A System for Real-Time Interactive Analysis of Deep Learning Training

Shital Shah
Microsoft Research
Redmond, Washington
shitals@microsoft.com

Roland Fernandez
Microsoft Research
Redmond, Washington
rfernand@microsoft.com

Steven Drucker
Microsoft Research
Redmond, Washington
sdrucker@microsoft.com

ABSTRACT

Performing diagnosis or exploratory analysis during the training of deep learning models is challenging but often necessary for making a sequence of decisions guided by the incremental observations. Currently available systems for this purpose are limited to monitoring only the logged data that must be specified before the training process starts. Each time a new information is desired, a cycle of stop-change-restart is required in the training process. These limitations make interactive exploration and diagnosis tasks difficult, imposing long tedious iterations during the model development. We present a new system that enables users to perform interactive queries on live processes generating real-time information that can be rendered in multiple formats on multiple surfaces in the form of several desired visualizations simultaneously. To achieve this, we model various exploratory inspection and diagnostic tasks for deep learning training processes as specifications for streams using a map-reduce paradigm with which many data scientists are already familiar. Our design achieves generality and extensibility by defining composable primitives which is a fundamentally different approach than is used by currently available systems.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; **Visualization systems and tools**; • **Computer systems organization** → **Architectures**; **Real-time systems**.

KEYWORDS

debugging; diagnostics; exploratory inspection; monitoring; visualization; deep learning; map-reduce; streams

ACM Reference Format:
Shital Shah, Roland Fernandez, and Steven Drucker. 2019. A System for Real-Time Interactive Analysis of Deep Learning Training. In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '19), June 18–21, 2019, Valencia, Spain.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3319499.3328231

1 INTRODUCTION

The rise of deep learning is complimented by ever increasing model complexity, size of the datasets, and corresponding longer training times to develop the model. For example, finishing 90-epoch ImageNet-1k training with ResNet-50 takes 14 days to complete on an NVIDIA M40 GPU[26]. Researchers and practitioners often find themselves losing productivity due to the inability to quickly obtain desired information dynamically from the training process without having to incur stop-change-restart cycles. While a few solutions have been developed for real-time monitoring of deep learning training, there has been a distinct lack of systems that offer dynamic expressiveness through conversational style interactivity supporting the exploratory paradigm.

In this paper we offer a new system that enables the dynamic specification of queries and eliminates the requirement to halt the learning process each time a new output is desired. We also enable the displays of multiple, simultaneous visualizations that can be generated on-demand by routing to them the desired information chosen by the user. The pillars of our architecture are general enough to apply our system design to other domains with similar long running processes.

Our main contributions are (1) system design based on dynamic stream generation using map-reduce as the Domain Specific Language (DSL) to perform interactive analysis of long running processes such as machine learning training (2) separation of concerns that allows to build dynamic stream processing pipeline with visualizations agnostic of rendering surfaces, and (3) abstraction to allow the comparison of previously generated heterogeneous data along with the live data in a desired set of visualizations, all specified at the runtime.

2 RELATED WORK

TensorBoard[25] is currently among the most popular of monitoring tools, offering a variety of capabilities including data exploration using dimensionality reduction, and model data flow graphs. However, its monitoring capabilities are limited to viewing only the data that was explicitly specified to be logged before the training starts. While the tool provides some interactivity in visualization widgets, no interactivity is provided in terms of dynamic queries. Furthermore, few pre-defined visualizations are offered in the dashboard in a pre-configured tabbed interface and thus somewhat limited in other layout preferences.

The logging-based model is also used by other frameworks, including Visdom[6] and VisualDL[1]. Several authors[6, 14, 23] have identified the research opportunities in diagnostic aspects of deep learning training and interactively analyzing it due to time consuming trial-and-error procedures.

Map-reduce is an extensively studied paradigm originated from the functional programming [21] and successfully utilized for constructing data flows and performing large scale data processing in the field of distributed computing [5, 7, 12]. Many variants of map-reduce has been created[2] for a variety of scenarios, and it has also gained wide adoption for the various data analysis tasks[9, 18].

Visualizations based on streams have been studied deeply for the real-time data scenarios[8, 24] with various systems aiming to enhance interactivity, adaptability, performance and dynamic configurations[10, 11, 15, 20]. Query driven visualizations have been popular for databases utilizing SQL and big data using custom DSLs[3, 19, 22]. This paradigm also becomes a cornerstone in our system design.

