1. Supplementary Material

1.1. Security proofs

We prove our protocols secure in the Universal Composability (UC) framework introduced by Canetti [3], as is standard practice for cryptographic protocols involving multiple parties. Protocols secure in the UC framework are secure even when arbitrary instances of the same or other protocols are run in serial or parallel, which is not the case for standalone security. We assume the reader is familiar with proofs in this framework, and consequently provide only a high-level review in this subsection.

UC security is proved as follows. First, one defines an ideal version of the protocol called the functionality, which acts as a trusted entity interacting with all parties to achieve some specified outcome. Then one creates a protocol between the parties that approximates the ideal version to achieve the same goal. Finally, one shows that for any adversary against the protocol, there is a simulator against the functionality such that any environment, in which the adversary operates, cannot tell if it is interacting as in the real-world protocol execution or in the ideal-world functionality execution. The inability to distinguish between these executions implies no adversary can learn anything from a protocol execution that cannot be learnt when instead interacting with the functionality, which is secure by construction.

At a high level, the UC composability guarantee says interacting with a trusted functionality is equivalent to executing a protocol that UC-securely realises it. This means the UC framework allows for cleaner exposition by supporting a modular approach to building complex protocols. When developing a protocol Π to realise functionality $\mathcal{F}$, we may make calls to any ideal functionality $\mathcal{F}'$ that can be UC-securely realised by some subprotocol Π'. This simplifies the process of proving security for Π since we no longer need to reason about computation in Π', instead dealing with ideal calls to $\mathcal{F}'$.

Throughout, we refer to the client performing queries as the 'receiver', and the data owners as 'senders', following standard conventions in the literature for comparable multi-party protocols.

Notation. We use κ to denote the computational security parameter. A variable x is assigned a value X by the notation $x \leftarrow X$. For a set X, we denote its cardinality by $|X|$. An arbitrary field is denoted by $\mathbb{F}$ and an n-dimensional column vector by $\vec{v} \in \mathbb{F}^n$. For a matrix $M \in \mathbb{F}^{m \times n}$ and a vector $\vec{v} \in \mathbb{F}^n$, we denote by $M \cdot \vec{v}$ the standard matrix/vector multiplication.

1.2. Communication Mechanism

Since the messaging system is not our focus, we do not provide the protocol and prove its security. The functionality we describe, given in Figure 1, is very weak (the adversary has considerable power: there is no guaranteed message delivery, authentication, or confidentiality) and far stronger messaging protocols have been implemented and proved UC secure, e.g. the Signal protocol [4]. We do not require authentication because it allows data owners not to reveal their identities until queriers request specific images. We essentially rely on the fact that the client messages are ciphertexts for confidentiality, and that if these ciphertexts are corrupted then the clients will usually halt the protocol.

$\underline{\mathcal{F}_{\mathsf{PubSub}}}$

The ideal-world adversary $\mathcal{S}$ may corrupt any entity at any point and prevent publishing any message or delivery of a message to a subscriber.

Initialise On input a message $(\texttt{sid}, \texttt{init})$, if it is the first message received, store `sid` and ignore any further messages with a different `sid`, and initialise an empty record $R \leftarrow \varnothing$.

Subscribe On input a message $(\texttt{sid}, \texttt{subscribe}, p)$ from player with identity `id`, store the entry $(\texttt{id}, p)$ in R.

Publish On input a message $(\texttt{sid}, \texttt{publish}, m, p)$ from any player, let $I = \{\texttt{id} : \exists(\texttt{id}, p) \in R\}$ and send the message $(\texttt{sid}, \texttt{new\_message}, m, p)$ to all players in I.

Figure 1. Functionality $\mathcal{F}_{\mathsf{PubSub}}$

$\underline{\mathcal{F}_{\mathsf{Mv}}}$

Query Await a message $(\texttt{query}, \vec{v})$ from the receiver and then do the following for each sender, indexed by $\texttt{id}$:

- Await a matrix $M_{\texttt{id}} \in R^{m \times n}$, where R is a ring, and if they are corrupt await errors $\vec{\varepsilon_{\texttt{id}}}$ from $\mathcal{S}$.
- Compute $\vec{v'_{\texttt{id}}} \leftarrow M'_{\texttt{id}} \cdot \vec{v} + \vec{\varepsilon_{\texttt{id}}}$.
- Send $\vec{v'_{\texttt{id}}}$ to the receiver.

