the pretrained NLP Transformer has a constraint on the maximal number of words in the input text (256 words in our case), we firstly split video transcript with more than 256 words into multiple transcript chunks, each of which contains 256 words. Then, for each transcript chunk, we use it as input of the NLP model and get the output embedding vector. We take the average of these embedding vectors for these transcript chunks to derive the final transcript embedding. We denote the transcript embedding for video v_i as $e_i^T \in \mathbb{R}^{384}$, which is a real vector with dimension 384. Note that if a video does not contain any transcript (e.g. music videos), we use the video title and description as an alternative of transcript to generate the transcript embedding. Last, we concatenate the metadata and transcript embeddings and derive the complete video embedding vector $e_i = [e_i^M, e_i^T] \in \mathbb{R}^{404}$.

B.3 YouTube Surrogate Model

Prior approaches to learn latent user-item relationships for recommendation systems (e.g., matrix factorization [33, 39, 40, 45, 56, 62, 63, 70], neural MF [8, 14, 20, 21, 24, 25, 65, 75]) are not scalable because they rely on a fixed set of users and items. To address this limitation, recent work has focused on embedding based recommendation systems that predict the next item clicked by users from their item-click history and thus can scale to a large and dynamic set of users and items [9, 11]. YouTube, deals with a large influx of videos

[8] Note that YouTube has 17 video categories, and we add an additional "none" category for videos without category metadata. Hence, the one-hot-embedding for category information has a dimension of 18.

and users everyday [22] and thus uses a scalable recommendation system that predicts the next watched videos based on the embeddings of the past watched videos and other factors [11]. Similar to YouTube's embedding based recommendation architecture, our surrogate model also takes as input the video embeddings. Slightly different from YouTube's embedding based recommendation architecture and as explained in Section 3.2, our surrogate model is designed to predict the recommended video class distribution, instead of making video-level recommendations.

C EXPERIMENTAL SETUP

C.1 Sock Puppet Based Persona Model

Specifically, we denote this model as $G(D, T)$ parameterized by D, the depth of the recommendation trail, and T, the total number of videos in the watch history, and we define the *recommendation trail* as a sequence of videos that are recommended and subsequently watched by a user starting from the given seed video. At each step of the recommendation trail, we randomly select one "up next" video to watch from the list of recommended videos with uniform probability. We repeat this process until the *recommendation trail* reaches the depth D at which point we check if the user has watched T videos. If not, we randomly select another seed video from the user's homepage and repeat the process until T videos have been watched. Note that we randomly select around 20,000 popular videos from a set of popular YouTube channels as our *seed videos*. For each *seed video*, we randomly generate a *recommendation trail* and use it as a synthetic user persona. Note that we collect seed videos used for generating sock puppet randomly from 200 popular YouTube channels, which include videos from all YouTube video categories.

C.2 Data Preparation

Video Embedding Preparation. We use `youtube-dl`, a free software for downloading YouTube videos [72], to collect metadata and transcripts of videos. For metadata, we extract the category, average rating, view count, title, and description of each video, which is then used to generate the metadata embedding of each video. For a transcript, after we download it, we extract the transcript text, split it into text chunks with 256 words each, and use the pretrained Transformer `all-MiniLM-L6-v2` from [55] to convert them into transcript embeddings. As described in Section 4.1, we combine the metadata and transcript embeddings to generate the final video embedding.

User Persona Dataset Collected for Surrogate Model. We construct 10,000 sock puppet based personas and 10,000 Reddit user personas with 40 videos each. For each of these personas, we collect the YouTube homepage recommended videos and derive the recommended video class distribution. We use these constructed personas as inputs (V^u) and the associated recommended video class distributions as labels (C^u) to build the dataset for surrogate model training and testing. As discussed in Section D, we use supervised learning to train the surrogate model.

User Persona Dataset Collected for Obfuscator and Denoiser. To evaluate the effectiveness of the *obfuscator* model against the real-world YouTube recommendation system, we need to construct both non-obfuscated and obfuscated user personas. Specifically, for each *obfuscator* model under an obfuscation budget α, we first construct 2,936 non-obfuscated user personas [9] with 40 videos each and the corresponding 2,936 obfuscated user personas generated by the *obfuscator* with on average $40 * \frac{\alpha}{1-\alpha}$ videos each. Then for each pair of non-obfuscated and obfuscated user persona (V^u and V^o), we collect their associated recommended videos from the YouTube homepage and derive their recommended video class distribution (C^u and C^o).

Moreover, we use the same user persona data collected for the *obfuscator* evaluation to create the dataset for the *denoiser* training and testing (see Section D). Specifically, each input of this dataset consists of one non-obfuscated user personas (V^u), the corresponding obfuscated user persona generated by the *obfuscator* (V^o), and its associated recommended video class distribution (C^o). Each label of this dataset is the recommended video class distribution of the non-obfuscated user persona (C^u).

Obfuscation Video Set. We create our obfuscation video set by combining played videos during persona construction and videos appearing in homepage recommendations of all personas. In total, we collect approximately one million YouTube videos and use them as the obfuscation video set. Note that the *obfuscator* will select one obfuscation video from the obfuscation video set at each obfuscation step.

C.3 Ethical Issues Related to Reddit User and Real-world User Personas

For the Reddit dataset, it is deemed exempt by IRB, and the dataset is publicly available and pre-crawled at https://files.pushshift.io/reddit/. We will de-identify usernames before public data release. For the YouTube users' dataset we obtained an IRB approval and conducted experiments along the Menlo Report guidelines [35]: Users consented to their data being collected for research purposes. We will not publicly release the dataset.

