Increasing Adversarial Uncertainty to Scale Private Similarity Testing

Yiqing Hua and Armin Namavari, Cornell Tech and Cornell University; Kaishuo Cheng, Cornell University; Mor Naaman and Thomas Ristenpart, Cornell Tech and Cornell University

https://www.usenix.org/conference/usenixsecurity22/presentation/hua
Increasing Adversarial Uncertainty to Scale Private Similarity Testing

Yiqing Hua\textsuperscript{1,2}, Armin Namavari\textsuperscript{1,2}, Kaishuo Cheng\textsuperscript{2}, Mor Naaman\textsuperscript{1,2}, Thomas Ristenpart\textsuperscript{1,2}
\textsuperscript{1} Cornell Tech \textsuperscript{2} Cornell University

Abstract

Social media and other platforms rely on automated detection of abusive content to help combat disinformation, harassment, and abuse. One common approach is to check user content for similarity against a server-side database of problematic items. However, this method fundamentally endangers user privacy. Instead, we target client-side detection, notifying only the users when such matches occur to warn them against abusive content.

Our solution is based on privacy-preserving similarity testing. Existing approaches rely on expensive cryptographic protocols that do not scale well to large databases and may sacrifice the correctness of the matching. To contend with this challenge, we propose and formalize the concept of similarity-based bucketization (SBB). With SBB, a client reveals a small amount of information to a database-holding server so that it can generate a bucket of potentially similar items. The bucket is small enough for efficient application of privacy-preserving protocols for similarity. To analyze the privacy risk of the revealed information, we introduce a framework for measuring an adversary’s confidence in inferring a predicate about the client input correctly. We develop a practical SBB protocol for image content, and evaluate its client privacy guarantee with real-world social media data. We then combine SBB with various similarity protocols, showing that the combination with SBB provides a speedup of at least 29× on large-scale databases compared to that without, while retaining correctness of over 95%.

1 Introduction

Faced with various policy-violating activities ranging from disinformation [59] to harassment [21, 34, 48] and abuse [10, 73], social media companies increasingly rely on automated algorithms to detect deleterious content. One widely used approach is to check that user content is not too similar to known-bad content. For example, to detect child sexual abuse imagery [10], some platforms utilize similarity hashing approaches like PhotoDNA [51] or PDQHash [1]. These approaches map user-shared images into unique representations that encode perceptual structure, enabling quick comparisons against a database of hash values. Such approaches could be helpful for combating other forms of bad content, such as the viral spread of visual misinformation on end-to-end encrypted messaging services [59]. For example, they could augment other efforts to provide users with important context about shared content [5, 16].

Currently deployed approaches rely on sending user content or a similarity hash of the content to a moderation service. This risks user privacy. As we detail in the body, the service can easily match a submitted similarity hash against known images to learn the content of a user’s image with overwhelming confidence. Privacy can be improved utilizing cryptographic two-party computation (2PC) [42, 77] techniques to only reveal matching content to the moderation service and nothing more. The recent CSAM image detection system proposed by Apple [3] goes one step further and notifies the platform only when the number of matching images surpasses a certain threshold. Automated notification of platforms necessarily raises concerns about privacy and accountability (e.g., how to ensure the system is not used for privacy-invasive search for benign images).

An alternative approach is to have only the client learn the output of similarity checks, to enable client-side notifications, warning or otherwise informing users. This may not be suitable for all classes of abusive content, such as CSAM, where the recipient may be adversarial, but could be useful for other abuse categories (misinformation, harassment, etc.). However, the scale of databases makes it prohibitive both to send the known-bad hashes to the client or, should hashes be sensitive, apply 2PC techniques to ensure as little as possible about the database leaks to clients. For an example of the latter, Kulshrestha and Mayer’s [42] private approximate membership computation (PAMC) protocol achieves state-of-the-art performance, but nevertheless requires about 27 seconds to perform a similarity check against a database with one million images. The protocol also has an average false negative rate of
almost 17% for slightly transformed images, meaning many similar images may be erroneously marked as dissimilar.

In this work, we target client-side detection, in order to warn users against abusive content. To this end, we explore the question of how to scale privacy-preserving image similarity protocols, while preserving correctness of the similarity testing. We introduce and formalize the concept of similarity-based bucketization (SBB). The idea is to reveal a small amount of structured information in a message to a database-holding server, so that it can determine a bucket of possibly relevant database entries. Ideally the bucket consists of only a small fraction of the full database, enabling use of a subsequent similarity testing protocol on the bucket to perform the final similarity check. We explore instantiating the testing protocol in a variety of ways.

The key technical challenge facing SBB is balancing the competing goals of minimizing bucket size (efficiency) with leaking as little information as possible (privacy). For example, one could modify a standard similarity hash, say PDQHash, to provide only very coarse comparisons. But as we will show, this still leaks a lot of information to the server, allowing high-confidence attacks that can associate the coarse hash to the specific content of a client request. More broadly we need a framework for navigating this tension.

We propose such a framework. It formalizes information leakage using a game-based definition. To be specific, an adversarial server attempts to learn, from an SBB message generated for some image drawn from an adversary-known distribution, a predicate about the underlying image. As an important running example, we use a “matching predicate” that checks if the underlying image has the same perceptual hash value as that of a known target image. Unlike in more traditional cryptographic definitions (e.g., [30]), we do not require the adversarial server to have negligible success (which would preclude efficiency) and instead offer a range of measures including accuracy improvement over baseline guessing, adversarial precision, and adversarial area under the receiver operating characteristic curve (AUC). Indeed, there is no one-size-fits-all approach to measuring privacy damage, and our framework allows one to more broadly assess risks.

We offer a concrete SBB mechanism that increases adversarial uncertainty compared to naive approaches. It converts any similarity hash that uses Hamming distance to a privacy-preserving coarse embedding; we focus on PDQHash because it is widely supported. We combine techniques from locality-sensitive hashing [29] with lightweight noise mechanisms. The ultimate algorithm is conveniently simple: apply a standard PDQHash to an image, choose a designated number \( d \) of bit indices randomly, flip each selected bit with probability \( \gamma \), and then send the resulting \( d \) bits and their indices to the server. An image in the server’s database is included in a bucket should \( k \) or fewer of the relevant \( d \) bits of its PDQHash mismatch with those that are sent from the client.

Using real-world social media data, we empirically assess correctness, efficiency and privacy under various definitions. We explore various settings of \( d \), \( \gamma \), and \( k \), and show that it is possible to ensure average bucket sizes of 9.3% of the database, while: (1) ensuring that the similar images are included in the bucket at least 95% of the time, and (2) an optimal adversary for the matching predicate achieves less than 50% precision, signifying low confidence in matching attacks. We caution that these empirical results are dataset-dependent, and may not generalize to every potential use case. Instead they can be interpreted as a proof-of-concept that SBB works in a realistic scenario.

We then combine our SBB mechanism with various similarity protocols, with different privacy guarantees for the server’s content. For the expedient approach of downloading the bucket of server PDQHash values and performing comparisons on the client side, SBB provides a speedup of 29× or more. A full similarity check requires less than 0.5 seconds for a database of \( 2^{23} \) images. We also explore using SBB to speed up an ad hoc similarity protocol based on secure sketches [20], as well as 2PC protocols implemented in the EMP [72] and CrypTen [41] frameworks. Our experiments indicate that SBB can provide speed-ups of 601×, 97×, and 67×, respectively, and often enables use of 2PC that would fail otherwise due to the size of the database.

We conclude by discussing various limitations of our results, and open questions that future work should answer before deployment in practice. Nevertheless, we expect that our SBB approach will be useful in a variety of contexts. Encrypted messaging apps could use it to help warn users about malicious content, with significantly better privacy than approaches that send plaintext data to third-party servers [5, 16]. In another setting, social media platforms that currently query their users’ plaintext data to third-party services to help identify abuse (e.g., [26, 69]) could use our techniques to improve privacy for their users. To facilitate future research, our prototype implementation is publicly available.\(^1\)

\(^1\)https://github.com/vegetable68/sbb

2 Background and Existing Approaches

In this section, we provide some background about a key motivating setting: providing client-side detection of bad content in end-to-end (E2E) encrypted messaging. That said, our approaches are more general and we discuss other deployment scenarios in the extended version [35].

Content detection and end-to-end encryption. Content moderation aims to mitigate abuse on social media platforms, and can include content removal, content warnings, blocking users, and more. Most moderation approaches rely on detecting objectionable content, particularly at scale where automated techniques seem to be requisite. Social media companies often maintain large databases of known adversarial content [10, 51] and compare a client message with items in
the databases to see if the message is sufficiently similar to some piece of adversarial content. However, this approach requires the client to reveal plaintext message content, which stands in tension with privacy-preserving technologies like E2E encryption. On the other hand, leaving contents unmoderated on the platform is unsatisfactory given the harms caused by abusive content such as misinformation, child sexual abuse material (CSAM), harassment, and more.

