# Addressing the Dark Side of Vision Research: Storage

Vishakha Gupta-Cledat
*Intel Labs*

Luis Remis
*Intel Labs*

Christina R. Strong
*Intel Labs*

## Abstract

Data access is swiftly becoming a bottleneck in visual data processing, providing an opportunity to influence the way visual data is treated in the storage system. To foster this discussion, we identify two key areas where storage research can strongly influence visual processing run-times: efficient metadata storage and new storage formats for visual data. We propose a storage architecture designed for efficient visual data access that exploits next generation hardware and give preliminary results showing how it enables efficient vision analytics.

## 1 Introduction

Data access is already becoming a bottleneck in visual data processing [11]. As the amount of visual data grows, it is becoming common for much of that data to be processed and analyzed by computers rather than seen by a human. This means there is an early opportunity to influence the way visual data is treated in the storage system. Unfortunately, there is an inherent segregation of research in computer science: vision researchers don't have the expertise to design a storage solution to meet their needs and storage researchers don't have the expertise to know how storage could impact visual processing.

We've identified two key areas where storage research can strongly influence visual processing run-times: efficient metadata storage and analysis friendly storage formats for visual data. Using metadata to provide a subset of relevant data reduces the amount of data that has to be processed as well as reducing data movement in the system, while new storage formats could provide faster transfer and processing of visual data. We are proposing a new architecture tailored for visual data that addresses the need to find relevant data and retrieve it efficiently. Our architecture stores visual data in analysis friendly formats, and efficiently identifies and retrieves relevant data by searching metadata that has been stored in a graph database.

To illustrate these two areas, we consider *image classification*, a basic component of vision research where a neural network is used to identify what the image contains. Suppose you want to find all the photos on your computer that contain your dog: given those photos, a pretrained neural network can identify which images contain a dog [27]. While the neural network will identify all objects in an image with some confidence, generally only the relevant information is kept: you want photos that have a 90% probability of containing your dog, not a 2% probability. However, if you then want to find images that contain your cat, a common approach is to run the neural network again, an approach which is not scalable. Storing the confidence values of image/object relations as metadata eliminates the need to process the same images with the same neural network again.

Another common practice in vision research is to keep visual data in its original format in the system rather than in a format to aid with analysis. Storing the visual data in a different form is rarely considered due to the belief that storage isn't an issue, because the time that it takes to read the data from the file system is dwarfed by the time spent in the neural network for classification. Running a neural network with different generations of hardware, the details of which are found in Section 4, proved that this hypothesis is not correct as shown in Figure 1.

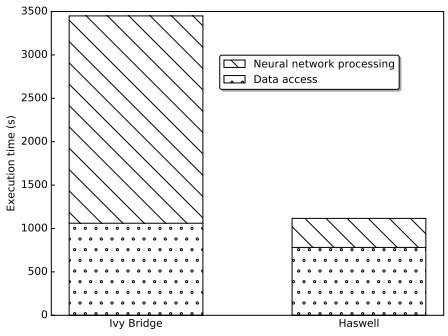


Figure 1: The newer generation of hardware (Haswell) with optimized software results in a 2.7x speedup in total time, and causes the neural network to run faster than the data is accessed.

In the rest of the paper, we give a brief explanation of how image classification works and present a case for storing visual metadata as a property graph. We outline our storage architecture and give preliminary results. It is our goal not only to develop a storage architecture designed for visual data with a focus on enabling analysis, but also to spark thought and discussion about this problem in the storage community.

## 2 Background and Related Work

Given our architecture is designed to enable better visual data analysis, we present a brief explanation of image classification, a common image processing operation, to provide clarity for our architectural design. For the sake of brevity we only present work relating to image processing in this paper, though our architecture can handle all types of visual data.

**Image Classification** In machine learning, a convolutional neural network is a type of feed-forward artificial neural network where the input is generally assumed to be an image [29]. One use for convolutional neural networks (and our use case in this paper) is image classification, where the goal is to determine with some confidence which classes the image contains.

A neural network is trained for a specific classification task using a set of images where the classes have been labeled, and provides the probability that an image contains each object class. Several end-to-end machine learning frameworks such as Caffe [17] and Tensorflow [1] have been developed that provide a complete toolkit for training, testing, finetuning, and deploying neural networks. Common practice is to start with a pretrained network that works best with the desired class set and finetune it for a specific set of data.

