CaPSuLe: A Camera-based Positioning System Using Learning

Yongshik Moon*‡ Soonhyun Noh*‡ Daedong Park*‡ Chen Luo*† Anshumali Shrivastava† Seongsoo Hong‡ Krishna Palem†

‡ Seoul National University
Seoul, Korea.
Email: {ysmoon, shnoh, ddpark, sshong}@redwood.snu.ac.kr

† Rice University
Houston, Texas, USA.
Email: {cl67, anshumali, kpalem }@rice.edu

***Abstract*—[1] We show the first camera based (privacy-preserving) indoor mobile positioning system, CaPSuLe, which does not involve any communication (or data transfer) with any other device or the cloud. The algorithm only needs 78.9MB of memory and can localize a mobile device with 92.11% accuracy. Furthermore this is done in 1.92 seconds of on-device computation consuming 3.77 Joules of energy, as evaluated on Samsung Galaxy S4 platform. At the core, our solutions uses a hashing-based image matching algorithm which is more than 500x cheaper, both in energy and computation cost, over existing state-of-the-art matching techniques. This significant reduction allows us to perform end-to-end computation locally on the mobile device. In contrast traditional approaches would consume 2100 Joules and takes more than 1000 seconds with a small accuracy increase of 0.89%. The ability to run the complete algorithm on the mobile device eliminates the need for the cloud, making CaPSuLe a privacy-preserving localization algorithm by design as it does not require any communication.**

I. Motivation

Indoor localization technology is expected to be a 4 billion dollar industry by 2018 [1]. Increased demand for accurate indoor localization market is due to venue-based marketing, poor performance of GPS in indoor environments [2], and government initiatives in developing positioning systems for public safety and urban security segments.

GPS signals are blocked indoors and therefore have poor accuracy. Therefore, there are a variety of algorithms utilizing other sensors, such as WiFi [3], for estimating the location indoor. Such algorithms rely on aggregating information from multiple sensors to get good accuracy, which makes then expensive and complicated.

Very recently, it was found that an elegant way of localizing a mobile more accurately is by utilizing the device's camera [4]. The idea is to match the current image from the camera with a database of geo-tagged images. Recent advances in vision have made image matching technology quite accurate, which makes camera based image positioning a very promising direction. However, current image matching algorithms are quite expensive from both latency and energy perspectives, and therefore they cannot run locally on a mobile device. For instance, we show that current state-of-the-art image matching algorithm when run on the database of 719 images require more than 1000 seconds using around 2100 Joules of energy for getting the current location; entirely impractical for use in a mobile context. An alternate is to consider a cloud-based service to perform image matching.

There are three major concerns with cloud-based image matching: **1) Communication, 2) Energy Consumption and 3) Privacy.**

1. Communication: Image matching requires transmitting the current image from the mobile device to the cloud, followed by the location, inferred in the cloud and transferred back. Communication often has unpredictable latency, as it requires WiFi, cellular networks, etc.

2. Energy Consumption: Image matching is an expensive operation. The cloud-based service even if very fast, is likely to consume a significant amount of energy. Thus cloud-based image matching service is not a sustainable solution.

3. Privacy: Transfer of data back and forth to the cloud compromises the privacy of user's information. It opens the possibility of potential privacy breaches.

Hope: Trading (insignificant) Quality for Energy. The philosophy of trading (a small amount of) quality for significant gains in energy has recently gained significant attention [5]–[7]. Capitalizing on this energy-quality tradeoff is deemed to be a future of SoC technology [8]. Here, we provide a concrete demonstration of this philosophy. Our proposed end-to-end system shows that by trading a small amount of accuracy we can get away with all the three shortcomings, as mentioned earlier, associated with the cloud-based image matching techniques.

