Scalable Image AI via Self-Designing Storage

Utku Sirin, Victoria Kauffman, Aadit Saluja, Florian Klein, Jeremy Hsu,
Vlad Cainamisir, Qitong Wang, Konstantinos Kopsinis, Stratos Idreos
Harvard University

Abstract

Image AI has the potential to improve every aspect of human life. Image AI, however, is very expensive. We identify that the root cause of the problem is a long-overlooked and largely unexplored dimension: storage. Most images today are stored as JPEG files. JPEG is designed for digital photography. It maximally compresses images with minimal loss in visual quality. We observe that JPEG is a fixed design. AI problems, however, are diverse; every problem is unique in terms of how data should be stored and processed. Using a fixed design, such as JPEG, for all problems results in excessive data movements and costly image AI systems. This paper presents Image Calculator, a self-designing storage system that finds the optimal storage for a given image AI task. The Image Calculator achieves this by identifying design primitives for image storage and creating a design space comprising thousands of storage formats based on these design primitives, each capable of storing and representing data differently, with varying accuracy, inference and training time, as well as space consumption trade-offs. It efficiently searches within this design space by building performance models and using locality among its storage formats. It exploits the inherent frequency structure in image data to efficiently serve inference and training requests. We evaluate the Image Calculator across a diverse set of datasets, tasks, models, and hardware. We show that Image Calculator can generate storage formats that reduce end-to-end inference and training times by up to 14.2x and consumed space by up to 8.2x with little or no loss in accuracy, compared to state-of-the-art image storage formats. Its incremental computation and data-sharing schemes over frequency components allow scalable inference- and training-serving systems.

1 Efficient Image AI Storage

Image AI Has Potential to Improve Every Aspect of Human Life. Image AI has shown great success in numerous areas, providing more productive and safer services and tools. Medical doctors now use image AI to support their decision-making process in the early and late detection of diseases. Companies deploy image AI tools to enhance worker safety conditions. Manufacturing companies use image AI in their production pipelines to increase productivity. Farmers use image AI to efficiently monitor the conditions of their crops and take preventive actions against potential diseases, further improving their yields [1, 2, 3].

Image AI is Prohibitively Expensive. Every year, billions of digital cameras capture trillions of images. These images consume thousands of petabytes of storage in the cloud. Numerous AI applications process these massive datasets for various purposes, such as personalized content management, image reconstruction, and visual

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

captioning. Applications access data over remote storage servers and need to move and process data in compute nodes using high-end processors. This incurs an immense dollar cost for cloud vendors and application developers. It costs millions of dollars to train and deploy a single model, which generates hundreds of thousands of pounds of carbon emissions, making image AI inaccessible [4, 5].

Inference and Training are Equally Costly. The lifetime of an AI model includes two main stages: training and inference. Training is where the model learns how to perform the task. Inference is where the model is deployed and performs a learned task. Both training and inference are important cost components. While some applications perform frequent re-trainings by newly arriving data, some applications are deployed across billions of devices concurrently and continuously performing inference, and some other applications perform both training and inference equally frequently [6, 7, 8, 9].

Where Does Time Go? In Figure 1, we examine the end-to-end inference times of four state-of-the-art AI models: ResNet, EfficientNet, ShuffleNet, and MobileNet. For this experiment, the server is an A100 GPU machine. The data resides on disk as JPEG files. The model resides on the GPU. Each inference call requires reading the data from disk to main memory, decoding it from the bytes stream into a human-recognizable image, transferring it from main memory to GPU, and executing the AI model on the GPU. The graph on the left-hand side presents the normalized number of floating-point operations (FLOPs) required to execute each model over an image, and the graph on the right-hand side presents the end-to-end inference time. The figure shows that, although models become smaller and smaller, and the number of FLOPs decreases by as much as 98%, the inference time remains constant. The reason is that reducing the number of flops reduces the model execution time, i.e., the GPU time, but not the other time components. When the GPU time is reduced, the other time components surface up, and the inference time remains the same[1]. This suggests that inference time is a complex function of multiple time components, and reducing inference time requires reducing all time components. We performed the same experiment for training time and reached the same conclusion.

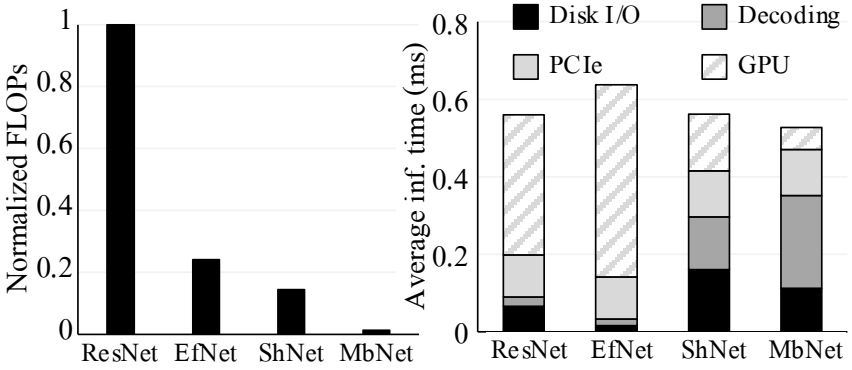

Figure 1: Reducing image AI cost requires improving all cost components.