Images are loaded from the storage system into the framework for processing, a step that is largely overlooked because of the belief that storage is not an issue. However, as hardware improves and software is optimized for vision processing algorithms, the data access begins to dominate the total time in the framework (as seen in Figure 1). In addition to finding a way to store images such that access time decreases, it is important to figure out how to search images as well [25]. Given large data sets, such as the one hundred million images in the YFCC100M dataset [30], having the ability to identify a subset of relevant images with a metadata query would be beneficial.

**Visual Metadata as a Graph** The output of a convolutional neural network for image classification associates a vector of values with an image that correspond to the probability of an object existing in the image. Generally, this information is used for the task at hand and then discarded; an image with a dog and a cat would have a probability for both, but if the goal is to find images with dogs the other information will be lost, preventing future use.

Given an image can contain one or more objects, other relationships become apparent: an image may be taken at a specific location as shown in Figure 2. Additionally, the way people interact with visual data focuses on the relationships ("find all the images that were taken at location Y that contain person A"), causing retrieval of this data from a different kind of database to be less efficient. Combined with the notion that the schema could be constantly changing as new information is learned from additional image processing, a graph database is a logical choice for representing visual metadata.

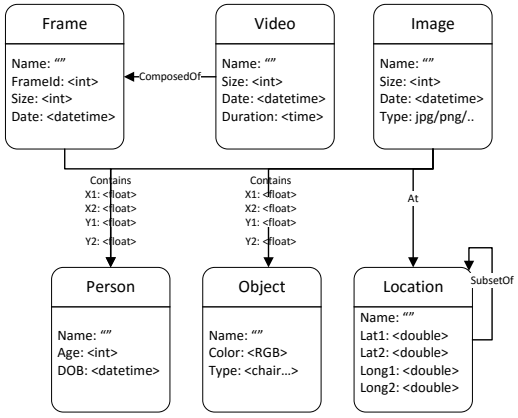


Figure 2: Example visual metadata represented as a property graph.

Many current graph database implementations are proprietary solutions, such as Allegro-Graph [12], InfiniteGraph [14], and Neo4j [6], with Titan [2] being the only fully open source implementation. Unfortunately, many of them have performance issues when supporting the traditional notion of transactions at low latency; in order to take advantage of the graph structure the database should be in memory but also persistent. New persistent memory technology [16, 21, 26] meets these needs; we further discuss the benefits and application to a graph database implementation in Section 3.

**Image Formats** Traditional lossy image formats such as JPEG are designed to compress the data by discarding high-frequency information that humans don't generally notice. However, as the quantity of visual data increases, more of it is processed before consumed by humans; information that is meaningless to humans may not be meaningless for analysis. Retaining all the infor-

mation results in lossless image formats such as PNG, but this comes at the cost of the compression rate.

Given these trends, we are exploring analysis friendly image formats such as TileDB [24]. TileDB was originally designed as a scientific data management system optimized for both dense and sparse arrays, a format in which images and videos can be expressed. TileDB is especially interesting because it allows efficient retrieval of specific areas of the array, making it possible to only return the area of an image where an object exists. Compared to other data management systems such as SciDB [5], TileDB performs orders of magnitude better in both reading and writing, but currently has fewer scientific capabilities.

**Related Data Management Systems** The Facebook architecture for photos combines TAO [4], Haystack [3], and f4 [22] for metadata, hot/recent data, and warm data respectively. While the social aspect of Facebook is logically suited to a graph, the bulk of their data is already stored in MySQL [23], so they chose to develop a graph-aware cache. A similar approach was taken by Grail [10], which argued that a syntactic layer written on top of a relational database could answer graph queries.

Diamond [13, 28] exploited active disks to perform *discard-based search* in order to shrink the amount of data returned to a reasonable size. We accomplish the same goal by exploiting persistent memory to store a graph database for metadata. There are several databases and data management systems that focus on enabling analytics or combining transactional and analytic workloads over large scale data, such as SciDB [5], BigTable [7], Shark [32], and Vertica [18]. While these systems do not focus on visual processing as a primary entity, they are valuable resources in distributing and analyzing very large scale data.