Governments and non-governmental organizations have for many years emphasized the need for technical innovations that could enable law enforcement access to encrypted data, while minimizing risks of privacy violation. However, security experts have repeatedly expressed concern that such ‘backdoor’ access would fundamentally break the privacy of E2E encryption or, if it provided content blocking functionality, enable problematic censorship.

In this work, we target mitigations that allow privacy-preserving client-side detection of content similar to known bad content. We focus on images, as discussed below. Our protocol is agnostic to how client software uses this detection capability, but we believe that client software should be designed to empower users with information and the ability to make their own decisions about content.

We target mitigations that allow privacy-preserving client-side detection of content similar to known bad content. We focus on images, as discussed below. Our protocol is agnostic to how client software uses this detection capability, but we believe that client software should be designed to empower users with information and the ability to make their own decisions about content.

Our techniques may be useful, for example, to mitigate the increasing use of E2E encrypted messaging for harmful disinformation campaigns. A widely discussed approach is to warn users against known disinformation. Recent research has shown that when carefully designed, such warnings are effective in guiding user behaviors to avoid disinformation. Our work provides a technical solution for the client-side warning mechanism. To be specific, the proposed system queries whether a client’s received content is similar to known disinformation and returns the answer only to the client. Such a design avoids both outright censorship and notifying platform operators that a particular client received a particular piece of content. This solution would enable the kinds of user-initiated known content detection approaches that have been suggested recently, and could help complement existing anti-abuse techniques that do not consider content, such as those used in WhatsApp.

But warning-style approaches that inform and empower users may not be suitable for threats like CSAM, where the recipients of messages can themselves be bad actors. Here client software would seemingly have to limit user choice, automatically blocking detected content and/or notifying some authority about it. Recent designs for CSAM mitigations include the Kulshrestha-Mayer protocol (when used to notify the platform) and the CSAM detection proposal by Apple. Cryptographers have, in turn, raised the alarm that, while efforts to combat CSAM are laudable, these platform-notifying systems represent a potential E2E encryption backdoor that is subject to misuse by platform operators or governments and that future work is needed to make such systems transparent and accountable. Our work is different, as we target client-side notification and not platform notification.

Another concern is that even client-side notification ends up a stepping stone towards riskier backdoor/censorship mechanisms, because once the former is deployed it will be easier to deploy, or justify deploying, the latter. Client-side functionality at least provides the opportunity for activists and others to detect changes to client-side software and understand their effects, adding some transparency and accountability. At the same time, arguments for, or against, various anti-abuse mechanisms would do well to delineate between approaches that empower users to understand and control their online experience (warnings, the ability to select users/content to block) and that disempower users (client-side or platform-side automatic censorship). We believe our techniques will be useful for the former, without intrinsically promoting the latter.

Client-side similarity testing and privacy. As mentioned above, we focus on private image similarity testing services. These allow a client, who receives some value w on an E2E encrypted platform, to submit a request to a service provider holding a database B; the response indicates to the client whether w is similar to any item in B. As the database B may be quite large, we need scalable solutions. The service provider could be the messaging platform, or a third party service. In the case when the provider is a third party service, the protocol runs between the client and the testing service, without involvement of platform servers.

A key concern will be the privacy risk imposed on clients by a testing service. Our threat model consists of an adversary in control of the service’s servers, who wants to learn information about a client’s image w by inspecting messages sent to the service in the course of similarity testing. This is often referred to as a semi-honest adversary, though our approaches will meaningfully resist some types of malicious adversaries that deviate from the prescribed protocol. In terms of privacy threat, we primarily focus on what we call a matching attack, in which the adversary wishes to accurately check whether w matches some adversarially chosen image (see Section 4 for a formalization). A matching attack enables, for example, adversarial service operators to monitor whether clients received any image on an adversarially chosen watchlist.

In this initial work we primarily focus on the risks against a single query from the client, and explicitly do not consider adversaries that just want to recover partial plaintext information, such as if the adversary wants to infer if an image contains a person or not. While we believe our results also improve privacy for such attacks, we do not offer evidence either way and future work will be needed to explore such threats. We also do not consider misbehaving servers that seek to undermine correctness, e.g., by modifying B to force clients to erroneously flag innocuous images. How to build accountability mechanisms for this setting is an interesting challenge.

---

https://www.missingkids.org/e2ee

---
open question. We simulate the scenarios of adversaries that somehow can take advantage of known correlations between queried images in the extended version [35] and propose potential mitigation solutions in Section 7.

Nevertheless there are already several challenges facing developing a service that prevents accurate matching attacks in our setting. While prior work has established practical protocols for private set membership [43, 70], these only provide exact equality checks. Even small manipulations such as image re-sampling, minor cropping, or format conversion make exact matching schemes fail. Second, the database \( B \) can be arbitrarily large and may require frequent update. For instance, the published dataset from Twitter with activities of accounts associated to the Russian Internet Research Agency consist of 2 million images in total [28].

**Existing approaches.** We review deployed systems and suggested designs for image similarity testing.

**Plaintext services.** Most current deployments have the client upload their image to a third party service. A prominent example is the PhotoDNA service. After a client submits an image to the service, it immediately hashes the image using a proprietary algorithm [51]. Importantly, the hash can be compared to other hashes of images in a way that measures similarity of the original images. Such hashes are often called similarity hashes [15,54] or perceptual hashes [78]. (We show examples later.) The original image that was sent to the service is deleted after hashing. This plaintext design has various benefits, including simplicity for clients and the ability to hide the details of the hashing used. The latter is important in contexts where malicious users attempt to modify an image \( w \in B \) in the service’s bad list to create an image \( w' \) that will not be found as similar to any image in \( B \) (including \( w \) ) [76].

Another example of a plaintext service is WhatsApp’s in-app reverse search function to combat visual misinformation [5], rolled out in June 2020. This feature allows users to submit their images to Google reverse image search for the source or context of a specific image. In this case, the user needs to reveal their image to both Google and WhatsApp, sacrificing user privacy.

**Hashing-based services.** For privacy-aware clients, revealing plaintext images represents a significant privacy risk. An alternative approach is to use a public hashing algorithm, have the client first hash their image, and submit only the resulting representation to the similarity checking service. While this requires making the hashing algorithm available to clients (and, potentially, adversarial users), it improves privacy because the original images are not revealed to the service. It also improves performance: hashes can be compact (e.g., 256 bits) and compared against a large database \( B \) in sublinear time [53]. This approach is used by Facebook’s ThreatExchange [26] service that allows organizations to share hashes of images across trust boundaries. They use a custom similarity hash called PDQHash [1].

Sharing hashes, however, still has privacy risk. For example, although the lossy process of PDQHash generation makes recovering the exact input impossible in general, revealing the hash allows inferring whether a queried value is similar to another image. An adversary at the service provider’s side may brute-force search a database of images to find ones close to the queried value.

**Cryptography-based services.** An alternative approach that preserves privacy is to employ a secure 2PC protocol [77] between the client and service. Existing 2PC protocols for similarity matching (e.g., [7, 13, 37]) can, in the best case, ensure that no information about the client’s image is leaked to the server and that nothing about \( B \) (beyond whether it contains a similar image) leaks to the client. However, existing 2PC protocols do not efficiently scale to large databases \( B \).

Recent work by Kulshrestha and Mayer [42] proposed private approximate membership computation (PAMC) to allow similarity testing of images encoded as PDQHashes. The protocol begins by splitting the database \( B \) into buckets. Using private information retrieval, a client retrieves a bucket from the server with the bucket identifier generated from the PDQHash of their image. The chosen bucket is not disclosed to the server. The two parties then perform a private similarity test to determine whether the client PDQHash has sufficiently small Hamming distance to any image in the bucket. The protocol is still rather expensive, with their initial experiments requiring 37.2 seconds for a one-time set up and 27.5 seconds for a query for a block list with the size of \( 2^{20} \). These times exclude network delays (measurements were performed with client and server on the same workstation). While a step closer to practicality, this remains prohibitive particularly since we expect that performance in deployment would be worse for lightweight client hardware such as mobile phones.

Concurrent work by Apple [3] proposed a protocol that encodes images from a user’s cloud storage into perceptual hashes. The perceptual hashing algorithm maps similar images into identical hashes with high probability. The protocol then performs private set intersection between the encoded hashes and a database of known CSAM images. The private contents are revealed to the platform only when the number of matches exceeds a certain threshold. Whether such a protocol, designed for CSAM detection, is fit for client-side detection remains a question for future work to explore.

