Rapid Retraining Architecture for Deploying AI/ML at the Speed of Relevance

Dennis Chen, Victoria Dorn, Arman Ommid, Rey Nicolas, Anastacia MacAllister

General Atomics

San Diego, CA

dennis.chen@ga-asi.com, victoria.dorn@ga-asi.com, arman.ommid@ga-asi.com,
rey.nicolas@ga-asi.com, anastacia.macallister@ga-asi.com

ABSTRACT

Artificial Intelligence (AI) and Machine Learning (ML) investment is exponentially increasing in both the DoD and industry, the DoD requested $130.1 billion in 2023 alone, due to its recognized ability to handle dull, dirty, or dangerous tasks. However, much of this investment goes towards early-stage research and development or proof of concept projects which fail to reach warfighters. Often work in early-stage R&D succumbs to the valley of death due to gaps in funding for operationalization of lab proven concepts. One such hurdle often encountered when attempting to operationalize AI/ML models is the lack of end-to-end infrastructure supporting all AI/ML life cycle stages such as development, testing, deployment, and maintenance.

End to end infrastructure is important for AI/ML models due to their unique data needs, however, this necessity is rarely mentioned in literature. Without robust end to end architecture, adeptly tracking and monitoring model parameters and data is often a cumbersome labor-intensive process. This lack of tracking and slow triaging of model performance issues often results in subpar model performance when fielded in operational settings. This paper looks at the critical components required to bridge the R&D valley of death to operationalize AI/ML based on a currently fielded rapid retraining architecture for object recognition. The paper starts by defining and describing three main stages in an end-to-end rapid retraining pipeline: 1) data collection and preprocessing; 2) experimentation and model development; and 3) model deployment and monitoring. From here the paper covers a use case using the pipeline, showing automation helps reduce time to run the pipeline by 75% and delivers a 10% increase in model accuracy. Ultimately, work in this paper demonstrates the importance of automating the AI/ML development and deployment process to effectively operationalize this critical new tool for the warfighter.

ABOUT THE AUTHORS

Dennis Chen is a Machine Learning Engineer at General Atomics Aeronautical Systems working on Autonomy and AI solutions. His work focus on creating the tooling and infrastructure for productizing machine learning algorithms. Throughout his career, Dennis has made contributions in various areas such as automated labeling, computer vision algorithms, and autonomous UAV flight. Dennis holds a bachelor's degree in Computer Science from Hong Kong University of Science and Technology.

Victoria Dorn is a Machine Learning Engineer at General Atomics Aeronautical Systems. She develops rapid retraining pipelines for machine learning models in areas such as object detection and prognostic health management. She utilizes this pipeline to run diagnostics and compare various models to improve model performance. She holds a B.S. in Computer Science and Engineering with a concentration in data science from Lehigh University.

Arman Ommid is a Machine Learning Engineering Intern for General Atomics Aeronautical Systems. His work looks at computer vision models for automated data annotation and reinforcement learning development tool chains. He received his B.S. in Computer Science from University of California San Diego, where he is currently pursuing a Master's in Artificial Intelligence and Machine Learning.

Rey Nicolas is Director of Software, Autonomy and AI Solutions and leads GA-ASI's Autonomy and AI teams developing next generation Autonomous Command and Control (C2) systems, Autonomous Processing, Exploitation, Dissemination (PED) systems, and AI edge processing for flight and sensor autonomy, air-to-air combat, and air-toground Intelligence, Surveillance, and Reconnaissance (ISR) missions. Rey is also an Executive Committee Leader for SAE Standard Works that is developing AI in Aviation Safety Standards. Previously, Rey worked for Intel Corporation and Motorola/Google leading AI R&D and product development for multiple product lines in the smart and connected home, autonomous driving, and healthcare applications. Rey holds a bachelor’s degree in Mechanical Engineering from University of California San Diego, Master’s in Computer Science from the University of Illinois, and MBA from San Diego State University.

Anastacia MacAllister, Ph.D., is Technical Director of Autonomy and Artificial Intelligence for General Atomics Aeronautical Systems (GA-ASI). Her work focuses on prototyping novel machine learning algorithms, developing machine learning algorithms using sparse, heterogenous or imbalanced data sets, and exploratory data analytics. Throughout her career she has provided key contributions in a number of interdisciplinary areas such as prognostic health management, human performance augmentation, advanced sensing, and artificial intelligence aids for future warfare strategies. This work has received several awards and commendations from internal technical review bodies. Dr. MacAllister has published over two dozen peer reviewed technical papers, conference proceedings, and journal articles across an array of emerging technology concepts. Dr. MacAllister holds a bachelor's degree from Iowa State University (ISU) in Mechanical Engineering. She also holds a Master's and PhD from ISU in Mechanical Engineering and Human-Computer Interaction.

Rapid Retraining Architecture for Deploying AI/ML at the Speed of Relevance

Dennis Chen, Victoria Dorn, Arman Ommid, Rey Nicolas, Anastacia MacAllister

General Atomics

San Diego, CA

dennis.chen@ga-asi.com, victoria.dorn@ga-asi.com, arman.ommid@ga-asi.com, rey.nicolas@ga-asi.com, anastacia.macallister@ga-asi.com

INTRODUCTION

Investment in Artificial Intelligence (AI) and Machine Learning (ML) is exponentially increasing in both the department of defense and industry. The United States Department of Defense (DoD) requested $130.1 billion for AI/ML in 2023 alone, due to its recognized ability to handle dull, dirty, or dangerous tasks. However, much of this and past investment goes towards early-stage research or proof of concept projects that fail to reach warfighters in a meaningful way. This phenomenon in the AI/ML space often starts when a promising technology, like automatic target recognition (ATR), is developed and demonstrated in a lab like setting. However, once the technology reaches a medium technology readiness level (TRL) it often languishes in this state without being operationalized. This follow through gap can be attributed to the valley of death early to mid-stage R&D often falls into, due to lack of scaling architecture. Unfortunately, this architecture development often falls outside of the scope and expertise of these R&D organizations. As a result, to effectively transition AI/ML technology, the DoD needs to carefully consider how to operationalize and support these models at scale throughout their lifecycle.

Model lifecycle stages to consider consist of model development, training, testing, deployment, and maintenance. The model development stage covers aspects like selection of AI/ML algorithms and data set collection. Decisions made during this stage are important to capture and justify. Early-stage information capture ensures proper testing and model monitoring is conducted later. Model training covers how each model is taught to perform its intended task. The testing phase during the lifecycle aims to benchmark model performance under certain expected deployment conditions. Understanding which algorithms were used and how the data was constructed will directly impact the testing plan. The deployment stage addresses the infrastructure required to integrate AI/ML models into a larger operationalized system of systems. Lastly the maintenance stage addresses model monitoring and retraining to ensure continuous effective performance. To operationalize AI/ML models effectively each of these phases need to be stitched together in a unified way to enable effortless trackability. Unfortunately, effective development of this infrastructure is rarely addressed in literature, resulting in a persisting gap between proof of concept and operationalizing technology critical to 21st century conflicts.