## 3 Proposed Storage Architecture

We propose a storage architecture to efficiently identify and retrieve relevant visual data, in order to enable analysis. This architecture, outlined in Figure 3, implements a graph database for visual metadata in order to efficiently identify which data is relevant and a library that handles processing on analysis friendly image formats in addition to traditional formats. Segregating the metadata and data allows us to map each to the most suitable hardware in a heterogeneous system and provides the request server the flexibility of identifying the most efficient way to handle a request for data.

**Metadata Storage** While we could have implemented a graph abstraction layer on top of a relational database, we wanted to have a highly efficient way of identifying relevant data. A query such as "find all the photos of Bob playing volleyball in Florida" would require two joins in a relational database, while it would be a 2-hop neighbor query in a graph database. Since we wanted a fast, persistent graph database, we have chosen to extend our previous work on a persistent memory graph database (PMGD) to be a fully distributed graph database. We are in the process of integrating PMGD with our architecture as a multi-modal metadata storage.

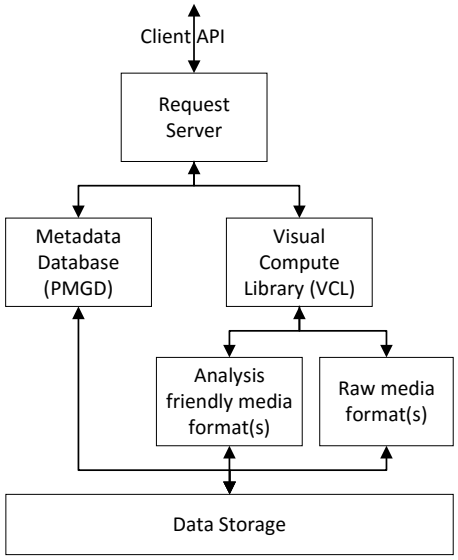


Figure 3: Major Components

For a graph database to persist the data on disk, current implementations serialize the data to write to a block sequential device. Unfortunately, this not only negates the benefits of the graph structure but also causes performance issues. To address these issues, PMGD takes advantage of developments in persistent memory (PM) technology [16, 21, 26]; specifically the low, predictable access times and byte-addressable nature. The result is a graph database that can effectively mix OLTP and OLAP functionality, bypassing the data serialization step.

PMGD is designed to exploit a memory hierarchy with data structures and transactional semantics that work with the PM caching architecture; reduce write requests (addressing PM's lower write bandwidth compared to DRAM); and reduce the number of flushes and memory commits. Our implementation organizes data in PM with the goal of achieving the best possible trade-off between storage density (how much information is stored per unit of storage) and speed of data retrieval [1].

[1] Full implementation and evaluation details of PMGD are beyond the scope of this paper.

**Data Storage** In order to reduce the data access time, we have been looking at storing images in an analysis friendly format, such as TileDB. The benefit of the TileDB format is that images are stored as a lossless compressed array of pixel values; when data is needed, it does not need to be decoded from the image format. This improves the speed at which data is retrieved and made usable; the trade-off being that the compression is not as good as the original format.

We have developed the Visual Compute Library (VCL), which interfaces with TileDB in order to store and access images. The base functions allow a user to convert images from traditional formats (such as JPEG and PNG) into the TileDB format. We have extended the VCL so that a user can interact with an image without worrying about the format it is stored in, unless specifically indicated. We are exploiting the array data management capabilities of TileDB to develop additional functionality in the VCL. The intention is to allow a user to do processing on the image directly, such as resizing or obtaining a specific region of the image.

## 4 Evaluation

To evaluate our early architecture and corresponding assumptions, we focus on understanding the individual components of PMGD and VCL. We ran our experiments on a Intel® Xeon® E5-4620 CPU (Ivy Bridge) server and a Intel® Xeon® E5-2699 CPU (Haswell) server. For image classification, we used the Intel version of Caffe [19] that has been optimized using the Math Kernel Library (MKL) [15], with a batch size of 5000 images.