In summary, all three existing design approaches for image similarity testing — revealing images as client requests, using similarity representations like PDQHash as client requests, and employing secure 2PC protocols — do not provide satisfying solutions. The first two designs do not provide sufficient privacy, while 2PC designs are currently not sufficiently efficient. Thus we need a new approach to similarity testing.
we refer to the similarity testing server as the server. Indicate the images depicted are perceptually similar.

PDQHash first converts a given image from RGB to luminance, then uses two-pass Jarosz filters to compute a weighted average of $64 \times 64$ subblocks of the luminance image. Given the $64 \times 64$ downsampling, the algorithm computes a two-dimensional discrete cosine transform (DCT), and keeps only the first 16 slots in both X and Y directions. After that, each entry of the $16 \times 16$ DCT output is transformed into a binary bit after being compared to the median, with 1 indicating larger than the median and 0 indicating otherwise.

Coarse embedding schemes. To allow bucketization via similarity, we define a coarse embedding scheme $E = (\text{Emb}, \text{Sim}, (\mathcal{W}, \Delta_W))$, as a pair of algorithms and an associated metric space. The (possibly randomized) embedding algorithm $\text{Emb}(w)$ takes as input a value $w \in \mathcal{W}$ and outputs a value $p \in \{0, 1\}^d$. Here $d$ is a configurable parameter. We call $p$ the embedding of $w$, or simply the embedding when $w$ is clear from context. The deterministic algorithm $\text{Sim}(p, w')$ takes as input $p \in \{0, 1\}^d$ and $w' \in \mathcal{W}$ and outputs a bit. The bit being one indicates that the embedding of $w'$ is similar to the embedding of $w$, which is denoted as $p$. It will be convenient to abuse notation, by letting $\text{Sim}(p, B)$ be defined to output the set $\{w' \mid w' \in B \land \text{Sim}(p, w') = 1\}$.

One idea for a coarse embedding scheme would be to simply use $\mathcal{F}$ directly, but with smaller $\ell$ and smaller $T$. To be specific, using PDQHash as an example, a coarse PDQHash scheme $E_{cPDQ} = (\text{Emb}_{cPDQ}, \text{Sim}_{cPDQ}, (\mathcal{W}, \Delta_W))$ can be implemented as follows: $\text{Emb}_{cPDQ}(w)$ computes the hash of $w$ on the first 4 x 4 slots of the DCT output, rather than $16 \times 16$ of the output, producing a 16-bit binary string. The 16-bit value can then provide much cruder similarity comparison. Then $\text{Sim}_{cPDQ}(p, B)$ iterates over all $w' \in B$, hashes them, and returns those with distance smaller than a coarse threshold $k$ as a bucket $B$. Unfortunately this scheme doesn’t meet our privacy goals, as we will explore in detail in Section 5.

Correctness and compression efficiency. We define $\Delta_W$ via an existing similarity embedding $\mathcal{F}$, i.e., $\Delta_W(w, w') = \Delta(\mathcal{F}(w), \mathcal{F}(w'))$. We say that a coarse embedding scheme is $(T, \varepsilon, \mathcal{D})$-correct if, for an image $w$ sampled from $\mathcal{D}$, a distribution over $\mathcal{W}$, and for any $w'$ such that $\Delta_W(w, w') < T$,
we have that \( \Pr[\text{Sim}(\text{Emb}(w), w') = 1] \geq 1 - \varepsilon \), where the probability is taken over the random coins used by Emb and the choice of \( w \) from \( D \). A trivial coarse embedding scheme is to just use \( F \) itself, which would be \((T, 0, D)\)-correct for any \( T \) and \( D \). But as mentioned, doing this will not provide the desired privacy.

Another type of trivial coarse embedding scheme is to have Sim always output one. Then Emb could output a fixed constant value regardless of input, meaning nothing leaks about \( w \). This would also be \((T, 0, D)\)-correct for arbitrary \( T \) and any given \( D \), but won’t be useful because, in our SBB application, the bucket would end up being the entire set \( B \).

We define a compression efficiency metric as follows. A coarse embedding scheme is \((B, \alpha, D)\)-compressing if for a distribution \( D \) over \( W \), \( B \subseteq W \), \( w \) drawn from \( D \) we have that \( E \left[ \frac{|B|}{|w|} \right] \leq \alpha \) where \( B = \{ w' : w' \in B, \text{Sim}(\text{Emb}(w), w') = 1 \} \) and the probability space is over the choice of \( w \) from \( D \) and the coins used by Emb. This measures the average ratio of bucket size to dataset size.

**LSH-based coarse embedding.** We propose a coarse embedding scheme that is based on locality sensitive hashing (LSH) [29]. An LSH function family allows approximate nearest neighbour search with high-dimensional data. Formally, the scheme \( E_{\text{LSH}} = (\text{Emb}_{\text{LSH}}, \text{Sim}_{\text{LSH}}, (W, \Delta_W)) \) is defined as follows (see also Figure 2). Let \( I \) be a family of hash functions that maps points from a high-dimensional input space \( I \) into a hash universe \( U \) of lower dimension. When \( I = \{0, 1\}^I \) and \( \Delta \) is Hamming distance, the construction of an LSH function family is intuitive. For an \( I \)-bit string \( v \), we denote the individual bits as \( v_1, \ldots, v_I \). An indexing function is a map \( I : \{0, 1\}^I \rightarrow \{0, 1\} \) and we let \( I \) be the set of all index functions, which is the LSH function family.

In our context, we randomize the selection of LSH functions for every individual query, and add noise to ensure privacy. \( \text{Emb}_{\text{LSH}} \) takes an image \( w \) as input, and computes the similarity embedding of it via \( v \leftarrow F(w) \). In our implementation, we use PDQHash for \( F \). Our protocol works on other types of embedding functions that use Hamming distance as a metric, such as pHash [78]. We sample \( d \) bits from \( v \) by sampling \( d \) LSH functions without replacement and flip each bit with probability \( \gamma \). The resulting embedding with added noise and the indices are shared with the server. The server performs \( \text{Sim}_{\text{LSH}} \) by comparing the received bits to the corresponding bits of \( F(w_i) \) for each \( w_i \in B \), adding \( w_i \) to the bucket \( B \) if sufficiently many of these bits match.

To formalize this, we abuse notation slightly. We denote \( I \) as a map \( \{0, 1\}^I \rightarrow \{0, 1\}^d \), a combination of \( d \) functions sampled uniformly from \( I \) without replacement. Similarly, one can easily encode an indexing function as a set of indexes; we treat \( I \) both as a function and its encoding. We let \( \text{Flip}_p \) be the randomized algorithm that takes as input a bit string \( p \) and outputs \( \tilde{p} \) of the same length, setting \( \tilde{p}_i = p_i \) with probability \( 1 - \gamma \) and \( \tilde{p}_i = \neg p_i \) with probability \( \gamma \). The full algorithms

\[
\begin{align*}
\text{Emb}_{\text{LSH}}(w) & \quad \text{Sim}_{\text{LSH}}((I, p), B) \\
v \leftarrow F(w) & \quad B \leftarrow \{} \\
I \leftarrow I & \quad \text{For } w \in B: \text{Sim}(\text{Emb}(w), B) \leq \text{Sim}(\text{Sim}_{\text{LSH}}((I, p), B), B) \leq \Delta \text{ then } B \leftarrow B \cup \{w\} \]
\end{align*}
\]

**Figure 2:** Coarse embedding scheme \( E_{\text{LSH}} \).

for \( E_{\text{LSH}} \) are shown in Figure 2. We use a threshold \( k \) for the Hamming distance over the randomly selected indexes. We formally analyze correctness of \( E_{\text{LSH}} \) in the extended version [35]. Different choices of the parameter sets of \( E_{\text{LSH}} \), i.e., embedding length \( d \), flipping bias \( \gamma \), and coarse thresholds \( k \) result in different combination of privacy loss, correctness, and bucket compression rate. We further explore this trade-off in Section 5.

One limitation of \( E_{\text{LSH}} \) arises should an adversary be able to collect many queries that it knows are for the same image. Eventually it will see all bit locations, and even have enough samples to average out the noise (e.g., via a majority vote for each bit location). We discuss this further in Section 7.

**Similarity protocols.** A coarse embedding scheme will not suffice to perform a full similarity check. Instead, we compose such a scheme to perform SBB with a similarity protocol where the server uses the resulting bucket \( B \leftarrow \text{Sim}(\text{Emb}(w), B) \). The composition achieves privacy levels related to the protocol’s for \( B \), and correctness proportional to the product of the coarse embedding and the protocol’s correctness. We discuss some examples and their properties here. These examples ensure perfect correctness, hence the correctness of the composition depends solely on that of the coarse embedding. In Section 5.4, we show that for various similarity protocols, both runtime efficiency and bandwidth are largely improved when combined with SBB.