Storage is Key. We observe that all the time components depend on one main factor: the amount of data moved/processed. This is determined by the storage format used for images. First, the storage format defines how we compress and store the data as well as how much data we read from the disk or network. Therefore, the storage format defines the I/O and decoding times. Second, PCIe time depends on the amount of data transferred over the PCIe link. PCIe stands for Peripheral Component Interconnect Express, and it serves as a bridge between the CPU and GPU. The amount of transferred data over PCIe depends on image size and, hence, storage format. Third, storage also determines the GPU time. The state-of-the-art AI models for images, i.e., CNNs and visual transformers, are primarily composed of transformers and convolutions, where inference and training times heavily depend on the height and width, i.e., the spatial dimensions of the image, which are part of the storage format.

JPEG is Designed for Digital Photography. Most images today are stored as JPEG files. JPEG is designed for digital photography. It lossily compresses images, where lost information minimally distorts the visual quality of the image. Users capture images for artistic, recreational, and/or communication purposes and store them

[1] EfficientNet has a higher GPU time than ResNet, although it requires only a quarter of FLOPs due to its more expensive pointwise operations (ReLU & BatchNorm).

using JPEG's algorithm so that stored images consume significantly less space than their raw versions. Thanks to JPEG's compression algorithm, storing and transferring images became much cheaper, which allowed the proliferation of images across the internet and social media.

AI is Diverse. We observe that JPEG is a fixed design. AI problems, however, are diverse. Every image AI problem has unique characteristics regarding how data should be stored and processed. We define an AI problem as a quadruple: dataset, AI model, machine, and performance budget. Any change in any dimension of an AI problem creates a completely new problem and requires a re-design of the storage. For example, a storage with five milliseconds of inference budget could be completely different than a storage with a hundred milliseconds inference budget. Therefore, using a fixed design for all problems is inefficient and can lead to severe performance issues due to excessive data movement.

Thesis. Our thesis is as follows.

Storage determines end-to-end image AI cost. Using a single storage for all problems results in inefficient image AI systems. Efficient systems need to tailor storage to the AI problem.

Solution: Self-Designing Storage for Image AI. We introduce Image Calculator, a self-designing storage system for image AI. Figure 2 presents an overview of the Image Calculator. Unlike the current state of the art, which uses fixed storage designed for a single purpose, such as minimizing interference with the human eye, Image Calculator automatically generates and manages new storage for every AI task. Image Calculator achieves this by identifying design primitives for image storage and creating a rich design space of storage formats based on these primitives, each capable of storing and representing images at a different trade-off between space, inference/training time, and accuracy. The Image Calculator determines the optimal storage format for inference and training, given source data, hardware, and performance budget [12]. It efficiently searches within its design space based on performance models and locality among its storage formats. Formats with similar features also have similar performances. Starting from information-dense formats, Image Calculator quickly identifies high-quality storage candidates, allowing for fast and scalable inference and training. It exploits the inherent frequency structure in image data to efficiently scale model serving across a large number of applications. The Image Calculator breaks images into pieces, i.e., frequency components, and stores, transfers, and processes images frequency by frequency, rather than image by image, as conventionally done. This dramatically reduces data communication between clients & servers, providing a fast and scalable inference and training serving [13].

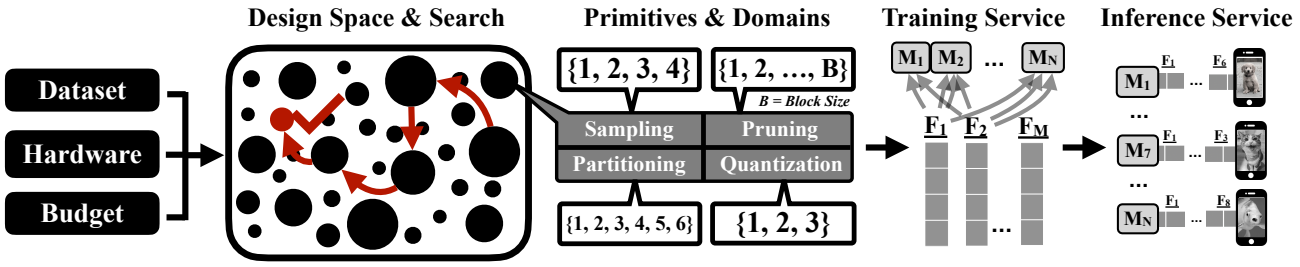

Figure 2: Image Calculator generates new image storage formats as combinations of fundamental design primitives. It exploits locality among storage formats for efficiently finding high-quality candidates. It stores, transfers, and processes images frequency by frequency instead of file by file for efficient serving of inference and training requests. Domains represent ID values rather than actual values. Sampling ID 4 means no sampling, for example. M_1 refers to first AI model, and F_1 refers to first frequency component.

2 Design Space

The Image Calculator adapts storage to a specific AI task. It achieves this by finding design primitives for image storage and creating a design space of storage formats based on design primitives.

