2022 19th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON) | 978-1-6654-8643-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SECON55815.2022.9918552

Co-Cache: Inertial-Driven Infrastructure-less Collaborative Approximate Caching

James Mariani
Michigan State University
mariani4@msu.edu

Yongqi Han
Michigan State University
hanyongq@msu.edu

Li Xiao
Michigan State University
lxiao@cse.msu.edu

***Abstract*—Many emerging multimedia mobile applications rely heavily upon image recognition of both static images and live video streams. Image recognition is commonly achieved using deep neural networks (DNNs) which can achieve high accuracy but also incur significant computation latency and energy consumption on resource-constrained smartphones. Recent efforts addressing these issues include cloud offloading and reducing the complexity of the DNNs, which, however, introduce increased network latency or reduced accuracy. In-memory caching has also been explored to assess the similarity of images as opposed to exact matching. However, such approximate caching systems often treat devices as static nodes, and do not fully utilize the mobile and collaborative nature of smartphones without outside infrastructure. Another consequence of treating nodes as static is the necessity of cache sizes larger than what is feasible for individual mobile applications.**

In this paper we introduce Co-Cache, a in-memory caching paradigm that supports infrastructure-less collaborative computation reuse in smartphone image recognition. Co-Cache utilizes the inertial movement of smartphones, the locality inherent in video streams, as well as information from nearby, peer-to-peer devices to maximize the computation reuse opportunities in mobile image recognition. Compared to other caching systems, our extensive evaluation shows that Co-Cache can reduce the required number of cache entries by 50-70% while lowering the average latency of standard image recognition applications by up to 94% with minimal loss of recognition accuracy.

I. Introduction

Emerging multimedia mobile applications are focusing on technology interacting with, and augmenting the real-world environment the user occupies. This interaction with the real-world relies heavily on image recognition. For example, augmented reality applications span navigation [27], gaming [2], education [29], etc. Contextual recognition and cognitive assistance applications, including Google Lens, constantly analyze the world around them in an effort to categorize images and offer users additional information and resources. These applications often require near-instant recognition to fulfill the latency and computation requirements needed to offer the user a seamless experience. A major obstacle these applications face in offering low latency is the inherent resource constraints of even the most modern and expensive smartphones.

Most techniques for mitigating the latency on smartphones include offloading to cloud or edge servers [4], [11], [25], [36], reducing the complexity of the deep neural netoworks (DNNs) often used for mobile image recognition [10], [15], [38], and caching results for reuse [17], [18]. Offloading can greatly reduce the computation requirements of the smartphone, however, also introduces significant networking latency and requires a reliance on outside infrastructure [35]. Reducing the complexity of a DNN can also reduce the latency and computational requirements of image recognition, but at the cost of reduced accuracy [9], [22], [34].

Caching for image recognition attempts to assess the similarity of raw images or video frames to allow for computation reuse. This paradigm of caching using similarity instead of exact matching is called approximate caching. In such a system, if an image is determined to be similar enough to a previously recognized image, the result of the previous classification is used and a neural network recognition is avoided. The use of approximate caching techniques is required for real-world recognition situations because as a user is moving around in the real-world they are capturing 30-60 raw images per second that each need recognition. To aid in the assessment of similarity, the cache structure of an approximate cache is often a Locality Sensitive Hash (LSH) or a KD-tree.

Current work in approximate in-memory caching for mobile image recognition focuses on optimizing the underlying data structure of the cache as well as the searching algorithms used to query the cache [17], [18]. The approximate nature of an LSH or KD-tree can often lead to false cache misses and missed reuse opportunities without this optimization. However, most current work in approximate caching for smartphone image recognition uses established cache replacement and insertion strategies that do not consider the inertial movement of devices. This leads to two issues: the necessity of very large caches, and the correct values not being cached or have been prematurely replaced in the cache. Large caches of 500+ [17] entries can consume 7-10 MB, and might be acceptable on powerful computers, however, smartphones have significant resource constraints and sometimes allow each individual application to use only 2 MB of memory [3], making current work infeasible in real-world situations. Each Android device is different, and while some expensive devices might offer large amounts of memory for individual applications, cheaper devices with limited memory and older versions of Android place significant restraints on the amount of memory per application. Our work is, however, not only important on current devices. As smartphones increase their memory, applications also increase their resource needs, requiring careful memory management both now and in the future.

In this paper we present Co-Cache, an approximate caching system that introduces caching strategies built around the inertial movement of mobile devices and allows for infrastructure-less collaboration between nearby devices through creation of ad-hoc peer-to-peer networks. In contrast to current work,

Co-Cache optimizes cache management using the inherent mobility and collaborative nature of smartphones. Co-Cache offers a suite of inertial-driven optimization tools, including predictive interpolative caching to automatically cache values before they are needed, peer-to-peer collaborative pre-caching, reuse scoring, and inertial-driven similarity thresholds to both reduce the latency of image recognition and maintain high accuracy. Our optimization tools allow us to achieve these results while requiring a cache size of only 150-200 entries, which allows Co-Cache to be deployed in real-world smartphones with no memory issues.