Our Contributions: We propose CaPSuLe for image based device positioning, a first of its kind system, which is free from all the three problems of communication, energy consumption, and privacy. At the heart of CaPSuLe lies an approximate image matching algorithm, based on fast locality sensitive hashing, which is more than 500x times cheaper than state-of-the-art image matching algorithm. Such a significant gain in computation and energy cost is a result of careful choices of hash tables, hash functions, and related operations. This massive reduction allows us to perform end-to-end image matching on the mobile device itself. Our algorithm takes 1.92

[1] * indicates equal contribution of authors

seconds requiring 3.78 Joules energy on Samsung Galaxy S4 archiving 92.11% accuracy in estimating the location. Since all computations are local and are performed on the device, our algorithm is free from privacy infringements as no information is transmitted. We hope that our work will lead to many new energy efficient machine learning algorithms where the need for cloud computing can be eliminated.

II. Devise Positioning via Image Matching

Image based positioning system [4], [9] takes the current picture of the location and matches it with images in a pre-collected database of geo-tagged images of the area of the building such as a shopping mall. The location of the matched image is deemed to be the current position of the device. The key observation is that building a densely sampled dataset consisting of images, tagged with their geo-location at different places in the indoor environment is a relatively easy task with the surge in the number of images. The accuracy of the system is then directly dependent on the ability to identify the right matching image in the database, which is a classical computer vision problem.

Formally, we are given a collection of geo-tagged images $\mathcal{C}$ consisting of images from the given indoor environment, e.g., shopping mall, campus, etc., where the device needs to be positioned. By using its camera, we create a query image q. The goal of the image matching algorithm is the find an image $I \in \mathcal{C}$ which maximizes the "similarity" with the query q. Formally,

$$I = \arg\min_{I \in \mathcal{C}} Sim(q, I) \tag{1}$$

The critical vision component in Equation 1 is the design of the similarity function or $Sim(.,.)$ which captures the notion of semantic similarity between different images. $Sim(.,.)$ must tolerate variations in pose, resolutions, shifts, etc [10], [11]. For better demonstration of the challenges associated with the state of the art, we first describe our setting and the dataset:

A. Dataset and Settings

We chose the LOTTE Department Store main branch, which is a major shopping mall in Seoul, Korea, for our positioning system. We collected a total of 871 images of different shops in the mall. Images were collected by using Naver Indoor Maps [12]. Besides, to get a good coverage of the mall, we also manually took pictures of stores by a cellphone camera. Overall, we covered 45 different location in the shopping mall. The images are taken with varying poses and lighting to ensure that the datasets reflect the real settings. Also, two separate sources of images make the setting more real and free of common bias.

Each image is annotated with its corresponding locations. We further downscaled each image to 640×360 pixels to reduce the computational cost associated with matching without significant loss of accuracy. Such downsampling of images are commonly adopted in many real applications [13]. For evaluation, we partition the data into two sets: 719 training and 152 query images. Figure 1 shows some query images and the matched training images for these query images using our CaPSuLe system. We can clearly see the complexity of the problem as the matching images can have varying poses.

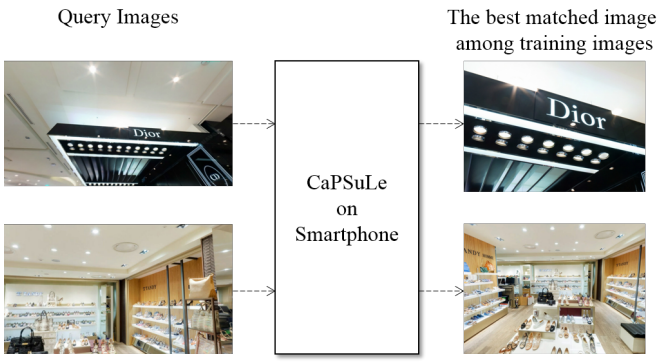

Fig. 1: Example Query and returned matches image by CaPSuLe system. The match is with varying pose and orientations showing the complexity of our dataset.

B. Device and Platform

We use aSamsung Galaxy S4 smartphone with Android 5.0.1 Lollipop running with a Linux kernel 3.4.5. The smartphone has an ARM processor that consists of four Cortex-A15 cores and 2 GB DRAM. We additionally used a Monsoon Power Meter to measure the power consumption of the smartphone. The detailed hardware and software configuration of the target system is shown in Table I.

