Artificial Intelligence in Resource-Constrained and Shared Environments

Sanjay Krishnan, Aaron J. Elmore, Michael Franklin, John Paparrizos,
Zechao Shang, Adam Dziedzic, Rui Liu
University of Chicago
{skr,aelmore,mjfranklin,jopa,zcshang,ady,ruiliu}@uchicago.edu

Abstract

The computational demands of modern AI techniques are immense, and as the number of practical applications grows, there will be an increasing burden on shared computing infrastructure. We envision a forthcoming era of "AI Systems" research where reducing resource consumption, reasoning about transient resource availability, trading off resource consumption for accuracy, and managing contention on specialized hardware will become the community's main research focus. This paper overviews the history of AI systems research, a vision for the future, and the open challenges ahead.

1 Introduction

Over the last 10 years, we have witnessed significant breakthroughs on Artificial Intelligence problems spanning from computer vision to natural language processing [17] [1]. While our mathematical understanding of modeling such problems has certainly improved, this rapid progress is equally a testament to novel computer systems architectures that have enabled high-dimensional model training over vast scales of data, running massive amounts of simulations in parallel, and interfacing AI techniques with user-facing applications to collect even more data.

The necessity of such large systems efforts is the genesis of the "AI-Systems" research community [25]. So far, research has focused on feasibility, i.e., aggressively scaling-up and scaling-out to solve a problem as accurately as possible in an acceptable amount of time. However, AI is a victim of its own success, and the popularity of such algorithms can lead to unsustainable growth in resource usage. The computational demands of training state-of-the-art neural networks are increasing dramatically, e.g., the recent OpenAI GPT-2 [24] ran for several weeks with an estimated training cost of $50K. Similarly, AlphaGo Zero used 64 GPUs, 19 CPU parameter servers for 40 days for training to achieve state-of-the-art results [28].

In the early days of general-purpose computing, managing shared resources emerged as a key problem [26]. The basic theme is still studied today in the form of multitenancy, provisioning, scheduling, and virtualization [31]. As AI applications start to dominate our computational infrastructure, the research community will have to understand this classical problem in a very new context. AI applications have widely varying time-scales spanning from model inference tasks that take milliseconds to training tasks that can take weeks. These applications are often accelerated by specialized hardware, like GPUs, and this means that systems will have to reason about complex placement constraints. Finally, AI applications have a fundamentally *softer* notion of correctness, so there are new opportunities for systems that can manipulate the accuracy of their tenants to control resource usage.

Resource optimization for AI is nascent subject area across the field. For example, many new projects explore compressing models for faster inference time on mobile devices [11–13], others reduce memory consumption during model training [4, 10, 33], compilation techniques for better resource utilization [3], and hardware-acceleration through reduced precision [14, 15, 35]. However, the prevailing focus is still on optimizing the resource usage of a single application, e.g., minimizing the memory usage for a neural network training procedure. We believe that there are interesting, unresolved research questions when we broaden to the perspective of *an entire workload*: a collection of concurrent applications composed of training and inference tasks. We identify three core challenges in building a new resource management for concurrent AI applications on shared infrastructure: (1) autoscaling, (2) multitenancy, and (3) graceful accuracy degradation.

Addressing these challenges requires both a mathematical and systems approach. We need new numerical optimization techniques and new deep learning models that can dynamically adapt to varying resource constraints, as well as a new management layer that handles resource allocation and performance isolation. We additionally have to rethink the design of system service level objectives (*SLOs*) for such applications where there might be global objectives, such as a preference to allocating resources to only the most accurate models. A comprehensive management layer will drastically reduce costs for an organization maintaining AI applications, enable faster development of these applications, and lead to a more sustainable proliferation of AI techniques.

[1] We refer to the trend of Deep Learning and it's applications as Artificial Intelligence (AI) in this paper.

We briefly overview the challenges and our initial work towards them:

Auto-scaling and elasticity. Data-intensive systems are increasingly elastic [18, 19, 23]. They can eagerly acquire more resources when needed or desirable. However, in AI applications, overly eager auto-scaling can lead to instability or even divergence. For parallelized stochastic gradient descent, low-level concerns like the level of parallelism and the frequency of synchronization significantly affect model accuracy and training stability [27]. Safely scaling modeling to use more resources is another important area of research.

Accuracy degradation. Data-intensive systems, such as streaming systems, have classically applied techniques like *load shedding* to degrade accuracy to maintain SLOs [32]. The analog to load shedding in an AI application is a model that degrades its prediction accuracy in response to memory or latency pressure. We explore new neural architectures that can smoothly and dynamically trade-off precision with resource usage, and believe that there is significantly more work to be done in this space.