II. Motivation of Co-Cache

A. Motivating Use Cases

Many popular mobile applications are vision-based and rely heavily on image recognition. Two of the main categories include augmented reality (AR) and cognitive assistance. Augmented reality is a technology that recognizes the environment through a camera and overlays virtual annotations and objects on a user's view of the real world. Throughout this paper we will explore and evaluate how Co-Cache can be used to provide a smoother user experience for cognitive assistance apps. Cognitive assistance applications provide the user with assistance in their everyday life through helpful information, instructions, and even directions.

These applications operate on a continuous stream of raw images from a camera. As such, it must perform image recognition to provide the user with useful information. Additionally, these applications are often used while on the go and in crowded areas, and therefore can be enhanced with Co-Cache's inertial-driven collaborative approach to aid in achieving a lower latency without sacrificing accuracy.

B. Similarity in Video Streams

Approximate caching is built on the idea that raw images from a video stream or from nearby users are often similar, but rarely exactly the same. A traditional cache that only performs exact matching would be of little to no use in this scenario, which provides an opportunity for approximate caching to take advantage of the temporal, spatial, and semantic similarity inherent in video streams.

Temporal similarity is the tendency for video frames near to each other in time to be similar to one another. Spatial similarity is a principle that says that physical locations near to each other are similar, and mainly differ in angle of viewing and lighting conditions. Semantic similarity shows that different instances of the same object appear in multiple locations and situations.

Co-Cache utilizes a smartphone's inertial movement and the **temporal similarity** in video streams to adaptively optimize our approximate similarity matching to capture as many high-probability reuse opportunities in successive video frames without a loss of accuracy. Co-Cache also uses the rotation of a smartphone along with known **spatial similarity** principles to predict when a cache miss is likely to occur and proactively inserts entries into the cache that will avoid a cache miss and reduce latency. Co-Cache offers infrastructure-less collaboration between users through ad-hoc Wi-Fi peer-to-peer networks and neighbor tracking through Bluetooth proximity detection. Our ad-hoc approach to collaboration and tracking allows us to fully utilize the **semantic similarity** present in video streams through nearby users sharing known valuable recognitions that allow users to recognize popular objects without ever needing a full DNN recognition.

III. Co-Cache Design

A. Overview

Co-Cache is an in-memory caching system that facilitates the reuse of image recognition computations to improve the overall latency and efficiency of DNN image recognition on mobile devices. Co-Cache leverages the collaborative nature and inertial movement of mobile devices, as well as the temporal, spatial, and semantic locality of real-world video feeds, to optimize cache searching and ensure that the correct computations are cached for reuse.

The core of Co-Cache's contributions lie in the Local Cache Management Module, and the Collaborative Network Communication Module, seen in Figure 1. The Local Cache Management Module handles all cache searching, cache insertion and replacement, sampling the on-device accelerometer, gyroscope, and magnetometer, and updating the cache entry reuse scores. The Collaborative Network Communication Module houses both the Wi-Fi Direct and Bluetooth components of Co-Cache. This includes the Bluetooth proximity detection used for sensing movement between users, and the Wi-Fi Direct peer-to-peer logic for actually sending over cache entries based on the Bluetooth proximity readings.

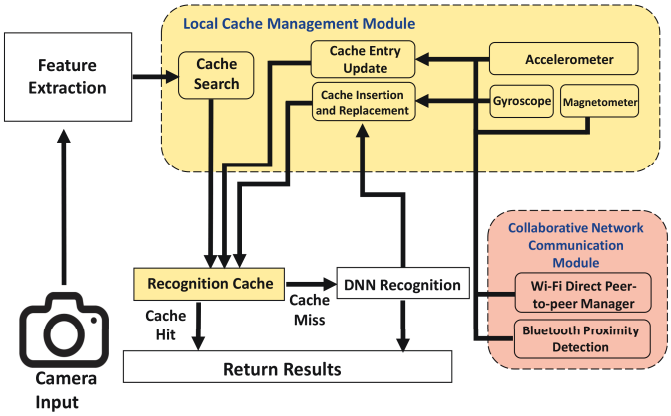

Fig. 1. System Architecture of Co-Cache

1) Background: Locality Sensitive Hashing: The underlying data structure of Co-Cache is a Locality Sensitive Hash (LSH). The keys of our approximate cache are feature vectors of raw input images, and the value of a cache entry is the class label given by the neural network recognition. An LSH is widely used for nearest matching in high-dimensional space [5]. The LSH contains multiple hash tables that each utilize a unique hash function. Each hash table also employs a set of buckets so that similar data will be hashed into the same bucket. Each of these buckets represent a sense of "locality," in

that similar images will all be hashed into the same bucket with a high probability. The cache entries present in the buckets from each hash table are then aggregated into a candidate set, from which the final output is selected based on a k-nearest neighbor search or a simple distance measure.

2) Background: Heading and Step Estimation: Traditional heading estimation and step detection are often used for localization and prioritize accuracy [31]. Heading estimation is a process to determine which direction a user is facing, and step detection determines whenever a step occurs, implying that the user is moving. However, as discussed previously, we do not need perfectly accurate step detection and heading estimation to achieve the goals of Co-Cache and therefore take an approximate approach to reduce computational complexity, which includes reducing filtering, correction algorithms, and real-world transformations for heading estimation and forgoing step length estimation for step detection.