C. The Image Matching Problem and its Computational (Energy) Barrier

With advancements in vision technology, image matching is quite accurate. However, they are far from being cheap. Modern matching algorithms require costly similarity measure Sim for good accuracy. Such expensive computations cannot be performed on the device because of their significant computational requirements. We elaborate our baselines which is the state-of-the-art algorithm, as implemented in the widely used OpenCV package [10], [11], for computing Equation 1.

The similarity measure, $Sim(.,.)$, used in the OpenCV package leads to 93% accuracy on our dataset. Other similarity measures based on Euclidian distance over Bag-of-Words

TABLE I: Target System Description

Hardware	System on Chip	Exynos 5410 Octa
	CPU	Quad-core 1.6 GHz Cortex-A15
	Main Memory	2GB
	Storage	16 GB NAND Flash
Software	OS Kernel	Linux kernel version 3.4.5
	Android Framework	Android 5.0.1
	OpenCV	OpenCV 3.1 for Android

(BoW) only yields 75% or less accuracy because our dataset contains many variations seen in the real environment which is not adequately captured by BoW methods.

OpenCV implementation for determining the similarity between the query q and any given image $I \in \mathcal{C}$ requires the following three steps:

1. Extract Features from Images: The first step is to extract a set of SURF [14] features from both the q and I. Each of these features is a 64-dimensional vector. In our system, we used 512 SURF features. We thus get 512 different 64-dimensional SURF features from each image. SURF features are the best-known features invariant to scale and other spurious transformations such as rotations. It is further known that SURF performs even better than traditional SIFT features [15]. Note, for every image I in the given training collection $\mathcal{C}$, feature extraction is done off-line. For the query, however, feature extraction needs to be done on the fly.

2. Threshold all pairwise features: The Euclidian distance between all possible feature combinations between q and I is then computed. This requires 512×512 Euclidian distance computations between 64 dimensional vector from q and I, totalling $512 \times 512 \times 64$ multiplications.

3. Compute the Similarity Values: The final score is the number of distances out of 512 x 512, which are smaller than a threshold. Roughly, this similarity measure scores based on the number of cross matches from the 512 different SURF features between the query and the image I.

The bottleneck is step number 2 which requires $512 \times 512 \times 64 > 16$ million multiplications for computing the similarity between the query and one image $I \in \mathcal{C}$. Thus, for 719 images in our datasets, a single query requires $(512 \times 512 \times 64 \times 719)$ more than 12 billion multiplications. If we plan to run this amount of computation on the mobile device, then to reiterate a single query takes more than 1030 seconds consuming more than 2100 Joules of energy. Step 2 is the primary reason why current image-based positioning algorithm needs the cloud to perform matching in reasonable time. However, as argued before, the cloud-based solution has many shortcomings. We will refer this as the Bruteforce Method.

D. Clustering (Bag-of-Words (BoW), sparse coding, etc.) does not seem to help

It might seem that step 2, requiring 512×512 distance computation can be side-stepped. The other most popular feature representation which does not require 512×512 distance calculation is the BoW [16] (or sparse coding [17]) feature representation. BoW tries to eliminate the need for multiple comparisons by pre-clustering all the extracted 512 SURF features. After clustering, BoW calculates the distances between all feature vectors of the current image and the cluster centers. BoW then produces a histogram expressing the closeness between cluster centers and the training image's SURF feature vectors. Image matching is finally performed by comparing the query image's histogram and the stored training image's histograms. This process is relatively cheap. However, it comes with a significant loss in accuracy. With our dataset, this approach barely reached 75% even with as many as 1000 clusters. Changing the cluster size of 5000 has no effect on accuracy.

Image matching is a harder task than object detection. For instance, two images may have the same categorical object (such as a chair), but they may not match with each other. This is probably the main reason why BoW is more common for object detection rather than image matching and popular state-of-the-art package OpenCV [10] implements more costly matching algorithms described earlier.