3 SCENARIOS

We describe a few real-world scenarios in this section to develop an intuition of the requirements and understanding of problems often faced by the practitioners.

Diagnosing Deep Learning Training

John is the deep learning practitioner with the task of developing a model for gender identification from a large labeled dataset of human faces. As each experiment takes several minutes even on a reduced subset of the data, John wishes to view training loss and accuracy trends in real time. In many experiments, training loss does not seem to be reducing and to understand the cause John needs to view the gradient flow chart. However, this requires John to terminate the training process, add additional logging for this information, and then restart the training. As John observes the gradient flow chart, he starts suspecting that his network may be suffering from the vanishing gradient problem. To be sure, John now wishes to view growth of weights and the distribution of initial values. This new information again causes a stop-change-restart cycle adding significant cost to obtain each new piece of information in the diagnostic process that is inherently iterative.

Analyzing The Model Interpretation Results

Susan is using a GAMs framework[13] to analyze the impact of each feature in her model. As a data scientist, she depends on Jupyter Notebook to perform her analysis. As the computation takes several minutes before generating the desired charts, Susan wants to display progressive visualizations displaying partial results as they evolve. Instead of designing and implementing a custom system for her one-off experiment, it would be ideal if she could simply generate a stream of data using map-reduce paradigm that she is already familiar with. This stream can then be easily painted to the desired rendering surface such as Jupyter Notebook to display progressive real-time visualization as her models evolve in time.

Diagnosing and Managing Deep Learning Jobs

Rachel spins up several dozens of deep learning jobs as part of her experiments in GPU cloud infrastructure. These long running jobs may take many days to complete and therefore are expensive to run in cloud. However, it turns out that many of the poorer performing jobs could be identified much earlier and be terminated, thus freeing up the expensive resources. However, designing and building such infrastructure is time consuming and requires additional engineering skills. It would be ideal for Rachel if she could simply output streams of performance data from her jobs and then create a small monitoring application that consumes these streams.

4 SYSTEM DESIGN

Key Actors

Our system design contains the following key actors as shown in Figure 1:

(1) A long running process P such as a deep learning training process.
(2) Zero or more clients that may be located on the same or different machines as P.
(3) An agent A that is embedded in P listening to requests from the clients

The Long Running Process

We abstract three specific characteristics of a long running process P:

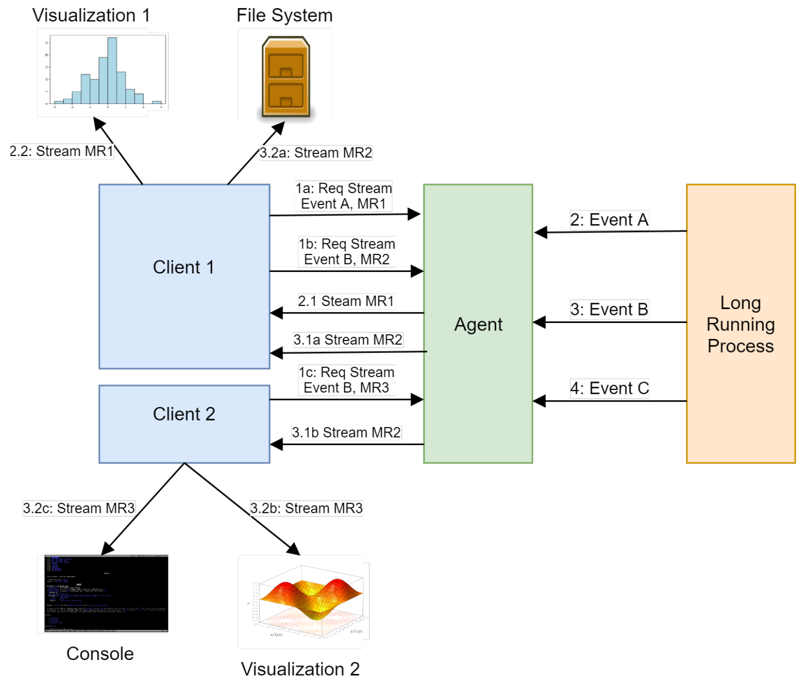

Figure 1: Collaboration diagram for our system depicting interactions between various actors. Standard notations are used with numbered interactions indicating their sequence with alphabet suffix denoting the potential concurrency. Our system includes the long running process generating various events, clients making requests for stream using map-reduce queries (denoted by MRx) for the desired events and the agent responding back with resultant streams that can be directed to desired visualizations or other processes.