### 4.1 Metadata Access

As previously discussed in Section 2, metadata consists of the output of a neural network about the contents of the visual data. In our experiments this is a set of confidence values, one for each object the neural network can detect (discarding objects with a confidence value lower than 0.1). A typical approach used when looking for a specific set of images (such as photos of your dog) is to store confidence values in a NumPy matrix, or for more complex queries in a relational database. As our baseline for comparison, the metadata was stored in either an in-memory NumPy [31] matrix or in a MemSQL in-memory row-store.

We constructed two tests for the metadata access: a timing test of a 2-hop neighbor query ("find all the images that contain cars on beaches") and the time of a simple query ("find all images that contain beaches") as the total number of images increased. Images were taken from the YFCC100M dataset [30].

**Setup** For emulating and evaluating PM, we use the Intel® Persistent Memory Emulation Platform (PMEP) that has been used in other persistent memory research [9, 8, 33]. PMEP is configured with an average predicted latency of 300ns to emulated PM; compared to the average latency of 90ns to local DRAM memory.

Our base comparison for relational databases is MemSQL [20], which provides high throughput and features an in-memory row store and an on-disk column store. We chose to use MemSQL because it does not require the use of a buffer pool (such as MySQL [23]) which can cause contention, and because the combination of code generation and lock-free data structures cause MemSQL to perform very fast in memory. Additionally, MemSQL is designed to be an in memory database, similar to PMGD (though the targeted memory is PM in the case of PMGD). We intend to scale out PMGD, and MemSQL is fully distributed, making it possible to continue to do direct comparisons.

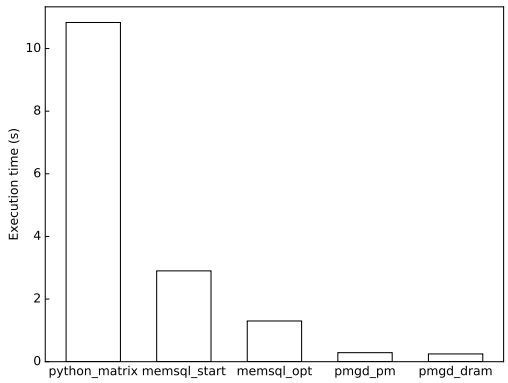


Figure 4: Total time to query metadata to find the desired set of images (cars on beaches). PMGD outperforms other solutions, even at the PM latency.

**Query Time** Figure 4 shows the total metadata query time, calculated as the amount of time it took to return a set of images satisfying the first part of the query ("find all images of beaches") and the amount of time it took to return a set of images satisfying the second part of the query ("of the beach images, find all images with cars"). The total query time with PMGD is 37.5x faster than keeping a NumPy matrix in memory. Due to the relatively small size of the query output, the additional persistent memory latency (pmgd_pm) does not affect PMGD numbers significantly.

We also compared PMGD with storing the metadata in a MemSQL database. Once we indicate which database to load, MemSQL always has data in local DRAM. In order to give MemSQL the best chance, we show the query

times during the execution of the experiment (implying no query caching, seen in Figure 4 as memsql_start) and when the query was run multiple times (memsql_opt). While the optimal MemSQL performance is significantly better than the Python matrix queries (8x), PMGD performs better even at the PM latency. This supports our hypothesis that a graph database is better suited for queries over visual data; more thorough evaluations are on-going work.

**Query Time at Scale** Figure 5 depicts how the performance of a simple query ("find all images of beaches") scales when the number of elements in the database increases. In the figure, python_matrix, memsql, and pmgd_dram were all executed at the DRAM latency with databases for MemSQL and PMGD persisted on a SSD. Pmgd_pm was executed at the PM latency.

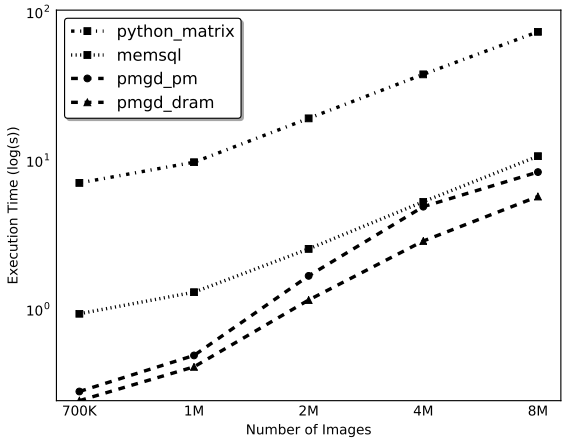


Figure 5: Effect of scaling the dataset size on the metadata query. Note the Y-axis is log-scale. PMGD outperforms other solutions, though could benefit from optimizations at larger scale.