III. HOPE: PROBABILISTIC HASHING ALGORITHMS

However, if we are willing to relax the need for accuracy by a small amount, then the picture changes completely. In particular, we will use the cheap Locality Sensitive Hashing algorithms combined with the careful choice of hash functions and estimation procedure to get more than 500x reduction in the computational and the energy cost.

Locality Sensitive Hashing (LSH) [18], [19] is popular for efficient sub-linear time matching. LSH generates a random hash map h which takes the input (usually the data vector) and outputs a discrete (random) number. For two data vectors x and y, the event $h(x) = h(y)$ is called the collision (or agreement) of hash values between x and y. The hash map has the property that similar data vectors, in some desired notion, have a higher probability of collisions than non-similar data vectors. Informally, if x and y are similar, then $h(x) = h(y)$ is a more likely event, while if they are not similar then $h(x) \neq h(y)$ is more likely. The output of the hash functions is a noisy random fingerprint of the data vector [20]–[22], which being discrete is used for indexing training data vectors into hash tables. These hash tables represent an efficient data structure for matching [18] and learning [23], [24].

The fundamental observation is that in Step 2 of image matching (Section II-C), for every SURF feature of the query image q, we search for matching SURF features from image I. This matching can be made efficient using hashing. However, only performing fast near neighbor search with hashing does not yield the desired benefit. It further requires many careful choices which we describe in the next section.

IV. THE CAPSULE SYSTEM: NEAR-CLOUD PERFORMANCE WITH ON-DEVICE COMPUTATION.

Our CaPSuLe system is summarized in Figure 2. At the heart of our system lies a set of lightweight hash tables which, for a given SURF feature of a query image, finds all the potential matching SURF features from the training set. This search is done in near-constant time, by querying hash tables indexed by LSH, which saves a significant amount of computational (and hence energy) overhead without compromising the accuracy.

CaPSuLe uses two parameters K and L which trades accuracy for gains notably in energy and in computational time. The algorithm works in two main phases [25] for device positioning. We first describe the two phases, and later we