**Similarity embedding retrieval.** A pragmatic similarity protocol has the server send to the client the similarity embeddings of all the elements in the bucket, i.e., send \( v_1, \ldots, v_B \) where \( v_i = F(w_i) \) for each \( w_i \in B \). The client can then compute \( F(w) \) and compare against each \( v_i \). This approach reduces the confidentiality for the server’s dataset, since now clients learn all the similarity embeddings in \( B \) that fall into the bucket. It may also reduce resistance to evasion attacks, but in contexts where client privacy is paramount this simple protocol already improves on existing approaches.

**Secure-sketch similarity protocol.** We can improve server confidentiality via a secure-sketch-based [20] similarity protocol. The protocol ensures that the client can only learn the similarity hashes that are close to a client-known value. If images in the database have sufficiently high min-entropy then the secure sketch ensures that the client cannot learn it. This assumption may not always hold (most obviously in the case that the client has a similar image), in which case confidentiality falls back to that achieved by similarity embedding.
retrieval. We defer details to the extended version [35].

2PC similarity protocols. Finally, one may compose SBB with an appropriate 2PC for similarity comparisons. Such an approach provides better confidentiality for $B$, but at the cost of larger bandwidth and execution time. We experiment with two frameworks: CrypTen [41] and EMP [72]. CrypTen is a secret-sharing-based semi-honest MPC framework for Python that is geared toward machine learning applications. CrypTen currently relies on a trusted third party for certain operations, including generating Beaver multiplication triples [9]. Generation of Beaver triples using Pailler [56] is actively under development. EMP is a circuit-garbling-based generic semi-honest 2PC framework that is implemented in C++.

Both frameworks above target semi-honest security. One could also compose SBB with a maliciously secure 2PC protocol, with caveat that a malicious server is not bound to correctly execute the SBB Sim algorithm and so could deviate by adding arbitrary values to the bucket. In our context, such an attack can anyway be performed by just modifying $B$ in the first place, but this could be relevant in future work, particularly as it relates to accountability mechanisms that monitor for changes to $B$.

**4 Privacy of Coarse Embeddings**

In this section, we detail our framework for reasoning about privacy threats against coarse embeddings. Our framework is designed to analyze the adversary’s confidence in assessing a predicate being true or not, when given one or multiple client requests as input. Here we only consider client privacy; privacy of the server’s dataset can be achieved by composing SBB with a suitable similarity protocol (see Section 3).

Proposed security measures. We consider settings where an adversary receives the embedding(s) of one or more images, and wants to infer some predicate over the images. Let $W^q$ be the Cartesian product of $q$ copies of $W$. We denote tuples of images in bold, $w \in W^q$ and $w[i] \in W$ for $i \in [1,q]$. Let $\text{Emb}(w)$ be the result of running $\text{Emb}$ independently on each component of $w$, denoted as $p$. That is, $p \leftarrow \text{Emb}(w)$ is shorthand for $p[i] \leftarrow \text{Emb}(w[i])$ for $i \in [1,q]$.

To start, consider a distribution $\mathcal{D}$ over $W^q$ and a predicate $f : W^q \rightarrow \{\text{false, true}\}$. We want to understand the ability of an adversary to infer $f(w)$ when given $\text{Emb}(w)$ for $w$ drawn from $W^q$ according to $\mathcal{D}$. As an example, let $q = 1$ and have $f$ indicate whether a client image has the same perceptual hash value with that of another image that is chosen by the adversary. We’d like to have a guarantee that revealing $\text{Emb}(w)$ doesn’t allow inferring that the images are similar with high confidence. We refer to a tuple $\pi = (\mathcal{D}, W^q, f)$ as a privacy setting.

We provide three measures of adversarial success: accuracy, precision, and area under the receiver-operator curve (AUC), thereby adapting traditional measures of efficacy for prediction tasks to our adversarial setting.

**Accuracy.** Let $A_{\text{acc}}$ be a randomized algorithm, called an accuracy adversary. We define a probabilistic experiment that tasks $A$ with inferring $f(w)$ given $\text{Emb}(w)$ for $w$ drawn according to $\mathcal{D}$. This probability space is over the coins used to sample $w$, to run $\text{Emb}$ a total of $q$ times, and to run $A_{\text{acc}}$. We let $\mathcal{A}_{\text{acc}}(\text{Emb}(w)) = f(w)$ be the event that $A_{\text{acc}}$ outputs the correct value of the predicate. We write this as a pseudocode game $\text{Pred}_{\text{Emb},\pi}$ shown in Figure 3, where the return value captures the event that $A$ succeeds. For skewed distributions, the trivial adversary that ignores its input and simply predicts the most likely predicate value may achieve high accuracy. We therefore define the advantage of $A_{\text{acc}}$ as the improvement over that trivial approach:

$$
\varepsilon_{\text{acc}} = \Pr[A_{\text{acc}}(\text{Emb}(w)) = f(w)] - \varepsilon_{\text{base}}
$$

where $\varepsilon_{\text{base}} = \max(\Pr[f(w) = 1], \Pr[f(w) = 0]).$

**Precision.** For adversaries that are mainly interested in inferring positive instances, $f(w) = \text{true}$, accuracy may appear misleading in cases with high skew, i.e., when $f(w) = \text{false}$ happens almost always [47]. In our running example, we expect that in practice most images handled by clients will be distinct from the adversary-chosen one.

We therefore also provide two other security measures. First, we measure the precision of a non-trivial adversary in inferring $f(w)$. By non-trivial, we mean that the adversary has to predict $f(w) = \text{true}$ at least once. We use the same probability space as in the previous definition. To emphasize that the best adversary for achieving high precision may differ from the best one for maximizing accuracy improvement, we use $A_{\text{pre}}$ to denote the adversary when considering precision. We want to measure the probability that $A_{\text{pre}}$ succeeds, conditioned on $A_{\text{pre}}$ outputting true. We denote this by $\Pr[f(w) = \text{true} \mid A_{\text{pre}}(\text{Emb}(w)) = \text{true}]$. To prevent $A_{\text{pre}}$ from using the trivial strategy of predicting all events as negative, we define an affiliate concept of recall $r$ as

$$
r = \Pr[A_{\text{pre}}(\text{Emb}(w)) = \text{true} \mid f(w) = \text{true}].
$$

We will restrict attention to adversaries $A_{\text{pre}}$ for which $r$ exceeds some threshold, e.g., $r > 0\%$. We let

$$
\varepsilon_{\text{pre}} = \Pr[f(w) = \text{true} \mid A_{\text{pre}}(\text{Emb}(w)) = \text{true}]
$$

denote the precision advantage for some adversary $A_{\text{pre}}$ that
achieves \( r > p \), with the exception of \( p = 100\% \), where the restriction is set as \( r = 100\% \).

**AUC.** Precision captures the adversary’s confidence in predicting the positive class, i.e., the likelihood of \( f(w) \) being true when the adversary predicts it to be true. However, it does not capture the adversary’s confidence regarding predicting the negative class. We therefore finally formalize a notion of AUC, where recall that AUC is the area under the receiver-operator curve, a popular measure of classifier efficacy. At a high level, AUC-ROC indicates the classifier’s capability in differentiating positive classes from negative ones. For a setting \( \pi = (D, B^s, f) \), let \( D_i \) be the distribution \( D \) over \( \mathcal{W}^q \) conditioned on \( f(w) = i \) for \( i \in \{true, false\} \). Then for an adversary \( A_{\text{auc}} \) that outputs a real value in \([0, 1] \) we measure the probability that \( A_{\text{auc}}(\text{Emb}(w_{true})) > A_{\text{auc}}(\text{Emb}(w_{false})) \) where \( w_i \) is drawn from \( B^s \) according to \( D_i \). The probability is over the independent choices of \( w_{true} \) and \( w_{false} \), as well as the coins used by the \( 2q \) executions of Emb and two executions of \( A_{\text{auc}} \). We provide a pseudocode game \( \text{AUCEmb.\pi} \) describing this probability space in Figure 3. Then we define the advantage of an AUC adversary \( A_{\text{auc}} \) by

\[
e_{\text{auc}} = 2 \cdot \Pr[A_{\text{auc}}(\text{Emb}(w_{true})) > A_{\text{auc}}(\text{Emb}(w_{false}))] - 1 .
\]

This formulation uses a well-known fact [4, 17] about AUC that it is equal to the probability that a scoring algorithm (in our case, the adversary) ranks positive-class instances higher than negative-class instances. For simplicity, we ignore ties (\( A_{\text{auc}} \) outputting the same value in each case). Without loss of generality, we can assume that the AUC adversary \( A_{\text{auc}} \) wins the game with probability greater than or equal to 0.5, and so the normalization maps to the range \([0, 1] \). (This corresponds to the classic Gini coefficient.)