As the number of elements grows, the metadata query time also increases for both the Python matrix and PMGD. However, the Python matrix suffers a much higher latency even though it is a simple query. The expected PM latency affects PMGD more and more as the data size increases but is still almost three times faster than the Python matrix in DRAM. Even reading data at the PM latency, PMGD is always better than MemSQL, though the differences start diminishing at higher data scales. Since we haven't done special optimizations for query caching or execution in our library, we aim to maintain the performance gap with MemSQL as we apply more optimizations at even larger scale.

Table 1: Time to read and resize various size and format images averaged over 400 images. As image size increases, so does the benefit of using the TDB format and the VCL.

| Image Size | Image Format | Resize Method | Time (ms) |
|---|---|---|---|
| 500x291 | JPEG | OpenCV | 2.85 |
| 500x291 | TDB | OpenCV | 0.37 |
| 500x291 | TDB | VCL | 1.36 |
| 1024x2048 | PNG | OpenCV | 68.04 |
| 1024x2048 | TDB | OpenCV | 4.63 |
| 1024x2048 | TDB | VCL | 2.89 |

## 4.2 Data Access

Since data access is becoming an issue with newer hardware and software (Figure 1), we are in the process of developing the Visual Compute Library (VCL), which interfaces with TileDB in order to provide fast access to images and regions of interest within an image. Since the image information in TileDB is stored as an array, VCL also introduces computation directly on the array.

Preliminary experiments tested reading the image from the original format (JPEG, PNG, or TileDB (TDB)) into OpenCV and using the OpenCV resize function, as well as reading a TDB image into the VCL and using the VCL resize function. Both large and small images benefit from being stored in TDB format, as seen in Table 1, and using the VCL to resize sees a 23.5x improvement for large images. The reason it is faster to resize the smaller TDB image using OpenCV is because the VCL resize function has not been optimized yet. This speedup comes at the cost of an increased size on disk when storing an image in the TDB format. Determining how to decrease the size on disk while maintaining the fast access time is on-going work.

# 5 Conclusions

Initial experiments with our architecture have shown the importance of developing storage solutions with visual data as a primary element. We have presented our architecture design, which seamlessly integrates cutting-edge hardware with novel software. It is our goal to spark thought and discussion about this problem in the storage community, and we hope that this paper provides a starting point for discussion and exploration in visual data storage.

# 6 Acknowledgments

Many thanks to Ian F. Adams and Scott Hahn for their contributions, as well as to the program committee.

## References


[1] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., AND ZHENG, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] AURELIUS. Titan documentation, 2015.

[3] BEAVER, D., KUMAR, S., LI, H. C., SOBEL, J., VAJGEL, P., ET AL. Finding a needle in haystack: Facebook's photo storage. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)* (2010), vol. 10, pp. 1–8.

[4] BRONSON, N., AMSDEN, Z., CABRERA, G., CHAKKA, P., DIMOV, P., DING, H., FERRIS, J., GIARDULLO, A., KULKARNI, S., LI, H., MARCHUKOV, M., PETROV, D., PUZAR, L., SONG, Y. J., AND VENKATARAMANI, V. Tao: Facebook's distributed data store for the social graph. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)* (San Jose, CA, 2013), USENIX, pp. 49–60.

[5] BROWN, P. G. Overview of scidb: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (2010), ACM, pp. 963–968.

[6] BRUGGEN, R. V. Learning neo4j, 2014.

[7] CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. E. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS) 26*, 2 (2008), 4.

[8] DEBRABANT, J., ARULRAJ, J., PAVLO, A., STONEBRAKER, M., ZDONIK, S. B., AND DULLOOR, S. A prolegomenon on OLTP database systems for non-volatile memory. In *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS 2014, Hangzhou, China, September 1, 2014.* (2014), pp. 57–63.