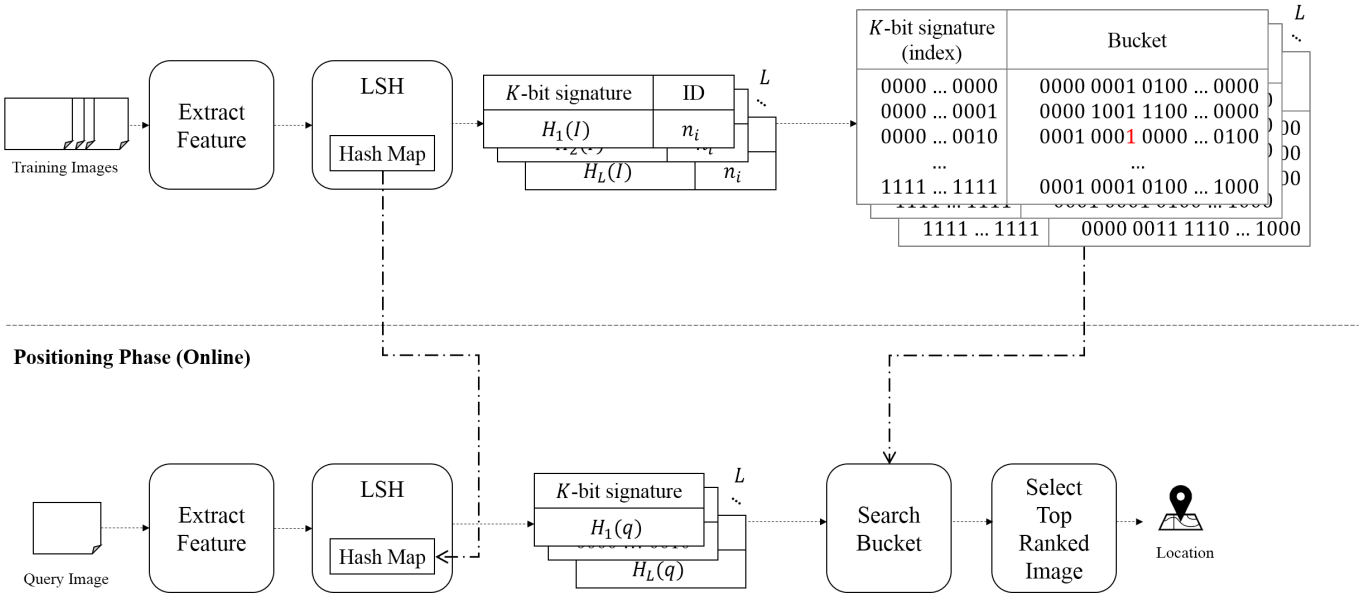

Fig. 2: The preprocessing and positioning phase of CaPSuLe.

provide more details about the design choices.

1) **Preprocessing** Phase (Offline): In the offline phase, following step 1 in Section II, we extract 512 SURF features (64 dimensional) from each geo-tagged image I in the training collection $\mathcal{C}$. We then create L different hash tables of size 2^K, i.e., K-bit keys (or indices). For every 512 SURF feature of I, we map it to a K-bit signature $H_j(I)$, using some LSH scheme H_j, for $j \in \{1,\ 2, ...,\ L\}$. Image I is then placed into hash table number j indexed by the K-bit signature $H_j(I)$ (as the key). Thus, every image is mapped to 512 keys (can be duplicate) in each of the L hash tables. The preprocessing step thus generates L independent hash tables.

2) **Query** Phase (Online): Given a query image q, we again extract 512 SURF features (64 dimensional each). For each of these 512 SURF features, we retrieve the bucket associated with the key $H_j(q)$ in hash table j. Overall, we get $512 \times L$ keys and probe the corresponding buckets (values) in the associated hash tables. Every image is then ranked based on the number of times it is observed in the $512 \times L$ buckets. The location of the top ranked image is returned as the current location as the final answer.

In this methodology, we made five novel and careful choices in CaPSuLe, all of which are critical. The system is prohibitively expensive if we remove any of the five choices. These choices are as follows:

1. Reduce Hashing Cost: The cost of computing L different K-bit hashes is expensive with popular LSH schemes such as signed random projections [26]. In particular, traditional LSH requires $K \times L \times 512 \times 64$ multiplications for computations of all the hashes (also the keys), which is very expensive. We instead used a cheap and sparse variant as described in [27] which reduces the total hashing cost per query to $\frac{1}{3}(K \times L \times 512 \times 64)$ additions/subtractions. This is a significant reduction also since multiplications are costlier than additions.

2. Buckets of Bit Arrays: Our hash tables need to store multiple images for every key. Even if we store only integer image IDs, the cost is significant. Since we have 719 images, we store a 719-bit array indexed by the K-bit key (2^K values). If image numbered n_i gets a particular key, we simply set the bit numbered n_i in the bit-array associated with the key as shown in Figure 2. This idea leads to around 32x reduction in the hash table size compared to the traditional scheme. Furthermore, we remove any memory associated with empty buckets during preprocessing to avoid unnecessary memory usage.

3. Cheap and Crude Ranking Estimation based on Bucket Matches: Hashing reports many images (sometimes multiples of a 100) as potential matches. For computing the best match, the recommended option in the literature is to rank candidates using the similarity function Sim. However, as argued in Section II, computing Sim is expensive. We utilize the property of LSH, and cheaply estimate the ranking by counting the number of times an image is hit by the query. Estimation using LSH signatures are significantly cheaper than similarity computation as reported in [28].

4. Ignoring Noisy Buckets: As our hash functions are cheap, there is a significant possibility that individual key values are likely due to bias in the LSH functions. Such bias will make some of the buckets unnecessarily crowded. Crowded buckets increase the computational time and energy since the algorithm retrieves unnecessary candidates. To eliminate this issue, we ignore buckets (treat it as empty) if they are overcrowded.

