Persistent Machine Learning for Government Applications

Joshua Haley, Chad Dettmering, Ross Hoehn PhD, Ryan Barrett
Ali Mizan, Alyssa Tanaka PhD & Brian Stensrud PhD
Soar Technology Inc.

12124 High Tech Ave 350
Orlando, FL 32826
first.last@soartech.com

ABSTRACT

Machine Learning (ML) offers benefits such as adaptive systems that are more performant initially and display continuously improved performance over time. However, we often encounter difficulties operationalizing commercial ML breakthroughs into the government and defense sectors, chiefly due to a lack of available training data. While data collection is at the forefront of most commercial entities who monetize data – such as Google and Facebook – it is often neglected outside the commercial scope, due to a lack of incentives. However, it is often the data, not the model architecture, that defines many commercialized breakthroughs. For instance, Google freely shares many of its models and tools through publications and open source repositories; yet they retain the training data, preventing result replication and product cloning. Given dataset size requirements for Deep Learning, dataset collection is critical for effective ML solutions. Even when datasets are available, too often ML algorithms are trained once and then never updated, effectively not using new data as it is collected. Often – in the absence of an on-line learning component – a system is initially as performant as it will be after hundreds of hours of use.

This paper presents an architecture developed to support and extend the lifespan of ML within the government and defense spaces. This architecture provides three key components for an effective Machine Learning architecture. First and foremost, it supports continuous acquisition and curation of new training data. Second, it provides computational resources to support Machine Learning exploration of data. Finally, it provides automatic and continuous ML, updating models in response to new data. We describe several sample domains – such as speech recognition – that impact military training needs. Finally, we discuss how this architecture addresses ML needs that are presently preventing operationalization of modern methods within the government space, as well as outstanding challenges.

Index Terms
Machine Learning, Data Acquisition, Model Maintenance

ABOUT THE AUTHORS

Joshua Haley is a Software Engineer in Soar Technology's Intelligent Training Division. He holds both Bachelor and Master degrees of Computer Engineering from the University of Central Florida (UCF) focusing in Machine Learning and Intelligent Systems. At SoarTech, Josh uses his Machine Learning experience to develop systems and scenarios for Intelligent Training Systems as well as Cognitive agents who play a role in such training systems. His academic and research interests include Artificial Intelligence, Machine Learning, Non-Declarative Knowledge Representation, and Computational Intelligence. Josh is currently a Doctoral student at UCF.

Chad Dettmering holds a Bachelor degree in Computer Science from the University of Central Florida. Since joining Soar Technology, Chad has been an expert in natural language processing systems specializing in both Machine Learning as well intent and grammar-based approaches. He has an active interest in developing more robust, data driven speech enabled systems.

Ross Hoehn, PhD is a research scientist in Soar Technology's Intelligent Training division. He earned is Ph.D. in Theoretical Chemistry from Purdue University in 2014, and continued research in chemical physics, quantum information and artificial intelligence until 2018. His research areas include: adaptive artificial intelligence, generative AI techniques, team learning and training, pedagogy, biological-based agent simulations, swarm mechanics, quantum information science, quantum computing and quantum mechanically-driven biophysical phenomenon. He was an active researcher and manager of the NSF Center for Chemical Innovation: Quantum Information for Quantum Chemistry, a multi-million-dollar multi-year research effort centered at Purdue University to utilize quantum computing for quantum mechanical calculations.

Ryan Barrett is a full stack Software Engineer with a diverse science background including chemistry, physics, spectroscopy, and computer science.

Ali Mizan is a Software Engineer in Soar Technology Intelligent Training division. He earned his Bachelors and Masters in Computer Engineering from the University of Central Florida in 2016

Dr. Alyssa Tanaka has worked with Soar Technology as a Research Scientist since April 2017. She has earned a Ph.D. and M.S. in Modeling and Simulation from the University of Central Florida, Graduate Certificates in Instructional Design and Training Simulations, and a B.S. in Psychology and Cognitive Sciences from the University of Central Florida. She also holds a diploma in robotic surgery from the Department of Surgery, University of Nancy, France. Before joining SoarTech, Dr. Tanaka led research initiatives directly related to improving the training and education of robotic surgeons through simulation as a Senior Research Scientist at Florida Hospital Nicholson Center, the premier location in the world both for training expert practitioners with the daVinci surgical robot and for advancing research in cutting-edge training to target this device. At SoarTech, she leads the initiatives aimed at leveraging AI and Intelligent Training solutions into the military medical community.