(1) *P* may generate many types of events during its lifetime. Each type of event may occur multiple times, but the sequence of events is always serialized, i.e., there is never more than one event of the same type occurring at the same time in the same process.
(2) As events of each type are strictly ordered so that we can optionally assign a group to any arbitrary contiguous set of events. This ability will enable windowing for the reduce operator discussed later.
(3) For each event, optionally a set of values may be available for the observation. For example, on a batch completion event the metrics for that batch may be available for the observation. The process informs the agent when an event occurs and provides access to these observables.

The Client

At any point in time multiple clients may exist simultaneously issuing the queries and consuming corresponding resultant streams. Each query can be viewed as a stream specification with the following attributes:

(1) The event type for which a stream should be generated. An event type may have multiple associated streams but each stream has only one associated event type.
(2) An expression in the form of map-reduce operations. This expression is applied to the observables at the time of event and the output becomes the value in the resultant stream.

The client may utilize the resultant stream by directing it to multiple processes such as visualizations chosen at the runtime. Thus the same stream may generate a visualization as well as become input to another process in the data flow pipeline.

The Agent

The agent runs in-process in the host long running process *P* and characterized by the following responsibilities:

(1) Listening to incoming requests for the creation of a stream. This is done asynchronously without blocking the host process *P*.
(2) When *P* informs the agent that an event has occurred, the agent determines if any active streams exist for that event. If so, the agent executes the map-reduce computation attached to each stream for that event and sends the result of this computation back to the client.

An important aspect of the agent design is that if there are no streams requested for an event then there is almost no performance penalty. Also, there is no performance penalty for having access to the large numbers of observables. This means that user may specify all of the variables of interest as observables beforehand and later use queries to use subset of them depending on the task.

Example: Implementation for Deep Learning Training

As an example of how above actors and abstractions may be utilized, consider the deep learning training scenario. This process performs computation in series of epochs, completion of each is an *epoch event*. During each epoch, we execute several batches of data, completion of each becoming a *batch event*. At each batch event we may observe the metric object that contains several statistics for the batch. Contiguous set of batch events within each epoch can be treated as one group. At the end of an epoch, we may want to compute some aggregated statistics which can easily be done by specifying the map-reduce expression that extracts the desired value from the metric object and performing the aggregation operation on it.

Multiple Processes and Streams

The above abstractions can easily be utilized to efficiently inspect many simultaneously running processes and make decisions such as early termination or modify desired parameters at the runtime. A user can also compare and visualize arbitrarily chosen subsets of jobs.

Modifying the State of a Long Running Process

Our design trivially enables a useful capability of changing the observables of the long running process. In the context of deep learning training, this can be used for interactive hyper parameter tuning guided by observations [4]. We simply allow users to send commands from interfaces such as Jupyter Notebook to the agent running in the host process. The agent then executes these commands on observables presented to it by the host process at the specified events.

Stream Persistence

One of the significant disadvantages of many current systems is the requirement that data of interest must be logged to the disk storage, which can become an expensive bottleneck. Our design with stream abstraction trivially enables pay-what-you-use model so that users can selectively specify at runtime to persist only those streams that they may be interested in viewing or comparison in the future.

5 STREAM VISUALIZATION

Once a stream is produced, it can be visualized, stored or processed further in users data flow graph.