Figure 2. Matrix / vector multiplication functionality

1.3. Matrix / Vector Multiplication Protocol

Our distributed search protocol involves the use of (a variant of) the functionality $\mathcal{F}_{\mathsf{Mv}}$ (see Figure 2) between one client 'receiver' and multiple data owner 'senders' that allows the receiver to obtain the image of their vector under a linear map (matrix under some choice of basis) held by the senders.

The protocol for $\mathcal{F}_{\mathsf{Mv}}$ could be built in many ways. One possibility is (Vector) Oblivious Linear function Evaluation ((V)OLE), which exactly enables one party to learn the image of their input under an affine linear function chosen by the other [1]. However, we prefer a solution that minimises communication rounds and allows multiple data owners to respond to a single client query so that the client query communication is a oneshot message. This better reflects the asynchronous nature of real-world distributed communication where users are not always online at the same time. VOLE protocols would also generally require preprocessing per client on the part of the data owners. Nevertheless, we leave investigation of a VOLE-based protocol as an interesting direction for future work. New directions in Private Set Intersection (PSI) and Private Information Retrieval (PIR) protocols may also prove fruitful for our use case [5].

Unfortunately, we cannot prove UC security of the 'clean' $\mathcal{F}_{\mathsf{Mv}}$ functionality using our homomorphic encryption approach, essentially because when senders are corrupt, the simulator is unable to extract the adversary's matrices without rewinding (which is forbidden in the UC framework). To sidestep this difficulty, we instead define a modified functionality $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$ that leaks cryptographic artefacts (public keys and ciphertexts) to the simulator. (The same approach is taken for the proofs in the celebrated SPDZ protocol [6, 7].) We would expect to be able to bootstrap our protocol to satisfy the cleaner functionality $\mathcal{F}_{\mathsf{Mv}}$ by adding commitments and zero-knowledge proofs (ZKPs), but our implementation would then have to use black-box ZKP techniques due to the limited availability of software libraries, which would significantly increase the computational overhead of our protocol, compromising our goal of billion-scale search. Since ZKPs for lattice-based cryptography are being actively researched (e.g. [8]), we anticipate that, in time, efficient proofs will become available to allow us to create a protocol for the cleaner functionality.

We emphasise that the end goal of our distributed search protocol is for data owners to sell the rights to images for successful matches, so they are generally incentivised to behave honestly (a client will stop the interaction if the matches are not close enough to their input message). The functionality $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$ therefore only (implicitly) offers passive security; in particular, the adversary could use the public parameters provided with a query ciphertext to encrypt *any* vector as a response. Indeed, an adversarial data owner can always refuse to hand over data items to the client at the last step. There does not, however, appear to be a reasonable incentive for doing so except to disrupt the system, which can be achieved more easily via non-cryptographic attacks (such as denial-of-service attacks). Mitigating these problems in an *asynchronous* protocol seems to be a complex task, potentially requiring commitments and Zero-Knowledge Proofs (ZKPs) and/or escrow systems (e.g. smart contracts). By contrast, our approach is extremely lightweight and scalable.

The advantage of realising $\mathcal{F}_{\mathsf{Mv}}$ rather than $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$ would be that our proof would admit a reduction to the problem of breaking IND-CPA security of the homomorphic encryption scheme [2] directly; at present, the functionality is vulnerable to non-black-box attacks on the encryption scheme. In brief, realising $\mathcal{F}_{\mathsf{Mv}}$ requires an IND-CPA homomorphic encryption scheme because the simulator needs to send a ciphertext to the adversary on behalf of an honest receiver without revealing the fact that the plaintext does not correspond to the receiver's input vector (since this is not revealed to the simulator by $\mathcal{F}_{\mathsf{Mv}}$).

In our description of $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$, the client is the receiver, and the data owners are the senders. For simplicity of exposition, we assume that a ciphertext $\vec{c}$ is a data object that includes any public information required to manipulate it under the homomorphic encryption scheme (e.g. ring moduli). Our protocol Π_{Mv} to realise $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$ in the $\mathcal{F}_{\mathsf{PubSub}}$-hybrid model is given

$\underline{\mathcal{F}^{\kappa}_{\mathsf{Mv}}}$

The functionality is parameterised by a computational security parameter κ, which bounds the environment's computational complexity. Once the functionality has been initialised on receipt of any message, the `sid` from this message is used as the session ID and all future messages with different `sid` are ignored. If an honest party receives the message $\perp$ at any point, they abort the protocol.