B. Local Cache Management Module

1) Key Computation: The cache keys in Co-Cache are variable-length feature vectors generated from input images. As our system runs on live video streams, each input image is a single frame from a video, and the corresponding cache key for any particular video frame is a feature vector that describes the image. Co-Cache supports any of the popular feature extraction algorithms, including SIFT [26], SURF [6], ORB [30], HoG [12], Harris [19], etc. The cache value is the label of the object recognized in the raw image. The only requirement when choosing a feature extraction algorithm is that there is an easily quantifiable concept of distance between two vectors. This distance measure is essential to the cache searching that uses a Locality Sensitive Hash (LSH) as well as for similarity scores used throughout various components of Co-Cache. Comparing extracted features of an image as opposed to comparing images directly allows us to more easily assess similarity between two images than if we compared the two raw images themselves.

2) Inertial-Driven Similarity Threshold: Co-Cache's inertial-driven similarity threshold updates the initial similarity threshold for matching images based on the inertial movement of the device as well as the temporal similarity that exists in the real-time video feed. The similarity threshold is what ultimately decides if any cache lookup results in a hit or a miss depending on how similar the current image is to the values in the cache. There is a clear tradeoff between the similarity threshold and the cache hit rate and error rate. A larger similarity threshold results in more cache hits, which reduces latency and computations, but also results in more errors. Existing work [18] attempts a threshold tuning algorithm, however, as their system ignores inertial movement their input images have no temporal similarity and their algorithm requires 100+ entries to have any effect. This is not feasible in a real-world scenario as it can take tens of minutes to recognize 100 unique objects and most smartphones do not have adequate memory to dedicate over 100 entries for threshold tuning. Co-Cache, however, requires only one positive recognition to show significant results of our inertial-driven similarity threshold, making it a realistic solution for live video feeds.

Algorithm 1: Inertial-Driven Similarity Threshold

Result: Updated Similarity Threshold

//Initial threshold chosen by user;
initialize thresh;
initialize init $\leftarrow$ thresh;
initialize α, β, $\Delta frame \leftarrow 0$;
//degree set to heading during recognition;
initialize degree ;
while *cache search does not miss* **do**
  $\Delta frame \leftarrow \Delta frame + 1$;
  $\Delta degree \leftarrow headingdifference$;
  $thresh \leftarrow MAX((thresh/\alpha + (thresh - (MAX(\Delta frame, \Delta degree)/\beta)/thresh)/2), init)$;
  if $diff(currframe, cacheentry) < thresh$ **then**
    return $originalframeresult$;
  else
    $LSHcachelookup()$;
    break;
  end
end

The process of loosening and tightening the similarity threshold can be seen in Algorithm 1. The main idea is that after any DNN recognition, the object that was recognized is highly likely to be seen in many consecutive frames of the video. As such, the similarity threshold is loosened significantly in the subsequent frames of the video and as the number of frames since the DNN recognition grows, the similarity threshold tightens back towards the original threshold. This takes full advantage of the temporal similarity that exists in consecutive frames of a live video feed.

Our evaluation in Section IV-C shows that there is a high probability that an object stays in the view of the camera for at least 10 frames, and usually 30+ frames. This observation is consistent with results in [17]. Because of this, the inertial-driven similarity updating algorithm is designed to capture as many reuse opportunities within the first 30 frames and then return to normal operation until the next new object is recognized. Coincidentally, the difference in heading between a user's view and their initial heading when recognizing an image follows the same principal as the sequence of frames. If the current heading of the user is within 10-30 degrees of the initial heading, then there is a high probability that the user is viewing the same object.

Our inertial-driven similarity threshold algorithm takes the initial similarity threshold, the larger value between heading difference and number of frames since initial recognition, and also a growth parameter, α, and dampening factor, β, that can both be changed based on the use scenario. The growth parameter is a value that determines how aggressive to grow the similarity threshold. As seen in Algorithm 1, when

calculating the updated similarity threshold the current value of the threshold is initially divided by the growth parameter which kickstarts the algorithm. We evaluate various growth parameters in Section IV-C. The dampening factor can also be decided based on the use scenario and is used to lessen the effect of rotation and frame sequence, as both of these values can be large enough to offset the rest of the algorithm if their effects are not kept in check.

The calculation of the new similarity threshold can be found on Line 10 of Algorithm 1. When updating the similarity threshold we add the threshold value divided by α to the difference between the current similarity threshold and the larger value between the number of elapsed frames and the change in heading. This larger value is then dampened by β and the difference is divided by the current similarity threshold. This maximum value between this calculated similarity threshold and the initial similarity threshold is chosen as the updated similarity threshold. An analysis of how the similarity threshold changes over time can be found in Section IV-C.

It is important to note that the process described above resets on every new DNN recognition and if a cache miss occurs and a new entry is inserted into the cache, the loosened threshold resets to the default similarity threshold.

3) Rotation-Driven Interpolative Caching: When using an approximate cache, we have identified a few scenarios that often lead to unnecessary cache misses. A cache miss often occurs when the distance between two images containing the same object is greater than the pre-determined threshold. Another scenario where unnecessary cache misses occur is when the LSH does not return the correct image in the candidate set for similarity matching. This can occur for several reasons, including an increase in image distance, and also when the user rotates their device in specific ways. We have found that rotation around the phone's x-axis often leads to avoidable cache misses. To the best of our knowledge, no existing work handles this issue, and we introduce our rotation-driven interpolative caching that reduces the cache misses due to rotation that often plague in-memory caching systems.