[9] DULLOOR, S. R., KUMAR, S., KESHAVAMURTHY, A., LANTZ, P., REDDY, D., SANKARAN, R., AND JACKSON, J. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 15:1–15:15.

[10] FAN, J., RAJ, A. G. S., AND PATEL, J. M. The case against specialized graph analytics engines. In *7th Biennial Conference on Innovative Data Systems Research (CIDR 15)* (2015).

[11] FOGAL, T., SCHIEWE, A., AND KRÜGER, J. An analysis of scalable gpu-based ray-guided volume rendering.

[12] FRANZ INCORPORATED. AllegroGraph 5.0 introduction, 2015.

[13] HUSTON, L., SUKTHANKAR, R., WICKREMESINGHE, R., SATYANARAYANAN, M., GANGER, G. R., RIEDEL, E., AND AILAMAKI, A. Diamond: A storage architecture for early discard in interactive search. In *Proceedings of USENIX Conference on File and Storage Technologies (FAST 04)* (2004).

[14] INC, O. InfiniteGraph, 2015.

[15] INTEL CORPORATION. Developer reference for intel math kernel library 2017 - c, 2017.

[16] INTELPR. Intel and micron produce breakthrough memory technology, 2015.

[17] JIA, Y., SHELHAMER, E., DONAHUE, J., KARAYEV, S., LONG, J., GIRSHICK, R., GUADARRAMA, S., AND DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia* (New York, NY, USA, 2014), MM '14, ACM, pp. 675–678.

[18] LAMB, A., FULLER, M., VARADARAJAN, R., TRAN, N., VANDIVER, B., DOSHI, L., AND BEAR, C. The vertica analytic database: C-store 7 years later. *Proceedings of the VLDB Endowment 5*, 12 (2012), 1790–1801.

[19] M, G. What is intel optimized caffe?, 2016.

[20] MEMSQL INC. How memSQL works, 2017.

[21] MICRON TECHNOLOGY, INC. Hybrid memory: Bridging the gap between DRAM speed and NAND nonvolatility, 2013.

[22] MURALIDHAR, S., LLOYD, W., ROY, S., HILL, C., LIN, E., LIU, W., PAN, S., SHANKAR, S., SIVAKUMAR, V., TANG, L., AND KUMAR, S. f4: Facebook's warm blob storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, 2014), USENIX Association, pp. 383–398.

[23] ORACLE. MySQL, 2017.

[24] PAPADOPOULOS, S., DATTA, K., MADDEN, S., AND MATTSON, T. The tiledb array data storage manager. *Proc. VLDB Endow. 10*, 4 (Nov. 2016), 349–360.

[25] PERONA, P. Vision of a visipedia. *Proceedings of the IEEE 98*, 8 (2010), 1526–1534.

[26] QURESHI, M. K., SRINIVASAN, V., AND RIVERS, J. A. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture* (Austin, TX, USA, 2009), ISCA '09, pp. 24–33.

[27] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., BERG, A. C., AND FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[28] SATYANARAYANAN, M., SUKTHANKAR, R., MUMMERT, L., GOODE, A., HARKES, J., AND SCHLOSSER, S. The unique strengths and storage access characteristics of discard-based search. In *Journal of Internet Services and Applications* (2010).

[29] STANFORD. Convolutional neural networks, 2015.

[30] THOMEE, B., SHAMMA, D. A., FRIEDLAND, G., ELIZALDE, B., NI, K., POLAND, D., BORTH, D., AND LI, L.-J. Yfcc100m: The new data in multimedia research. *Commun. ACM 59*, 2 (Jan. 2016), 64–73.

[31] VAN DER WALT, S., COLBERT, S. C., AND VAROQUAUX, G. The numpy array: A structure for efficient numerical computation. In *Computing in Science and Engineering*, vol. 13. 2011.

[32] XIN, R. S., ROSEN, J., ZAHARIA, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Shark: Sql and rich analytics at scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (2013), ACM, pp. 13–24.

[33] ZHANG, Y., YANG, J., MEMARIPOUR, A., AND SWANSON, S. Mojim: A reliable and highly-available non-volatile memory system. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (2015), ASPLOS '15, pp. 3–18.