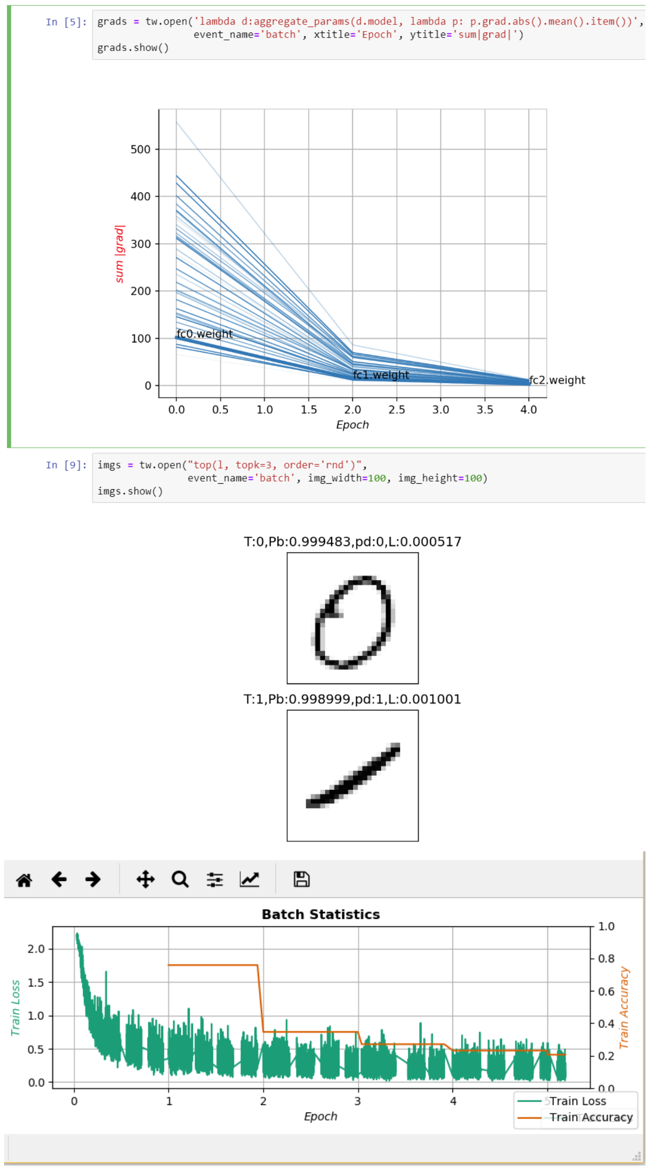
```
In [5]: grads = tw.open('lambda d:aggregate_params(d.model, lambda p: p.grad.abs().mean().item())',
                        event_name='batch', xtitle='Epoch', ytitle='sum|grad|')
        grads.show()
```
```
In [9]: imgs = tw.open("top(l, topk=3, order='rnd')",
                       event_name='batch', img_width=100, img_height=100)
        imgs.show()
```

Figure 2: Screenshot of three simultaneous real-time visualizations on two different surfaces generated dynamically by an user for the MNIST training process. On the top is an interactive session in Jupyter Notebook where the user specifies map-reduce queries in a cell for each desired visualization. The first output cell at the top shows evolution of average absolute gradients for each layer with lighter lines indicating the older plots. The second output cell shows random sample of predictions so far. At the bottom is the plot of two batch statistics rendered in a separate native application.

Adaptive Visualizers

As we allow users to generate arbitrary streams dynamically, it becomes important that visualization widgets are specifically designed for automatic configuration by reflecting on data available in the stream. We adopt the adaptive visualization paradigm[16, 17] for this purpose. For example, a visualizer may decide to paint a stream that has a tuple of two numeric values as a 2D line chart, tuple of 3 numeric values as a 3D line chart and tuple of 2 numeric and 1 string value as annotated 2D line chart. The user may override to select a precise rendering for a given stream.

A visualizer may allow adding or removing streams dynamically. The streams may not have the same data type allowing for the heterogeneous visualizations such as display of a histogram and a line chart overlays. If a visualizer receives incompatible streams than it may display an error. In the context of deep learning, this enables capabilities such as viewing multiple related metrics in the same visualization or comparing data generated by multiple experiments in the same visualization.

Frame Based Animated Visualizations

Many useful visualizations may consume values in a stream one after another as they arrive, e.g. , line charts. Another interesting scenario is to consider each value in the stream providing the complete data for each *frame* in the visualization. This enables users to create dynamic specification for the animated visualizations using familiar map-reduce paradigm. In the context of deep learning training, this allows users to create on-demand custom visualizations such as per-layer gradient statistics change over time, display of sample predictions sorted by loss value and so on.

6 STREAM GENERATION USING MAP-REDUCE

Background

There are many variants of the map-reduce model[2] and differences in various implementations. We will focus on the variant that is popular among data scientists and readily available in widely used programming languages such as Python.

The map-reduce paradigm consists of two higher order operators: *map* and *reduce*. The map operator accepts a function M and a list of values V. The M is applied to each value in V to transform it to some other value or choose not to output any value, i.e. , the filter operation.

