The Case for Machine-Oriented Compression

Dan Jacobellis

Contents

1 Head in the clouds

We stand at a crossroads where technological innovation has dramatically expanded the limits of what we can achieve with computation. In particular, the past few decades have seen an unprecedented surge in two areas: cloud computing and machine learning. Our newfound capacity to ingest massive amounts of data and perform super-human analysis on it for just pennies worth of computation has dramatically changed our approach to problem solving.

This explosive growth has not been without its challenges and disruptions.Scientists and engineers have been cast into a whirlwind of change where the boundaries of what's possible are being redefined almost daily. As we navigate this terrain, it is critical to understand exactly how we got here.

1.1 A reading machine for the blind

In 1976, not long after the first digital image sensors became available, Ray Kurzweil unveiled his reading machine for the blind.[1] It was capable of optical character recognition (OCR) in any font and for different types of documents including, books, letters, and receipts. It was capable of scanning the document to identify sequences of characters and reading it aloud using a speech synthesizer at roughly 150 words per minute.

How is it that in 2023, companies like Amazon,[2] Google,[3] and IBM[4] can sell OCR services that require you to surrender your documents to a data center miles away? After all, the computational power in our smartphones vastly exceeds what Kurzweil would have had access to in 1976. Surely it would be better to do this processing locally! It would prevent wasted transmission of millions of bits of raw image data to the cloud. It would work without reliable network access. It would eliminate a host of security and privacy related issues.

I am not alone in putting forth this argument.[5] So why hasn't the needle moved?

1.2 Attention deficit disorder

The tongue-in-cheek title "Attention is All You Need" has been taken quite literally by the ML community.[6] The attention mechanism is no doubt a landmark achievement.[7] allowing sequence models to scale to billions of parameters

[1] A Description of the Kurzweil Reading Machine and a Status Report on its Testing and Dissemination (1977)

[2] Amazon Web Services OCR

[3] Google Cloud Platform OCR

[4] IBM Cloud OCR

[5] Semianalysis: On Device AI – Double-Edged Sword

[6] Google search trends: 5 year comparison between "Transformer model" and "Self-supervised learning"

[7] Wikipedia: Timeline of Machine Learning

and enabling the pretrain-finetune paradigm used by foundation models like ChatGPT[8] and Stable Diffusion.[9]

Unfortunately, it has also been seen as a blank check to make models bigger and more complex. This unending arms race for the best model performance has many casualties. Academic researchers are forced to compete with multi-billion dollar companies for the privilege to buy NVIDIA chips with exorbitant markups.

Meanwhile, other chip makers like AMD, Qualcomm, and ST Microelectronics, each of whom mass produce cheaper and more power efficient inference hardware[10][11][12], are fighting tooth and nail for just a sliver of market share from the NVIDIA-Google AI hardware Duopoly.[13]

Governments and non-tech firms are easily fooled into shoveling huge sums of money into contracts to develop "AI" technologies,[14] many of which are thinly veiled interfaces to free and open source software.[15]

1.3 Filter bank robbery

The roots of the problem are multi-faceted and complex. The deep learning community, including academic researchers, educators, and industry have each contributed to these issues in different ways.

With the rapid pace of development, It has been difficult for university educators to incorporate deep learning into their curricula.[16] In their place, a plethora of free, online courses have sprung up that promise to make you a deep learning expert overnight.[17] Intimidating signal processing jargon, like "polyphase interpolation filterbank"[18], is dropped in favor of equally confusing deep learning jargon, like "transposed convolution with dilation and stride"[19]. The historical thread of technological development is broken in the process, leading to a mysticism surrounding deep learning algorithms. With demand for formal and rigorous ML engineering education outpacing its supply, the process of industrial adoption has been chaotic and expensive.