Design Primitives for Image Storage. Creating a design space requires breaking down image storage formats and understanding their main design primitives. A rich design space encompasses a large number of candidates, each offering a distinct trade-off between space, inference/training time, and accuracy. We studied most used image/video storage formats such as JPEG [14], JPEG2000 [15], Learned JPEG [16], PNG [17], Bitmap [18], WebP [19], HEIF [20], AVC [21], and HEVC [22]. We show that four main design primitives cover all standard storage formats: (i) **Sampling**, (ii) **Partitioning**, (iii) **Pruning**, and (iv) **Quantization**. Sampling defines removing a set of columns and/or rows from the image. Partitioning refers to decomposing an image into blocks and storing or reading them block by block rather than as a whole image. Pruning refers to the removal of unnecessary data from the image. Quantization refers to reducing the magnitude of the values in image data so that they can be encoded with a smaller number of bits.

These four primitives govern all canonical storage formats. JPEG in its default settings, for example, samples images at every other row and column, partitions them into 8x8 blocks, and uses a specific 8x8 quantization matrix to quantize every data value within the block with a specific quantization factor. Learned JPEG, on the other hand, in addition to performing the same sampling, partitioning, and quantization as standard JPEG, also prunes unuseful data based on a learned model defining which data items are useful and which are not. Video storage formats follow image storage formats, except that they use sophisticated algorithms for exploiting the temporal relationship within the video data. They, too, use similar primitives, such as partitioning, sampling, and quantization when storing data.

Frequency-Domain Representation. There is one design primitive that all standard formats make the same decision: image representation. Standard formats use a human-recognizable color space representation of images, such as the red-green-blue (RGB) representation. Recently, several studies have shown that AI problems can be solved equally successfully by using frequency-domain representation [16, 23]. This representation enables fast image reconstruction, thereby significantly reducing inference time. Image Calculator follows this trend of frequency-domain representation, aiming for high efficiency in AI inference and training. When reading an image, it simply converts the byte stream into a set of frequency coefficients in a single step and uses it as is to train AI models and perform inference with them. This significantly reduces image reconstruction time and also enables scalable image representation, thanks to efficient data pruning, as described later in this section.

Domains for Design Primitives. Design primitives define the fundamentals when storing images. They are, however, useful only when we know how to perform each design primitive. A domain defines the set of possible ways to perform a specific design primitive. Every design primitive has its domain. Every combination of every domain value across all design primitives defines a different storage format, and all such combinations collectively define the total design space. There are, however, exponentially many ways to perform each design primitive. An RGB image, for example, can be sampled in an exponentially large number of ways. Similarly, we can partition the image into blocks of any size and prune them in an exponentially large number of ways. Consider a block size of 8x8, i.e., images are partitioned into 8x8 blocks when storing, and assume that we apply the same pruning strategy to all blocks of the image. How many possible pruning strategies are there for, say, six different block sizes: $8x8$, $16x16$, $32x32$, ..., $256x256$? For $8x8$ block size, there are $2^{8x8} = 2^{64}$ possible pruning strategies. For $16x16$ blocks, there are $2^{16x16} = 2^{256}$ strategies. In total, this makes $2^{8x8} + 2^{16x16} + ... + 2^{256x256} > 10^{150K}$, which is beyond any computational capacity to search within.

Dimensionality Reduction. We perform sensitivity analysis for each design primitive to determine a feasible domain. Our goal is to reach conclusions for each design primitive such that certain values of their domains perform significantly better than others. This way, we aim to determine the most valuable set of design decisions for each primitive and create a feasible yet high-quality design space. When performing the sensitivity analysis, we fix values of all design primitives except the primitive under test. We then vary the values of the primitive

under test from a very small value to a very large value and analyze how various AI problems, using different datasets and AI models, perform across different ranges of values. If we identify a group of domain values that are worse than others, we exclude them from the domain. If all domain values perform similarly, we uniformly sample a feasible set of domain values. This way, we identify the most valuable design decisions for each design primitive and create a high-quality design space.

#1 Sampling. Sampling primitive removes some rows and columns of pixels from the data. It has two aspects: which sampling strategy to use and which channels to sample. For the first aspect, we use four sampling strategies: sampling every other column, sampling every other row, sampling every other row and column, and no sampling. We observe that these strategies effectively cover the range. For the second aspect, we perform the following analysis. Once they are captured, images are typically stored in the red-green-blue (RGB) color space. Most standard formats transform them into Y-Cr-Cb color space before storing them. The reason is that Y-Cr-Cb color space separates brightness information from the color information. Y channel carries the brightness information, i.e., it is the black-and-white version of the original image. Cr and Cb channels carry the color information. Standard formats observe that the human eye is more sensitive to brightness information than to color information. Hence, they use a more aggressive sampling strategy for the color channels than for the brightness channel. We test this hypothesis against AI problems and observe that it does not hold for AI problems. Color information can be equally successful as brightness information for performing various AI tasks. Hence, once the sampling strategy is chosen, we sample all channels using the same sampling strategy.