Query If the receiver is honest, do the following:

- Await a message $(\texttt{sid}, \texttt{query}, \vec{v})$ from the receiver, where $\vec{v} \in \mathbb{C}^n$ for some n.
- Sample a new key pair $(\texttt{pk}, \texttt{sk}) \leftarrow \texttt{KeyGen}(\kappa)$ for a homomorphic encryption scheme and leak a ciphertext $\vec{c} \leftarrow \texttt{Enc}_{\texttt{pk}}(\vec{v})$ to $\mathcal{S}$. (Recall that we assume the ciphertext data structure embeds public parameters including the public key $\texttt{pk}$.)
- Await input matrices $M_{\texttt{id}}$ from honest senders and compute $\vec{v}_{\texttt{id}} \leftarrow M_{\texttt{id}} \cdot \vec{v}$.
- Await response vectors $\vec{c'_{\texttt{id}}}$ from $\mathcal{S}$ on behalf of each corrupt sender (if any), and attempt to decrypt, $\vec{v}_{\texttt{id}} \leftarrow \texttt{Dec}_{\texttt{sk}}(\vec{c'})$.
- Await a set C of indices of honest senders whose messages $\mathcal{S}$ wants to corrupt, and send $(\vec{v}_{\texttt{id}}, \texttt{id})$ to the receiver (for a chosen k), where $\vec{v}_{\texttt{id}}$ may be $\perp$ if decryption failed or if $\texttt{id} \in C$.

If the receiver is corrupt and there is at least one honest sender, await a message $(\texttt{sid}, \texttt{query}, \vec{c})$ from $\mathcal{S}$ on behalf of the receiver and then for each honest sender with identity `id`, do the following:

- Await a matrix $M_{\texttt{id}}$ from the sender.
- Await a ciphertext $\vec{c_{\texttt{id}}}$ from $\mathcal{S}$.
- Compute $\vec{c'_{\texttt{id}}} \leftarrow M_{\texttt{id}} \cdot \vec{c}$.
- Send $\vec{c'_{\texttt{id}}}$ to $\mathcal{S}$.

Figure 3. Functionality $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$

in Figure 4.

Proposition 1. *The protocol Π_{Mv} UC-securely realises $\mathcal{F}^{\kappa}_{Mv}$ with abort, with computational security κ, in the $\mathcal{F}_{PubSub}$-hybrid model.*

Proof. To prove the theorem, we must show that for any adversary $\mathcal{A}$ against Π_{Mv}, there is a simulator $\mathcal{S}$ against $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$ such that the view of the environment $\mathcal{Z}$ is indistinguishable between the two cases. We construct the simulator as follows. The simulator executes a copy of $\mathcal{F}_{\mathsf{PubSub}}$, runs the **Initialise** procedure, and responds to future queries as the functionality would.

Corrupt receiver If the receiver is corrupt, the simulator acts as a relay between $\mathcal{A}$ and $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$. If there is at least one honest sender, $\mathcal{S}$ proceeds as follows. The simulator awaits some ciphertext $\vec{c}$ from $\mathcal{A}$ in the call $(\texttt{sid}, \texttt{publish}, (\vec{c}\|\texttt{qid}), \texttt{PUBLIC})$ to $\mathcal{F}_{\mathsf{PubSub}}$ on behalf of the corrupt receiver. The simulator forwards $\vec{c}$ to $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$ via the message $(\texttt{sid}, \texttt{query}, \vec{c})$. and sends the message $(\texttt{sid}, \texttt{new\_message}, (\vec{c}\|\texttt{qid}), \texttt{PUBLIC})$ to $\mathcal{S}$ if there are any corrupt senders, as $\mathcal{F}_{\mathsf{PubSub}}$ would. For each honest sender, indexed by `id`, the simulator receives back one ciphertext $\vec{c'_{\texttt{id}}}$ from $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$, samples $\texttt{rid}_{\texttt{id}}$ as an honest sender would, and sends the message $(\texttt{sid}, \texttt{publish}, (\vec{c}, \texttt{rid}_{\texttt{id}}), \texttt{qid})$ to its copy of $\mathcal{F}_{\mathsf{PubSub}}$, sending the resulting messages to $\mathcal{A}$ as in an honest execution.