5. Reducing Main Memory: Although hash tables are significantly small (few hundred MBs), for mobile devices, loading all of them in main memory is still a concern. Our hash tables are organised into contiguous buckets, i.e., 2^K indices

each of 719 bits (see Figure 2) We, therefore, store the hash tables in device memory and load the L buckets (719 bits for each bucket) on demand during runtime (using the fseek function) without noticeable I/O overhead. This ensures that our application needs low main memory.

A. Dynamic Updates

One of the unique characteristics of CaPSuLe is that it can be incrementally updated. In particular, adding/deleting images to/from the database with only amounts to flipping a few bits, to add the new image (with labels) in the corresponding buckets, into the appropriate hash table. Thus, increasing the number of images or locations can be handled with no modification to the algorithm and minimal change to the data structure.

V. Evaluations of CaPSuLe

We implemented CaPSuLe on the platform described in Figure 1. We evaluate CaPSuLe on four metrics: 1) Response Time, 2) Energy Consumption, 3) Accuracy and 4) Main Memory. Response time and energy consumption are measured for the complete end-to-end process, i.e., including the feature extraction, retrieval, and final ranking. Accuracy is measured over the test set as the percentage of time the algorithm identifies the correct location. Main memory usage is the amount of DRAM used. It is imperative that all of these four metrics are properly balanced for the system to be practical.

Cost-Quality Tradeoff through K and L: There are two main parameters in the CaPSuLe system, K, and L. To reiterate, K determines the range of the hash table (K-bits), which is also its size. L specifies the number of hash tables. K and L are the knobs which give us finer control over the cost-quality tradeoff. If we increase K and L, the recall is better, but the space required grows significantly.

See Figure 3 for the plot of memory utilization with varying K. If we use $K = 24$ the amount of main memory needed by a single hash table easily grows to around 1GB which for $L > 1$ hash tables is infeasible. If we lower K, then the accuracy drops by around 10%. We found that other than memory, the computational, energy, and response time costs are not sensitive to variations in K and L. Memory-accuracy is the main tradeoff. We found $K = 22$ and $L = 24$ to be the sweet-spot that balances both accuracy and memory nicely. Our system uses these values for K and L. Note, we have two parameters which can be tuned offline.

Competing Solutions: Our primary goal is to approximate the accuracy of the brute force algorithm described in Section II-C. However, we want our solution to run with limited energy, memory, and latency range, which are crucial for a device positioning system. Our primary baseline is, therefore, the bruteforce algorithm in the state-of-the-art package OpenCV.

In addition to bruteforce and CaPSuLe, we tried two classical and cheaper baseline approaches:

1) BoW based image matching: As described in Section II-D, we used popular BoW based features which exploit clustering over SURF features to make matching efficient. However, there is a significant drop in the accuracy in this case. With 1000 Bag-of-words (or cluster centers), we could barely achieve 75% accuracy even after fine tuning. Increasing BoW to 5000 lead to no significant gains.

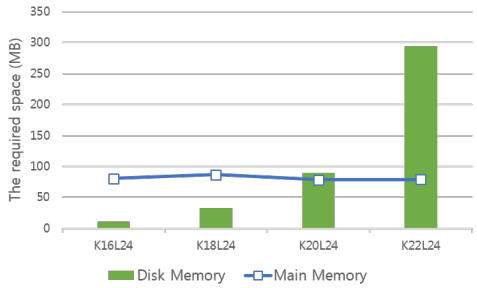

Fig. 3: The required memory/(main memory) with varying K. We load the buckets in main memory on demand.