**Possible predicates.** We focus on the matching predicate in our analyses. An adversary chooses an image \( w_{adv} \), and wishes to determine if the client request \( \text{Emb}(w_c) \) corresponds to an image that is very similar to \( w_{adv} \), i.e., \( \mathcal{F}(w_c) = \mathcal{F}(w_{adv}) \). If the adversary has perfect knowledge of the distribution \( D \) as well as \( D^s \)’s support, which is unlikely in practice. While we do not explicitly model side information that an adversary might have about a client’s image, it is possible to include it indirectly in this framework, for example, by changing the distribution or modifying the privacy predicate.

**Bayes optimal adversaries.** To allow simulations that evaluate privacy, we focus on adversaries that maximize advantage. Recall that we assume that the adversary knows the distribution \( D \) from which the clients are sampling images for their requests. Upon receiving client submitted requests \( p = \text{Emb}(w) \), the Bayes optimal adversary computes the exact likelihood of the predicate being true — \( \Pr[f(w) = true | \text{Emb}(w) = p] \), probabilities are over the choice of \( w \) being sampled from \( \mathcal{W}^q \) and coins used by the executions of Emb. The Bayes optimal adversary for the precision metric, \( A_{\text{prec}} \), chooses a threshold \( T_{adv} \), such that \( A_{\text{prec}}(w) = true \) if and only if \( \Pr[f(w) = true | \text{Emb}(w) = p] > T_{adv} \). The adversary may choose \( T_{adv} \) to maximize \( \varepsilon_{\text{prec}} \). A similar strategy can be used by \( A_{\text{acc}} \). However when \( f(w) = true \) is especially rare, the adversary may achieve larger \( \varepsilon_{\text{acc}} \) by predicting all predicates using the majority class, \( f(w) = false \). When doing so, the optimal \( \varepsilon_{\text{acc}} \) is zero. Note that in our simulations we consider all possible threshold values for the sampled dataset, and report on the one that provides the best success rate. A real attacker would have to pick a threshold a priori, meaning our analyses are conservative.

The Bayes optimal adversary for \( A_{\text{auc}} \) doesn’t have to choose a threshold \( T_{adv} \). The adversary is given two scenarios to rank: \( w_{true} \) and \( w_{false} \), one has \( f(w_{true}) = true \) and the other has \( f(w_{false}) = false \). The adversary wins the game when they correctly rank the true scenario over the false one, i.e., when \( A_{\text{auc}}(p_{true}) > A_{\text{auc}}(p_{false}) \). As \( \text{Emb}(p) \) is the only information that the adversary gained from our SBB protocol, the optimal strategy to utilize the information is hence to use \( \Pr[f(w) = true | \text{Emb}(w) = p] \) as \( A_{\text{auc}}(p) \).

## 5 Balancing Security, Correctness, Efficiency

In this section, we demonstrate how to balance security, correctness and compression efficiency of SBB when using the LSH-based coarse embedding scheme \( E_{LSH} \). We do so via simulations using real-world image sharing data collected from social media sites. Using our framework, we evaluate the security of \( E_{LSH} \) with varying parameter settings. We then fix the security requirement and explore the trade-off between correctness and compression efficiency.

### 5.1 Experimental Setup

**Data collection.** Recall that our deployment scenario in Section 2, an ideal dataset should represent the image sharing behaviors among users on an end-to-end encrypted messaging platform. However, data of one-to-one shares among users on any private messaging protocol is by definition, private.
We simulate a workload for similarity testing as follows. Any larger threshold, there are more requests containing images. Images in the client requests (84%) share the same similarity for the case when \( T = 70 \) (first row from top, darkest column with purple).

**Implementation.** We compare the privacy of SBB when using \( E_{\text{LSH}} \) with different embedding lengths \( d \) to the baseline method \( E_{\text{PDQ}} \). We focus on the security guarantees against the matching attack and explain our implementation details.

We formally define the matching attack as \( \pi_{\text{match}}^w = (D, W, f_{\text{match}}^w) \), where \( D \) is the distribution over \( W \) that the client requests are sampled from and \( w_{\text{adv}} \) is an image chosen by the adversary. For any client submitted request with an image \( w \), the adversary wishes to learn the value of \( f_{\text{match}}^w(w) \). We have \( f_{\text{match}}^w(w) = \text{true} \) if and only if the corresponding PDQHash of the two images are the same, i.e., \( F(w) = F(w_{\text{adv}}) \). When trying to match the client image to \( w_{\text{adv}} \), an adversary who receives the client request through a bucketization protocol is able to filter out images that are not in the bucket. When the client image is in fact similar to \( w_{\text{adv}} \), it should most likely be included in the same bucket by definition of correctness. In this case, a \( w_{\text{adv}} \) whose similarity hash value is shared more frequently than any other image in the same bucket may boost the adversary’s confidence in asserting that the client image is a similar match. Hence, having an image with a more popular hash value as \( w_{\text{adv}} \) increases the adversarial advantage. In the following experiments, we use the image that has the most popular \( F(w) \) as \( w_{\text{adv}} \). The same similarity hash appeared in 0.2% of all requests.

To evaluate the privacy guarantees provided by \( E_{\text{LSH}} \) and \( E_{\text{PDQ}} \), we iterate over all requests in our dataset and simulate the client, server, and adversary behavior, to compute the security metrics \( \epsilon_{\text{acc}}, \epsilon_{\text{aux}}, \) and \( \epsilon_{\text{cond}} \).

**Coarse PDQHash embedding scheme (\( E_{\text{PDQ}} \)).** Recall the algorithm of \( E_{\text{PDQ}} \) described in Section 3. The embedding algorithm \( \text{Emb}_{\text{PDQ}}(w) \) takes an image \( w \) as an input, and follows the PDQHash algorithm but with modified parameter settings, to generate a 16-bit coarse PDQHash. This allows coarse grained similarity comparison. When receiving a request \( p = \text{Emb}(w) \), the Bayes optimal adversary follows the strategy as described in Section 4. To be specific, the adversary computes the likelihood of the predicate being true, \( \Pr(F(w) = F(w_{\text{adv}}) | \text{Emb}_{\text{PDQ}}(w) = p) \), and makes a binary prediction based on the computed likelihood.

**LSH-based embedding scheme (\( E_{\text{LSH}} \)).** Recall that in Figure 2, the LSH-based embedding scheme \( E_{\text{LSH}} \) consists of two algorithms \( \text{Emb}_{\text{LSH}} \) and \( \text{Sim}_{\text{LSH}} \). \( \text{Emb}_{\text{LSH}} \) takes an image \( w \) as an input, and outputs the selected indexing function \( I \) from \( \mathcal{I} \), and computes \( \hat{p} = \text{Flip}_p(I(F(w))) \). The function is parameterized by two parameters: the length of the indexing function \( d \) and the bias \( \gamma \) to flip an index that was chosen.

**Dataset statistics.** Users on social media share similar images frequently. The \( T \)-neighborhood size of an image \( w \) is the number of images that are \( T \)-similar to it. Two images are \( T \)-similar if and only if their similarity embeddings have a Hamming distance smaller than \( T \). We choose the values of \( T \) according to the recommendations from the white paper on PDQHash [1], where 32 and 70 were specified as the lower and upper bounds of recommended similarity thresholds. We also include \( T = 0 \) and \( T = 64 \) for comparison.

Figure 4 shows the distribution of \( T \)-neighborhood sizes (shades of color) of the images in client requests, with different \( T \) (in different rows). The lightest shades (left) are requests with neighborhood size of one, i.e., the neighborhood only contains the single image. The following darker shades are the images with neighborhood size in the range \((1, 10)\), \((1, 100)\), and \((100, \infty)\). The bottom row shows the neighborhood size distribution with \( T = 0 \). In our dataset, most of the images in the client requests (84%) share the same similarity embedding with more than one, but fewer than 10 neighbors. The distributions of neighborhood size are mostly similar to each other, especially for \( T = 64 \) and \( T = 70 \). Naturally, with a larger threshold, there are more requests containing images with a larger neighborhood size. For example, only 2.85% of all request images have a neighborhood size larger than 100 with \( T = 32 \) (third row from top, darkest column with purple), while 5.23% of the requests satisfy the same condition when \( T = 70 \) (first row from top, darkest column with purple).
We first repeat this process for at least 10 times. For parameter (round markers), Theorem 1. Let $\epsilon$ estimate for all experiments after at most 20 iterations. We were able to obtain a stable setting that result in large fluctuation in the results, we repeat for another 10 iterations. This information is revealed to the adversary and the adversary all requests in the dataset. We fix $I$ choose an index function $\text{ indexing function}$ takes the output from $\text{Emb}$ $\text{LSH}$. Note that $d$ is also the length of the coarse embedding. $\text{Sim}_{\text{LSH}}$ takes the output from $\text{Emb}_{\text{LSH}}$ and a dataset $B$ as input, and outputs a candidate bucket as an output. The function has one parameter $k$, a coarse threshold to choose items for the candidate bucket.