This paper aims to begin the conversation about what an end-to-end rapid retraining process looks like for the DoD. The goal is to develop a flexible architecture to organize, track, and automate key pieces of AI/ML development to enable model deployment at speeds relevant to today's information driven fight. The paper begins by categorizing and providing background information on key phases in the AI/ML model development lifecycle. From here the paper describes a rapid retraining architecture design that connects these key phases into an end-to-end deployment and monitoring tool for supervised object detection models. After describing the rapid retraining architecture, the paper presents a use case describing how it was used to deploy and rapidly retrain automatic target recognition (ATR) models. Ultimately, the goal of this paper is to start the discussion surrounding sound operationalization practices for AI/ML models and what it might look like in practice.

BACKGROUND

There are three major components to any complete AI/ML infrastructure: data collection & preprocessing, experimentation & model development, and model deployment. Each play an important role in the process of operationalizing AI/ML models and are most effective when fit together into a coherent end-to-end infrastructure.

This section will provide a review of previous work in each of these components to motivate why they are necessary for effective AI/ML development and to inform how previous literature was extended upon to construct the architecture presented below.

Data Collection and Preprocessing

The Data Collection and Preprocessing stage aims to control data related tasks like storage, collection, and preprocessing to meet the needs of downstream training processes. The objectives are to be able to efficiently collect, store, retrieve, and process domain representative data to minimize effort required for preparation stages of AI/ML training cycles. Work by Mahmood et al. explores solutions around optimal data collection leveraging ideas from statistical theory, neural scaling laws, and data-centric meta-learning (Mahmood, Lucas, Alvarez, Fidler, & Law, 2022). They formulate what they describe as the "optimal data collection problem" for determining data needs of a project through their approach "Learn-Optimize-Collect." Their work foreshadows how the benefits of a data-model feedback loop can be realized with end-to-end infrastructure. Krishman et al. addresses the problem of data cleaning and preprocessing by framing it as an iterative and progressive process that preserves convergence guarantees to optimize later model performance (Krishnan, Wang, Wu, Franklin, & Goldberg, 2016). Their work highlights the importance of being able to tie how data cleaning methods used early in the pipeline later impact model performance. Related to general data management, Li et al. demonstrates MLog, a high-level declarative language that integrates machine learning into data management (Li, Cui, Chen, Wu, & Zhang, 2017). They design a declarative programming language that bridges the gap between database navigation and machine learning development. This is important because it allows data management to be immediately accessible during model development. It's clear that traceable data collection and preprocessing procedures are critical parts of the AI/ML pipeline, due to model's data centric needs. Without it AI/ML cannot be scaled and operationalized for the warfighter in a meaningful way.

Experimentation and Model Development

Experimentation and Model Development stages focus on training and testing machine learning models. This stage's goal is to conduct experimentation, test, and record different experiment parameters. During experiments various augmentation, architecture, and hyperparameter choices for a classification task are tested to produce the best model possible. Gharibi et al. (2019) outlined a lightweight system for experimentation management called ModelKB that automatically extracts experiment metadata locally to facilitate model selection and reproducibility (Gharibi, Walunj, Alanazi, Rella, & Lee, 2019). The quality of automation in ModelKB's experimentation framework allows the core AI/ML development process to scale. Extending this work, Gharibi et al. (2021) moved to a networked cloud architecture for even greater scalability (Gharibi, Walunj, Nekadi, Marri, & Lee, 2021). While helpful this work still does not completely encompass the scope of end-to-end AI/ML operations. Ferenc et al. also explored solutions for storing experiment results, however, they also included experimental design features (Ferenc, Viszkok, Aladics, & Jász, 2020). Research shows that state of the art MLOps practices are moving towards frameworks that consolidate supporting protocols like artifact management, performance analytics, and orchestration with model experimentation to operationalize the experimentation stage. However, research is lacking that connects each stage of the process in a wholistic manner.

Model Deployment and Monitoring

Model deployment and monitoring focuses on integrating trained models into larger systems for warfighter use. Poor processes and procedures in early stages can make operationalization at this stage challenging. Sculley et al. demonstrated the importance of understanding the complexity of upstream components in machine learning systems to minimize unwanted behavior (Sculley, et al., 2015). Their work highlighted the importance of managing underlying datasets, models, and input signals such as hyperparameters and preprocessing steps. Olston et al. prototyped a uniform model deployment method to solve configuration management challenges by fusing version, configuration and a standard interfaces for software systems to interact with machine learning models (Olston, et al., 2017). Their work aimed to mitigate integration debt incurred between model development and deployment stages by ensuring a shared single standard configuration. Klaise et al. looked at how to ensure high model performance in live deployment by mitigating failure scenarios that occur due to outliers or data drift from the training data set (Klaise, Looveren, Cox, Vacanti, & Coca, 2020). Their drift detection informs the system during deployment if the model should be retrained on a new data. In addition, opensource libraries such as Alibi Detect (Loovern et al.) also provide off the shelf anomaly