#2 Partitioning. Partitioning primitive decides how to decompose images into blocks. We analyzed six block sizes of $8x8$, $16x16$, ..., and $256x256$. Each block size provides a different representation with varying accuracy offerings for different AI tasks. We observed that no block size is better than the others. Hence, we keep all six block sizes and use them as the domain of the partitioning primitive.

#3 Pruning. Pruning primitive removes some data values from the image. It is similar in definition to sampling but differs in its implementation. Most standard formats store data in the so-called frequency domain. Once an image is captured, it is usually in a color space, such as RGB. When storing images, most standard formats perform a frequency transformation, such as the Discrete Cosine Transformation (DCT) [10], which converts color-space representation into a frequency-space representation. This transformation is on top of the color-space transformation mentioned in the previous paragraph. Frequency-space representation allows reasoning about the data. Every value is a weight, i.e., a coefficient for a specific signal with a specific frequency. High-frequency signals represent fine-grained details on the image, whereas low-frequency signals represent coarse-grained information, such as the shape of a human body. Standard formats profile the human eye and observe that it is more sensitive to low-frequency signals than to high-frequency signals. Hence, standard formats compress high-frequency signals more aggressively than low-frequency signals, providing efficient storage.

Low-Frequency Coefficients Are Much More Useful Than High-Frequency Coefficients. We use the frequency structure in image data to design the pruning domain. We test four pruning strategies. As an image is a two-dimensional data, i.e., a two-dimensional signal, frequency transformation includes signals whose frequencies increase vertically and horizontally. In our first pruning strategy, we always retain the lowest horizontal and vertical frequency signals and prune everything else. As we prune less and less, we add higher and higher frequency signals, both in the horizontal and vertical dimensions. In our second pruning strategy, we always retain the highest frequency signals in both horizontal and vertical directions. As we prune less and less, we add lower and lower frequency signals both at horizontal and vertical dimensions. In our third and fourth pruning strategies, we retain the highest horizontal and lowest vertical frequency signals, as well as the highest vertical and lowest horizontal frequency signals. Again, we add more and more signals in the opposite direction as we prune less and less. The third and fourth strategies test whether horizontal or vertical frequencies play a different role in image AI problems. We compare each strategy based on their performance using various AI problems. We observe that low-frequency signals have a clear superiority over all other types of signals. Hence, when pruning, we first prune the highest frequency coefficients at both horizontal and vertical dimensions and prune lower and lower frequency signals as we prune more. This constitutes the domain of the pruning primitive. For

a given image, there is now a strict order and a small number of possible pruning strategies in terms of what signals should be pruned and what signals should be retained.

Removing Frequency Coefficients Allows Scalable Representation of Images. Eliminating the unuseful frequency signals has two advantages. First, it allows for significantly reducing the data size, as we physically remove some values from the dataset. Secondly, it provides a scalable representation of images in the main memory. We eliminate unuseful frequency signals outside the boundaries of the chosen triangle, as shown in Figure 3. As a result, the spatial dimensions of the image are reduced. For example, the image size in Figure 3 is reduced from 256x256 to 24x24 after we remove the frequency signals outside the chosen triangle. This reduction in terms of the spatial dimensions of the image allows for reducing disk I/O, decoding, PCIe times, and also GPU time.

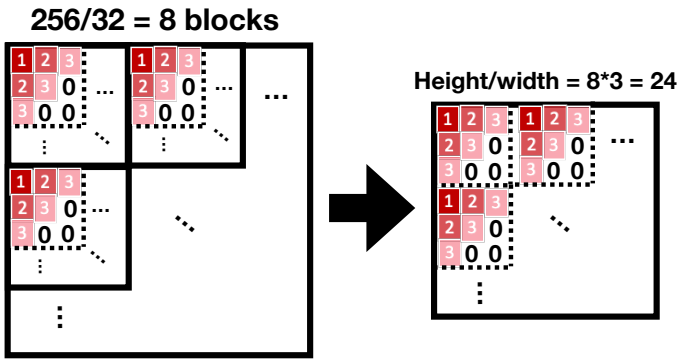

Figure 3: Image Calculator removes frequency coefficients outside the boundary of the chosen triangle.

#4 Quantization. Quantizing a value refers to dividing it with a quantization factor and rounding it to its nearest integer. This reduces the magnitude of the values, allowing them to be encoded with a smaller number of bits. Quantization is essential for reducing the consumed space, though it does not affect PCIe and GPU times and only minimally affects disk I/O and CPU decoding times. Even a small degree of quantization, e.g., quantizing with a small factor of 2, dramatically reduces the amount of data stored. This is because every value becomes smaller and hence needs a smaller number of bits to encode. Furthermore, as the values decrease, the number of repetitions across all values within the dataset increases. Modern encoding algorithms such as DEFLATE [11] exploit repetitions in the data and use a symbol table, where the most frequently occurring data items are encoded with the smallest number of bits. As values are quantized more and more, they get closer to each other and, hence, benefit more and more from repetitions. The trade-off is to quantize values as much as possible without losing much accuracy. In our experiments, we observed that quantizing values up to a factor of 20 typically does not result in any accuracy loss. Quantizing with factors larger than 20 results in an accuracy loss, and typically in proportion to the quantization factor. Hence, we choose 20 as the minimal quantization factor and select 50 and 100 for medium and high degrees of quantization, respectively.