While most agree that simpler, more efficient models are needed, we are simultaneously tempted by the ease of using massive foundation models like Google's

[8] ChatGPT: Optimizing Language Models for Dialogue (November 30, 2022)
[9] Stable Diffusion Launch Announcement (August 5, 2022)
[10] ML Commons: Datacenter Inference
[11] ML Commons: Edge Inference
[12] ML Commons: Tiny Inference
[13] MarketWatch: Nvidia 'should have at least 90%' of AI chip market with AMD on its heels
[14] Stanford University: The AI Index Report
[15] Palantir Edge AI
[16] Weeping and Gnashing of Teeth: Teaching Deep Learning in Image and Video Processing Classes (2020)
[17] Neural Networks: Zero To Hero
[18] Wikipedia: Polyphase quadrature filter
[19] Pytorch: Transposed Convolution

vision transformer[20], which receive millions of downloads each month. After all, it's hard to argue with the convenience of writing ten lines of python code that instantly converts millions of bits of image data into convenient, information-dense floating point matrix[21] which can immediately be used for tasks like classification[22] and segmentation.[23]

1.4 Deep learning compute myths

When discussing the state of deep learning with fellow students and colleagues, I often find them surprised by the low cost of running deep learning models. They are astonished to learn it's possible to buy a $250 GPU that is capable of cranking out over 10 trillion operations per second[24] and with enough VRAM and train the latest text-to-image models on your own dataset.[25] They are surprised to learn that most smartphones have enough power to run that same text-to-image model locally.[26] They are surprised that an $10 microcontroller which consumes less than 500mW of power[27] can easily run deep learning models for speech recognition and computer vision in real-time.[28]

There is a perception that these models must be expensive, and they must be run on supercomputers with expensive GPUs. We are all witness to the deluge of machine learning apps that have recently emerged. Their simple interfaces for everything from image enhancement to document analysis hide the complex orchestration of cloud resources underneath. For example, Hugging Face Spaces[29] has become ubiquitous for deploying such apps. When running a model on this free platform, the type of AI supercomputer hardware used is proudly displayed. "*Running on T4*," or "*Running on A10G*."

1.5 Embarrassingly parallel, embarrassingly redundant

When I describe common machine learning practices to my colleagues in other fields like robotics, high performance computing, or wireless communication systems, they are shocked by practices that are routine in the machine learning industry. For example, consider the inefficiencies in a typical ML image processing pipeline:

- Generate an image on an **edge device**, such as a smartphone or microcontroller.

[20] Hugging Face: google/vit-base-patch16-224
[21] Vision transformer example
[22] Fine-Tune ViT for Image Classification with HF Transformers
[23] Meta AI: Segment Anything (2023)
[24] Wikipedia: RTX 3060
[25] Using LoRA for Efficient Stable Diffusion Fine-Tuning
[26] Qualcomm: World's first on-device demonstration of Stable Diffusion on an Android phone
[27] Wikipedia: STM32 H7
[28] STM32Cube.AI Developer Cloud
[29] Hugging Face Spaces

- Apply **lossy compression**, which discards most of the bits of the image by **reducing redundancies.**
- Upload the data to powerful **GPUs** in the cloud.
- **Decode the image**, thus reintroducing redundancies, then artificially increase its precision to floating point for compatibility with neural networks.
- Apply expensive deep learning "**feature extraction**" routines to eliminate the redundancies and simplify analysis.
- Apply deep learning models to perform the desired **analysis**, e.g. optical character recognition or object detection.

The paradigm of **machine-oriented compression**[30] addresses these inefficiencies. It is a field in its infancy, but its potential impact is immense. It has the potential to reduce the design complexity of ML systems, making them fairer and more transparent. Moreover, it opens the door for other chip makers to compete with NVIDIA and Google by establishing highly **versatile** and **standardized** foundation models. Compare this with the current flood of foundation models, many of which are deprecated within months. Machine-oriented compression has the potential to create a better ecosystem for offline ML apps running on smartphones, embedded systems, and robotics platforms.

2 Representation learning

Let's revisit Kurzweil's reading machine for the blind. Considering that modern approaches to document analysis and speech synthesis require massive deep neural networks and 2020s supercomputers, how was Kurzweil's machine possible? Or, more importantly, how can we reclaim the efficiency of this decades-old system and apply it our present situation?