TABLE II: Evaluations for CaPSuLe and Bruteforce

	K22L24	**Bruteforce**
Accuracy	92.11%	93.42%
Energy Consumption	3.78J	2103.22J
Response Time	1.92sec	1030.43 sec
Required Storage Space	294.39MB	363MB
Required Memory Space	78.90MB	171.41MB

2) Supervised Learning. We tried another possibility of treating location identification as a multi-class classification problem. We treat each location as a class label and use training images labeled with the location as the standard supervised multi-class classification. However, supervised learning fails to achieve more than 80% of accuracy. We used VLFeat [29], an open source package for image classification in this experiment.

A. Performance Summary

We used $K = 22$ and $L = 24$ for our settings. The response time and energy consumption for bruteforce and our approach are evaluated.

1) **Accuracy:** For our dataset, the accuracy of bruteforce is 93%, Bow 75%, supervised learning 77% and CaPSuLe 92.11%, as shown in Figure 4. Bruteforce method yielded the highest accuracy among three methods, while CaPSuLe is very close- off by only 0.89%. This phenomenon is not surprising as our approach is approximation of the Bruteforce method. BoW and supervised learning methods have poor performance, and therefore, we do not evaluate their time and energy consumption.

2) **Response Time:** We estimated the response time of Bruteforce and CaPSuLe. The response time using the Bruteforce method is 537 times more than CaPSuLe on the target mobile device. CaPSuLe takes only 1.92 seconds in the positioning phase on the device. However, the response time using the

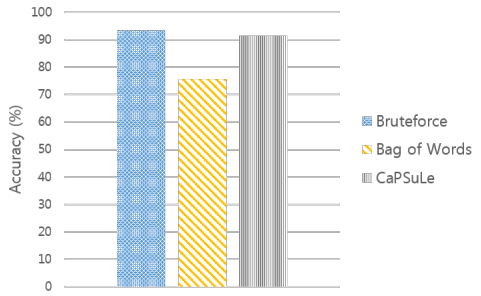

Fig. 4: The accuracies of bruteforce, BoW and CaPSuLe.

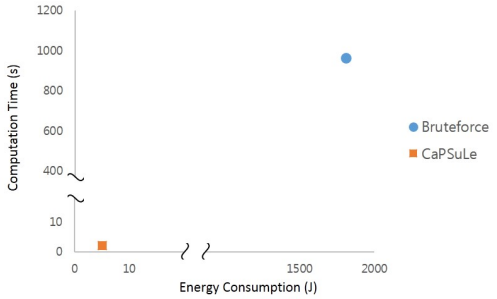

Fig. 5: Energy consumption Vs response time.

Bruteforce method is 1030.43 seconds in the online phase, which is unacceptable.

3) **Energy** Consumption: The Bruteforce method requires 2103.22J on our mobile system in the online phase. This amount of energy consumption further makes current algorithms non-sustainable. However, CaPSuLe consumed mere 3.78J for localization, which is 557x smaller than Bruteforce method.

The energy-time comparisons are illustrated in Figure 5. The overall comparison between CaPSuLe and the state-of-the-art Bruteforce matching algorithm on our platform and dataset are summarized in Table II. By sacrificing only 0.89% of the accuracy, CaPSuLe is 537 times faster in the response time and 557 times cheaper in energy consumption.

VI. Conclusion

It is widely assumed that cloud-based Machine Learning Solutions are the future. However, cloud-based applications are not ideal for the societal problem of sustainability and privacy. We have shown that by trading a small (insignificant) amount of quality, modern machine learning solutions can be made private and sustainable, thus eliminating the need for the cloud. We capitalize on the cost-quality control provided by randomized hashing algorithms and demonstrate an end-to-end indoor camera-based positioning system CaPSuLe which can localize a mobile device, with 92.11% accuracy, in 1.92 seconds of local (on-device) computations consuming 3.78 Joules of energy, using a Samsung Galaxy S4 platform.

With the ever increasing computational power of mobile devices, we believe that such cloud-independent private and sustainable solutions are the future of SoCs. We hope many works will follow this line of thought.

Acknowledgment

Dr. Krishna Palem and Dr. Anshumali Shrivastava would like to acknowledge the support of DARPA Grant FA8750-16-2-0004.