Interpolative caching avoids unnecessary cache misses by prefetching selected frames and inserting them into the cache before they are needed. Co-Cache avoids cache misses that require DNN recognitions by predicting when a miss might occur and preemptively inserting a new frame into the cache that will result in a cache hit and avoid unnecessary DNN recognitions.

Our interpolative caching tracks both the changing similarity distance between consecutive frames as well as the rotational movement of the smartphone to predict when the above scenario might occur. Based on factors including gyroscope readings of the phone's x-rotational axis and the incremental distance measures, Co-Cache's interpolative caching system will preemptively insert an image into the cache to avoid a costly cache miss. If the rotation of the device corresponds to a growing similarity difference between the cached entry and the current raw image we prefetch the current video frame and insert it into the cache preemptively when the difference in similarity approaches the similarity threshold. Every cache miss and DNN recognition resets the rotation calculations to a new base which allows Co-Cache to avoid any drift that normally accumulates after repeated integrations. This both avoids expensive drift mitigation calculations and also avoids expensive DNN recognitions. Results of our rotation-driven interpolative caching can be found in Section IV-F.

4) Collaborative Pre-Caching: We introduce Co-Cache's collaborative pre-caching as an infrastructure-less solution to sharing environment and cache information between users. CARS [36] introduces collaborative caching with a cloud server and nearby devices running the same application, however, their strategy for collaboration does not take into account movement of devices or relative distance between users. Co-Cache creates ad-hoc peer-to-peer networks of smartphones to facilitate collaboration between nearby users. While connected, devices routinely monitor the relative movement of nearby devices using our Bluetooth proximity detection. Other work, including Psephos [23], attempt peer-to-peer collaboration for image recognition, but their solution requires a cloud server and does not account for relative movement between users. Co-Cache operates on realistic live video feeds and better informs cache sharing through calculation of the relative movement between users to allow for users to take full advantage of the semantic similarity that exists in video streams.

As users move throughout the environment, Co-Cache monitors any nearby devices that are approaching the user. If a device is approaching the user, they exchange cache information. Collaborative pre-caching is vital to the success of Co-Cache, as a significant source of cache misses and latency is the physical act of moving to a new location. As a user enters a new location, they do not have any information of the objects that exist in the new area. Collaborative pre-caching alleviates these issues through the sharing of known valuable cache entries to any users who enter a new area.

When sharing cache entries to nearby users, the cache entries with the highest reuse score (discussed in Section III-B5) are shared, as those entries are deemed to be the most valuable. In the cache, a subset of the total entries are dedicated to pre-cached entries. The value of the subset size is configurable depending on the use scenario. Pre-cached entries are given their own subset of the overall cache because as more users enter the network, individual nodes can receive tens, even hundreds of entries from neighbors. If the general cache and pre-cached entries are not separated then the valuable entries from our local inertial-driven optimizations would be replaced quickly and not fully utilized. When a pre-cached entry is accessed for computation reuse, that entry is moved from the pre-cached subset to the general cache as it can now have a reuse score that is informed by our inertial optimizations.

5) Cache Warm-up, Insertion, and Replacement: **Reuse score** is a metric developed for Co-Cache to assign a single number value to predict the potential reuse opportunities of any given cache entry. The reuse score is the main criteria taken into consideration by Co-Cache's cache replacement

strategy. When determining what entries to potentially evict, the existing cache entry with the lowest reuse score will be evicted first.

$$reusescore = \frac{\frac{1}{\frac{s+1}{10} * \lceil \frac{\Delta heading}{\beta} \rceil} + \frac{accesses+1}{elapsedframes}}{\#instances} \quad (1)$$

Most existing caching systems determine their replacement strategies based on metrics such as access frequency and access time. Co-Cache's reuse score, however, has a bevy of metrics at its disposal, including standard access frequency, access time, but also number of instances of an entry, and also various movement-based information. In an approximate caching paradigm there is often duplication of cache entries due to the inherent accuracy issues regarding cache searching, and therefore the number of instances of an entry in the cache is a valuable metric. The movement-based metrics are derived from the Local Cache Management Module's sampling of the smartphone's inertial sensors. The equation for calculating the reuse score can be seen in Equation 1, where *s* is the number of steps taken since the last cache access for a particular entry, $\Delta heading$ has a maximum value of 180 and is the difference in heading between the current heading of the user and the heading during the last cache entry access, β is the same dampening factor discussed in Section III-B2, *accesses* is the number of times an entry has been accessed, *elapsedframes* is the number of frames that have elapsed since an entry was last accessed, and finally *#instance* is the number of entries that map to the same value as a particular entry.

When calculating the reuse score, one is added to *s* to avoid issues of dividing by zero. One is also added to *accesses* to avoid issues that arise with zero. If one was not added to *accesses*, then an entry that has zero accesses after two frames would be too similar to an entry that has zero accesses after 100 frames. The ceiling of the heading divided by the dampening factor is taken because small changes in heading should not have a large impact on reuse score.