Multitenancy. We envision a system that is given a workload of concurrent AI tasks and will decide how to multiplex limited resources among them. This approach is fundamentally a form of multitenancy, drawing inspiration from how databases and operating systems support multitenancy or virtualization [9, 29, 30]. We describe how different multitenancy protocols can support inference and training. These protocols specify how the tasks share the underlying resources and achieve resource/performance isolation between tenants.

2 Today's AI Stack

We first overview the modern AI software stack and describe the missing pieces. Figure 1 presents a coarse outline of how AI software is designed today.

2.1 Hardware Interfaces

At the lowest level of the stack there are hardware interfaces for different chipsets used in AI, e.g., CPU, GPU, FPGA. These interfaces define optimized numerical computation primitives, such as matrix multiplications, that can be composed into higher-level operations. These interfaces are proprietary and designed by their respective chip manufacturers. Notable examples include Intel's Math Kernel Library (MKL) [34] and Nvidia CUDA [16]. Resource management, such as cache usage, in these libraries happens within each operation. Scheduling and thread management in these libraries similarity is also at a similar operation-level granularity.

2.2 Execution Libraries

The next higher-level of the stack consists of libraries that compose hardware primitives into higher-level operations used in AI applications today. Such libraries are particularly useful when there are multiple ways to implement a basic operation, and making the choice of implementation requires higher-level information about the program. For example, nVidia's cuDNN [5] library implements various convolution algorithms (the workhorse of computer vision and NLP neural network models) using the low-level CUDA interface. cuDNN automatically selects the best choice of algorithm and tunes internal parameters for the best performance, while maintaining correctness up-to machine precision differences. cuDNN [5] and TVM [3] are the best examples of such execution frameworks.

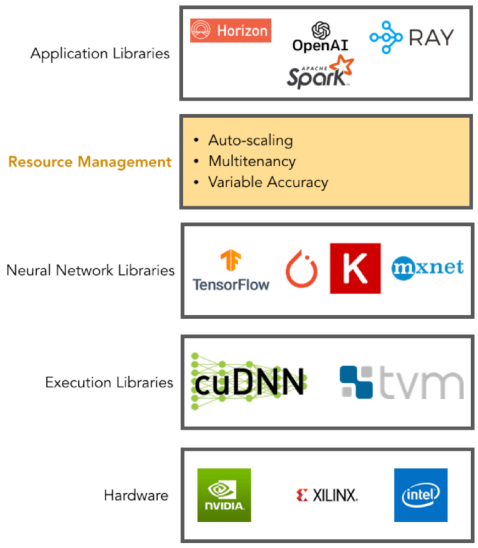

Figure 1. Today's AI stack is missing a resource management layer that interfaces applications to the lower-level frameworks. We see a need for a new layer that allocates and manages resources given to each concurrent application.

2.3 Neural Network Libraries

The popular neural network libraries, such as TensorFlow [1] and PyTorch [22] are built on the aforementioned execution libraries. Neural Network libraries allow composable definitions of neural network *models*. Each model essentially represents a parametrized predictive function: given an input observation, predict some output. These libraries maintain a symbolic description of each model, which is used to generate a *computation graph*: a DAG of numerical operations that represents training and inference.

2.4 Application Libraries

As neural networks mature, they are increasingly interfaced to other system components through application libraries. For example, OpenAI "Baselines" [7] implements Reinforcement Learning algorithms using TensorFlow models, and the Ray project implements a general asynchronous framework to write programs that evaluate and update models [20]. Data

processing systems such as Apache Spark can now interface with Neural Network libraries such as TensorFlow, leading to new systems such as Spark ML and ML Flow. Several projects serve and connect models to external applications via REST [6, 21]. Today each of these applications implements its own resource management and scheduling–as if it was the only application using its allocated resources.

3 Resource Management Layer

Given this architecture, we see a need for a new layer of the stack that allocates and manages resources available to each concurrent application. We outline a basic framework, and then introduce motivating examples.

3.1 What is an AI application?

From a resource management perspective, how should we think of an AI application? There are different granularities of optimizations for AI systems spanning from the chipset level to the macro task level. We focus our attention at the scale of entire models. For this paper, a model is the basic object that our layer will query and manipulate. Each model represents a predictive user-defined function:

```
prediction = model.predict({obs})
```

For example, a model could be a neural network that predicts labels from images, or a linear regression model that predicts a spam score from a word frequency vector. It might also not be statistical at all but rather a simulator that returns a final state of a chemical reaction given an initial input. *A model is a user-defined function with an input and output data type and an associated runtime library (e.g., Tensorflow).*

3.1.1 Updating Models

Models can be updated given new observations and labels. They are often updated incrementally in an iterative algorithm. We abstract the update algorithm, e.g., Stochastic Gradient Descent or Random Search, with an update method that changes the state of the model (takes a single step):

```
model.update({obs}, {labels})
```

This function is also a user-defined routine defined in the model's runtime framework.