The reduce operator accepts a function R and a list of values V. The R processes each value in V to produce an aggregated output value. For instance the operation of sum over a sequence can be done as reduce operation with R that initializes aggregated value to 0 and then consumes each value in the sequence to produce new aggregated value.

Extending Map-Reduce

While the map operator consumes a stream and outputs a stream, the reduce operator consumes stream and outputs an aggregated value instead of a stream. The reduce operator's output is not generated until the entire stream ends. In several of our scenarios, we rather desire that the reduce operator works on a group of contiguous values in the stream, aggregating values in that group and outputting a stream. For instance, we may want to compute the average duration for batches within each epoch and generate a stream with these averages as epochs progresses.

To achieve this, we introduce an extension to allow us leveraging the existing infrastructure and avoid need for entirely new domain specific language. In our extension, we simply require that each value in the stream is accompanied by an *optional* binary value B which when $true$ triggers the output from the reduce operator.

There are two advantages offered by this design:

(1) B can be set at any time by the host process P enabling many of our core scenarios trivially.
(2) B can also be set by a client at any time. This enables the scenarios where the user dynamically defines the aggregation window. For example, the user may wish to view a metric averaged over every 5 minutes.

7 IMPLEMENTATION

We implement our design using Python and other frameworks described in this section. We will be releasing our implementation as an open source cross-platform offering.

For networking stack we utilize the ZeroMQ library to implement publisher-subscriber model between the agent and the client. Out of the box, we offer implementations for MatplotLib as well as Plotly frameworks for various visualizations including line charts, histograms and image matrix. MatplotLib allows a variety of UX backends, many of which can run as native application or in Notebook interface for exploratory tasks. The Jupyter Lab allows transforming Notebook in to the user defined dashboards.

One of the key requirements in our system model is the implementation of the map-reduce extension described in Section 6. We achieve this by implementing a component we call *postable iterator*. The postable iterator allows to post input sequence of tuple $\{value, B\}$, where B is group completion flag described in the Section 6. The postable iterator then evaluates the map-reduce expression and returns the output value of map or reduce operator or signals the caller that no output was produced for the posted value.

One of the key difficulties in implementation using languages such as Python and frameworks such as ZeroMQ, MatplotLib, and Jupyter Notebook is managing the limitations imposed for multi-threading. We adopt the cooperative concurrency model with callbacks combining with the producer-consumer pattern to work around many of these limitations.

8 CONCLUSION

We described the design of a system that brings data streaming and map-reduce style queries to the domain of machine learning training for enabling the new scenarios of diagnosis and exploratory inspection. We identified several advantages of our system over currently popular systems, including the ability to perform interactive queries, dynamic construction of data flow pipelines, and decoupled adaptive visualizations as nodes in such pipelines. We plan to release our system as an open source cross-platform offering to help researchers and engineers perform diagnosis and exploratory tasks more efficiently for the deep learning training processes.

ACKNOWLEDGMENTS

We would like to thank Susan Dumais for her guidance and advice on this project.

REFERENCES

[1] [n. d.]. Visualize your deep learning training and data flawlessly. https://github.com/PaddlePaddle/VisualDL

[2] Foto N. Afrati, Vinayak Borkar, Michael Carey, Neoklis Polyzotis, and Jeffrey D. Ullman. 2011. Map-reduce Extensions and Recursive Queries. In *Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT '11)*. ACM, New York, NY, USA, 1–8. https://doi.org/10.1145/1951365.1951367

[3] Shivnath Babu and Jennifer Widom. 2001. Continuous Queries over Data Streams. *SIGMOD Rec.* 30, 3 (Sept. 2001), 109–120. https://doi.org/10.1145/603867.603884

[4] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2546–2554. http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

[5] Bryan Catanzaro, Narayanan Sundaram, and Kurt Keutzer. 2008. A map reduce framework for programming graphics processors.

[6] Jaegul Choo and Shixia Liu. 2018. Visual Analytics for Explainable Deep Learning. *IEEE Computer Graphics and Applications* 38, 4 (jul 2018), 84–92. https://doi.org/10.1109/mcg.2018.042731661

[7] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce. *Commun. ACM* 51, 1 (jan 2008), 107. https://doi.org/10.1145/1327452.1327492

[8] Chandrajit Bajaj (Ed.), Chandrajit Bajaj, and C Fl John Wiley. 1998. Data Visualization Techniques.