Brian Stensrud, PhD is a senior research scientist at Soar Technology and currently serves on the executive management team for its Intelligent Training division. On staff since 2003, Brian has provided scientific and technical leadership for over $25M of DoD research efforts in the areas of interactive human behavior models for simulation, serious games, adaptive training, intelligent user interfaces and robotic platforms. Brian received a Ph.D. in Computer Engineering from the University of Central Florida (2005), and holds bachelor's degrees in Electrical Engineering and Mathematics from the University of Florida (2001).

Persistent Machine Learning for Government Applications

Joshua Haley, Chad Dettmering, Ross Hoehn PhD, Ryan Barrett
Ali Mizan, Alyssa Tanaka PhD & Brian Stensrud PhD

Soar Technology Inc.
12124 High Tech Ave 350
Orlando, FL 32826
first.last@soartech.com

INTRODUCTION

Over the past several years Corporations such as Google, Facebook and Apple have ushered in a new wave of public consciousness concerning both Machine Learning (ML) and Deep Learning (DL) with impressive results in image recognition, predictive analytics, and solutions to strategic gameplay (Silver *et al.* 2017; Girshick *et al.* 2018). Given the promise of ML and its effects on modern technology, it is natural to expect that we should be able to better support warfighter training by adapting these breakthroughs to the simulation, yet there exist several key issues preventing these technologies from being leveraged as they currently exist. We shall detail two of these issues here:

1. A dearth of available training data. ML methods fundamentally all use a model to translate a large amount of data to a series of iteratively learned parameters, all to relate a specific input to a desired output. Commercial companies are collecting as much data as they are legally permitted to; they do this in case they can later monetize it. Outside the commercial space, data is only collected after a requirement has been identified. With many Deep Learning methods requiring several thousand data instances to perform effective training, the systems we build must be outfitted with the proactive and continued collection of data in mind.

2. A lack of cloud connectivity. Most commercial providers execute and train their models on a cloud-based server environment. This means that when you verbally request that your phone search for ‘bacon’, that audio selection is being sent to the Google cloud for remote recognition. This method has several advantages. First, Google has effectively already collected new data in the form of your query. Second, Google does not need to worry about deploying an updated capability to your phone, as it is already being executed on their servers. With most training systems deployed in the field and across the globe, expecting this level of connectivity is unreasonable.

Herein we present a conceptual architecture capable of supporting the needs of persistent ML for the DoD training and simulation spaces. In this context, we define ML as a process that uses data to train a model capable of being used advantageously by determining specific relationships between initial data and class/policy/forecast. We do not distinguish between supervised, unsupervised or semi-supervised approaches as our architecture is general enough to encode all of these use cases; we consider this to be a fundamental requirement of any persistent learning platform. It is our hope that we can both facilitate and enhance the governmental simulation and training capabilities by collectively adopting these practices to better support the utilization of data-intensive ML solutions.

Within the introduction of this work, we motivated the driving need for the design and development of persistent learning environments for use on military and Defense-related projects and motive this need by presenting a military-relevant example domain. In the architecture section, we present the requirements, overall design elements – both Frontend and Backend – and outline the training schema used. Within the initial results, we discuss specific implementations and initial results of the persistent training environment via implementation of an Image Classifier. In our conclusions, we provide closing comments and future directions.

Example Training Domain: Speech-Enabled Training

A commonly encountered training domain is one in which we train a military occupation with voice communications as a core interface to support the occupation. In such a role, the warfighter must communicate with other entities to both gather information and direct the actions of others. Training such a system with virtual role players requires a natural human interface that matches the modality of the operational task (Iwahashi 2004; Stensrud *et al*. 2008; Stensrud *et al*. 2006; Majewski and Kacalak 2016). One such domain is the performance of medical hand-offs. During these interactions, the responsibility of care for a patient is being transferred from one provider to another; this transfer of responsibility comes with a cognitive off-load of patient information. A lapse in information transfer can be detrimental to the well-being of the patient. In fact, 80 percent of medical errors originate during hand-offs (Derlet, Richards and Kravitz 2006; Alvarado *et al*. 2006; Joint Commission. 2012). These critical moments in a patient's care are even more complex and important within the context of combat casualty care, where casualty information is being transferred between providers in austere and stressful environments and the providers may be managing multiple casualties that have multiple severe injuries, over long periods of time.