Desing Space: ∼4K Storage Formats. Having defined the domains, we can now count the total number of elements in our design space. There are three quantization factors, four sampling strategies, six block sizes, and as many pruning strategies as the dimension of each block size. This variation in total sums up to 4104 storage formats, which is large yet feasible enough to search within.

3 Inference Time

Having defined the design space, Image Calculator first reduces inference time by adapting storage to the AI task. It assumes the AI model is part of the AI task and, hence, user-specified. It aims to find the best-performing storage formats across various inference time budgets, allowing users to decide which storage format works best for their own use case. Image Calculator uses **performance models** to achieve this. A performance model is a predictive model that maps every element in the design space to a metric of interest, such as inference time, space consumption, or accuracy. Building a performance model for inference time and space consumption is straightforward, as hardware is easily accessible today through various cloud vendors, and it is inexpensive to profile hardware and obtain real-life measurements. Building a performance model for accuracy is a challenging

task. The brute-force method would train an AI model for each storage format in the design space and offer an accuracy-cost trade-off across ∼4K candidates. This, however, means thousands of AI model training and is prohibitively expensive.

Bucket Sampling Reduces Accuracy Model Construction Time by 3x. Image Calculator uses sampling, interpolation, and transfer-learning-based methods to construct a performance model for the accuracy metric efficiently. We first observe a diminishing return relationship between inference time and accuracy. As storage formats become increasingly costly in terms of inference time, they also deliver higher accuracy. This is because high inference time implies expensive storage formats, which are characterized by high information density and, therefore, deliver high accuracy. Image Calculator exploits the diminishing returns curve and performs bucket sampling over the design space of storage formats. It first partitions the design space into a set of buckets, where each bucket contains storage formats with similar inference times. It then performs bucket sampling such that the sampled buckets define the overall shape of the trade-off curve reasonably well. It then trains an AI model specifically for the storage formats included in the sampled buckets. We show that this reduces the number of storage formats for which Image Calculator requires training an AI model from scratch from ∼4K to ∼1.3K, i.e., provides about 1/3 of savings.

Transfer Learning Further Reduces Accuracy Model Construction Time by 10x. We observe that storage formats in Image Calculator's design space are correlated. Consider the storage format that uses 16x16 blocks, prunes all frequency signals except the lowest four, and employs no sampling, with a quantization factor of 20. This storage format stores the data only slightly differently than the storage format that uses the same block size, sampling strategy, and quantization factor but prunes all frequency signals except the lowest five. Therefore, an AI model trained with a specific storage format could share its learned model parameters, i.e., weights, with other storage formats to accelerate their learning. In deep learning literature, this is referred to as transfer learning. Image Calculator trains only one AI model from scratch using the storage format that includes the highest amount of information, i.e., the format that is least pruned, sampled, and quantized. It then shares its weights with all other sampled storage formats. We analyzed four transfer learning strategies: (i) successively going backward from the most information-dense format to the least information-dense format in a sequential manner, (ii) going forward from the least information-dense format to the highest information-dense format in a sequential manner, (iii) parallel transfer learning where all formats learn from the highest information-dense format, and (iv) parallel transfer learning where all formats learn from the lowest information-dense format. We observe that the first transfer learning strategy, which involves successively transitioning from the most information-dense format to the least information-dense format in a sequential manner, yields the highest performance. Hence, we chose this as the Image Calculator's transfer learning algorithm. We show that transfer learning further reduces the accuracy model construction cost by 10x, as every storage format now needs ∼10x less number of epochs to train its AI model for achieving a similar level of accuracy.

Interactive Exploration of Trade-Off. Overall, bucket-sampling, interpolation, and transfer learning bring about ∼30x reduction in computing a performance model for accuracy, compared to the brute-force method of training ∼4K AI models. Once performance models are built, the user can interactively explore the design space and its various trade-offs, making an informed decision on which storage format to use and what accuracy to expect at runtime. Furthermore, all applications using the same dataset but different resource budgets can use the same accuracy-time/space models to make their decisions. As AI applications are typically deployed at a scale of millions across various types of hardware and software platforms and capacities, performance models offer efficient and scalable means for determining the right storage format for each application. We demonstrate that Image Calculator's storage formats can reduce inference time by up to 14.2x and consume space by up to 8.2x with little to no loss in accuracy compared to JPEG and its recent variants across diverse datasets, AI models, and AI tasks [12].

4 Training Time

Next, the Image Calculator aims to reduce training time. Training time encompasses storage format search time, AI model search time, specifically neural architectural search time, and hyperparameter optimization time. There are two challenges in reducing the training time. The first one is reducing storage format, architectural, and hyper-parameter search times. The second one is performing data augmentation in the frequency domain.