In short, the answer is **representation learning**. Unfortunately the term "representation learning" has been confusingly redefined many times, so its worth clarify exactly what we mean. In Kevin Murphy's biblical treatise on machine learning[31], Poole and Kornblinth offer this definition:

"Representation learning is a paradigm for training machine learning models to transform raw inputs into a form that makes it easier to solve new tasks. Unlike supervised learning, where the task is known at training time, ***representation learning often assumes that we do not know what task we wish to solve ahead of time."***

Compare this to Kurzweil's description of his optical character recognition algorithm:

[30] End-to-End optimized image compression for machines, a study
[31] Probabilistic Machine Learning: Advanced Topics

"Each individual character is analyzed by a set of feature extraction routines. The features, or properties, extracted are those that have been found to be relatively invariant for the same character with respect to the kinds of changes that occur across different typestyles. These properties are basically geometric—line segments, concavities, loops, loop extensions, and the positional relationships among these elements. For example, the properties of a standard capital "A" include a single loop and a single south-facing concavity . Once the properties have been extracted, they are compared to stored lists describing each character in the identification set."

By meticulously crafting these features, **Kurzweil, not the machine, learned the representation**. Because this representation was so effective at distilling the structure of text into a few bits of information, he could get by using 1970s microprocessors. Modern foundation models also learn representations. The only difference is that **the machine, not the human, does the learning**.

Machine-oriented compression is the natural progression of representation learning; in addition to being useful for any task, the representation is also **compact** so that it can be efficiently stored or distributed across networks. To understand the emerging field of machine-oriented compression, it's essential to understand the basic tools and techniques that it is being built upon.

2.1 Information theory and data compression

Because of the high density and precision of our the signals we interact with (e.g. audio, images, and video), storage and transmission of the raw signal is rarely practical. For example, a standard 1080p video stream is roughly three billion bits **per second** (2.98 Gbps) before compression, roughly ten times the typical capacity of a 5G cellular network (100-400 Mpbs).

Lossy compression[32] techniques aim to reduce the data's size without excessively compromising its quality. Rate-Distortion Theory[33], a branch of information theory[34], provides a theoretical framework for understanding the trade-off between the amount of data (rate) needed to represent a source and the loss of quality (distortion) of the representation.

While the bit rate is well defined, "distortion" is much trickier. Simple distortion metrics like mean squared error are only loosely correlated to the quality metrics that we actually care about. In audio, image, and video signal processing, a variety of **human perceptual quality metrics**[35][36] were co-developed with lossy compression algorithms. These perceptual quality metrics are now also understood to have extremely deep connections with natural signal statis-

[32] Wikipedia: Lossy Compression
[33] Wikipedia: Rate-Distortion Theory
[34] Wikipedia: Information Theory
[35] Wikipedia: Image quality
[36] Wikipedia: Perceptual Evaluation of Audio Quality

tics and efficient learning.[37] Machine-oriented compression can be viewed as a progression of rate distortion theory from human perceptual quality to **machine perceptual quality**: the ability to analyze and make accurate predictions from lossy representations.

2.2 Conventional codecs

Codecs (short for coder-decoder) such as MPEG and JPEG were standardized in the 1990s and are still widely used for audio, images, and video. Generally, these codecs adhere to the paradigm of transform coding, which involves three main steps:

1. **A time-frequency or space-frequency sub-band decomposition.** For example, JPEG uses the two-dimensional type-II DCT[38] applied to 8x8 blocks. In MPEG layer III[39], an audio signal is efficiently divided into 32 frequency bands using a polyphase filterbank, then each sub-band is further decomposed using the modified DCT. These types of transforms have the effect of decorrelating signal components to eliminate redundancy.
2. **Perceptual quantization.** While data from an audio ADC or image sensor typically have a precision in the range of 4-16 bits, the sub-band decomposition may be carried out with higher precision (often 32-bit floating point) and then quantized to lower precision. a perceptual model is used to allocated bits to each sub-band. The quantization matrix in JPEG[40], for example, results in between 2-8 bits for each DCT coefficient. Most of the complexity of standard codecs is in the sub-band bit allocation procedure.
3. **Entropy coding.** The lossy compression performed in steps (1) and (2) is combined with lossless compression to achieve very high compression ratios. Simple prefix codes such as the Huffman code, Golomb code, and run-length encoding[41] are usually preferred to keep the decoding cheap.