In a training system for patient handoff – where role players have been automated – the training environment must be able to accurately recognize what the trainee has said via Speech to Text (STT) recognition. Grammar-based approaches work well for very narrow domains where the entirety of what will be said can be predefined or scripted; making these approaches rather brittle to out-of-grammar utterances. Additionally, commercial off-the-shelf grammar-based approaches will not adapt overtime. If they misrecognize an utterance when first deployed, the same misrecognition will occur throughout the lifetime of the deployed system. DeepSpeech, in contrast, is a ML approach that uses a volume of training data to parameterize an implicit recognition model (Hannun *et al.* 2014). Using an open corpus such as Mozilla Common Voice the system can be pre-trained to recognize general English utterances (Mozilla 2019). While a useful start for tasks such as dictation, it will likely display sub-optimal performance initially when used for a specific domain. By using Machine Transfer Learning (Pratt 1993; Baxter 1998; Pan and Yang 2009), this general English STT recognizer can then later be adapted to the specific domain. This works by using the open data corpus to train the model to recognize the higher-level computational features important in the general domain. Operationally, this process would work by first collecting domain utterances during use and storing them locally. When the system can sync with a central repository, these examples would be sent back to remotely re-train the backend recognition model, exploiting the greater computational resources available at the central repository. The updated model can then be sent back to the training system, analogous to how software updates are distributed. Thus for our casualty hand-off domain, we will start with a system that has good performance for general English and will become more attuned to the specific phraseology for the domain over time. An added benefit is with the right data collection agreements, that data collected over time can help pre-train other models for speech recognition in the medical speech domain; this new bootstrapped model will benefit from the collection of domain-related grammar and vocabulary, assuring that subsequent model generations will display better initial performance in their specific domains.

ARCHITECTURE

The architecture required to support ML within DoD training and simulation space has two functional components. 1) The Backend where data is stored and models are trained. 2) The Frontend Client, or training system, with which a user is actively interfaces. We will discuss the functional requirements of both components, as well as how those requirements were addressed within our initial implementation.

Backend

The Backend houses several components relating to two high-level functions of the architecture: 1) The acquisition and storage of data. 2) The automatic training and maintenance of ML models in response to new data. While the second function is useful for engineering convenience, the first function is critical to furthering the application of ML in environments without existing data sets.

Data Storage

The large sum of data that we hope to collect is only as useful as curation and organization allow. Given the mixed nature of data (audio, images, video, text, etc...), simply storing all the information within a database is not a viable solution. Additionally, the data must be organized in a generalized and search-able manner with consistent metadata tagging. The metadata tags encoded within the data store allowed for generalized search and exploration via a code interface. This interface allows for models to make specific requests such as "find all speech, with format PCM, and transcripts" in order to train.

Model Maintenance

When a new model is imported, additional model metadata is captured; this metadata indicates what type of data the model operates upon, as well as an appropriate differential in training data size to trigger a re-training event. This information is used by a Model Maintenance Service to periodically query the data store and determine which models to re-train. We identified several requirements to define a re-training period, including but not limited to:

1. Whenever a new pertinent data instance has been added to the data store.
2. Whenever the number of new data instances (not included in the previous training cycle) exceeds a predefined threshold.
3. Upon some specified time period, if new data has been added.

When it has been determined that a model requires re-training, the model is then passed to a Model Computation Service. A requirement of the Model Computation Service is that it is capable of managing several models and should support the use of multiple instances and classes of computing resources.