3.1.2 Controlling Model Knobs

We envision that each model can additionally expose knobs to the resource management system that affect its accuracy, latency, and memory usage. We would like to be able to automatically set parameters to temporarily degrade accuracy to meet a SLO. For example, if we could sacrifice accuracy for a faster result, our system should be able to make that decision:

```
model.set_param(param, value)
```

3.1.3 AI Applications

For the purposes of our work, every AI application of interest can be described as a sequence (or workload) of concurrent model update() and predict() for possibly many model objects. Supervised learning model training workloads are mostly update() procedures:

```
while not converged:
    {obs}, {labels} = sample(data)
    model.update({obs}, {labels})
```

Serving workloads are exclusively predict() calls:

```
while true:
    {obs} = next(input_stream)
    yield model.predict({obs})
```

And, online learning and reinforcement learning tasks have a mix of both:

```
while true:
    {obs} = observe()
    {res} = action(model.predict({obs}))
    model.update({obs}, {res})
```

In the case of RL, the simulator call itself observe() might be another model for the layer to manage.

3.2 Resource Management Granularity

We argue that a *model* is the right granularity for resource management. Myopic solutions exist at each level of the AI stack but a global resource manager that can consider several concurrent applications is missing. The update() and predict() interface abstracts the algorithmic details but still gives us introspection into which models an application is querying and updating. Our envisioned framework's job is to allocate resources and schedule these update() and predict() calls, and potentially tune model parameters to meet SLOs. Consider the following example scenarios.

Example 1. Multiple Model Training

Modern deep learning training systems increasingly rely on specialized hardware like GPUs or TPUs. Neural network development is a trial-and-error process where parameters are tuned, architectures are tweaked, and a large number of trial models are trained. The consequence is that organizations have many more training jobs than specialized resources available. GPUs today have significantly more onboard memory than in the past: up-to 32 GB in commercial offerings. On the other hand, the largest computer vision neural network models are still in single-digit GBs [2] and the rise of neural networks optimized for mobile devices [13] has led to a number of small-footprint architectures that take up a significantly smaller space. *Allocating one model to one device is wasteful*, and an AI resource management layer

should be able to dynamically multiplex and share available resources among training tasks.

Example 2. Timely Decision v.s. No Decision

Consider a cloud machine learning model serving system serving recommendation requests from music videos for users accessing a website. Suppose the service has reached their budget for scaling out their computational infrastructure, but the request rates are still increasing. Eventually, the prediction latency of the service will start to drop as the processing power on the cloud service and the network saturate. User experience in web applications is highly dependent on service latency and significant delays would render the application useless. In this case, *an inaccurate but timely decision is more desirable than a very accurate but delayed one.* Ideally, an AI resource management layer should automatically degrade accuracy to maintain constant service latency.

4 Towards a Resource Management Layer

In this section, we overview initial work towards a resource management layer. We address both the systems challenges in building such a layer as well as the mathematical challenges in making models that have fine-grained resource control.

4.1 Accuracy-Resource Tradeoffs in Convolutional Networks

Convolutions are core building blocks of neural networks in computer vision and natural language processing. These operations are often the most expensive operations in neural network training and inference. Fast methods for computing convolutions can require a large amount of memory. We explore whether it is possible to gracefully degrade the precision of the convolutional operations when less memory is available during both training and inference. The key insight is to artificially constrain the frequency spectra of convolutional operations—the more highly constrained the more memory efficient. The details of this technique are described in our prior work [8]. The table below shows how test accuracy degrades as a function of compression.

Table 1. We vary the compression ratio from 0 (less compression) to 50% (more compression) and measure the test accuracy of ResNet-18 on Cifar-10 and DenseNet-121 on Cifar-100.

CIFAR	0%	10%	20%	30%	40%	50%
10	93.69	93.42	93.24	92.89	92.61	92.32
100	75.30	75.28	74.25	73.66	72.26	71.18

We observe minimal degradation in accuracy despite the high (i.e., 50%) compression ratio, which leads to significant speedup and reduction in allocated memory (Figure 2). We also found that this tradeoff is smoother and easier to control than reduced precision arithmetic. It requires no further modification to the architecture.