Reducing Storage Format Search Time via Unsupervised Learning. Image Calculator reduces storage format search time by using unsupervised learning. It exploits locality among storage formats in its design space. Closely related storage formats also exhibit similar performance. Image Calculator trains an AI model that creates one embedding for each storage format. It then starts with the densest storage format and traces lower-budget storage formats that are closest to it by keeping at least one storage format per inference time budget. It stops when it hits the most sparse storage format. Training an AI model that computes embeddings requires a single AI model training. Finding the densest storage format is a constant-time operation, as we know which storage format is densest in Image Calculator's design space, which is usually among the top-performing storage formats. Tracking back to lower-budget formats that are close to the densest format is also inexpensive, as it involves simply computing cosine similarity across a range of storage formats. As a result, finding high-quality storage formats using unsupervised learning is highly efficient, costing roughly the same as training a single AI model and allowing for scalable storage format search times. We demonstrate that unsupervised learning can identify storage formats with up to 10% higher accuracy than randomly chosen storage formats. Furthermore, they have only 3-5% lower accuracy than the optimal storage formats, which would take hundreds of AI model training using transfer learning, as described in the previous section.

Scaling Neural Architecture Search and Hyper-Parameter Optimization Time via Cheap Storage Formats. Having found a family of high-quality storage formats, Image Calculator performs neural architecture search and hyper-parameter optimization with one of these formats. The user provides two performance budgets: training time budget and inference time budget. The Image Calculator performs architectural and hyper-parameter searches using a storage format that fits within the inference budget, as much as the training budget allows. To illustrate, if the user has an inference budget of 100 microseconds, the Image Calculator determines which storage format, among those produced by its unsupervised learning algorithm, fits within the 100-microsecond budget. It then performs the architectural and hyper-parameter search with that format. This way, it finds the optimal architecture and hyper-parameter for that specific storage format, which allows searching for small training budgets that would otherwise not be possible and/or yield very poor qualities. Image Calculator can work with any neural architecture search algorithm, such as weight-sharing [7], neural predictor [24], and zero-cost [25] algorithms, and any hyper-parameter optimization algorithm, such as sequential [26], and synchronous [27] and asynchronous [28] parallel methods.

Data Augmentation in Frequency Domain. Most neural architecture search studies perform repeated model evaluations, i.e., they train different candidate neural architectures at least partially to evaluate how high/low quality they are. Training an AI model requires data augmentation, such as randomly cropping and resizing different parts of the image and horizontally or vertically flipping the images. Data augmentation today is performed in the visual RGB domain. The Image Calculator uses data augmentation algorithms in the frequency domain. This enables the elimination of the costly image reconstruction phase during training, significantly improving individual training times. It employs existing digital signal processing algorithms for basic image manipulations in the frequency domain, such as block composition/decomposition and sub-band approximation, and implements nine popular data augmentations based on various combinations of these algorithms. This way, it can accelerate individual training time by up to three times.

5 Model Serving

Model-serving systems aim to provide fast and efficient inference [29, 30, 31, 32, 33] and training [34] at scale, where scale is defined by the number of applications concurrently submitting requests with varying latency/time and accuracy requirements. In its next step, Image Calculator scales inference and training times across a large number of applications. Model-serving systems use sophisticated algorithms to optimize resource efficiency, prediction latency, and accuracy jointly. The trade-off typically arises from using various types of AI models with differing cost-accuracy trade-offs. Existing studies all rely on a fixed storage, e.g., JPEG. Image Calculator replaces JPEG with its own design space of storage formats, thereby dramatically reducing all data movement costs. Additionally, the Image Calculator leverages the inherent frequency structure in image data to incrementally perform inference and efficiently share data during training. It breaks images into pieces, i.e., frequency components, and transfers, stores, or processes images frequency component by frequency component rather than file by file, as conventionally done.

Inference Serving: Adaptive AI Model and Storage Selection with Incremental Computation. When serving for inference, Image Calculator communicates with registered applications and asks only for the first set of frequency components for a batch of images. It then performs inference by using only these first frequency components for all images. For those images in which the Image Calculator reaches a threshold of confidence score, it finalizes the image AI task and returns the results. For those who do not, the Image Calculator requests additional frequency components and continues performing inference until it reaches a threshold of confidence score for all images in the batch. This way, it transfers only as much data as necessary for each image, which dramatically reduces data movement and processing costs when serving for inference. The Image Calculator performs inference using an ensemble of AI models. It uses an adaptive model and storage selection algorithm. Given a latency budget, it starts with a large ensemble of AI models over sparse storage formats. As more and more frequency components are transferred over the network, it reduces its ensemble size while increasing the density of the images with newly arriving frequency components. Its goal is to support sparse storage formats with large ensembles, allowing it to reach high confidence early and minimize data movement over the network. Image Calculator proactively prefetches the next frequency components while running the ensemble of AI models, overlapping AI model execution time with data transfer time to reduce end-to-end latency further.