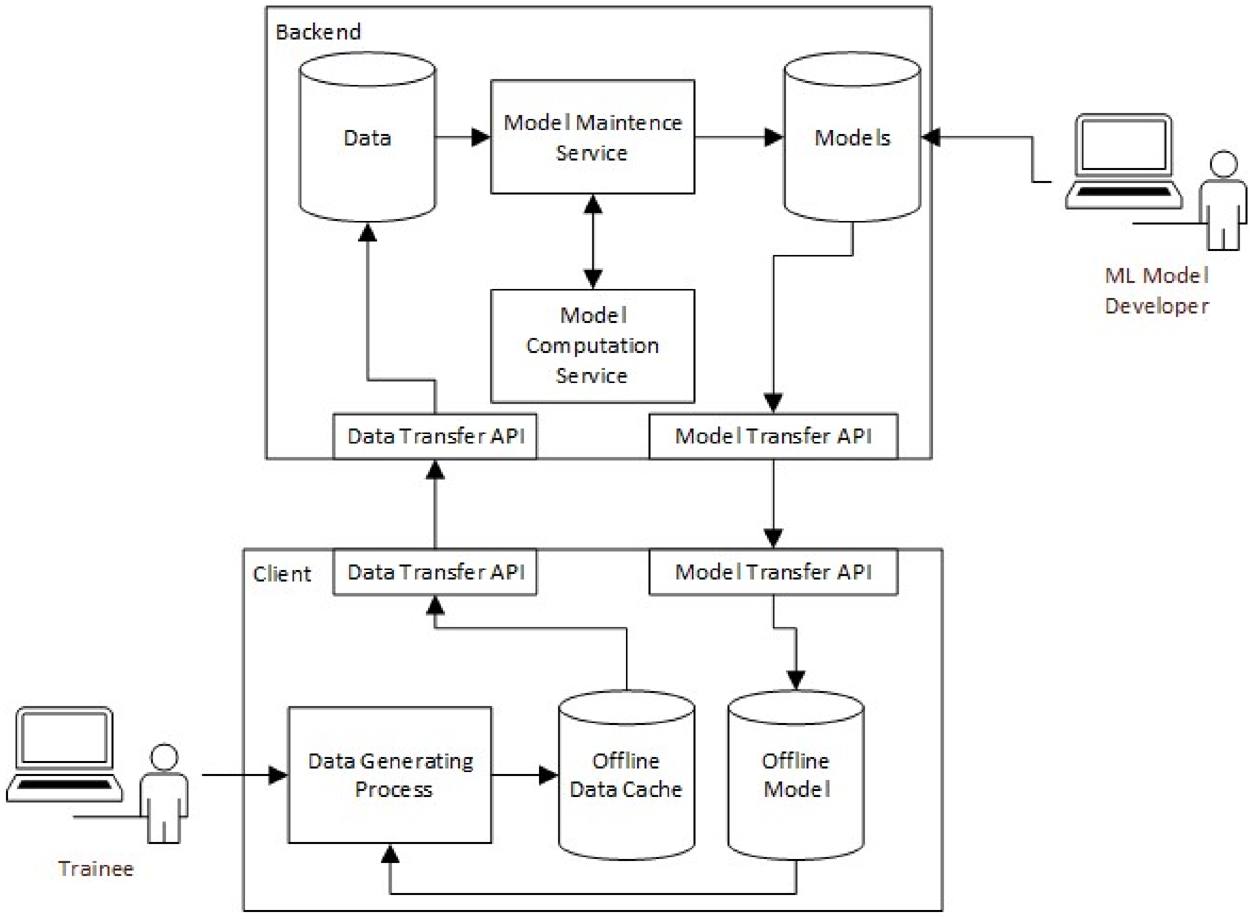

Figure 1 Architecture Components

Model API

In order to provide a general model maintenance capability, an API defining the steps of model execution must be defined and consistently enforced. This model API prescribes the functions required during implementation by an ML engineer for integration into the architecture. This API must be general enough to support various classes of ML (*e.g.* supervised, unsupervised, etc) and tasks (*e.g.* regression, classification, and clustering). Given the wealth of existing libraries, and that Python has become the *lingua franca* of machine learning, our interface design is based upon Python and heavily inspired by the SciKit Learn API (Pedregosa *et al.* 2011). Our general API and lifecycle for a ML model is described below.