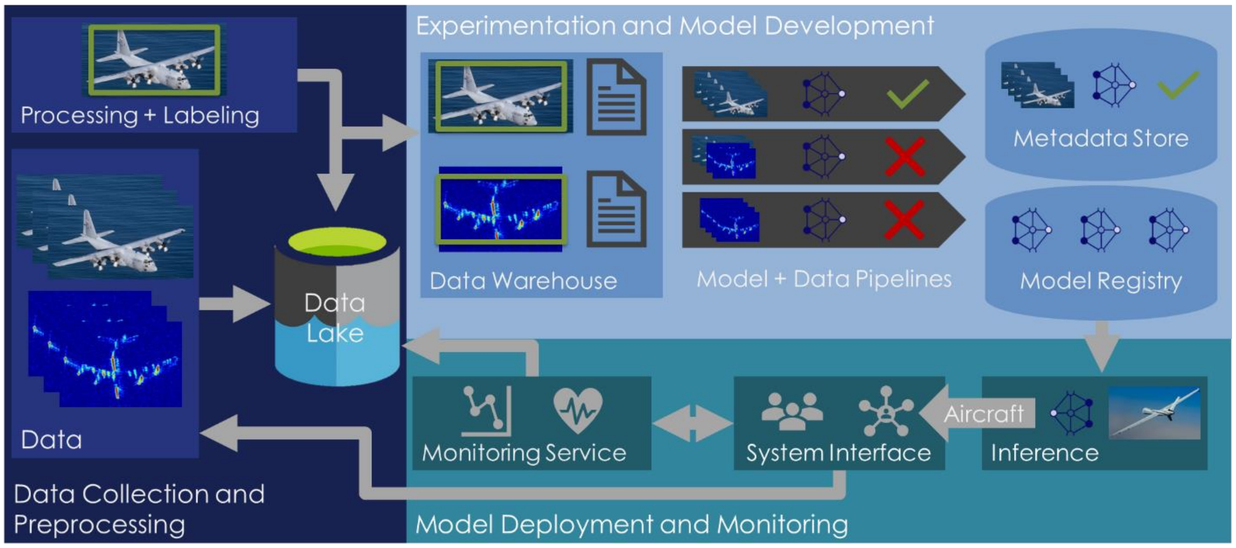

Figure 1. The rapid retraining pipeline is an interconnected set of COTS and custom tools that automate model training and tracking to improve operationalization of AI/ML technology

detectors as well as drift detection which can be used to finetuned to fit a problem domain (Looveren, 2023). However, these deployment validation measures can only be used if the proper upstream practices are used.

In the end, operationalizing machine learning requires an understanding of the data a model was trained on, an understanding of the model training process, and an understanding of how those decisions impact model deployment. Without robust end to end pipelines for storing, tracking, and monitoring this information truly realizing the potential of AI/ML technology in real day to day operations is challenging.

RAPID RETRAINING ARCHITECTURE DESIGN

While literature points to the importance of understanding the provenance of decisions that go into the machine learning model creation process, very few examples exist of a truly end-to-end system. This section begins to describe an automated rapid retraining process that was designed to begin fulfilling this need. This section starts by describing the overall architecture of the rapid retraining pipeline, as shown in Figure 1. It then describes each of the three stages in detail and what design components were utilized for each stage. The paper does not, however, describe the specific software tools used in each stage. There are many open-source tools and commercial tooling frameworks that can be used to fulfill roles described in the pipeline components below. The aim of this paper is to be vendor agnostic and provide an abstract overview of the stages and components that are required to build such an end-to-end system.

System Overview

The rapid retraining pipeline can be broken down into three stages: 1) data collection and preprocessing; 2) experimentation and model development; 3) model deployment and monitoring. When developing machine learning models, processes that allow iterative improvement of data and models are key. Quick iteration enables higher performance and adaptability to changing requirements. The first part of the pipeline is data collection and preprocessing. During this stage, data is rapidly collected so it is imperative to have a scalable means to process and organize it for developers to use. For applications like ATR, AI/ML model performance is extremely dependent on the quality and quantity of labelled data. Only with scalable and automated processes can organizations translate large quantities of data into useful knowledge.

Experimentation and model development stages are where model training and improvement iterations occur. Here developers use high quality data from the previous step to initiate parallel experiment runs with various hyperparameters, data augmentations, and data subsets. These tunable parameters are often modified experimentally by developers to create high performance AI/ML models. In a research setting, these tweaks may be performed

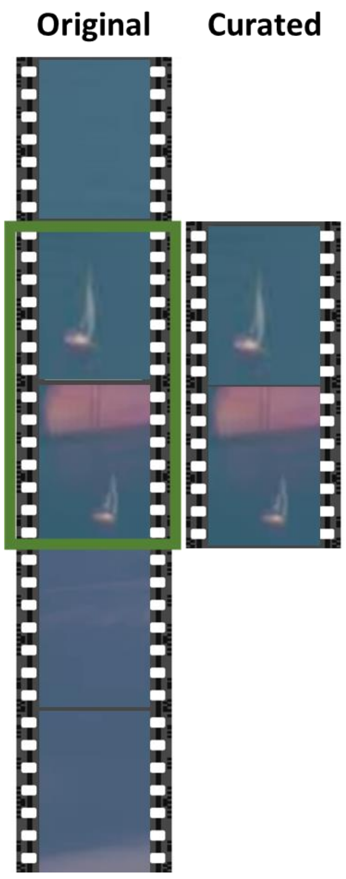

Figure 2. Data curation ensures the model only trains on relevant images

manually and described in code, but this is often not scalable or traceable in later steps. As a result, this framework aims to automate experiment parameter collection as much as possible. Once an optimal model is finalized, a model with its descriptive parameters is registered. From here the model can be moved to the last stage in the process and be deployed to the warfighter. Since model development and deployment are usually at least two distinct teams, many variables need to be communicated across these two stages to ensure performance engineered by model developers translates into the end software product. During the deployment stage model monitoring also occurs. This records model performance in real time to capture any model misclassifications so feedback can be provided back into the rapid retraining system to iteratively improve models.

Data Collection and Preprocessing

The goal of the data collection and processing step is to clean and format data into a usable form for machine learning models. Effectively executing this stage is critical to producing deployable models. Unfortunately, real-world data required for these models is often scattered and requires a significant amount of formatting to make usable. Formatting or cleaning actions at this stage can include labeling, time synchronization, outlier removal, and missing value replacement. The main software components used for data collection include: a data lake, a data warehouse, a workflow orchestration framework, labeling tools and/or external labeling services. A data lake is used to collect and store raw data from the field along with byproducts of data processing pipelines. A data warehouse is used transform raw data into organized useful data. Workflow orchestration tools allow scheduling of code workflows, which often involve extracting data, transforming data, and loading data into either the data lake or data warehouse. Often, data needs to be labelled for supervised learning problems like ATR, so organizations will opt to either provide a labeling tool that an inhouse team can use to label data quickly or outsource it to an external service.

The overall workflow begins by collecting large amounts of unorganized and uncleaned data. The data is then organized for analysis and labeling in a semi-automated fashion with both humans reviewing the data and code executing automated checks. With workflow orchestration frameworks, developers write pipelines that perform standard preprocessing steps and then load it into the data warehouse. By ingesting raw data into data warehousing software, data selection can occur with human reviewers or subject matter experts (SMEs) by using queries to find the most relevant and important snippets of data and applying tags. For example, with FMV data a clipping functionality could be used to extract only the important ten second events out of a four-hour long flight log as shown in Figure 2. Only after curating the most important events and organizing them into searchable and quarriable collections, can developers begin to iteratively improve the data. The most common data process step is transmitting the data to a label tool or external labeling service, where labels are obtained for supervised learning tasks. However, other data improvement processes are possible such as running them through automated data pipelines that extract or generate metadata that could be used as input into model training. For example, a data pipeline could include determining if the video is night or day, the presence of heads-up display (HUD) graphics, or how obscure the target object is. These generated datapoints are then re-ingested back into the data warehouse for querying and analysis. Obtaining metrics and understanding of the data is crucial to planning what type of additional data collects are necessary to mitigate gaps in model performance. Traditional processes may involve a human reviewer looking through folders of data trying to find the relevant data, but this is expensive and slow for the fast iteration needed for AI/ML development. By offloading as much of the data improvement process to automated pipelines, human labor is only limited to the review and labeling process, which can be augmented with automated tooling. Workflow orchestration tools form various processes in a Directed Acyclic Graph (DAG) so that the completion of one data process can trigger multiple others dependent on it, this type of processing is easily extensible and robust to partial failures. For example, upon collecting a set of images to label, they can be sent to three different services: manual labeling, night vs day classification, and sending an email notification to waiting developers. Ultimately, automating large parts of data processing enables higher quality models and faster release cycles, leading to more robust AI/ML products for the warfighter.