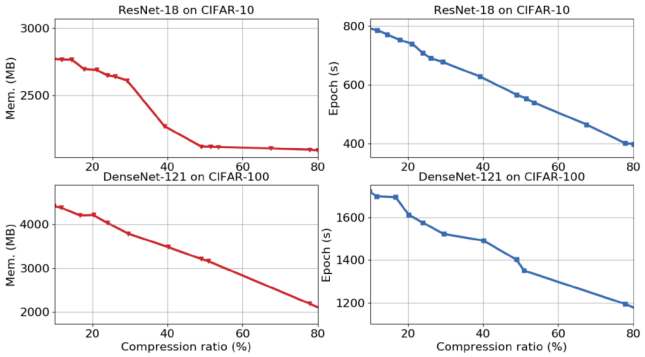

Figure 2. *The resource utilization of compressed model training as a function of the compression ratio. We measure memory and epoch time for ResNet-18 and DenseNet-121 with compression.*

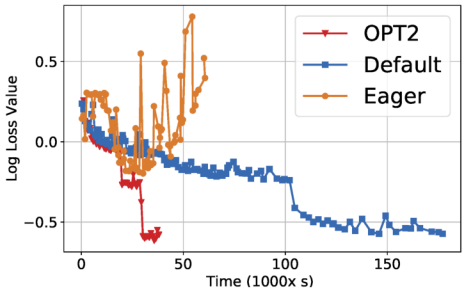

Figure 3. *We built a system called Opt^2, which safely auto-scales SGD tasks to avoid instability. It converges faster than the default (constant parallelism) but safer than overly eager auto-scaling.*

4.2 Auto-Scaling Stochastic Gradient Descent

Large-scale optimization algorithms, including Stochastic Gradient Descent (SGD) and its variants, have become one of the most important workloads for distributed data-intensive systems. We started with a seemingly simple goal of creating an auto-scaling framework in which, if more system resources are available, they can be adaptively allocated to the training task. Unfortunately, low-level concerns like the level of parallelism, the frequency of synchronization, and the number of examples that can fit into memory significantly affect model accuracy and training stability [27]. The coupling between algorithmic convergence and the physical layer limits opportunities for dynamic resource allocation and management.

To enable auto-scaling, we propose a coarse equivalence class expresses hyper-parameter settings and the degree of

Framework	Isolation	Description	Advantage	Disadvantage
Shared Model	Neurons	Combining computation graphs for co-training & inference	Easy to share computation when possible	No accuracy or resource isolation guarantee.
Shared AI System	Model	The neural network library controls execution and placement of different via context switching	Ease of use, and explicit control of resource and scheduling.	No accuracy or resource isolation guarantee, no computation sharing.
Shared Hardware	VM / GPU	Virtualization provided via hypervisor or GPU	Easy to use and strong isolation	Requires new hardware API for specialized hardware and worse overall resource usage

Table 2. A taxonomy of deep learning multitenancy approaches

parallelism in the same units. This model abstracts and summarizes the algorithmic properties related to convergence. Using this model, we devise a meta-optimizer for SGD. Our optimizer (called Opt^2) searches for the best choice of parameters that trains the model in the fastest way, but ensures that it does not scale the model too aggressively. We show an example of a predictive logistic regression model trained on the Criteo dataset [2]. Figure 3 illustrates the results. The default model does not change its level of parallelism and converges slowly. An over-eager method rapidly scales out the training and introduces instability. Opt^2 is able to safely and reliably scale-out the training process.

4.3 Multitenancy for Model Training

Our resource management layer should be able to flexibly choose a multitenancy framework depending on a desired SLO. A multitenancy framework specifies how a task shares the underlying resources and defines the degree of resource isolation between tenants. The AI setting has a number of key differences compared to a classical multitenancy or shared-resource setting. First, AI workloads often consist of repetitions of very similar tasks, e.g., during hyper-parameter optimization a user may train many very similar neural network models. For many use cases, such as for static neural networks (the predominant architecture in computer vision), the system has access to a full white-box description of the model. Therefore, there are significantly more opportunities to share computation between concurrent AI tasks. However, the drawback is that increased sharing reduces the degree of isolation between these tasks. Relating back to experiment on auto-scaling, the problem is subtle because changes in system-level parameters (e.g., latency of weight updates) can actually affect the accuracy of the task.