Cache Warm-up, Insertion and Replacement. If the cache is not entirely filled with entries, a cache entry is simply added to the LSH and can immediately be used. However, if the cache is entirely full then an entry must first be evicted. The eviction process relies solely on the reuse score discussed previously. Co-Cache finds the entry with the lowest reuse score, removes it from the LSH, and inserts the new value into the LSH. Cache warm-up is the process of populating the cache with values whenever the user enters the application and they have an empty cache. Any entries added to the cache during the warm-up phase come entirely from other users near to the new user. Proximity detection is based on received signal strength (RSSI), and gives a rough estimate of who is nearest to the user. Based on the RSSI values returned from the proximity detection, the new device will request a proportional amount of cache entries from various nearby users.

C. Collaborative Network Communication Module

1) Wi-Fi Direct Peer-to-peer Collaboration: Wi-Fi Direct is an ad-hoc Wi-Fi protocol that exists on Android smartphones. Phones use their Wi-Fi radio to make direct connections between users which allows for the transfer of information without any outside infrastructure [16]. Co-Cache's peer-to-peer system allows users physically near to one another to exchange caching information quickly. When a user enters a new location, Co-Cache automatically scans for existing peer-to-peer networks. If a network already exists, then the new user requests to join the network and once approved will request cache entries from certain users in the network to perform cache warm-up. If no network exists, the user will start their own network and wait for other users to join.

2) Bluetooth Proximity Detection: Proximity detection is used by Co-Cache to determine both what users are physically the closest to each other as well as the relative movement between users over time. Proximity detection is achieved using Bluetooth's discovery process. Discovery in Bluetooth is a process in which a device broadcasts a bluetooth signal to everyone near to them. While in discovery, devices also listen for the broadcasts of nearby devices to potentially connect to. However, for our purposes, the transfer of information between users is handled by Wi-Fi Direct. Instead, when a device receives a broadcast it also receives a signal strength value known as RSSI, or a received signal strength indicator. A large RSSI value indicates that the user broadcasting the discovery message is physically close. Co-Cache periodically performs Bluetooth discovery to collect RSSI values of all nearby nodes.

If the RSSI value of a neighbor has been increasing over the past few cycles, the users are potentially becoming physically closer to each other. For each user device, the step detector described in Section III-A2 is used to determine if this user is currently moving. Once it has been determined that this user is getting closer to another user, it will exchange messages via Wi-Fi Direct to let the other user know if it is currently moving. If only one user is moving, then the stationary user will send cache entries to the moving user as the moving user is most likely approaching the stationary user. If both users are moving, then based on the heading of each device, either one or both users exchange cache entries with each other to help reduce the flurry of cache misses that often accompany moving to a new location.

IV. Implementation and Evaluation

A. Evaluation Methodology

To test the viability of Co-Cache, we develop an image recognition Android application built upon various popular DNN models. The application was developed using Java, OpenCV [1] and TensorFlow to facilitate the usage of multiple DNNs. Our cloud baseline was deployed on Amazon Web Services (AWS) with 8 vCPUs and 32 GB RAM. Our edge baseline is a local PC with an i7-7700 CPU (4 cores, 8 logical processors, @ 3.6 GHz).

Co-Cache was evaluated using ResNet [20] and Inception v4 [32] trained on the standard Imagenet [13] dataset as the DNN models. Both of these models are widely available and widely used. We also explored using mobile focused

DNN architectures such as MobileNet [21] or GoogleNet [33], however, these architectures sacrifice significant accuracy and are not well suited to our mobile image recognition tasks. Co-Cache was tested on a variety of Android smartphones, including Google Pixel 4 XL, Samsung Note 10+, Xiaomi Mi 8 Lite, Samsung Galaxy S9, and Google Pixel. This collection of devices has a wide range of age and computation capacity, which makes for a good real world experiment.

To evaluate Co-Cache in the real world, four testers were brought to a local shopping supercenter and asked to browse naturally throughout the store together using our image recognition application. The store contains a wide variety of products, including grocery, produce, athletics, electronics, clothing, home essentials, etc. The shopping supercenter use case is favorable as there were many object classes that could be found in the store. As the users are moving throughout the store, we save every piece of information needed for Co-Cache to function so that we can recreate the exact same test scenarios in different situations, including different DNNs, devices, or even with more or less neighbors. To assess the effectiveness of Co-Cache we investigate overall application latency, computation reuse opportunities, inertial-driven similarity threshold, recognition accuracy, and the effects of cache size on the overall performance of Co-Cache. **Latency** is measured in total milliseconds from the arrival of a new video frame until the result of the recognition is returned. Latency is calculated and averaged only on frames that returned a positive recognition result. **Accuracy and error rate** are measured against a ground truth of the DNN recognition. An error is counted only if the cache search returns an incorrect result. It is not considered an error if the cache returns no match. **Reuse rate** is determined based on how many cache hits occur compared with how many cache hits were possible. A cache hit is not possible when there is no object in the video frame or the correct cache entry is not in the cache.

B. Comparison with Existing Work

We also compare with existing work, *CARS* [36], in which collaboration is proposed for improving the latency in image recognition, involving a small recognition cache on the local device, and a cloud server. CARS also introduces techniques specific to augmented reality. For our comparison, we implemented the image recognition pipeline of CARS, and forgo implementation of any Augmented Reality-specific tasks to make the image recognition comparison as fair as possible. Additionally, we use the same LSH as Co-Cache for our CARS implementation. This was done because the cache described in CARS is not well suited to the unlabeled dataset we are using. Also, because we are interested in the performance of collaboration specifically, the choice to use the same cache structure in both implementations will make our comparison as equal as possible.