Experimentation and Model Development

The experimentation and model development stage is comprised of: independent model training pipelines managed by workflow orchestration tools, metadata store, model registry, and data warehouse. In contrast to the data iteration steps above, model iteration is not constrained by human labor but rather computation power. Finding the optimal model requires extensive testing of many combinations of data and model hyperparameters. To facilitate this rapid experimentation the pipeline leverages workflow orchestration tools to read data augmentation and model configurations automatically to deploy experiments instantly. As a result, scaling the number of experiments running becomes as simple as adding more servers to the cluster.

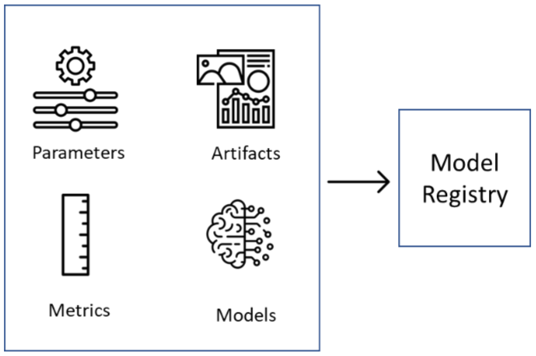

Figure 3. Automatically storing relevant parameters, train/test artifacts, metrics, and models in a structured registry helps improve

However, model improvement is often also very reliant on the quality and quantity of data not just model hyperparameters. As a result, the pipeline also logs metrics about the data set that are computed in the data processing stage. These metrics can include the distribution of classes and number of unique images. These logged data set metrics and hyperparameters are used by developers to adjust model performance. During this experimentation, metadata stores are used to track various combinations of data and hyperparameters to allow collaborative sharing of results between model developers. Previously, traditional experimentation processes may involve using a spreadsheet to collaborate data points; however, this is both tedious and unreliable. Metadata stores offer API integrations into code, which allow developers to easily log experiment metadata and view them. It also offers the additional benefit of allowing developers to have data lineage and traceability when a model inference error appears in production. Just as with traditional software development code version control, data and model versioning are equally important when it comes to developing AI/ML models. Once a high performing combination of data and hyperparameters are found, the model and its artifacts are sent to the model registry. The model registry, Figure 3, is where a finalized model is pulled from deployment. This is the communication link between the model developers and deployers, so it's important as much information about upstream steps are included for troubleshooting issues in the field. By automating experiment and parameter logging in upstream steps, the handoff at this stage has a high chance of success.

Model Deployment

The final stage of the pipeline is model deployment and monitoring. For deployment, models are often placed into a larger software service that uses AI/ML models as a subsystem. While running the models the monitoring service will ideally capture any errors, which will be provided as feedback to model developers for improving future performance. However, to generate this error information the deployment service needs to understand baseline model performance and for developers to receive this information the proper feedback loops need to be built into the pipeline. By packaging metrics and parameters logged in the metadata store above, deployment servers can load metadata parameters automatically to ensure that the performance seen by the model developers are the same during deployment. If it is not, then automated reports are generated and sent back into the data lake. Where they can be ingested by model developers to make improvements. Ultimately, reducing lag time between field reports of model performance and retraining is the key to improving model performance quickly. Manual processes often take months or years, whereas the automated pipeline described allows models to learn new information from the battlefield in the order of weeks and days. This quick iteration allows model adjustments to be made at the pace of 21st century conflicts.

In the end by automating key steps in the AI/ML pipeline the rapid retraining system can iterate and operationalize models more effectively. Automating data processing minimizes expensive manual labeling and sorting of data. Parallel execution of experiments allows developers to test large combinations of data/model parameters quickly and find an optimal model, while allowing room for collaboration by using a metadata store. Model monitoring ensures errors and new classes found in the field are provided as feedback into the retraining system, allowing the model to adapt quickly to new conditions. Together, these processes allow the robust and effective operationalization of AI capabilities in the real world.

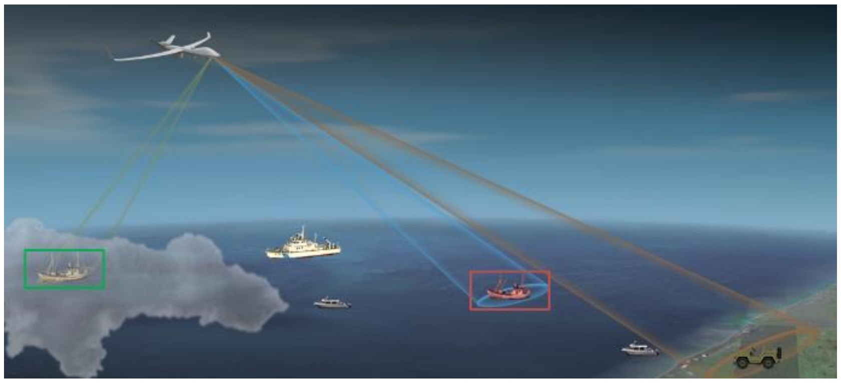
Figure 4. Warfighter uses ATR model on live field data, data is collected and sent back to rapid retraining process to improve model performance

TESTING RESULTS & DISCUSSION

This section covers a real-life use case demonstrating how an end-to-end rapid retraining pipeline helps improve machine leaning model operationalization over more traditional manual methods. The use case compares the manual release process used for initial model releases with the automated rapid retraining pipeline. The use case is shown in Figure 4. The model developed was for automatic target recognition (ATR) in a maritime environment. The aircraft is equipped with sensors that feed an ATR model which provides object classifications, shown as red and green bounding boxes, to an operator. These detections or any misclassifications are recorded and sent back to model developers to help improve the model. The following sections describing the manual model pipeline along with a task-based analysis using Natural Goals Operators Methods, Selection Language (NGOMSL). After the manual process is presented, a similar analysis is conducted for the automated pipeline. The section concludes with a discussion of how differences between the pipelines impact the ability to operationalize the ATR model. Ultimately, the goal is to demonstrate how automated pipelines can decrease model release timelines, increase traceability of model parameters, and increase model accuracy.