1. Setup Model(intermediate_state):
   This function instantiates the specific ML approach, wither it's a Deep Learning Neural Network, Bayesian Classifier, etc.. It is agnostic to the underlying method or library to be as general as possible and provide maximal control to the ML engineer.
2. Format Data(datacursor):
   Using the data query tools provided by the architecture and the model metadata describing the data it operates upon, the model is provided a data cursor pointing to the pertinent data. The format_data function is called to transform this data into a specific input representation required by the model. For instance, if a model operates on images, but requires a 128x128 pixel grayscale input, this function is where images of other representations and sizes would be transformed.
3. Fit(datafile):
   The fit call takes as input a pre-formatted data file of inputs and outputs and then tunes the model to the data presented. This function calls the underlying model or library's training function and optionally would collect any training metrics as implemented by the ML engineer. While this data file is a generic type, in practical applications it will be an HDF5 (Folk *et al.* 2011).
4. Predict(x):
   This function takes a single input *x*, and runs it through the pre-trained model to produce output. This is the function called for run-time usage of the model. See Figure 3.
5. Test(data_file):
   This optional function allows for additional testing of the model. For example, one might have an evaluation dataset that will not be used in training, but is the judge of model performance.
6. Serialize():
   Depending upon the underlying libraries, there are many different model-dependent serialization schemes to store the saved model. Additionally, there are a set of artifacts (training metrics, test results, etc.) that the ML engineer need to save for analysis. This function performs those actions and returns the artifacts as a binary blob for saving.
7. Deserialize(blob):
   This function reproduces the model from a previously saved state. In practice, this serves as the constructor for the operational use of the model.
8. Compare(other_model):
   In order to support automatic model maintenance, it is critical to determine if your newly trained model is better than your previous best model. The definition of "better" is model and domain-dependent, so this function is left to the ML engineer for maximum flexibility.

Given the above API, the order by which the functions are called while executing along the model training life cycle is given in Figure 2.

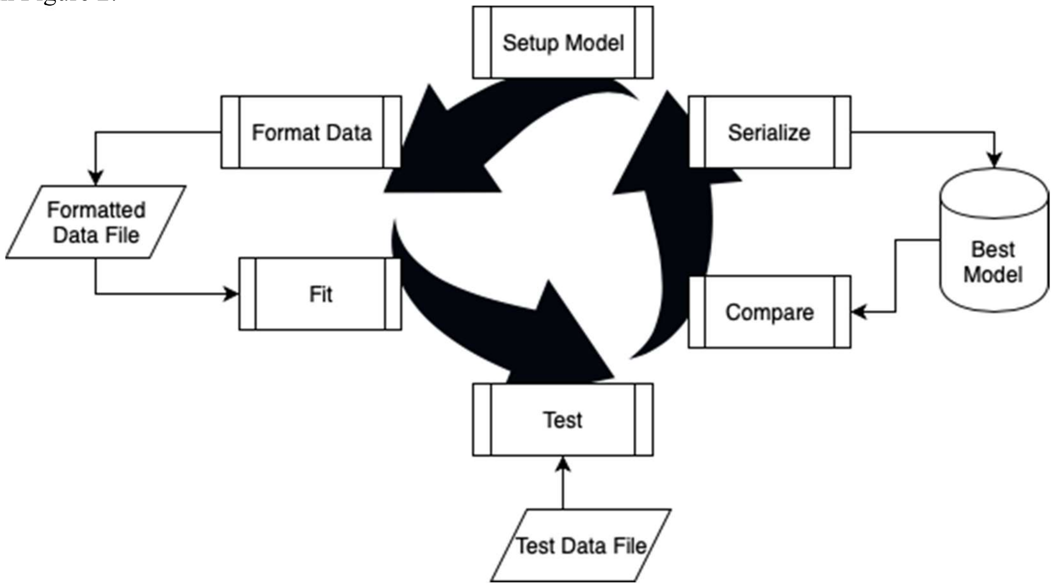

Figure 2 Model Lifecycle

In operation, we will simply deserialize the model and then use the predict function as needed in application as seen in Figure 3.