We analyse the security guarantee of $E_{\text{LSH}}$ against the Bayes optimal adversary under the setting of a matching attack $p_{\text{match}}$. When receiving a request $p$, as in $p = \text{Emb}(w)$, the Bayes optimal adversary wants to predict $f_{\text{match}}(w) = \text{true}$. The adversary bases their prediction on $\Pr[\mathcal{F}(w) = \mathcal{F}(w_{\text{adv}}) | \text{Emb}_{\text{LSH}}(w) = p]$, and computes the probability by using all the information that is revealed to them: the indexing function $I$, the resulting coarse embedding $\beta$ and the added noise $\gamma$. We define the distribution $D_x$ of similarity hashes of images sampled from $D$ as follows: $D_x(x) = \Pr[\mathcal{F}(w) = x] = \sum_{w' \in D} D(w')$. Theorem 1. Let $\Delta_x(v, p) = \Delta(I(v), p)$ for a similarity hash $v \in \{0, 1\}^l$. Consider a fixed image $w_{\text{adv}} \in W$ and a sampled image $w \leftarrow D$. Let $v_{\text{adv}} = \mathcal{F}(w_{\text{adv}})$ and $v = \mathcal{F}(w)$. Then

$$\Pr[v = v_{\text{adv}} | \text{Emb}(w) = (I, p)] = \frac{\gamma^{|v_{\text{adv}}|} \cdot (1 - \gamma)^{|w - v_{\text{adv}}|} \cdot D_x(v_{\text{adv}})}{\sum_{v' \in \{0, 1\}^l} \gamma^{|v'|} \cdot (1 - \gamma)^{|w - v'|} \cdot D_x(v')}$$

where the probability is over the coins used by Emb and the choice of $w$ sampled from $D$.

We prove this theorem in the extended version [35]. When analysing the security guarantee of $E_{\text{LSH}}$, we vary the parameter settings of $d$ and $\gamma$ only, as $k$ has no impact on the adversarial advantage. For any choice of $d$ and $\gamma$, we randomly choose an index function $I$, and then execute the protocol for all requests in the dataset. We fix $I$ in the simulation because this information is revealed to the adversary and the adversary computes the likelihood conditioned on the indexing function. We first repeat this process for at least 10 times. For parameter settings that result in large fluctuation in the results, we repeat for another 10 iterations. We were able to obtain a stable estimate for all experiments after at most 20 iterations.

5.2 Privacy of SBB

Using different security metrics. To obtain a broad understanding of the information leakage from our protocol, we present all three security metrics $\epsilon_{\text{acc}}, \epsilon_{\text{prec}},$ and $\epsilon_{\text{adv}}$ in Figure 5 (left). To be specific, we show the results of using (1) $E_{\text{LSH}}$ without added noise (i.e., $\gamma = 0$), but with varying embedding length $d$, and (2) $E_{\text{PDQ}}$, a baseline method that embeds a client request into a 16-bit coarse PDQHash. The Y axis denotes the value of the security metrics, ranging from 0 to 100%. The X axis denotes different methods used. From left to right, we list $E_{\text{LSH}}$ with $d$ ranging from 8 to 16 (with round markers). In the red box, to the very right, the diamond markers represent the security metrics for $E_{\text{PDQ}}$. A naive baseline of using the plaintext similarity embedding (as mentioned in Section 2) achieves 100% for all security metrics (not included in figure).

In our setting, accuracy measures the adversary’s performance in predicting the correct class; precision measures the adversary’s confidence in predicting the positive class; AUC measures the adversary’s ability in differentiating negative classes from positive ones. Our dataset is highly skewed, with more negative predicates than positive ones: only 0.2% of all requests trigger positive matches. This property may lead to a biased view when measured by certain metrics.

Accuracy. In datasets with a skewed distribution, a trivial algorithm that always outputs the majority class, i.e., $f_{\text{match}}^\text{false}(w) = \text{false}$, may achieve higher accuracy than any meaningful algorithm that tries to differentiate positive cases from negative cases. In fact, in some cases, when experimenting with $E_{\text{LSH}}$ with $d = 8$, we have $\epsilon_{\text{acc}} = 0$ (Figure 5 on the left, green markers). This indicates that the adversarial advantage as measured by $\epsilon_{\text{acc}}$ was based on the performance of the trivial algorithm, hence was considered as none. Meanwhile, other metrics ($\epsilon_{\text{prec}}$ and $\epsilon_{\text{false}_R}$) show that the adversarial advantage is non-zero, e.g., $\epsilon_{\text{false}_R} = 37%$ for $d = 8$. Both $E_{\text{PDQ}}$ and $E_{\text{LSH}}$ allow the adversary to have perfect accuracy improvement when performing the matching attack when of similar coarse embedding length. However, applying $E_{\text{LSH}}$ with a smaller coarse embedding length, e.g., $d = 10$ decreases the accuracy improvement to 45%. Figure 5: Left: Simulation results of the three security metrics, evaluated on (1) $E_{\text{LSH}}$ with embedding length $d$ from 8 to 16 (round markers), $\gamma = 0$ and (2) $E_{\text{PDQ}}$ (diamond markers in red box). Right: The conditioned precision metric $\epsilon_{\text{prec}}$ of matching attack, at different recall threshold, with $d = 9$ and varying $\gamma$. Error bars in both plots represent the 95% confidence interval.
**Precision.** When there’s no noise added in $E_{LSH}$ ($\gamma = 0$), $\epsilon_{prec}$ remains the same regardless of varying recall thresholds $\rho$. We will expand on this point later. With the same value of precision, a larger recall specifies more true positive predicates that the adversary may correctly classify with high confidence, hence larger privacy damage. When using $E_{LSH}$, decreasing the length of coarse embeddings ($d < 12$) decreases adversarial precision, improving security. However, with embeddings of similar length, $E_{LSH}$ and $E_{PDQ}$ both behave poorly ($d = 15, 16$ compared with $E_{PDQ}$).

**AUC.** Regardless of the embedding schemes, $\epsilon_{adv}$ is almost 100%. The reason is that there are disproportionately many images for which the predicate evaluates as negative that can be easily differentiated from positive ones. Hence, most images with different PDQHashes from $w_{adv}$ are assigned to different buckets than $w_{adv}$. Therefore, when measured by the adversary confidence of differentiating negative cases from positive ones, as most of the negative cases can be distinguished correctly, the embedding schemes behave poorly. Note that the definition of AUC is in direct conflict with the utility of an adversary with high confidence, hence larger privacy damage. When using $E_{LSH}$, predicting a positive answer, this leads to a larger likelihood of being impacted from the uncertainty introduced by the flipping bias. Nevertheless, when having $\gamma \geq 0.05$, $\epsilon_{prec}$ is smaller than 50% (noted by the horizontal line) for all recall thresholds. This indicates that for any given query that the $A_{prec}$ predicts as true, the adversarial success rate is lower than randomly flipping a coin.

In summary, these results show that using coarse PDQHash $E_{PDQ}$ fails to provide privacy for clients. The naive solution of revealing the plaintext similarity embeddings to the server also provides no privacy. Different security metrics demonstrate different aspects of adversarial advantage. We focus on $\epsilon_{prec}$ as its definition fits our privacy goal the best. For our purpose, we consider $\epsilon_{prec}^{\rho=0} < 50\%$ as our security goal, i.e., when the majority of an adversary’s positive guesses (given that there’s at least one) are wrong. Given our empirical analysis, we suggest that a reasonable choice of parameters is embedding length $d = 9$ and flipping bias $\gamma = 0.05$, but caution that the privacy performance may vary in practice should the image distribution be very different (see Section 7).

### 5.3 Correctness and compression efficiency

We show the tradeoff between correctness and bucket compression rate under the security parameters suggested above ($d = 9$, $\gamma = 0.05$). We vary the value of the coarse threshold $k$, which specifies the bucket being sent from the server side when performing $Sim_{LSH}$. Formally, we refer to the notion of correctness as $\epsilon$ in the definition of $(T, \epsilon, D)$-compressing (see Section 3). We use the dataset as $D$, which we refer to as $D_{\text{twitter}}$, and all the possible values of PDQHash in the dataset as $B$. We randomly select 2 million requests with replacement and perform the protocol, and take the average value of the correctness and compression rate from all iterations.