We chose CARS for our comparison because CARS is the paper most similar to Co-Cache. CARS includes ad-hoc sharing of image data between users, and a caching system on the local device. However, Co-Cache differs from CARS in significant areas. CARS requires a cloud server, and doesn't take into account any inertial movement during its recognition, and CARS' image sharing policy doesn't take into account any movement or distance between users. CARS also does not implement any caching optimizations on the local device, and requires nearby neighbors to achieve any optimizations, which is not always available in the real world. Overall, this may lead to a non-optimized caching experience.

C. Inertial-Driven Similarity Threshold

As discussed in Section III-B, the growth parameter, α, controls the aggressiveness of our inertial-driven similarity threshold algorithm. A smaller α indicates aggressive threshold updating, allowing for greater computation reuse, but risking errors. We designed the inertial-driven similarity threshold algorithm to capture as many reuse opportunities without sacrificing accuracy. A small initial similarity threshold means a high accuracy is desired, at the cost of higher latency.

Figure 2 shows the effects of the threshold parameter on both reuse opportunities taken and error rate. As the α gets smaller, the reuse percentage and error rate increase dramatically. With no inertial-driven similarity threshold updating, we have a reuse rate of about 78% and an error rate of around 4%. As we decrease α, both the error rate and reuse rate increase slowly until we reach a growth parameter of .7. With an α of .7, we achieve a reuse rate of 89% and an error rate of 10%. As α decreases the reuse rate and error rates both skyrocket as the safeguards of frame sequence and heading can no longer contain the similarity threshold. It is important to note that the inertial-driven similarity threshold resets on every cache miss or new object detection. A too small growth parameter does not allow for this reset to occur frequently enough to contain the similarity threshold. An α of .7 gives a good tradeoff of latency reduction and error rate.

Fig. 2. Effect of Growth Parameter on Reuse and Error Rate

D. Cache Size

The number of cache entries required for caching is important for mobile applications as it directly affects the reuse opportunities and accuracy. Additionally, individual Android applications are granted limited amounts of memory. The final amount each application gets is dependent on the specific device, but it is sometimes capped at 8 MB [3].

Co-Cache, however, can achieve better results than many other systems while maintaining a cache size 50-70% smaller than other applications. As can be seen in Figure 3, with a cache size of 150 entries, Co-Cache achieves a reuse rate of just under 90% with an error rate of around 10%. As the cache size increases, the reuse rate also increases, but the accuracy correspondingly decreases. At a cache size of 250 entries, we achieve a reuse rate of around 95%, but an accuracy of only around 80%. Because of this, we choose 150 cache entries as the optimal cache size. The underlying data structure of approximate caching is what leads to a decrease in accuracy as the cache size increases. In exact caching situations, more cache entries leads to higher accuracy, but in approximate caching more entries leads to a higher rate of false positives. With more entries in the cache, there is a higher probability that one of the images is similar enough to the current frame, even when it is not an actual match. Work in [17], [18] combats the issue of accuracy with larger cache sizes by having many entries of the same object. This, however, is not feasible in real-world situations where a smaller cache size is required on smartphones and smart glasses.

Fig. 3. Effects of Cache Size on Accuracy and Error Rate

E. Latency

The most important metric for evaluation of our system is the overall latency of the application. Latency is the motivating factor for most attempts of both offloading and caching. Work in human-computer interaction has shown that image recognition should take no longer than 100 ms to give a seamless user experience [7], [8], [28]. The latency of image recognition applications using Co-Cache was evaluated on five different devices of varying age and quality, as well as two popular neural network architectures. We also developed cloud and edge offloading baselines to compare against Co-Cache. Our cloud and edge baselines receive images from the smartphone's camera, run the DNN recognition, then return the results to the device. Finally, we compare our results against CARS, an existing work in this area.

One of the motivations of Co-Cache is the infeasibility of the current state of the art cloud and edge offloading for image recognition. As can be seen in the *Cloud* and *Edge* columns of Table I, the latencies achieved by cloud and edge offloading do not meet the 100 ms threshold required for a seamless user experience. Running Co-Cache on the local device with no neighbors achieves lower latency than cloud offloading on all device/model combinations. When compared against edge offloading, running Co-Cache locally with no neighbors out performs edge offloading in 7 out of 10 device/model combinations. With the addition of one neighbor, however, Co-Cache outperforms both standard cloud and edge computing in every scenario. Co-Cache's peer-to-peer communication is so effective because devices are not sending large images, instead sending only extracted features of images, and also the communication is done completely in the background, not impacting latency negatively at all.

We also explored baseline cloud and edge offloading with a vanilla approximate cache. While this did provide better latencies of around 230-270 ms for cloud and 170-200 ms for edge, the increased error rate of around 20-23% was not feasible for real-world image recognition. Additionally, the overhead involved with offloading inertial measurements made using Co-Cache optimizations on the cloud not possible.