Training Serving: Sharing Data Across Frequency Columns. When serving for training, Image Calculator uses a collaborative training algorithm for all AI models it maintains in its pool and their variants to perform inference with varying numbers of frequency components. Like its inference-serving component, Image Calculator uses the inherent frequency structure within image data for efficient training. It stores images frequency-component by frequency-component instead of file by file. Every image has a fixed number of frequency components, and each frequency component is represented as an array of values. Storing images using the frequency structure is similar to storing a relational database table column by column rather than row by row. In this analogy, a relational table corresponds to a batch of images, a row corresponds to an individual image, and a column corresponds to a specific frequency component across the entire batch of images. In Image Calculator's pool, there are AI models that recognize an increasing number of frequency components. Storing images based on frequencies allows for efficient data sharing across AI models that use a similar set of frequency components. The Image Calculator co-locates such AI models and shares the I/O cost among them during training. This dramatically reduces the I/O cost, as otherwise, Image Calculator would need to create a separate file for every version of the image, even though they are part of the same storage, and would need to perform a separate I/O operation for each. We demonstrate that Image Calculator reduces I/O costs by up to 9x, thanks to its data sharing scheme over frequency columns [13].

6 Summary & Future Work

Image Calculator is a self-designing storage system that creates and manages storage tailored to a specific image AI task. It builds a design space of storage format, where every format offers a different accuracy-cost trade-off in terms of accuracy, inference/training time, and space consumption. It uses performance models and locality among storage formats to efficiently search for high-quality candidates. It breaks images into frequency components and stores, transfers, or processes them frequency by frequency, rather than file by file, for efficient serving of inference and training requests.

The AI stack has been heavily optimized, from the infrastructure layer, which uses massively parallel architectures with high memory bandwidth [35], to the framework layer, which employs libraries with efficient performance tuning methods, such as operator fusion and just-in-time compilation [36]. Storage has been the only component that is "general purpose", meaning it is not specialized for AI. Image Calculator challenges this convention and makes the case for an adaptive storage for image AI tasks. It opens the black box of image storage and sets an ambitious goal: to have AI storage for images that can replace JPEG.

In its next steps, we aim to expand Image Calculator's design space for better and cheaper formats, as well as for new domains and problems such as health [37], autonomous vehicles [38], 3D image reconstruction [39], and video games [40]. Given that neural architectures have design spaces that are orders of magnitudes larger than Image Calculator's design space (10^{21} vs. $4K$), the potential is huge. The Image Calculator's idea is simple and can be applied to other data types, such as video and text. For example, information retrieval systems use similar primitives to Image Calculator for efficient text processing, such as sampling [41], pruning [42], and quantization [43]. The Image Calculator enables the reimagining of the entire image AI pipeline in terms of how data is moved and processed, from bits to the system level. We plan to enhance Image Calculator to an end-to-end data system that holistically manages all data objects produced and consumed by an image AI pipeline, such as activation maps and gradients [44]. Image Calculator makes inference a viable option on CPUs, offering a new cost-performance trade-off. CPUs are equipped with a variety of micro-architectural features that database workloads have long been using [45, 46, 47, 48, 49, 50, 51]. We plan to study novel data systems that provide fast and scalable performance on CPUs by efficiently using micro-architectural resources.

References

[1] Chooch. How to use Computer Vision AI for Detecting Workplace Hazards? *https://www.chooch.com/blog/computer-vision-ai-safety-technology-to-detect-workplace-hazards*, 2023. Accessed on Nov 14, 2024.

[2] Alice Gomstyn and Alexandra Jonker. How to use Computer Vision AI for Detecting Workplace Hazards? *https://www.ibm.com/topics/smart-farming*, 2023. Accessed on Nov 14, 2024.

[3] Gang Yu et al. Accurate Recognition of Colorectal Cancer with Semi-supervised Deep Learning on Pathological Images. *Nature Communications*, 12 (6311), 2021.

[4] Matic Broz. How Many Pictures are There (2024): Statistics, Trends, and Forecasts. *https://photutorial.com/photos-statistics*, 2024. Accessed on July 31, 2024.

[5] Edge Delta Team. Breaking Down The Numbers: How Much Data Does The World Create Daily in 2024? *https://edgedelta.com/company/blog/how-much-data-is-created-per-day*, 2024. Accessed on May 5, 2025.

[6] Emma Strubell et al. Energy and Policy Considerations for Deep Learning in NLP. *ACL*, 2019.

[7] Han Cai et al. Once-for-All: Train One Network and Specialize it for Efficient Deployment. *ICLR*, 2020.

[8] Forbes. Google Cloud Doubles Down On NVIDIA GPUs For Inference. *https://www.forbes.com/sites/moorinsights/2019/05/09/google-cloud-doubles-down-on-nvidia-gpus-for-inference*, 2019. Accessed on May 16, 2023.

[9] HPCwire. AWS to Offer Nvidia's T4 GPUs for AI Inferencing. *https://www.hpcwire.com/2019/03/19/aws-upgrades-its-gpu-backed-ai-inference-platform*, 2019. Accessed on May 16, 2023.

[10] Vasudev Bhaskaran and Konstantinos Konstantinides. Image and Video Compression Standards. *Springer*, 1995.

[11] Peter Deutsch. DEFLATE Compressed Data Format Specification version 1.3. *https://datatracker.ietf.org/doc/html/rfc1951*, 1996. Accessed on December 19, 2024.

[12] Utku Sirin and Stratos Idreos The Image Calculator: 10x Faster Image-AI Inference by Replacing JPEG with Self-designing Storage Format. *SIGMOD*, 2024.