2.3 Vector embedding

The concept of a vector embedding[42] (which we will henceforth refer to as "∇_x embedding" for reasons that will soon become clear) is best explained in the context of natural language processing, the field that gave birth to the attention mechanism and the transformer architecture.

Consider an input character sequence from a discrete alphabet (for example the 128 valid ASCII characters). The standard training procedure for neural

[37] On the relation between statistical learning and perceptual distances
[38] Wikipedia: M-D DCT-II
[39] Wikipedia: MPEG Audio Layer III
[40] Wikipedia: JPEG Quantization
[41] Wikipedia: Entropy Coding
[42] Pytorch: Embedding

networks consists of a high precision (16 or 32 bit floating point) representation at the input neurons and some form of normalization. This works extremely well, in part because of the remarkable ability of the floating point number system to efficiently represent real numbers. When training a neural network, how should we represent our 7-bit characters which represent a category rather than a real number? One approach is to simply one-hot encode.[43] This works, but leads to unreasonable memory requirements and dramatically increases the number of network parameters. The "∇_x embedding" technique instead consists of the following:

- Create a lookup table that maps the each of the 128 possible input tokens to a unique vector in an M dimensional space. We will call this 128 x M matrix the "embedding matrix" and we will initialize it with i.i.d. samples from a standard Gaussian.
- When training, map the input tokens to vectors according to the embedding matrix/lookup table. Use the result of this mapping as the input to the network instead of the categorical variables.
- Train the neural network as usual. That is, compute the gradient of the loss function with respect to the neural network parameters, $\nabla_\theta L$. Moving in the opposite direction of this gradient will adjust the parameters to minimize the loss.
- In addition, compute the gradient of the loss function with respect to the **input**, $\nabla_x L$. Since the input is simply the entries of the embedding table, moving in the opposite direction of this gradient will **learn a representation** that minimizes the loss.

2.4 Autoencoders

A plethora of architectures for autoencoders have been proposed. One commonality among them is a **reconstruction objective**. In other words, the target output should be similar to the input. Stated another way, the loss function includes a **distortion metric**.

The distortion metric can be chosen for mathematical convenience (e.g. squared error), but for an autoencoder to achieve our design goals it is preferable to choose an application specific metric. For example, to optimize for human perceptual quality of images, we might use structural similarity (SSIM) as the distortion metric.

Some notable flavors of autoencoders include:

- The **Variational autoencoder (VAE),** which adds an additional loss term that forces the learned representation towards a specific probability distribution (typically Gaussian). Recently, foundation models for audio,

[43] Wikipedia: One-hot

images, and video have utilized this architecture to improve the efficiency of diffusion models.[44]

- **Vector-quantized VAE (VQ-VAE)**. In addition to the reconstruction loss, a codebook loss similar to the standard vector quantization/k-means objective is added. Neural codecs including Google Soundstream[45] and Meta's EnCodec[46] have recently been standardized and use variants of the VQ-VAE to achieve extremely high compression ratios. Recent audio synthesis models for for speech[47] and music[48] use representations learned from a VQ-VAE.
- The **denoising autoencoder**[49] and **masked autoencoder**[50] leave the target the same, but only provide partial access to the input. These techniques have been extremely successful in modeling language, audio, and images. These are foundational tools used in the paradigm of **self-supervised learning**.[51]
- The **rate-distortion autoencoder**, aka "**nonlinear transform coding**"[52] penalizes the entropy of the latent representation. When originally proposed for image compression, Ballé et. al. established an important theoretical link between this technique and the VAE.[53]

2.5 Implicit neural representations

An newer technique that is rapidly gaining traction is the concept of implicit neural representations (INRs), now commonly referred to as functa. This field of study is still in its infancy, so I will defer to Dupont's recent dissertation on the topic[54], which provides this excellent description:

"Data is often represented by arrays, such as a 2D grid of pixels for images. However, the underlying signal represented by these arrays is often continuous, such as the scene depicted in an image. A powerful continuous alternative to discrete arrays is then to represent such signals with an implicit neural representation (INR), a neural network trained to output the appropriate signal value for any input spatial location. An image for example, can be parameterized by a neural network mapping pixel locations to RGB values."