The data shown in Table I for our CARS implementation is assuming four neighboring devices, which is also used in our Co-Cache evaluation. It is interesting to note that CARS achieves significantly better results than a pure Cloud or Edge implementation, partly in thanks to its small local cache. On our two highest quality devices, Co-Cache with no neighbors has a lower average latency than CARS with four neighbors. When comparing CARS with four neighbors to Co-Cache with two neighbors, Co-Cache performs better in 8 out of 10 scenarios. Co-Cache with three devices outperforms CARS with four devices in every scenario. And finally, Co-Cache with four devices significantly outperforms CARS with four devices in every scenario. In many cases, the average latency of Co-Cache is less than half that of CARS, and in the best scenario, Co-Cache achieves 81ms of latency compared with CARS latency of 247ms, which is almost a 70% reduction in latency.

Co-Cache also boasts significant latency reduction in older, less powerful smartphones. Without caching support, ResNet and Inception v4 running on the Mi 8 Lite, Galaxy S9, and Pixel have latencies as high as 3,102.99 milliseconds. With Co-Cache support and four devices, this latency is reduced to around 196 milliseconds in the worst case and 119 milliseconds in the best case. This takes image recognition from unusable to nearly seamless and in the best case can achieve a latency reduction of around 94%.

Co-Cache supports both local caching and collaborative caching. Figure 4 shows the latency reduction achieved by Co-Cache in cases from one to four users. With only one device, Co-Cache achieves latency reduction of anywhere from 80% to 84% depending on what device is being used. With two devices collaborating, we achieve latency reduction anywhere from 87% to 90%. With three users we see latency reduction in the range of 89% to 92%. Finally, with four devices we achieve latency reduction anywhere from 91% to 94%.

These results are significant in two areas. First, when there are many neighbors nearby, Co-Cache achieves very small la-

TABLE I
END-TO-END LATENCY OF CO-CACHE (MS)

Smartphone	DNN Model	No Cache	Cloud	Edge	CARS	LRU Cache	Local Co-Cache	Two Devices	Three Devices	Four Devices
Pixel 4	ResNet	1,128.63	524.57	385.23	253.58	340.37	215.23	135.25	117.78	98.02
	Inception v4	1,086.60	630.91	429.42		328.63	208.30	131.39	114.59	95.59
Note 10 +	ResNet	898.82	529.18	377.99	247.26	272.90	175.37	113.04	99.43	84.03
	Inception v4	839.84	631.22	440.64		257.25	166.45	108.42	95.74	81.40
Mi 8 Lite	ResNet	2,584.20	544.65	399.03	271.03	704.81	421.63	240.65	201.12	156.41
	Inception v4	2,395.53	653.93	434.41		659.36	397.78	230.59	194.07	152.77
Galaxy S9	ResNet	2,171.79	536.17	382.82	269.84	622.23	378.28	222.37	188.32	149.80
	Inception v4	1,592.38	639.59	438.17		463.87	286.13	172.58	147.78	119.72
Pixel 1	ResNet	3,102.99	531.46	384.13	273.97	869.00	521.82	299.93	251.47	196.65
	Inception v4	2,959.91	635.62	431.96		836.52	505.70	294.27	248.08	195.85

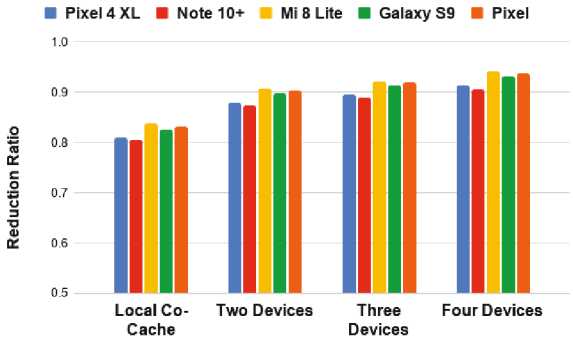

Fig. 4. Latency Reduction of Co-Cache with Multiple Users

tencies, and **even in situations when there are no neighbors, Co-Cache still achieves significant latency reduction.**

F. Interpolative Caching and Collaborative Pre-Caching

Co-Cache's success is directly affected by the performance of our interpolative caching and collaborative pre-caching. Figure 5 shows the reuse rate and error rate of Co-Cache as a whole as well as the cache entries added by our interpolative caching and collaborative pre-caching. Reuse rate is determined by the percentage of video frames that have an object and also result in a cache hit. Reuse rate is different from cache hit rate because for reuse rate we do not consider video frames that have no objects.

Fig. 5. Reuse Rate and Error Rate of Co-Cache

The overall reuse rate when using a standard LRU caching scheme without Co-Cache is about 73%, with an error rate of around 7%. Using Co-Cache on a local device increases the reuse rate by 12% over standard LRU, coming in around 85% with an error rate of around 10%. With two devices, we see a reuse rate of about 90% with an error rate of 11%. With three devices we increase the reuse rate to 92% with an error rate of 13%. Finally, with four devices, we see an overall reuse rate of 94% and an error rate of 12%.

We also evaluate the hit and error rate on entries cached via our interpolative and pre-caching. Interpolative caching on a local device has a reuse rate of about 89%, and an error rate of 18%. The error rate for interpolative caching comes from the use of the DNN recognitions as the ground truth. Because interpolative caching preemptively caches values without running a DNN recognition, often a few frames after the interpolative caching the DNN sees a new object in the frame, but the LSH still labels it as the old object.