[13] Utku Sirin et al. Frequency-Store: Scaling Image AI by A Column-Store for Images. *CIDR*, 2025.

[14] Gregory K. Wallace. The JPEG Still Picture Compression Standard. *IEEE Transactions on Consumer Electronics*, 38 (1), 1992.

[15] JPEG. JPEG 2000. *https://jpeg.org/jpeg2000*, 2023. Accessed on Jan. 02, 2023.

[16] Kai Xu et al. Learning in the Frequency Domain. *CVPR*, 2020.

[17] Mark Adler et al. PNG Specification. *https://www.w3.org/TR/2003/REC-PNG-20031110*, 2003. Accessed on December 19, 2024.

[18] DigicamSoft. *https://www.digicamsoft.com/bmp/bmp.html*. Accessed on May 5, 2025.

[19] Google for Developers. An Image Format for the Web. *https://developers.google.com/speed/webp*. Accessed on May 5, 2025.

[20] HEIF Technical Information. *https://nokiatech.github.io/heif/technical.html*. Accessed on May 5, 2025.

[21] Gregory K. Wallace. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13 (7), 2003.

[22] Gary J. Sullivan et al. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22 (12), 2012.

[23] Lionel Gueguen et al. Faster Neural Networks Straight from JPEG. *NeurIPS*, 2018.

[24] Colin White et al. How Powerful are Performance Predictors in Neural Architecture Search? *NeurIPS*, 2021.

[25] Mohamed S. Abdelfattah et al. Zero-cost Proxies for Lightweight NAS. *ICLR*, 2021.

[26] Stefan Falkner et al. Robust and Efficient Hyperparameter Optimization at Scale. *ICLR*, 2018.

[27] Stefan Falkner et al. A Generalized Framework for Population based Training. *KDD*, 2019.

[28] Liam Li et al. A System for Massively Parallel Hyperparameter Tuning. *MLSys*, 2020.

[29] Daniel Crankshaw et al. Clipper: A Low-Latency Online Prediction Serving System. *NSDI*, 2017.

[30] Francisco Romero et al. INFaaS: Automated Model-less Inference Serving. *USENIX ATC*, 2021.

[31] Jashwant Raj et al. Cocktail: A Multidimensional Optimization for Model Serving in Cloud. *NSDI*, 2022.

[32] Ferdi Kossmann et al. CascadeServe: Unlocking Model Cascades for Inference Serving. *arXiv/2406.14424*, 2024.

[33] Alind Khare et al. SuperServe: Fine-Grained Inference Serving for Unpredictable Workloads. *NSDI*, 2025.

[34] Wei Wang et al. Rafiki: Machine Learning as an Analytics Service System. *VLDB*, 2018.

[35] NVIDIA. H100 Tensor Core GPU. *https://www.nvidia.com/en-us/data-center/h100*. Accessed on May 5, 2025.

[36] William Wen. Introduction to Torch.compile. *https://pytorch.org/tutorials/intermediate/torch_compile_tutorial.html*, 2023. Accessed on May 5, 2025.

[37] Google AI. Health AI. *https://ai.google/applied-ai/health*. Accessed on May 5, 2025.

[38] Autonomous Vehicles Factsheet. *Center for Sustainable Systems, University of Michigan*, Pub. No. CSS16-18, 2024.

[39] Rao Fu et al. GigaHands: A Massive Annotated Dataset of Bimanual Hand Activities. *CVPR*, 2025.

[40] Hongchi Xia et al. Video2Game: Real-time, Interactive, Realistic and Browser-Compatible Environment from a Single Video. *CVPR*, 2024.

[41] Zhen Yang et al. TriSampler: A Better Negative Sampling Principle for Dense Retrieval. *AAAI*, 2024.

[42] Yueqian Lin et al. SpeechPrune: Context-aware Token Pruning for Speech Information Retrieval. *ICME*, 2025.

[43] James O' Neill and Sourav Dutta. Improved Vector Quantization For Dense Retrieval with Contrastive Distillation. *SIGIR*, 2023.

[44] Utku Sirin and Stratos Idreos. Data Storage and Management for Image AI Pipelines. *SIGMOD*, 2025.

[45] Utku Sirin et al. Micro-architectural Analysis of In-memory OLTP. *SIGMOD*, 2016.

[46] Utku Sirin et al. OLTP On A Server-grade ARM: Power, Throughput and Latency Comparison. *Damon*, 2016.

[47] Utku Sirin et al. A Methodology for OLTP Micro-architectural Analysis. *Damon*, 2017.

[48] Utku Sirin and Anastasia Ailamaki. Micro-architectural Analysis of OLAP: Limitations and Opportunities. *VLDB*, 2020.

[49] Utku Sirin et al. Performance Characterization of HTAP Workloads. *ICDE*, 2021.

[50] Utku Sirin et al. Micro-architectural Analysis of In-memory OLTP: Revisited. *The VLDB Journal*, 2021.

[51] Utku Sirin. Micro-architectural Analysis of Database Workloads. *PhD Thesis. EPFL.* 2021.