Honest receiver If the receiver is honest and any sender is corrupt, the simulator awaits $\vec{c}$ from $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$. (If no senders are corrupt, there is nothing to simulate.) The simulator samples `qid` as an honest receiver would and sends the message $(\texttt{sid}, \texttt{publish}, (\vec{c}\|\texttt{qid}), \texttt{PUBLIC})$ to its copy of $\mathcal{F}_{\mathsf{PubSub}}$ and sends the resulting messages to $\mathcal{A}$ as in an honest execution. On receipt of a message $(\texttt{sid}, \texttt{publish}, (\vec{c'}\|\texttt{rid}), \texttt{qid})$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{PubSub}}$, the simulator sends $\vec{c'}$ to $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$. If $\mathcal{S}$ does not receive a message on behalf of any corrupt sender, $\mathcal{S}$ sends the corresponding party sender indices to $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$ as the set C.

Π_{Mv} in the $\mathcal{F}_{\mathsf{PubSub}}$-hybrid model

Initialise The senders and the receiver do the following:

- Send the message $(\texttt{sid}, \texttt{init})$ to $\mathcal{F}_{\mathsf{PubSub}}$.
- Subscribe to some public channel to which queries will be published by sending the command $(\texttt{sid}, \texttt{subscribe}, \texttt{PUBLIC})$ to $\mathcal{F}_{\mathsf{PubSub}}$.

Query The senders and receiver do the following:

- The receiver samples a new query ID $\texttt{qid}$ and subscribes to the relevant channel by sending the message $(\texttt{sid}, \texttt{subscribe}, \texttt{qid})$ to $\mathcal{F}_{\mathsf{PubSub}}$.
- The receiver with input vector $\vec{v}$ samples a new key pair $(\texttt{pk}, \texttt{sk}) \leftarrow \texttt{KeyGen}(\kappa)$.
- The receiver computes $\vec{c} \leftarrow \texttt{Enc}_{\texttt{pk}}(\vec{v})$ and sends the message $(\texttt{sid}, \texttt{publish}, (\vec{c} \| \texttt{qid}), \texttt{PUBLIC})$ to $\mathcal{F}_{\mathsf{PubSub}}$.
- The senders await the message $(\texttt{sid}, \texttt{new\_message}, (\vec{c} \| \texttt{qid}), \texttt{PUBLIC})$ from $\mathcal{F}_{\mathsf{PubSub}}$ and sample a response ID $\texttt{rid}$.
- The senders compute $\vec{c'} \leftarrow M \cdot \vec{c}$ using their input M, and send the message $(\texttt{sid}, \texttt{publish}, (\vec{c} \| \texttt{rid}), \texttt{qid})$ to $\mathcal{F}_{\mathsf{PubSub}}$.
- On receipt of any message $(\texttt{sid}, \texttt{new\_message}, (\vec{c} \| \texttt{rid}), \texttt{qid})$, the receiver computes $\vec{v'} \leftarrow \texttt{Dec}_{\texttt{sk}}(\vec{c})$ and (locally) returns $\vec{v'}$ or $\perp$ if decryption fails.

Figure 4. Protocol Π_{Mv}

Indistinguishability We now argue that the view of the environment interacting with the simulator and the ideal world instance is computationally indistinguishable from the view when performing Π_{Mv}. First, note that since $\mathcal{F}_{\mathsf{PubSub}}$ is honestly simulated, there is no way to distinguish between worlds using its behaviour.