D TRAINING AND TESTING

Surrogate Model. We split the user persona dataset collected for the surrogate model in 80% for training and 20% for testing. We use stochastic gradient descent for the surrogate model to minimize its loss, which is defined as the KL divergence between its output distribution and the actual recommended video category distribution of input user persona. We train our surrogate model for 50 epochs, where all the training samples are used once at each training epoch. We report that the average loss of our surrogate model on the testing dataset is 0.55.

Obfuscator. Recall that the *obfuscator* needs to take as input the non-obfuscated user personas. We use the training and testing user personas in the dataset collected for the surrogate model as the non-obfuscated user personas, and train the *obfuscator* to generate obfuscated user personas that maximize privacy (see Section 3.4). Specifically, we train the *obfuscator* against the surrogate model for 50 epochs, where all the training user personas are used once at

[9] Note that 2,936 non-obfuscated user personas consist of 1,000 sock puppet based personas, 1,000 Reddit user personas, and the 936 real user personas from real-world users.

each epoch. After that, we use the testing user personas to evaluate the *obfuscator* against both the surrogate model and the real-world YouTube recommendation system, and report the average privacy metrics (P and P^{Norm}). Note that to evaluate the performance of the *obfuscator* against YouTube, we construct non-obfuscated and obfuscated user personas to collect real-world data from YouTube (see Section 5.2). Moreover, when training the *obfuscator*, we use the on-policy RL algorithm A2C (Advantage Actor and Critic)[50], which is one of the state-of-the-art on-policy RL algorithms. Note that we choose the on-policy RL algorithm since it fits our application well, where the *obfuscator* (RL agent) needs to keep interacting with the YouTube recommendation system (environment) to improve the policy in an online fashion due to the dynamics of the YouTube recommendation system.

Denoiser. As described in Section 5.2, we create a dataset with 1,800 samples to train and test the *denoiser*, where 80% of the samples are used for training and 20% are used for testing. Specifically, the *denoiser* is trained via stochastic gradient descent to minimize the KL divergence between the output of the *denoiser* $\hat{C}^u$, i.e. the estimated recommendation video category distribution of a non-obfuscated user persona, and the actual distribution C^u. We train the *denoiser* for 50 epochs, where all the training samples are used once at each training epoch. We test the *denoiser* using the remaining 20% samples and report the average utility metrics (U^{Norm}_{Gain} and U^{Loss}).

Note that when we test De-Harpo on sock puppet based persona dataset, we use the models of De-Harpo trained on sock puppets dataset; when we test De-Harpo on Reddit user persona dataset, we use the models of De-Harpo trained on sock puppets dataset; when we test De-Harpo on Real-world YouTube user dataset, we use the models of De-Harpo trained on sock puppets dataset.

E SYSTEM OVERHEAD ANALYSIS

We evaluate the system overhead of De-Harpo in terms of CPU and memory usage and the video page load time using a an Intel i7 workstation with 64GB RAM on a campus WiFi network. As described in Section 4.6, De-Harpo consists of an *obfuscator* component that always runs in the background and a *denoiser* component that only runs when the user visits the YouTube homepage. We separately report their overhead below.

1) Obfuscator: We select an obfuscation budget α from {0.0, 0.2, 0.3, 0.5}, where $\alpha = 0.0$ is used as the baseline (i.e. no obfuscation videos). For each obfuscation budget α we construct 10 user personas with 15 user videos each, and the browser extension visits $15 \cdot \alpha$ obfuscation videos in the background. We find that the increased CPU usage is less than 5% and the increased memory usage is less than 2%, even for obfuscation budget $\alpha = 0.5$. Moreover, the change in video page load time of user videos is less than 2% as α increases. Hence, we conclude that the *obfuscator* component in De-Harpo has a negligible impact on the user experience overall.

2) Denoiser: The YouTube's homepage load time with De-Harpo is 1.79 seconds, which represents just a 37.8 millisecond increase as compared to the homepage load time without De-Harpo. Specifically, it takes less than 24.6 millisecond to get the “noisy" recommended videos from the homepage, 13.0 millisecond for the *denoising* module to get “clean" recommended videos, and 0.2 millisecond for showing these videos in the homepage. In terms of the CPU and memory usage, the *denoiser* of De-Harpo will increase them by 27.1% and 2.2% respectively, which is mainly due to running the ML model in the *denoising* module. Note that the increase of the CPU usage (from 12.9% to 40.0%) lasts for just 13 milliseconds while the ML model runs and returns to the normal level right after that. It is worth noting that the aforementioned measurements are conducted for the *live* version of De-Harpo. In practice, we can reduce the overhead even further by implementing a *cached* version of De-Harpo, which caches the YouTube homepage periodically in the background and simply shows the cached homepage when the user navigates to the YouTube homepage. Hence, we conclude that the *denoiser* component in De-Harpo has a negligible impact on the user experience overall.

F DISCUSSION OF JOINT TRAINING OF OBFUSCATOR AND DENOISER

The *obfuscator* and *denoiser* in De-Harpo are separately trained and their joint training might be much more effective. We experimented with jointly training the *obfuscator* and the *denoiser* using multi-objective reinforcement learning. Specifically, we started by training a *denoiser* model. Then, we trained the *obfuscator* to maximize the privacy against the surrogate model, while minimizing the loss of the *denoiser* with obfuscated user personas as inputs. After we trained the *obfuscator*, we retrained the *denoiser* and repeat the above process until both the *obfuscator* and the *denoiser* converge. We found that jointly training did not improve privacy or utility because of our use of the surrogate model, instead of YouTube in the wild, for practical reasons. When trained against the surrogate model, *denoiser* was able to trivially replicate the surrogate model. While in theory we could jointly train the *obfuscator* and the *denoiser* in the wild to avoid this issue, it would not be practical due to its time consuming nature. Future work can look into hybrid surrogate and in the wild joint training of *obfuscator* and *denoiser*.