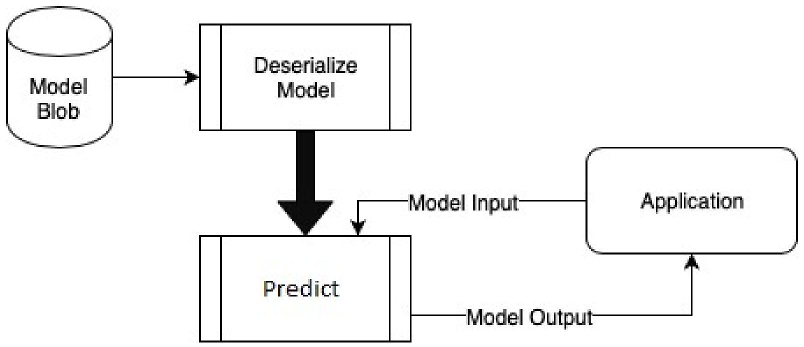

Figure 3 Model Utilization

Frontend/Client

In contrast to the Backend, the Frontend is fairly simple, yet should be designed specifically for a domain. The data generating component of the training system has been abbreviated to 'Data Generating Process' in Figure 1 as we do not need to know the full logic of the target application. It is important to note that the training system is the interface interacting with the user, collecting data and using the ML model. In our medical hand-off example, this is the speech recognition component that is receiving audio and actively translating from speech to text.

Offline Cache/Model

A general lack of connectivity mandates that both the model and the gathered data must be stored offline until connectivity can be reestablished. While the training of such a model is resource intensive it benefits both from specialized hardware and from the minute computational power called upon to execute the model; therefore, offline execution is rarely limited by resource.

Transfer APIs

The Data Transfer API is a plugin component that must have sufficient logic to determine if it can connect with the Backend, and then transfer data upon request. Similarly, the Model Transfer API must be able to detect a connection to the Backend as well as the presence of a new model to download and store for offline use. Just like a software update, this 'update' process can be automatic, scheduled, or manually actuated. This offline component is a critical difference from what is commonly practiced by commercial organizations such as Facebook or Google.

INITIAL RESULTS

As part of an internal research and development effort, an initial implementation of this architecture was developed. Here we discuss specific implementation decisions as well as an example classifier that was incrementally improved as more data was added to the system.

Implementation Choices

For the Data Store implementation, a hybrid solution of raw file storage combined with a MongoDB© for indexing was used in our reference implementation. By using a noSQL database such as MongoDB©, we were able to have an extensible representation of data that did not require systematic change as new data types are added. Initial work used a traditional SQL database, but as the data types evolved when new metadata requirements were identified, it necessitated a database migration which was costly and disruptive to the online capability of the system. All of the raw data itself was stored on a network attached storage device using a network file system allowing for the computational nodes of the Model Maintenance and Model Computation services to transparently access training data over a dedicated storage area network.

The Model Maintenance Service was implemented in Python and instantiated as a background daemon constantly monitoring the contents of the data store for changes. These changes were compared to model metadata enumerating the set of models, the data parameters that pertain to the model, and the desired update parameters. It was decided for convince that the persistence of models and associated metadata would be stored in the same MongoDB© instance as the data store. When a data change was matched to an interested model and the update parameters were met, a call to the Model Computation Service to re-train the model was made via a standard operating system call.

The Model Computation Service requires the ability to manage several models and computational resources. To fulfill these requirements, we used the Slurm Workload Manager©, a free and open source job scheduling package. Slurm – used by the majority of HPC clusters – brought to bear several features out of the box. It allowed for the management and scheduling of available resources, the queuing of model re-training if a backlog exists, and the logging of resource utilization. By giving each model its own Slurm account, we are able to generate metrics detailing which models – and thus which projects – are utilizing computation resources. Additionally, in an environment where projects contribute additional computers or have higher priority, model update prioritization can be assigned. Any other job scheduling software system, such as Portable Batch System© or Moab HPC Suite©, may be capable of fulfilling the requirements of the Model Computation Service.

To satisfy the requirements of the Data and Model transfer services, we chose to use a RESTful API implemented in Python to transfer data in between the Frontend and Backend with local file storage serving as our offline cache.