In Figure 6, we present the trade-off between correctness and compression efficiency. We plot the correctness $\epsilon$ (Y-axis) as the average compression rate $\alpha$ varies (X-axis). The dashed horizontal line represents 95%. We experiment with different definitions of similarity, i.e., with different values of $T$, denoted by different colors. Each node represents a pa-
Datasets. For client requests, we sample half of the requests from the union of the COCO, T4SA, and Webvision 2.0 datasets. For client requests, we sample half of the requests from the union of the COCO, T4SA, and Webvision 2.0 datasets. For instance work [42].

For each set of experiments with a randomly generated blocklist to simulate requests that don’t match any image in the block list, and the rest from the generated blocklist to simulate client requests that return a match.

For each set of experiments with a randomly generated blocklist to simulate requests that don’t match any image in the block list, and the rest from the generated blocklist to simulate client requests that return a match.

In conclusion, these experiments suggest that for $D_{\text{twitter}}$, one can achieve $\epsilon_{\text{prec}} < 50\%$, over 95% correctness for the investigated values of $T$ and 9.3% compression rate using $E_{LSH}$ with $d = 9$, $\gamma = 0.05$ and $k = 3$. Hence, the resulting scheme achieves an almost order of magnitude reduction in the amount of data input to a second-stage similarity protocol.

5.4 End-to-end Simulation

We now perform end-to-end simulation on varying sizes of blocklists $B$ to demonstrate the improvement of execution time and bandwidth for different similarity protocols combined with SBB. For the experiments, we use parameters suggested in Section 5.3, i.e. $d = 9$, $\gamma = 0.05$ and $k = 3$.

Datasets. To form varying sizes of blocklist $B$, we randomly sample images from the datasets that were used in prior work [42]. The details of the datasets are listed in Table 2.

Table 2: Dataset statistics and description.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th>Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB-WIKI [61, 62]</td>
<td>Faces.</td>
<td>523,051</td>
</tr>
<tr>
<td>COCO [45]</td>
<td>Common objects.</td>
<td>123,403</td>
</tr>
<tr>
<td>T4SA [71]</td>
<td>Twitter images.</td>
<td>1,473,394</td>
</tr>
<tr>
<td>Webvision 2.0 [44]</td>
<td>Flickr and Google images.</td>
<td>13,907,566</td>
</tr>
</tbody>
</table>

Figure 6: Correctness (with varying similarity threshold $T$) and compression rate tradeoff. The dashed line marks 95%.

In conclusion, these experiments suggest that for $D_{\text{twitter}}$, one can achieve $\epsilon_{\text{prec}} < 50\%$, over 95% correctness for the investigated values of $T$ and 9.3% compression rate using $E_{LSH}$ with $d = 9$, $\gamma = 0.05$ and $k = 3$. Hence, the resulting scheme achieves an almost order of magnitude reduction in the amount of data input to a second-stage similarity protocol.

5.4 End-to-end Simulation

We now perform end-to-end simulation on varying sizes of blocklists $B$ to demonstrate the improvement of execution time and bandwidth for different similarity protocols combined with SBB. For the experiments, we use parameters suggested in Section 5.3, i.e. $d = 9$, $\gamma = 0.05$ and $k = 3$.

Datasets. To form varying sizes of blocklist $B$, we randomly sample images from the datasets that were used in prior work [42]. The details of the datasets are listed in Table 2. In particular, for each experiment, we generate $B$ of the requisite size by uniformly selecting images (without replacement) from the union of the COCO, T4SA, and Webvision 2.0 datasets. For client requests, we sample half of the requests from the IMDB-WIKI dataset to simulate requests that don’t match any image in the block list, and the rest from the generated $B$ to simulate client requests that return a match.

For each set of experiments with a randomly generated $B$ tested with similarity embedding retrieval (the server simply sends the embeddings of bucket entries to the client), we provide measurements over 40 iterations. With secure sketch, we use 20 iterations and with 2PC protocols, we use 10 iterations because these take significantly longer to run.

Implementation. We use an AWS EC2 t2.large instance with 61 GiB of memory and 200 GiB disk storage for the server side computation. The instance is initialized with the deep learning AMI provided by Amazon. An AWS EC2 t2.small instance with 2 GiB of memory and 64 GiB storage in the same region acted as a client. The measured bandwidth between the two instances was 1 Gbits/sec for both directions and the network latency was 0.9 ms. The server side implementation uses Python and is parallelized using the GPU. The client side implementation uses Go. For the secure sketch protocol, we use an oblivious pseudorandom function (OPRF), implemented by the circl Go library from CloudFlare [27].

For the 2PC protocols our setup is identical except that we use an AWS EC2 t2.large instance with 8 GiB of RAM as the client to be able to handle the 2PC frameworks. The bandwidth was measured to be 1.01 Gbits/sec from server to client, 721 Mbits/sec from client to server. The observed network latency was 0.3 ms. For both the CrypTen and EMP frameworks, we used the computed functionality that checks if there exists a hash among the server’s input that has Hamming distance less than $T$ to the client-provided hash. The server’s input is the entire $B$, and the generated SBB bucket, in the non-bucketized and bucketized setting, respectively. We further XOR the output of this comparison with a randomly-generated client-provided bit so that only the client learns the result. For CrypTen, for simplicity we configured the trusted third-party for Beaver triple generation to run on the same EC2 instance as the client (so-called trusted first-party mode). This is not a secure configuration but provides lower bounds on performance (moving Beaver generation to another server would decrease performance). Experimental results for CrypTen should therefore be considered to be lower bounds on performance for secure deployments. Note that our 2PC prototypes are not optimized, and absolute timings would be improved using custom protocols for our setting such as the Kulshrestha-Mayer protocol [42]. However, it is unclear if the protocol can be combined with SBB since it requires generating all buckets at setup time.

Results. We present the average total execution time, average total bandwidth, and the speedup provided by SBB for varying $|B|$ in Table 3. The execution time and total bandwidth do not vary much between client requests that match images in $B$ and those that do not. Many of the similarity
testing protocols do not scale well, and we denote by dashes in the table experiments that failed to complete. Typically this was due to the client instance running out of memory. In all these cases, SBB was able to increase scaling to complete executions with the available resources.

Our results show that SBB drastically improves the similarity protocol’s performance, both in terms of execution time and total bandwidth. For similarity embedding retrieval, SBB provides a 29× speedup ($|B| = 2^{22}$) in execution time. For large-scale datasets ($|B| \leq 2^{23}$), similarity embedding retrieval with SBB returns the answer in real time, under 0.5 seconds. For the secure sketch protocol, the improvement is even larger, the speedup is at least $601 \times$ for $|B| = 2^{18}$. For 2PC protocols, the improvement provided by SBB grows larger as $B$ becomes bigger, when $|B| = 2^{16}$, the speedup in execution time is $67 \times$ and $97 \times$ for CrypTen and EMP, respectively. For $|B| = 2^{20}$, EMP with SBB takes less time than the Kulshreshtha-Mayer protocol (20.24s vs 27.5s), however it requires larger bandwidth.

### Table 3: Average time and bandwidth of similarity protocols without and with SBB for four different similarity testing protocols. Dashes (−) indicate when execution failed due to poor scaling. Numbers in parentheses are standard deviations.

| $|B|$ | CrypTen | EMP |
|------|---------|------|
| No SBB | SBB | No SBB | SBB | No SBB | SBB | No SBB | SBB |
| Execution Time (s) | Total Bandwidth (MiB) | Execution Time (s) | Total Bandwidth (MiB) | Execution Time (s) | Total Bandwidth (MiB) | Execution Time (s) | Total Bandwidth (MiB) |
| $2^{18}$ | 0.76 (0.00) | 0.02 (0.00) | 18.41 (0.00) | 0.21 (0.01) | 1664.98 (870.78) | 2.77 (0.17) | 78.81 (0.36) | 0.89 (0.05) |
| $2^{19}$ | 1.55 (0.01) | 0.03 (0.01) | 36.84 (0.02) | 0.46 (0.15) | 5702.44 (3049.44) | 6.03 (0.41) | 158.66 (2.39) | 1.90 (0.14) |
| $2^{20}$ | 3.09 (0.01) | 0.06 (0.02) | 73.65 (0.00) | 0.83 (0.06) | – | 12.09 (0.99) | – | 3.56 (0.26) |
| $2^{21}$ | 6.17 (0.01) | 0.10 (0.01) | 147.31 (0.01) | 1.69 (0.12) | – | 26.70 (2.40) | – | 7.18 (0.55) |
| $2^{22}$ | 12.41 (0.07) | 0.43 (0.99) | 294.61 (0.03) | 3.33 (0.25) | – | 61.99 (6.63) | – | 14.35 (1.17) |
| $2^{23}$ | – | 0.40 (0.02) | – | 6.70 (0.41) | – | 157.57 (14.41) | – | 28.40 (1.68) |

### 6 Related Work

**Secure proximity search.** Secure proximity search based on general multi-party computation has been used in many applications, including privacy-preserving facial recognition [24, 64], biometric authentication [8, 25], querying sensitive health data [7], and more. However, these works don’t scale sufficiently for our use case. More scalable solutions include fuzzy extractors [20, 38], which give a small piece of client information to the server, that does not leak information about the secret, in order to derive secrets from noisy readings such as biometrics. However, the security guarantee is based on the assumption that the distribution of the secret (for example, fingerprints) has enough minimum entropy, which is not necessarily true for image distributions.