[9] Jaliya Ekanayake, Shrideep Pallickara, and Geoffrey Fox. 2008. MapReduce for Data Intensive Scientific Analyses. In *2008 IEEE Fourth International Conference on eScience*. IEEE. https://doi.org/10.1109/escience.2008.59

[10] B. Ellis. 2014. *Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data.* Wiley. https://books.google.com/books?id=DFnOAwAAQBAJ

[11] Stephen Few. 2006. *Information Dashboard Design: The Effective Visual Communication of Data.* O'Reilly Media, Inc.

[12] Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. 2009. Building a High-level Dataflow System on Top of Map-Reduce: The Pig Experience. *Proc. VLDB Endow.* 2, 2 (Aug. 2009), 1414–1425. https://doi.org/10.14778/1687553.1687568

[13] Trevor Hastie and Robert Tibshirani. 1986. Generalized Additive Models. *Statist. Sci.* 1, 3 (aug 1986), 297–310. https://doi.org/10.1214/ss/1177013604

[14] Shixia Liu, Xiting Wang, Mengchen Liu, and Jun Zhu. 2017. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics* 1, 1 (mar 2017), 48–56. https://doi.org/10.1016/j.visinf.2017.01.006

[15] Ivan Logre and Anne-Marie Déry-Pinna. 2018. MDE in Support of Visualization Systems Design: A Multi-Staged Approach Tailored for Multiple Roles. *Proc. ACM Hum.-Comput. Interact.* 2, EICS, Article 14 (June 2018), 17 pages. https://doi.org/10.1145/3229096

[16] Constantinos Mourlas. 2009. *Intelligent user interfaces : adaptation and personalization systems and technologies.* Information Science Reference, Hershey, PA.

[17] Kawa Nazemi. 2016. *Adaptive Semantics Visualization.* Springer International Publishing. https://doi.org/10.1007/978-3-319-30816-6

[18] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. 2009. A Comparison of Approaches to Large-scale Data Analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD '09)*. ACM, New York, NY, USA, 165–178. https://doi.org/10.1145/1559845.1559865

[19] B. Plale and K. Schwan. 2003. Dynamic querying of streaming data with the dQUOB system. *IEEE Transactions on Parallel and Distributed Systems* 14, 4 (apr 2003), 422–432. https://doi.org/10.1109/tpds.2003.1195413

[20] Jonathan C. Roberts. 2007. State of the Art: Coordinated & Multiple Views in Exploratory Visualization. In *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*. IEEE. https://doi.org/10.1109/cmv.2007.20

[21] Guy L. Steele. 1995. Parallelism in Lisp. *ACM SIGPLAN Lisp Pointers* VIII, 2 (may 1995), 1–14. https://doi.org/10.1145/224133.224134

[22] K. Stockinger, J. Shalf, Kesheng Wu, and E.W. Bethel. [n. d.]. Query-Driven Visualization of Large Data Sets. In *VIS 05. IEEE Visualization, 2005.* IEEE. https://doi.org/10.1109/visual.2005.1532792

[23] Nako Sung, Minkyu Kim, Hyunwoo Jo, Youngil Yang, Jingwoong Kim, Leonard Lausen, Youngkwan Kim, Gayoung Lee, Dong-Hyun Kwak, Jung-Woo Ha, and Sunghun Kim. 2017. NSML: A Machine Learning Platform That Enables You to Focus on Your Models. *CoRR* abs/1712.05902 (2017). arXiv:1712.05902 http://arxiv.org/abs/1712.05902

[24] Jonas Traub, Nikolaas Steenbergen, Philipp Grulich, Tilmann Rabl, and Volker Markl. 2017. I^2: Interactive Real-Time Visualization for Streaming Data. https://doi.org/10.5441/002/edbt.2017.61

[25] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B. Viegas, and Martin Wattenberg. 2018. Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (jan 2018), 1–12. https://doi.org/10.1109/tvcg.2017.2744878

[26] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. 2018. ImageNet Training in Minutes. In *Proceedings of the 47th International Conference on Parallel Processing (ICPP 2018)*. ACM, New York, NY, USA, Article 1, 10 pages. https://doi.org/10.1145/3225058.3225069