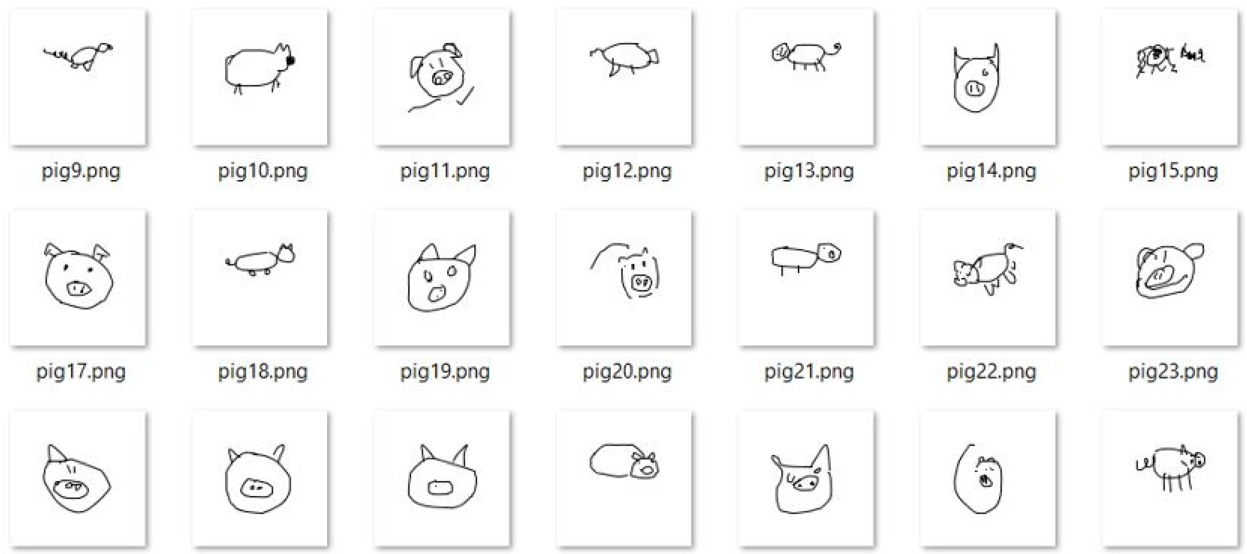

Figure 4 Quick, Draw! Pig doodles

Machine Learning Model Example

As a test domain, we decided to develop a simple image classifier using a Deep Convolution Network as a specific implementation of the ML Model API. As a domain we selected the Pig subset within the 'Quick, Draw!' image library, example illustrations are provided in Figure 4 (Ha and Eck 2017; Google 2018). The goal was to have a novel and unique domain in which we could observe a classifier becoming more effective as additional training data was added to the model's host persistent learning platform. A supervised classification application was developed, instantiated as an Android mobile application that allowed for easy human labeling of pig images and automated data upload to the data store. Over time as more pig images were manually labeled and appended to the training library, we were able to increase the accuracy of the 'pig body vs face' classifier as the model was contiguously re-trained in response to new data. Figure 5 shows the accuracy vs Epoch curve representing a full training run for three different quantities of available data. In all cases, the model is able to over-fit the training data, but a lack of training data in the first two cases is preventing the model from generalizing, as evidenced by poor performance on validation data (the data held out of training for evaluation).

While this image classifier may seem trivial, it contains many of the basic requirements for more DoD relevant domains such as identification of high value targets from aerial surveying.

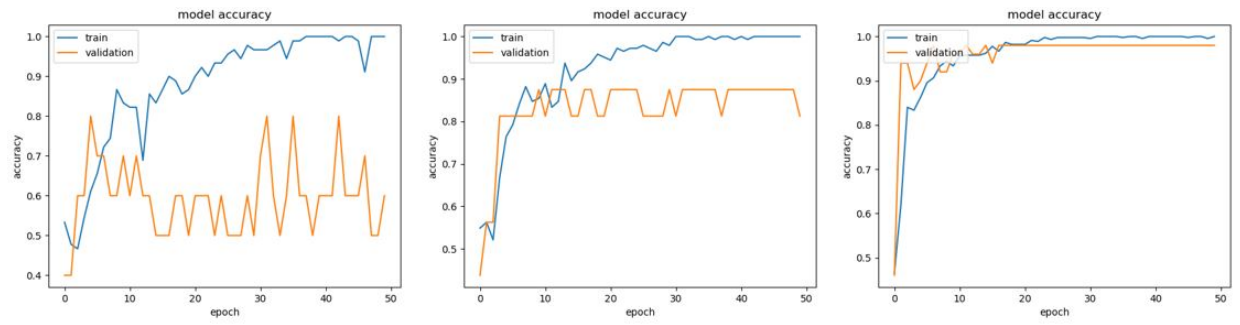

Figure 5 Model accuracy vs response data (number of data instances from left to right: 100, 250, and 500)