V. RELATED WORK

Offloading systems such as CloudAR [37] choose to offload the entire image recognition pipeline, including feature extraction, image recognition/object detection, and object tracking. Others, including VisualPrint [24] choose to offload only the image recognition and compute the feature extraction and object tracking locally. As discussed previously, CARS [36] introduces a cloud offloading system that also has a small recognition cache and collaboration between users.

Approximate caching for image recognition is not as widely studied as cloud and edge offloading. Recent studies, including Cachier [14], Potluck [18], and FoggyCache [17] all study approximate caching for image recognition. Cachier utilizes edge caching for approximate cache entry optimization and and modeling query patterns. Potluck introduces cross-application deduplication, allowing multiple applications residing on one device to reuse recognitions from one another. FoggyCache is a cross-device approximate computation reuse system that optimizes the locality sensitive hashing and k-nearest neighbor searching that many approximate caches rely on.

VI. CONCLUSIONS

In this paper we introduce Co-Cache, an inertial-driven infrastructure-less collaborative approximate caching system where computation results of similar images can be reused to improve the latency of mobile image recognition. Approximate caching enhanced with Co-Cache achieves low latency without sacrificing accuracy. We design interpolative caching, reuse scoring, collaborative pre-caching, and inertial-driven similarity thresholds to optimize the cache management

of our caching system using the inertial measurements of smartphones as well as the inherent similarity that exists in video streams. We evaluate Co-Cache in a realistic real-world situation. Our evaluation shows that Co-Cache can reduce the overall latency of image recognition by up to 94%, while also maintaining a cache size around 50-70% smaller than competing approximate caching systems. We also show that Co-Cache outperforms existing work in collaborative image recognition.

ACKNOWLEDGMENT

This work was partially supported by the U.S. National Science Foundation under Grant CCF-2007159.

REFERENCES

[1] Opencv: Open computer vision library, 2000.
[2] Pokemon go augmented reality game, 2016.
[3] Android compatibility definition, 2020.
[4] Khadija Akherfi, Micheal Gerndt, and Hamid Harroud. Mobile cloud computing for computation offloading: Issues and challenges. *Applied computing and informatics*, 14(1):1–16, 2018.
[5] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 47–66. SIAM, 2017.
[6] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.
[7] Stuart K Card. *The psychology of human-computer interaction*. Crc Press, 2018.
[8] Stuart K Card, George G Robertson, and Jock D Mackinlay. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pages 181–186, 1991.
[9] Lukas Cavigelli, Philippe Degen, and Luca Benini. Cbinfer: Change-based inference for convolutional neural networks on video data. In *Proceedings of the 11th International Conference on Distributed Smart Cameras*, pages 1–8, 2017.
[10] Guoguo Chen, Carolina Parada, and Georg Heigold. Small-footprint keyword spotting using deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4087–4091. IEEE, 2014.
[11] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168, 2015.
[12] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.
[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
[14] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Cachier: Edge-caching for recognition applications. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pages 276–286. IEEE, 2017.
[15] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 115–127, 2018.
[16] Colin Funai, Cristiano Tapparello, and Wendi Heinzelman. Enabling multi-hop ad hoc networks through wifi direct multi-group networking. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 491–497. IEEE, 2017.
[17] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. Foggycache: Cross-device approximate computation reuse. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 19–34, 2018.
[18] Peizhen Guo and Wenjun Hu. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 271–284, 2018.
[19] Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.
[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
[21] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.
[22] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 82–95, 2017.
[23] Stratis Ioannidis, Laurent Massoulie, and Augustin Chaintreau. Distributed caching over heterogeneous mobile networks. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 311–322, 2010.
[24] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. Low bandwidth offload for mobile ar. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 237–251, 2016.
[25] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery.
[26] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
[27] Wolfgang Narzt, Gustav Pomberger, Alois Ferscha, Dieter Kolb, Reiner Müller, Jan Wieghardt, Horst Hörtner, and Christopher Lindinger. Augmented reality navigation systems. *Universal Access in the Information Society*, 4(3):177–187, 2006.
[28] Jakob Nielsen. *Usability engineering*. Morgan Kaufmann, 1994.
[29] Iulian Radu. Augmented reality in education: a meta-review and cross-media analysis. *Personal and Ubiquitous Computing*, 18(6):1533–1543, 2014.
[30] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.
[31] Zuolei Sun, Xuchu Mao, Weifeng Tian, and Xiangfen Zhang. Activity classification and dead reckoning for pedestrian navigation with wearable sensors. *Measurement science and technology*, 20(1):015203, 2008.
[32] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016.
[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.
[34] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 129–144, 2018.
[35] Wenxiao Zhang, Bo Han, and Pan Hui. On the networking challenges of mobile augmented reality. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 24–29, 2017.
[36] Wenxiao Zhang, Bo Han, Pan Hui, Vijay Gopalakrishnan, Eric Zavesky, and Feng Qian. Cars: Collaborative augmented reality for socialization. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*, pages 25–30, 2018.
[37] Wenxiao Zhang, Sikun Lin, Farshid Hassani Bijarbooneh, Hao Fei Cheng, and Pan Hui. Cloudar: A cloud-based framework for mobile augmented reality. In *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, pages 194–200, 2017.
[38] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.