Manual vs Rapid Retraining Results

Sections above have described the reasoning and architecture behind the automated rapid retraining pipeline. This section seeks to contextualize those decisions by providing a more quantitative look at the benefits by comparing execution times of both manual and automated processes. Figure 5 depicts steps required to execute the manual

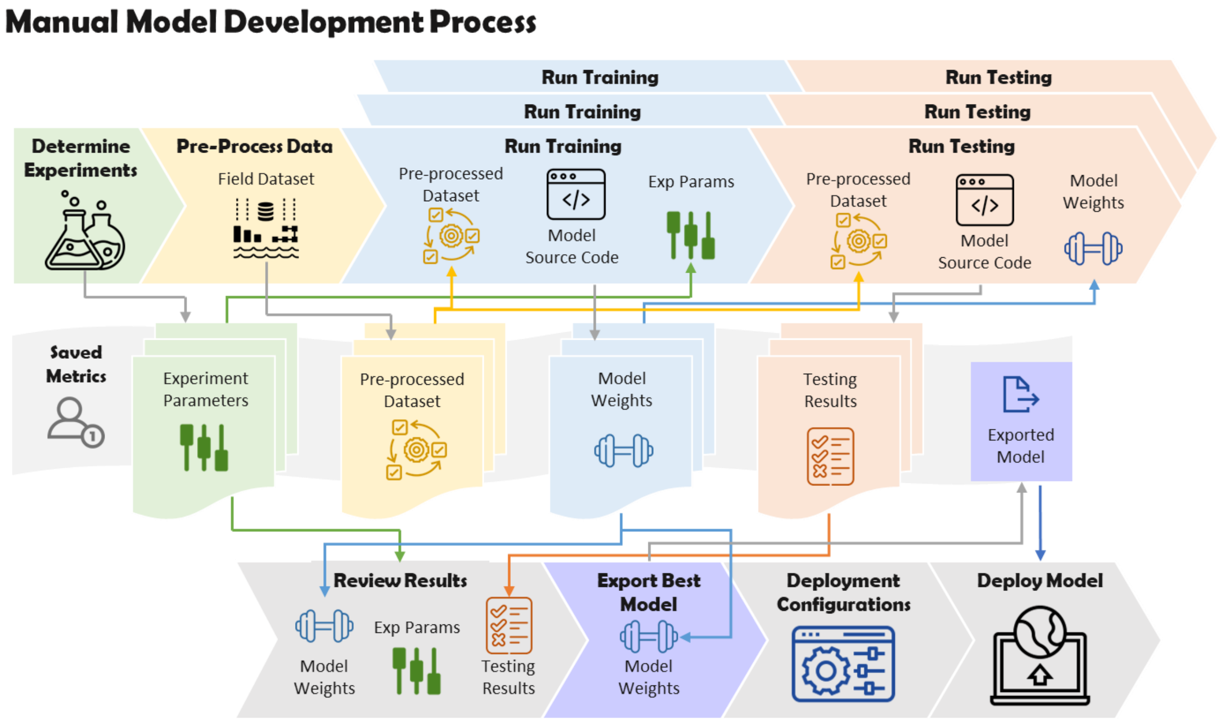

Figure 5. The manual model training process consists of fragmented model development stages where metrics are saved in local user directories, limiting process automation and optimization

pipeline. Note the graphic does not show steps where data is sent back from the field and manually labeled. For the comparison, this step is excluded because it is consistent across manual and automated methods. In the manual pipeline there are eight sequential steps. If a user wanted to run multiple experiments, they must send off separate training and testing runs for each. Looking at inputs (color coded arrows in the graphic) and outputs (grey arrows in the graphic) for each step, shows metrics are saved locally on a per user basis and without a unifying structure. This is a downfall of the manual process because it does not facilitate easy collaboration between team members, outputs could easily be deleted, and deployed model weights were not uniformly tracked after release.

To help quantify the overall time it took for a user to work through this process, a GOMS based task analysis was selected due to the discrete nature of each of the steps. This type of task analysis was originally proposed by Card, Moran, and Newell (Card, Moran, & Newell, 1983) and stands for Goals, Operators, Methods, and Selection (GOMS). There are four variations: the original (CMN-GOMS) which focuses on how to express a "goal hierarchy, methods and operators, and how to formulate selection rules" (Kieras, A Guide to GOMS Model Usability Evaluation using NGOMSL, 1996); the Keystroke-Level Model (KLM) which only shows the operators; the Natural GOMS Language (NGOMS) which takes a GOM and expresses it in pseudo code like syntax; the CPM-GOMS which uses cognitive, perceptual and motor operators to show how activities can be performed in parallel (Kieras, A Guide to GOMS Model Usability Evaluation using NGOMSL, 1996). NGOMSL specifically was selected because it estimates execution time as well as time it takes to complete higher level complex operators, both components of the pipeline. The downfall with this evaluation method is that it is unable to express parallel multitasking, but with the processes discussed parallel operations are not commonly used.

Before diving into the NGOMSL model, first assumptions need to be addressed. The model was created with a specialized user in mind that is familiar with the system, knows the location of datasets, and has used the system before. Additional assumptions are written out at the beginning of the analysis for both the manual and rapid retraining process. For the system execution time a single experiment is assumed, however, scaling effects as experiments are batched will be discussed later.

TABLE 1

	MANUAL MODEL RELEASE PROCESS NGOMSL ANALYSIS	# of Executions	External Operator Times
ASSUMPTIONS: EXERPIENCED USER, AVERAGE TYPIST, PROVIDED DATASET AND EXPERIMENTS, CUSTOMER FEEDBACK IS OPEN, MODEL SOURCE CODE IS OPEN, DATASET LOCATIONS ARE ALREADY KNOWN, COMPLEX MENTAL OPERATORS ARE ABSTRACTED.			
DATA COLLECTION & PREPROCESSING	Method for goal: PREPROCESS DATASET.	* # of ex.	
	Step 1. Determine location of dataset (already known).	1	0.00
	Step 2. Select dataset (CTRL+A).	1	0.40
	Step 3. Copy dataset (CTRL+C).	1	0.40
	Step 4. Move mouse to data storage location (location known).	1	1.10
	Step 5. Click mouse button.	1	0.20
	Step 6. Paste dataset (CTRL+V).	1	0.40
	Step 7. Recall file structure for model type.	1	1.35
	Step 8. Verify dataset folders are in correct file structure.	1	1.35
	Step 9. Verify dataset folders are labeled in correct schema.	1	1.35
	Step 10. Verify dataset does not contain duplicates.	1	1.35
	Step 11. Remove low quality images.	1	1.35
	Step 12. Select new preprocessed dataset (CTRL+A).	1	0.40
	Step 13. Move mouse to training code location.	1	1.10
	Step 14. Click mouse button.	1	0.20
	Step 15. Paste dataset to training location (Same location as model source code).	1	0.40
	Step 16. Retain location of testing dataset.	1	1.35
	Step 17. Return with goal accomplished.	1	
			= 12.7 sec