When the receiver is corrupt, the simulator simply relays messages between the adversary and $\mathcal{F}_{\mathsf{Mv}}^{\kappa}$, so the distributions are identical in both worlds. When the receiver is honest, computations of the simulator and functionality together are the same as what an honest receiver does in Π_{Mv}, so the revealed ciphertexts are identically distributed in this case too. Note that the ciphertext is leaked to the adversary regardless of whether any parties have been (adaptively) corrupted by this point in the execution. This is because the ciphertext is published publicly using $\mathcal{F}_{\mathsf{PubSub}}$. This allows us to simulate even adaptive corruptions, where the adversary does not initially corrupt any sender. □

To instantiate our final protocol using $\mathcal{F}_{\mathsf{Mv}}^{\kappa}$, receivers (queriers) and senders (data owners) use a vision-language model to generate embedding vectors for the query and datasets as their inputs. When the querier has received a vector of inner products as the output, they can request data items corresponding to the indices of close matches (where they decide for themselves how close a 'successful' match is, and how many items to request).

1.4. Privacy Leakage

The UC proof shows that Π_{Mv} securely realises $\mathcal{F}_{\mathsf{Mv}}^{\kappa}$; we now briefly clarify what each party observes in the protocol to demonstrate that Π_{Mv} combined with the matrix row permutation described in Sec. 4.4 ensures the client cannot learn anything about the data owner's dataset.

Note that once a client has received an item from the dataset, they can freely recompute the embedding vector if they choose (if an open-weight VLM was used); embedding vector privacy is not required at this point since a corrupt client now knows the data item itself and the data owner has already 'sold' their content.

First, we consider the privacy of the client from the data owner. The data owner only observes the client's ciphertexts in the protocol, and receives a final set of indices for items the clients has selected. In our protocol, the client uses one SHE key per 'session' (query), and this key is used a number of times that is logarithmic in the size of the dataset. As such, the data owner is extremely limited in types of attack they can mount on the encryption scheme to determine the client's input vector (e.g. selective failure). An adversary could pretend to be multiple data owners to increase the number of ciphertexts they can test, but data owners do not know if a client has ceased computation because they have not found any 'reasonably close' match, because they received a ciphertext they could not decrypt, or because they simply chose to abort the protocol. Moreover, if it was found in a real deployment that many clients were receiving ciphertexts they could not decrypt, disincentives such

as nominal costs for posting and receiving messages could be added. Another mitigation could be for clients to respond occasionally even if they cannot decrypt data owner responses to hide decryption failure. The need for these mitigations depends on what behaviours are observed in a real deployment. Adding ZKPs for CKKS (as discussed in the paper body) could be a low-effort way to prevent these kinds of attack in a future iteration of PRISM.

We now consider the privacy of the data owner from the client. The client submits a ciphertext $\vec{c}$ of an embedding vector $\vec{a} \in \mathbb{F}^n$ and receives back a vector $\vec{c'}$ from a data owner. After decryption, the client learns a vector $\vec{b} \in \mathbb{F}^m$, which can be viewed as the inner product of their input vector $\vec{a}$ with each row of the data owner's matrix $M \in \mathbb{F}^{m \times n}$. If the client submits several linearly independent vectors $(\vec{a_i})_{i=1}^m$, then they can theoretically learn some subspace of the rowspace of M (i.e. the embedding space of their inputs), and possibly partially reverse-engineer the data owner's dataset as described in Sec. 4.4.

To mitigate this issue, we have prescribed that the data owner permute the rows of the matrix on each invocation of $\mathcal{F}^{\kappa}_{\mathsf{Mv}}$. We also require that M be padded with random vectors up to the ciphertext packing parameter ($4,096$ in our implementation) so this permutation is always meaningful.

The best strategy for the client would appear to be to submit low Hamming-weight vectors to learn small linear combinations of columns of M and from these attempt to reconstruct embedding vectors. Supposing the client submits standard basis vectors (e.g. $(0, 0, 1, 0, 0, \ldots, 0)$ to obtain columns of M, there are $4,096$ choose 512 (i.e. approximately 3.16×10^{668}) vectors to construct to find 4096 embedding vectors. While embedding vectors are clearly readily distinguishable from random vectors, we believe the task of determining embedding vectors from these known values to be suboptimal compared to