This basic technique has been massively successful in computer graphics[55] and

[44] High-Resolution Image Synthesis With Latent Diffusion Models
[45] Google Soundstream
[46] EnCodec
[47] VALL-E Neural Speech Codec
[48] MusicLM: Generating Music From Text
[49] Wikipedia: Denoising Autoencoder
[50] Masked Autoencoders Are Scalable Vision Learners
[51] Self-supervised learning
[52] Nonlinear Transform Coding
[53] End-to-end Optimized Image Compression
[54] Neural Networks as Data
[55] Neural Radiance Fields

researchers have recently started to close the gap for other applications like inverse problems[56] and video compression.[57]

3 Machine-oriented compression

I have only begun to scratch the surface on the plethora of representation learning techniques that have been enormously successful at building the current generation of foundation models. I hope that this has helped cultivate an appreciation for the breadth of different architectures and objectives that are used by these foundation models; there is much more to them than simply scaling their size with the attention mechanism. As impressive as these foundation models are, I will argue that **machine-oriented compression will become the next generation of foundation models**.

3.1 First generation foundation models

At present, foundation models are typically trained in two stages (1) **self-supervised pretraining** and (2) **task-specific fine-tuning**. This paradigm works remarkably well for a wide range of tasks, complexities, and model sizes. For example:

- A self-supervised vision transformer is trained using masked patch prediction[58], then fine-tuned to classify pictures of food.[59]
- A self-supervised audio transformer is trained to minimize a contrastive loss[60], then fine-tuned to perform speech recognition.[61]
- A self-supervised text transformer is trained using masked token prediction,[62] then fine tuned using reinforcement learning with human feedback to follow instructions.[63]

Clearly, the representations produced by these models are incredibly powerful. However, such representations were never designed to be able to **reconstruct the original data**. Additionally, design choices (e.g. the training objective and loss function) in the self-supervised training step will affect the type of tasks that the foundation model can be used for. In most cases, only a handful (and in many cases a single) fine-tuning task is considered when designing the self-supervised pretraining model.

What if we **expanded the scope of pretraining** from general-purpose objectives like masked prediction to include one or more anticipated fine-tuning

[56] Wavelet Implicit Neural Representations
[57] Video Compression With Entropy-Constrained Neural Representations
[58] An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale
[59] Fine-tuning image classifier on Food-101 dataset
[60] wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations
[61] Speech Recognition with Wav2Vec2
[62] Improving language understanding with unsupervised learning
[63] ChatGPT: Optimizing Language Models for Dialogue

objectives? This can be considered a type of **multitask learning**[64] which would yield representations that meet several objectives. Anchoring secondary objectives to a core **rate-distortion objective** is a promising way to achieve this. Additionally, there are compelling justifications stemming from the study of **algorithmic information theory**[65].

3.2 Next-generation media codecs

I am not alone in making this case. Machine-oriented compression has been explicitly expressed as the goal of next-generation media codecs, most notably **JPEG AI**. In May 2023, JPEG announced a set of design goals for JPEG AI,[66] including:

- High compression efficiency in terms of rate distortion performance.
- Hardware platform agnosticism
- Effective compressed domain processing for (1) classification, (2) super-resolution, (3) denoising, and more tasks to be specified in the future.

The current JPEG AI proposal only specifies these desired requirements. It only puts forth a very rough picture of a system that might achieve these goals. It our job to materialize these goals into something real. It will likely be several years before any specific architectures are standardized. In fact, It's very possible that JPEG AI will never take off, and suffer the same fate as JPEG 2000 and the plethora of other codecs which have become footnotes instead of seeing widespread adoption. However, this general approach offers several compelling benefits:

- Multitask learning has been shown to have several advantages, especially in terms of **data efficiency** and training convergence speed.[67]
- ML engineers are already building complex systems around representations from foundation models and building custom **vector databases**[68] specifically for them. Machine-oriented compression will grant the critical property of **memory efficiency**.

Still, the most compelling argument for machine-oriented compression is the effect it will have on our deployment strategy for ML models and **hardware agnosticism**.