		* # of ex.	
EXPERIMENTATION & MODEL DEVELOPMENT	Method for goal: DEVELOP MODEL		
	Step 1. Open model source code (already open).	1	0.00
	Step 2. Move mouse to new terminal button.	1	1.10
	Step 3. Click mouse to open new terminal.	1	0.20
	Step 4. Move mouse to terminal.	1	1.10
	Step 5. Click mouse button.	1	0.20
	Step 6. Type in training command with necessary experiment flags. (Average number of keystrokes = 225).	1	45.0
	Step 7. Press enter button.	1	0.20
	Step 8. Wait for training code to complete.	1	t
	Step 9. Type in testing command with necessary experiment Flags. (Average number of keystrokes = 200).	1	40.0
	Step 10. Press enter button.	1	0.20
	Step 11. Wait for testing code to complete.	1	t
	Step 12. Move mouse to testing output location.	1	1.10
	Step 13. Click mouse button.	1	0.20
	Step 14. Review results.	1	1.35
	Step 15. Move mouse to best model weights.	1	1.10
	Step 16. Click mouse button to select model weights.	1	0.20
	Step 17. Copy best model weights path (CTRL-C).	1	0.40
	Step 18. Retain best model weights path.	1	1.35
	Step 19. Return with goal accomplished.	1	= 93.7+ 2t
MODEL DEPLOYMENT	Method for goal: DEPLOY MODEL		
	Step 1. Open model source code (already open).	1	0.00
	Step 2. Move cursor on top terminal.	1	1.10
	Step 3. Click mouse button.	1	0.40
	Step 4. Recall best model weights path is saved to clip board.	1	1.35
	Step 5. Type in exporting command with necessary flags (Average number of keystrokes = 75).	1	15.0
	Step 6. Wait for exporting code to complete.	1	t
	Step 7. Move mouse to exported model location.	1	1.10
	step 8. Click mouse button to select exported model.	1	0.20
	Step 9. Copy exported model (CTRL-C).	1	0.40
	Step 10. Type in command to make new directory.	1	2.00
	Step 11. Press enter button.	1	0.20
	Step 12. Type in command to enter the new directory.	1	2.00
	Step 13. Copy exported model into the new directory.	1	0.40
	Step 14. Select deployment testing data subset (3 images).	1	4.05
	Step 15. Copy deployment testing data subset (CTRL-C).	1	0.40
	Step 16. Paste deployment testing data subset in export directory.	1	0.40
	Step 17. Determine deployment configurations (i.e. metrics).	1	1.35
	Step 18. Determine sha256sum for verification.	1	1.35
	Step 19. Write deployment configurations in document inside of the export folder.	1	75.0
	Step 20. Open application to send to integration team.	1	1.10
	Step 21. Copy over export folder into message.	1	0.80
	Step 22. Move cursor to send button.	1	1.10
	Step 23. Click send button.	1	0.20
	Step 24. Return with goal accomplished.	1	= 109.9 + t

TOTAL TIME ESTIMATION: (60 STEPS * 0.1 SECS) + (12.7 + 93.7 + 109.9) + SYSTEM IDLE TIME = 222.3 + SYSTEM IDLE TIME

Table 1 shows the high-level goals, methods, and operators of the manual process. This task analysis and established NOGMSL framework shows total time execution time for the manual process is 222.3 + System Idle Time. This is derived from the execution time formula that Bovair, Kieras, & Polson developed (Bovair, Kieras, & Polson, 1990). The general formula is: "NGOMSL statement time + Primitive Operator Time + Analyst-defined Mental Operator Time + Waiting Time" (Bovair, Kieras, & Polson, 1990). In this formula the mental operator time refers specifically

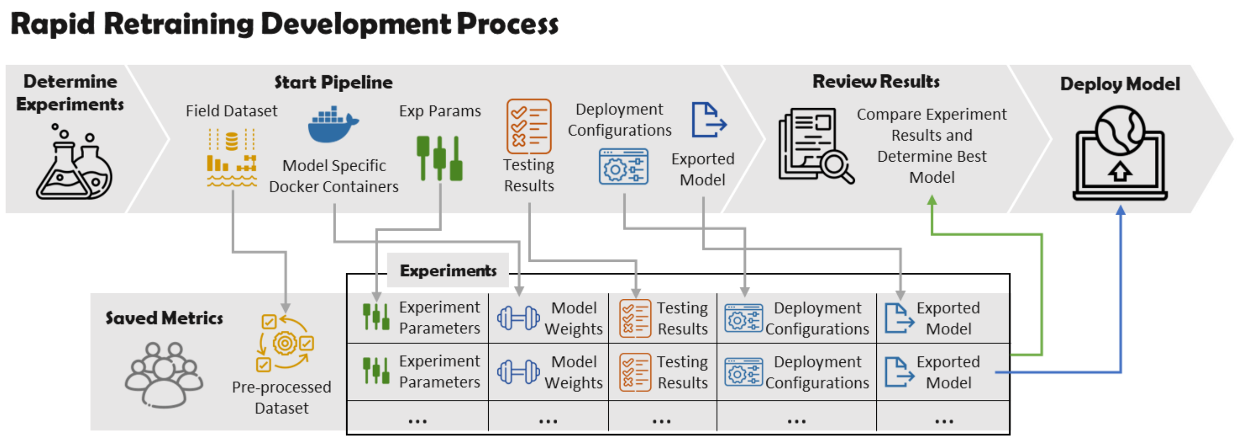

Figure 6. The rapid retraining architecture uses automation to batch experiments and uses automated logging to track artifacts for future model analysis and improvements

to the mental operator time and is calculated in with the primitive operator time in the table. An important note when analyzing this process is that creating a model is more complex and requires high level thinking, something that is not represented in GOMS. We determined that this high-level cognitive time would be the same for both the manual and rapid retraining processes because each process requires a user to think about new experiments to run and go through the testing results to determine which model should be deployed. These steps are captured in the diagram of how each process works, but not included in the task analysis that focuses on procedural run time difference between the two pipelines. In addition to not quantifying the high-level cognitive time we opted to not quantify the idle time of the system. This is because the wait time of the system is highly dependent on the amount of data that is being worked with, the number of experiments, and overall computing power of the system. As a result, running through the manual process and the rapid retraining process with the same parameters and dataset would equate to a similar system idle time, therefore it was not included in overall execution time of each system. Accordingly, we arrived at an execution time of 222.3 seconds for the manual pipeline.