**Private information retrieval.** Chor et al. [14] introduced the concept of private information retrieval, a type of protocol that enables a client to retrieve an item from a database such that the identity of the item is not revealed to the server. The protocol requires clients to supply the index of the data item that they are querying about for retrieval purposes. However, this is unlikely to be applied to our use case. The most likely solution for similarity lookup is content-based privacy preserving retrieval [46, 66, 75]. Previous works [46, 75] in this area assume three parties involved in this type of protocol, a data owner, the client who queries the service and an actual server where the service is hosted. The data owner encodes images or other types of multimedia into feature vectors that are further encoded with searchable encryption schemes and used as indices. The server is made oblivious to the actual content of indices and the content of corresponding data items. The client queries the server by generating indices from their images and receives the data items as answers. The threat model doesn’t prevent the case when the data owner colludes with the server, hence cannot be directly applied to our scenario, where the data owner is the server.

**Secure k nearest neighbors search.** Another possible solution to use secure k nearest neighbors search (k-NNS) to look for similar items at the server side without revealing client information. Most of the k-NNS solutions require a linear scan of the database that is queried against. Recent work [13] proposed a sub-linear clustering-based algorithm, yet the solution requires significant preprocessing time for each client. Similar to our work, Riazi et al. [60] used locality sensitive hashing to encode client queries for efficient k-NNS. Different from our approach, they preserved user privacy by...
converting the LSH encodings to secure bits. However, the
notion of security is different in their work, in particular, an
adversary can estimate the similarity of two given data points
based on the encodings. This makes the protocol vulnerable
to the matching attack, which we addressed in our work.

**Location-based services.** We draw comparisons between
our bucketization setting and that of location-based services
(LBSs). Andrés et al. [6] propose a privacy-preserving mech-
anism in which mobile clients send their noise-perturbed lo-
cations to a server in order to obtain recommendations for
nearby restaurants. One may view the noise-perturbed lo-
cation as a coarse embedding and the server-provided list
of restaurants as a similarity bucket. Similar to our coarse
embedding scheme, the mechanism of Andrés et al. suffers
from privacy loss when applied repeatedly to the same user
input. These connections suggest that our framework could
have applications to reasoning about LBS privacy. Conversely,
insights from location privacy may serve as inspiration for
improved SBB mechanisms.

**Privacy measures.** Prior work has proposed measuring pri-
vacy using an adversary’s expected error when making in-
ferrances based on a posterior distribution on user inputs
[55, 67, 68]. Recent work has explored the Bayes security
measure [12], which is similar to $\varepsilon_{acc}$, but involves a security
game in which the adversary attempts to recover a secret in-
put as opposed to guessing a predicate on the secret input.
Local differential privacy [22] has also proven to be a popular
worst-case privacy measure, but often incurs high correctness
penalties. Although similar, these metrics cannot be directly
applied to our scenario nor replace $\varepsilon_{auc}$ and $\varepsilon_{prev}^\rho$.

## 7 Limitations

Our work naturally suffers from several limitations that should
be explored further before deployments are considered. Most
notably, use of an SBB mechanism fundamentally must leak
some information to the server to trade-off client privacy
for efficiency. In some contexts leaking even a single bit of
information about user content would be detrimental, in which
case our techniques are insufficiently private. We speculate
that leaking some information about client images is, however,
fundamental to achieve practical performance in deployment
for large $B$. How to provide a formal treatment establishing
that scaling requires some leakage and what that means for
moderation mechanisms remain open questions.

Second, our empirical analyses focus on matching attacks
for a single query, which excludes some other potential threats.
In particular it does not address adversarially-known corre-
lations between multiple images queried by one or more
clients. A simple example, mentioned in Section 3, is an
‘averaging’ attack against our LSH-based coarse embedding
in which the adversary obtains a large number of embed-
dings all for the same image $w$. Then the adversary can av-
erage out the per-bit noise and recover the granular embed-
ding $\mathcal{F}(w)$. We discuss simulation results for this scenario
in the extended version [35]. The results indicate that, simi-
lar to privacy-preserving mechanisms for location-based ser-
VICES [6, 12, 55, 67], repeated queries on the same content
drastically weaken the privacy guarantee of SBB: an adver-
sary that sees multiple SBB outputs that it knows are for the
same image can obtain near-perfect matching attack precision
for almost all recall thresholds.

To address risk here, client software might cache images
they’ve recently queried. The client would not query the sim-
ilarity service if a new image is too close to a prior image,
and instead just reuse the cached result for the latter. Caching
may not be feasible in all cases, and doesn’t speak to cross-
user sharing of images, which may be inferrable from traffic
analysis should the adversary have access both to the simi-
larity service and the messaging platform. Another approach
would be to somehow ensure that the same noise is added
to the same image, regardless of which client is sending it.
This could possibly be done by having some clients share a
secret key, and use it to apply a pseudorandom function to
the image (or its PDQHash) to derive the coins needed for
the random choices underlying our coarse embedding. Here
an adversary’s $\varepsilon_{prev}^{\rho}$ advantage remains at 50% regardless of
the increasing number of repeated queries. But this doesn’t
account for other potentially adversarially-known correlations
across images (e.g., they are almost identical), and may be
fragile in the face of malicious client software. Moreover,
sending the same SBB embedding for the same image would
seem to increase susceptibility to linking attacks in which an
adversary infers when two or more queries correspond to the
same image. We are unsure which scenario bears more risk
in practice. We leave the exploration of these mitigations to
future work.

A related limitation is the exclusive use of empiricism for
evaluation. While we focus on Bayes-optimal adversaries, it
would be preferable to couple empiricism with analyses pro-
viding bounds on adversarial success. While our definition
al framework provides the basis for proving bounds on, e.g.,
precision for particular data distributions, we do not yet have
proofs and it appears to be challenging. We emphasize that
such results cannot fully replace empirical work, because even
formal results would necessarily make assumptions about data
that must be empirically validated. Nevertheless, we consider
the empirical results presented in this initial work as a proof-
of-concept of the SBB framework and encourage future works
to further examine the theoretical bounds for this approach.

Finally, the public perceptual hash algorithms that SBB
relies on increases the risk of evasion attacks that seek to
modify images just enough to avoid detection. This risk seems
particularly acute when using a similarity testing protocol
that sends a bucket of PDQHash values to the client, as the
adversary could extract these values from a client to inform
their attacks. Allowing users to report misinformation images
to frequently update the database may mitigate this risk.
8 Conclusion and Future Directions

In this paper, in order to allow efficient privacy-preserving similarity testing, we defined the framework of similarity-based bucketization and formalized a set of privacy goals that are important to this application. We consider the information that the adversary wants to infer from a client input as the answer to a prediction task. An adversary’s advantage is measured by their uncertainty regarding the prediction, using metrics that are widely applied in machine learning.

Towards a realistic prototype for SBB, we focus on image similarity testing. Driven by the privacy formalization, we ran simulations on real-world social media data and analyzed the SBB protocol’s security against a “matching attack”. The attack refers to the scenario where an adversary tries to infer if a client input is similar to an adversary-chosen image. Using our framework, deployments can tune the performance/privacy trade-off depending on the application context. We then test SBB’s performance when composed with four similarity protocols with varying server privacy guarantees for the server content. We show that the composition with SBB significantly reduces the protocol latency and required bandwidth. While further research is needed to address various open questions and limitations of our results, we nevertheless believe that SBB represents a promising approach to scaling private similarity testing in practice.

9 Acknowledgements

We thank Lucas Dixon, Nirvan Tyagi, and Paul Grubbs for useful discussions and feedback about this work. We thank Laurens van der Maaten and Brian Knott for answering our questions regarding CrypTen, and Xiao Wang for answering questions about EMP. This research was supported in part by NSF award #1704527.

References


[50] India Mckinney and Erica Potnoy. Apple’s plan to “think different” about encryption opens a backdoor to your private life. The Electronic Frontier Foundation, 2021.


[66] Jagarlamudi Shashank, Palivela Kowshik, Kannan Srinathan, and CV Jawahar. Private content based image