Therefore, the resource management layer will have to analyze the AI task to determine how best to isolate *both accuracy and completion time* from the other tenants. At one end of the spectrum, we have full virtualization, i.e., as if the models residing in their own containers. This offers the strongest level of isolation, but may be wasteful in terms of resource usage (e.g., duplicate execution sessions, context switching overheads). At the other end of the spectrum is model-level consolidation, where we concurrently run training and inference routines in the same execution environment. While this may not isolate performance, it may allow for improved opportunities to share common computation. We envision that different use-cases will require different levels of isolation. In Table 2, we summarize each these frameworks and at what granularity they isolate tenants.

5 A Vision for the Future

In this paper, we have presented a number of related projects that are integral steps towards a long-term vision of automated resource management for AI. Such a layer will resemble the multitenancy and resource management frameworks widely employed in programmed systems today but with added knobs to control accuracy.

Long-term, we believe that understanding resource management in AI requires accounting for model accuracy in SLOs. Classical systems have a concept of performance isolation; where the running time and resource utilization of one tenant is independent of all others on the same infrastructure. We envision that this concept has to be amended to include accuracy considerations. In particular, we envision a new concept of *accuracy isolation*, where the system's scheduling and resource allocation decisions will not affect the final accuracy of any model more than a certain amount. Designing an API for describing such accuracy objectives over tasks and groups of tasks will be one of the overarching challenges.

We choose to focus on this problem because the computational demands of modern AI techniques will soon overwhelm our data-intensive systems. We envision a forthcoming era of AI-Systems research where reducing resource consumption, reasoning about transient resource availability, trading off resource consumption for accuracy, and managing contention on specialized hardware will become the community's main research focus. All of these components will have to be integrated into a new resource management layer that solves the familiar systems problems of multitenancy and isolation, but also addresses the key differences in AI tasks and workloads.

[2] https://labs.criteo.com/category/dataset/

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[3] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy. Tvm: end-to-end optimization stack for deep learning. *arXiv preprint arXiv:1802.04799*, pages 1–15, 2018.

[4] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[5] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[6] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A low-latency online prediction serving system. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 613–627, 2017.

[7] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines. *GitHub, GitHub repository*, 2017.

[8] A. Dziedzic*, J. Paparrizos*, S. Krishnan, A. Elmore, and M. Franklin. Band-limited training and inference for convolutional neural networks. *ICML*, 2019.

[9] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi. Towards an elastic and autonomic multitenant database. In *Proc. of NetDB Workshop*, 2011.

[10] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in neural information processing systems*, pages 2214–2224, 2017.

[11] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[12] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[14] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2592–2600, 2016.

[15] N. Jouppi, C. Young, N. Patil, and D. Patterson. Motivation for and evaluation of the first tensor processing unit. *IEEE Micro*, 38(3):10–19, 2018.

[16] D. Kirk et al. Nvidia cuda software and gpu parallel computing architecture. In *ISMM*, volume 7, pages 103–104, 2007.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[18] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 239–250. ACM, 2015.

[19] S. Loesing, M. Hentschel, T. Kraska, and D. Kossmann. Stormy: an elastic and highly available streaming service in the cloud. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 55–60. ACM, 2012.

[20] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.

[21] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

[22] A. Paszke, S. Gross, S. Chintala, and G. Chanan. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration*, 6, 2017.

[23] B. Peng, M. Hosseini, Z. Hong, R. Farivar, and R. Campbell. R-storm: Resource-aware scheduling in storm. In *Proceedings of the 16th Annual Middleware Conference*, pages 149–161. ACM, 2015.

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners.

[25] A. Ratner, D. Alistarh, G. Alonso, P. Bailis, S. Bird, N. Carlini, B. Catanzaro, E. Chung, B. Dally, J. Dean, et al. Sysml: The new frontier of machine learning systems. *arXiv preprint arXiv:1904.03257*, 2019.

[26] D. M. Ritchie and K. Thompson. The unix time-sharing system. *Bell System Technical Journal*, 57(6):1905–1929, 1978.

[27] C. J. Shallue, J. Lee, J. M. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl. Measuring the effects of data parallelism on neural network training. *CoRR*, abs/1811.03600, 2018.

[28] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[29] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 275–287. ACM, 2007.

[30] G. Somani and S. Chaudhary. Application performance isolation in virtualization. In *2009 IEEE International Conference on Cloud Computing*, pages 41–48. IEEE, 2009.

[31] A. S. Tanenbaum and H. Bos. *Modern operating systems*. Pearson, 2015.

[32] N. Tatbul, U. Çetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB*, 2003.

[33] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[34] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon PhiâĎć*, pages 167–188. Springer, 2014.

[35] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.