References

[1] J. Liu, "Survey of wireless based indoor localization technologies," 2014.
[2] R. Mautz, "The challenges of indoor environments and specification on some alternative positioning systems," in Positioning, Navigation and Communication, 2009. WPNC 2009. 6th Workshop on. IEEE, 2009, pp. 29–36.
[3] Z. Farid, R. Nordin, and M. Ismail, "Recent advances in wireless indoor localization techniques and system," Journal of Computer Networks and Communications, vol. 2013, 2013.
[4] J. Z. Liang, N. Corso, E. Turner, and A. Zakhor, "Image based localization in indoor environments," in Computing for Geospatial Research and Application (COM. Geo), 2013 Fourth International Conference on. IEEE, 2013, pp. 70–75.
[5] K. V. Palem, "Inexactness and a future of computing," Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, vol. 372, no. 2018, p. 20130281, 2014.
[6] J. Markoff, "A climate-modeling strategy that won't hurt the climate," http://www.nytimes.com/2015/05/12/science/inexact-computing-global-warming-supercomputers.html, 2015.
[7] K. V. Palem, "Computational proof as experiment: Probabilistic algorithms from a thermodynamic perspective," in Verification: Theory and Practice. Springer, 2003, pp. 524–547.
[8] ——, "Energy aware computing through probabilistic switching: A study of limits," IEEE Transactions on Computers, vol. 54, no. 9, pp. 1123–1137, 2005.
[9] H. Kawaji, K. Hatada, T. Yamasaki, and K. Aizawa, "Image-based indoor positioning system: fast image matching using omnidirectional panoramic images," in Proceedings of the 1st ACM international workshop on Multimodal pervasive video analysis. ACM, 2010, pp. 1–4.
[10] Itseez, "Open source computer vision library," https://github.com/itseez/opencv, 2015.
[11] The OpenCV Reference Manual, 2nd ed., Itseez, April 2014.
[12] http://map.naver.com/.
[13] A. Youssef, "Image downsampling and upsampling methods."
[14] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in European conference on computer vision. Springer, 2006, pp. 404–417.
[15] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," International journal of computer vision, vol. 60, no. 2, pp. 91–110, 2004.
[16] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in Workshop on statistical learning in computer vision, ECCV, vol. 1, no. 1-22. Prague, 2004, pp. 1–2.
[17] H. Lee, A. Battle, R. Raina, and A. Y. Ng, "Efficient sparse coding algorithms," in Advances in neural information processing systems, 2006, pp. 801–808.
[18] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in STOC, Dallas, TX, 1998, pp. 604–613.
[19] A. Shrivastava, "Probabilistic hashing techniques for big data," Ph.D. dissertation, CORNELL UNIVERSITY, 2015.
[20] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," in STOC, 1977, pp. 106–112.
[21] M. O. Rabin, "Fingerprinting by random polynomials," Center for Research in Computing Technology, Cambridge, MA, Tech. Rep. TR-15-81, 1981.
[22] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," IBM Journal of Research and Development, vol. 31, no. 2, pp. 249–260, 1987.
[23] R. Spring and A. Shrivastava, "Scalable and sustainable deep learning via randomized hashing," arXiv preprint arXiv:1602.08194, 2016.
[24] P. Li, A. Shrivastava, J. Moore, and A. C. König, "Hashing algorithms for large-scale learning," in NIPS, Granada, Spain, 2011.
[25] A. Andoni and P. Indyk, "E2lsh: Exact euclidean locality sensitive hashing," Tech. Rep., 2004.

[26] A. Shrivastava and P. Li, "Densifying one permutation hashing via rotation for fast near neighbor search," in ICML, Beijing, China, 2014.
[27] D. Achlioptas, "Database-friendly random projections," in PODS, Santa Barbara, CA, 2001, pp. 274–281.
[28] A. Shrivastava and P. Li, "Fast near neighbor search in high-dimensional binary data," in ECML, Bristol, UK, 2012.
[29] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms," http://www.vlfeat.org/, 2008.