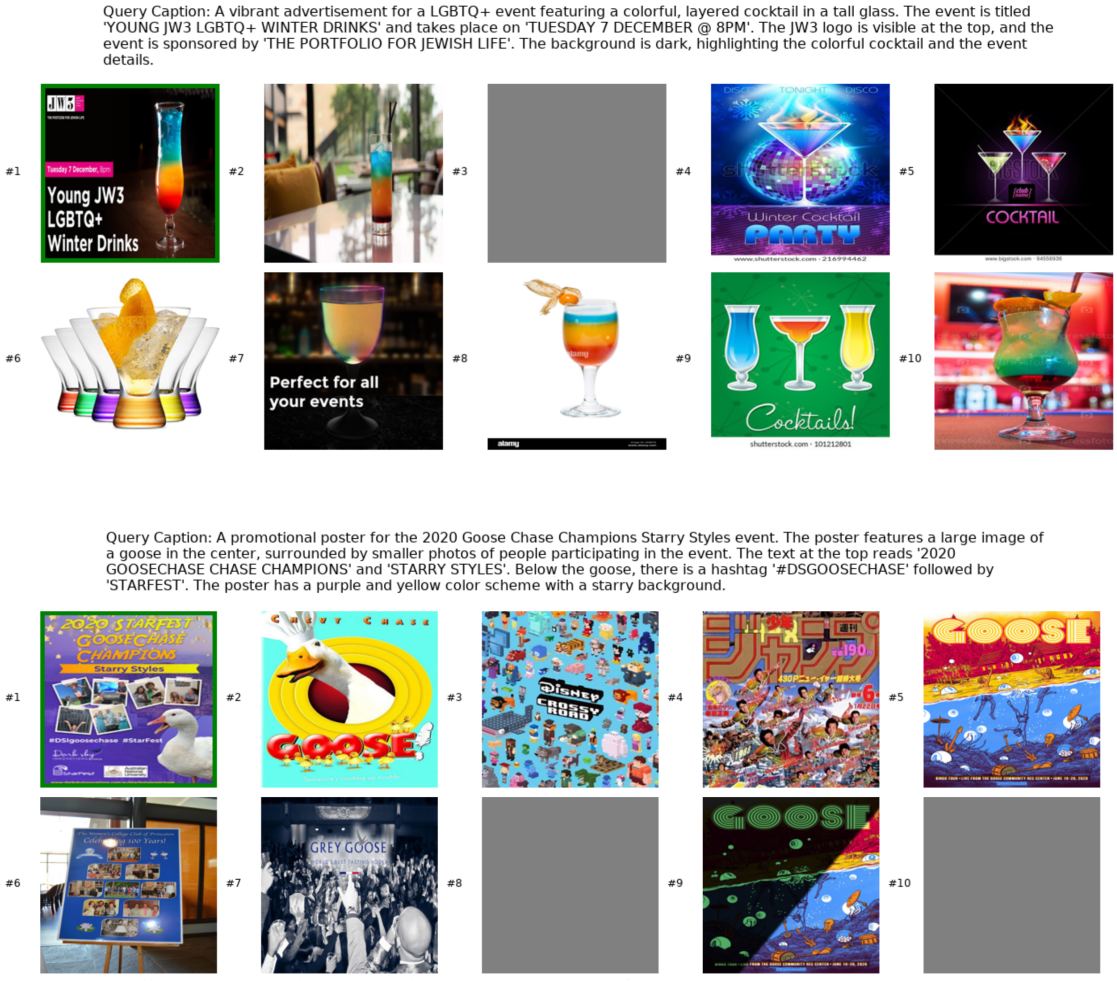

Figure 5. Examples of **correct retrievals** where the top-ranked result matches the ground truth image for the given query. *Note:* Gray images indicate items that were correctly retrieved by the index but were no longer available at their source URL at the time of visualization (link rot).

Figure 6. Examples of **failed retrievals** where the system did not return the exact ground truth image. Despite being classified as incorrect, the retrieved images often exhibit high semantic relevance to the text query, highlighting the robustness of the fuzzy search capability.

non-cryptographic ways to attack the system (e.g. spoofing, human factors, etc.). There may be other much better attack strategies, but it is not clear to the authors what these might be as the dimensions of the vectors involved are considerable.

1.5. Qualitative Results

In this section, we provide visual examples of the system's retrieval performance on the Recap-DataComp-1B dataset. Figure 5 demonstrates successful exact matches, while Figure 6 illustrates cases where the exact ground truth was missed, but the retrieved content remains semantically aligned with the query. Note: Gray images indicate items that were correctly retrieved by the index but were no longer available at their source URL at the time of visualization (link rot).