The next step is to now analyze the automated rapid retraining pipeline to determine execution time. Figure 6 shows the automated rapid retraining pipeline. This pipeline has a shared centralized location for saving standardized metrics. Sharing these outputs from different experiments ensures users do not waste time repeating experiments and that experiments can be recreated for testing purposes. As mentioned previously, this diagram does not include data labeling, assumed to happen externally to the pipeline. Following the same assumptions from the manual pipeline, the rapid retraining pipeline was analyzed using NGOMSL and shown in Table 2. Note due to automation and more streamlined experiment batching the rapid pipeline combines some of steps seen in the manual pipeline. Based on this analysis the total execution time was calculated to be 54.0 seconds.

TABLE 2

RAPID RETRAINING MODEL RELEASE PROCESS NGOMSL ANALYSIS		# of Executions	External Operator Times
ASSUMPTIONS: EXERPIENCED USER, AVERAGE TYPIST, HAS GIVEN DATASET AND EXPERIMENTS, CUSTOMER FEEDBACK IS OPEN, RAPID RETRAINING PIPELINE UI AND CODE IS OPEN, DATASET LOCATIONS ARE KNOWN, COMPLEX MENTAL OPERATORS ARE ABSTRACTED.			
DATA COLLECTION & PREPROCESSING EXPERIMENTATION & MODEL DEVELOPMENT EXPORTING MODEL	Method for goal: RUN AUTOMATED PIPELINE.		
	Step 1. Determine location of dataset (already known).	1	0.00
	Step 2. Select dataset (CTRL+A).	1	0.40
	Step 3. Copy dataset (CTRL+C).	1	0.40
	Step 4. Type location of dataset and experiment parameters into input file (Average number of keystrokes = 200).	1	40.0
	Step 5. Move mouse to UI to start run.	1	1.10
	Step 6. Click mouse button to send off the rapid retraining pipeline.	1	0.20
		1	
	Step 7. Wait for system to complete.	1	t
	Step 8. Return with goal accomplished.		= 42.1 + t

MODEL DEPLOYMENT	Method for goal: DEPLOY MODEL		
	Step 1. Move cursor on top of the model for deployment.	1	1.10
	Step 2. Double click mouse button.	1	0.40
	Step 3. Move cursor to register model button.	1	1.10
	Step 4. Click mouse button.	1	0.20
	Step 5. Type name with model release version number (Average number of keystrokes = 10).	1	2.00
	Step 6. Move cursor to save button.	1	1.10
	Step 7. Click mouse button.	1	0.20
	step 8. Select deployment configuration file and exported model.	1	0.40
	Step 9. Copy selected files (CTRL-C).	1	0.40
	Step 10. Open application to send to integration team.	1	1.10
	Step 11. Paste selected files (CTRL-V).	1	0.40
	Step 12. Move cursor to send button.	1	1.10
	Step 13. Click send button.	1	0.20
	Step 14. Return with goal accomplished.	1	= 9.7 secs
TOTAL TIME ESTIMATION: (22 STEPS * 0.1 SECS) + (42.1 + 9.7) + MENTAL OPERATOR TIME + SYSTEM IDLE TIME = 54.0 + MENTAL OPERATOR TIME + SYSTEM IDLE TIME			

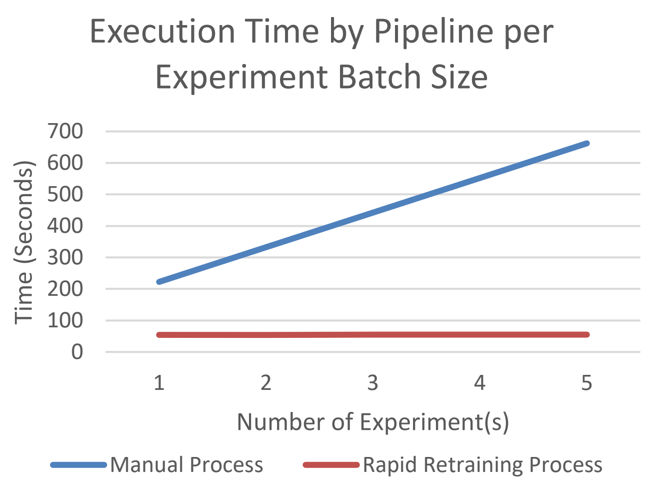

Figure 7. Performance of Manual verse Rapid Retraining for multiple experiments.

Task analysis results demonstrate that automation in the rapid retraining pipeline results in a four times speed improvement over the manual process. That means each time model training is run using the more highly automated and integrated rapid retraining pipeline gives 75% back to the user to do other tasks like determine experiments, label data, or review past results. This gives model developers time to solve other complex problems, ultimately producing better products for the warfighter.

In addition to single experiment execution time, batch experiment time can also be computed. Due to automated batching, adding an experiment to the automated pipeline does not increase its execution time, the difference would be possibly adding more lines to the input file. This addition comes at an execution time of 0.20*(# of keystrokes). The manual model development process on the other hand requires running through most of the sequential process for each experiment. Adding experiments in this case would mean for each additional experiment the execution time would increase according to the formula: $y = (36x+24)*0.1+106.4x+109.9$. The differences and cumulative time savings for up to five experiments are shown in Figure 7. Another benefit the more automated pipeline is the number of saved metrics. The automated pipeline saves over forty metrics and model parameters whereas manual process only saved fifteen. The centralized automated storage in the automated pipeline means more artifacts are recoded for model retraining and delivery to the deployment team. Both key factors impacting the success of operationalized models over time.

Ultimately, the goal of an end-to-end pipeline is to produce a more accurate model. Figure 8 shows the trajectory of normalized model accuracy improvements over time. Since the pipeline came online model accuracy has been improved by roughly 10.6%, a large increase by deployed AI/ML model standards. The chart also shows the increase in data overtime, which was around 11%. The pipeline allowed the team to keep up with the increase in data flow and ensure that it was fed into the model in a timely manner. Without all the automation and end-to-end infrastructure, the team would not have been able to keep pace with the incoming field data to update the mode in a timely manner.