[64] Wikipedia: Multi-task learning
[65] Wikipedia: Algorithmic Information Theory
[66] The JPEG AI Standard: Providing Efficient Human and Machine Visual Data Consumption
[67] Multi-Task Learning with Deep Neural Networks: A Survey
[68] Wikipedia: Vector database

3.3 Model compilation and deployment

ML deployment scenarios often face stringent size, weight, and power (SWaP) constraints. Depending on these constraints, we can group applications into one of four clusters:

- Online datacenters, which receive lossy compressed input data from the internet and have minimal model inference constraints. Servers in these data centers may utilize general-purpose coprocessors suitable for training (for example, NVIDIA H100) or coprocessors specifically optimized for inference (such as the Qualcomm Cloud AI 100).
- Edge devices, which receive inputs directly, bypassing lossy compression. Examples of such devices include the NVIDIA Jetson[69] or the Thundercomm Edge AI Station.[70]
- Mobile devices, which often include a Neural Processing Unit (NPU) or a Digital Signal Processor (DSP) that shares resources with other applications[71] and can be used for ML acceleration.
- Embedded systems and microcontrollers, which may also include a NPU or DSP, but usually operate on bare metal or using a real-time operating system.

During model training, ML researchers and engineers almost universally use frameworks and libraries (like PyTorch and CUDA) which were not designed with mobile or embedded deployment in mind. Consequently, device manufacturers have begun to develop their own model compilation tools. Some notable examples include the Qualcomm's Neural Processing SDK [72]and Microelectronics's STM32CubeAI.[73]

From the perspective of an ML engineer, the deployment process has several cumbersome steps:

1. Train the models using standard frameworks and libraries like Pytorch and CUDA.
2. Use an interchange format (e.g. ONNX[74]) to store the model and weights.
3. Use the mobile or embedded hardware manufacturer's compilation tools to optimize the model for a specific target.
4. Validate whether the compiled model meets all design objectives. In particular:

[69]Wikipedia: Nvidia Jetson
[70]Thundercomm Edge AI Station
[71]Wikipedia: Quallcomm Snapdragon
[72]Qualcomm Neural Processing SDK
[73]STM32CubeAI
[74]Wikipedia: Open Neural Network Exchange

- Predictive performance, which could be diminished compared to training due to model compression techniques like quantization or pruning.
- Computational performance, which can be evaluated in terms of latency, CPU utilization, memory usage, and so forth.

5. If the model fails to meet one or more design goals, go back one or more steps, possibly retraining or choosing a new hardware platform.

As difficult as this process already is, the supremacy of large foundation models and the pretrain-finetune paradigm leads to even more complexity. In many cases, it might be optimal to distribute a single model across several edge devices. For example, consider the deployment of a voice-activated home automation system, which uses all four types of inference (datacenter, edge, mobile, and embedded).

- The system makes API calls to a large language model running in a datacenter to respond to complex queries.
- An edge device, permanently stationed in the home, is used to fine-tune the speech recognition model for user specific patterns and commands.
- A mobile device runs a speech-to-text model.
- A low power embedded system runs a voice detection model to decide when to engage the speech recognition model.

Machine-oriented compression offers and elegant solution to the deployment problem. Rather that force hardware manufacturers to support the incredibly complex processing pipelines of the latest foundation models, they could simply implement a handful of encoders and decoders specified my a machine-oriented compression algorithm. This is not unlike the status quo prior to ML and cloud computing revolutions; since the inception of media compression algorithms like MPEG in the 1990s, our devices have used ASICs and DSPs to offload the expensive but very predictable computation of media compression away from general purpose CPUs.

3.4 Rate Extortion

Machine-oriented compression is an exciting frontier. It leverages nearly a century of technological progress in information theory and signal processing to amplify the abilities of machine learning systems, and will help us build the next generation of foundation models.

By abstracting the complexities of model deployment, machine-oriented compression will help streamline the adoption of these foundation models into industry. It can reduce the design complexity of ML systems, allowing educators to keep pace in providing a relevant but rigorous understanding of these systems. These are critical challenges as we attempt to safely adopt ML technologies into every sector of our economy from entertainment to defense.

The widespread adoption of machine-oriented compression is fraught with challenges. We should learn from the mistakes of the dozens of media codecs which failed to meet their potential. We must do so by making the development of these technologies open and inviting. It's crucial for the community to come together, sharing expertise, resources, and insights to bring this promising technology to fruition.