CONCLUSIONS

Herein we have described and presented both a technical approach to and conceptual architecture designed to alleviate the pervasive under-availability and underutilization of ML data to supported limited connectivity, field deployment applications common in the government space. While this architecture supports both data collection, as well as model maintenance and deployment, these individual functions can be separated and need not exist as a monolith; this has been demonstrated by the use of the Slurm Workload Manager as the computational resource within our reference implementation. The goal of this architecture is to address the unique concerns of operating within the government and defense spaces, without limiting the use of existing open source offerings, e.g. Tensorflow. While the notion of a Backend/client model is not novel in the context of commercial offerings, the ability to operate without cloud-based connectivity is rare for all but the most basic models. Additionally, most offerings do not currently support the use of non-cloud based servers; this cloud-based only implementation leads to concerns about data security and export control. Here we extend these capabilities to more advanced, and therefore operationally significant models, while allowing for non-cloud based servers for data storage and the update of model parameters.

Perhaps more important than the technological solution and conceptual architecture, is the need to widely adopt more forward facing data gathering philosophies. While the data gathered today may be of limited initial use, the ownership of large data sets gathered over time become a distinct competitive advantage. The conceptual architecture presented here will allow for present and active data storage, compiling, and curation; this capability allows users to collect and store datasets that are of interest, yet too small to train a model. This forward-thinking philosophy would allow government institutions to maximally leverage their extant data for future use in training ML routines.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the early contributions of Mr. Adam Wearne to the project. His love of pigs inspired the selection of the test domain.

REFERENCES

Alvarado, K., Lee, R., Christoffersen, E., Fram, N., Boblin, S., Poole, N., ... & Forsyth, S. (2006). Transfer of accountability: transforming shift handover to enhance patient safety. Healthc Q, 9(suppl), 75-79. J. Baxter, Theoretical Models of Learning to Learn, pp. 71–94. Boston, MA: Springer US, 1998.

Common Voice by Mozilla. (2019). Retrieved June 6, 2019, from https://voice.mozilla.org/en

Derlet, R. W., Richards, J. R., & Kravitz, R. L. (2001). Frequent overcrowding in US emergency departments. Academic Emergency Medicine, 8(2), 151-155.

Folk, M., Heber, G., Koziol, Q., Pourmal, E., & Robinson, D. (2011, March). An overview of the HDF5 technology suite and its applications. In Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases (pp. 36-47). ACM.

Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P., & He, K. (2018). Detectron.

Ha, D., & Eck, D. (2017). A neural representation of sketch drawings. arXiv preprint arXiv:1704.03477.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... & Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567.

Iwahashi, N. (2004, September). Active and unsupervised learning for spoken word acquisition through a multimodal interface. In RO-MAN 2004. 13th IEEE International Workshop on Robot and Human Interactive Communication (IEEE Catalog No. 04TH8759) (pp. 437-442). IEEE.

Joint Commission. (2012). Transitions of care: the need for a more effective approach to continuing patient care. Oakbrook Terrace, IL: Author. https://www.jointcommission.org/assets/1/18/Hot_Topics_Transitions_of_Care.pdf

Majewski, M., & Kacalak, W. (2016, September). Intelligent speech-based interactive communication between mobile cranes and their human operators. In International Conference on Artificial Neural Networks (pp. 523-530). Springer, Cham.

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), 1345-1359.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

Pratt, L. Y. (1993). Discriminability-based transfer between neural networks. In Advances in neural information processing systems (pp. 204-211).

Quick, Draw! The Data. (n.d.). Retrieved June 13, 2019, from https://quickdraw.withgoogle.com/data

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Chen, Y. (2017). Mastering the game of go without human knowledge. Nature, 550(7676), 354.

Stensrud, B., Taylor, G., & Crossman, J. (2006). IF-Soar: A virtual, speech-enabled agent for indirect fire training. In Proceedings of the 25th Army Science Conference.

Stensrud, B., Taylor, G., Montefusco, J., Schricker, B., & Maddox, J. (2008). An intelligent user interface for enhancing computer generated forces. SOAR TECHNOLOGY INC ANN ARBOR MI.