In the end, these results and comparisons show that the automated pipeline reduces the cost of training and retraining models, because it streamlines processes in an end-to-end manner. The current use case has proven this to be an effective solution to a deployed machine learning model. Through a NGOMSL analysis we show the rapid retraining pipeline has a four times execution speed up verse the manual process. This allows users running model development to spend time on more high-level cognitive thinking, rather than clicking through and setting up fragmented scripts. Results also show the increased time advantage when running multiple experiments because the rapid retraining pipeline's batching. All these benefits add up to helping the warfighter have increased model accuracy in a shorter amount of time.

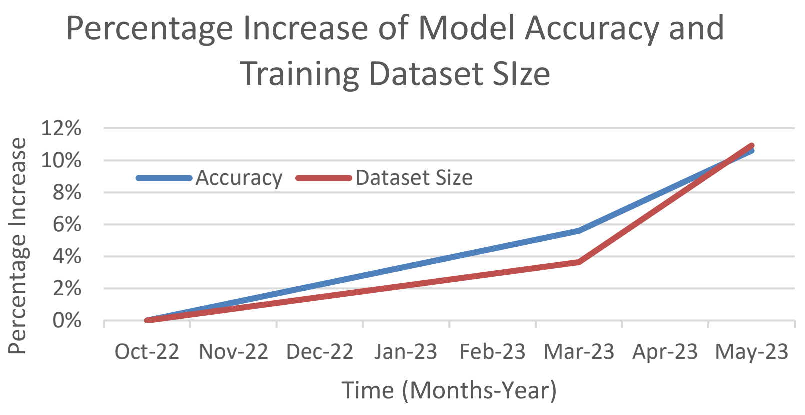

Figure 8. Automation in the rapid retraining process helps more quickly increase model accuracy as new data becomes available

CONCLUSIONS AND FUTURE WORK

The work presented in this paper outlines an end-to-end rapid retraining architecture that leverages best practices in literature to operationalize machine learning models. Each of the rapid retraining pipeline stages were broken down into detail and described. The first part of the pipeline, data collection and preprocessing, highlighted the importance of curating and preparing data for training. The second stage, experimentation and model development, highlighted the need to track and execute well thought out experiments that fit the data. The last stage, model deployment and monitoring highlighted the importance of tracking key metrics throughout development and making sure deployment feedback can make it back to model developers. From here a use case was presented showing the execution time and model accuracy improvements from using the automated rapid retraining pipeline. In the end, this work demonstrates to the community the importance of leveraging end-to-end tools for operationalizing machine learning to ensure it moves out of the lab and into the hands of the warfighter.

Moving forward the team anticipates adapting the system larger quantities of data by including more servers and utilize data parallelization frameworks to handle distributed data processing and distributed training. As the number of use cases increases, the variance of models and data types will require infrastructure distributed in nature to handle the increased computation and storage load. The team would also like to profile the performance of models with various types of hardware and explore optimization techniques such as pruning and quantization to add to the model deployment workflow.

REFERENCES

Bovair, S., Kieras, D., & Polson, P. (1990). *The Acquisition and Performance of Text-Editing Skill: A Cognitive Complexity Analysis*. Retrieved from https://www.researchgate.net/publication/234817321_The_Acquisition_and_Performance_of_Text-Editing_Skill_A_Cognitive_Complexity_Analysis

Card, S., Moran, T., & Newell, A. (1983). *The Phychology of Human-Computer Interaction* (1st ed.). (S. K. Card, Ed.) Hillsdale, New Jersey: Erlbaum.

Ferenc, R., Viszkok, T., Aladics, T., & Jász, J. (2020). Deep-water framework: The Swiss army knife of humans working with machine learning models. *Elsevier*, 1 of 7. Retrieved from https://pdf.sciencedirectassets.com/312019/1-s2.0-S2352711020X00021/1-s2.0-S2352711019303772/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEIz%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJHMEUCIBRArqTslPQo4pKGdjtX60wqteCA7%2BiiOmHJXWLNf%2FDqAiEAufQVqYJt

Gharibi, G., Walunj, V., Alanazi, R., Rella, S., & Lee, Y. (2019). Automated Management of Deep Learning Experiments. *The Third Workshop on Data Management for End-to-End Machine Learning* (p. 1 of 5). ACM.

Gharibi, G., Walunj, V., Nekadi, R., Marri, R., & Lee, Y. (2021). Automated End-to-End Management of the Modeling Lifecycle in Deep Learning. *Emperical Software Engineering*, 1 of 33. Retrieved from https://par.nsf.gov/servlets/purl/10312568

Kieras, D. (1988). Toward a Practical GOMS Model Methodology for User Interface Design. Ann Arbor, Michigan. Retrieved from https://www.researchgate.net/publication/229067883_Toward_a_practical_GOMS_model_methodology_for_user_interface_design

Kieras, D. (1996). A Guide to GOMS Model Usability Evaluation using NGOMSL. Ann Arbor, Michigan. Retrieved from https://web.eecs.umich.edu/~kieras/docs/GOMS/NGOMSL_Guide.pdf

Klaise, J., Looveren, A. V., Cox, C., Vacanti, G., & Coca, A. (2020). Monitoring and explainability of models in production. *Challenges in Deploying and Monitoring Machine Learning Systems* (p. 1 of 7). ICML.

Krishnan, S., Wang, J., Wu, E., Franklin, M., & Goldberg, K. (2016). ActiveClean: Interactive Data Cleaning For Statistical Modeling. *Very Large Data Base Endowment. 9*, p. 1 of 12. VLDB Endownment. Retrieved from http://www.vldb.org/pvldb/vol9/p948-krishnan.pdf

Li, X., Cui, B., Chen, Y., Wu, W., & Zhang, C. (2017). MLog: Towards Declarative In-Database Machine Learning. *Very Large Data Base Endowment* (p. 1 of 4). VLDB Endowment. Retrieved from https://pages.cs.wisc.edu/~wentaowu/papers/vldb17-mlog.pdf

Looveren, V. (2023). *Alibi Detect: Alogithms for outlier, adversarial and drif Detection*. Retrieved from GitHub: https://github.com/SeldonIO/alibi-detect

Mahmood, R., Lucas, J., Alvarez, J., Fidler, S., & Law, M. (2022). Optimizing Data Collection for Machine Learning. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2022/file/c1449acc2e64050d79c2830964f8515f-Paper-Conference.pdf

Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, F., . . . Soyke, J. (2017). TensorFlow-Serving: Flexible, High-Perfomance ML Serving. *NIPS 2017 Workshop on ML Systems*, (p. 1 of 8). Retrieved from https://arxiv.org/pdf/1712.06139.pdf

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., . . . Dennison, D. (2015). Hidden Technical Dept in Machine Leanring Systems. *Advances in Neural Information Processing Systems 28*, (p. 1 of 9). Retrieved from https://papers.nips.cc/paper_files/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html