Machine learning model cards toward model-based system engineering analysis of resource-limited systems

Thomas M. Booth[a] and Sudipto Ghosh[b]

[a]USAF/309th SWEG, Hill AFB, UT, USA
[b]Colorado State University, Fort Collins, CO, USA

ABSTRACT

Sensor fusion combines data from a suite of sensors into an integrated solution that represents the target environment more accurately than that produced by individual sensors. New developments in Machine Learning (ML) algorithms are leading to increased accuracy, precision, and reliability in sensor fusion performance. However, these increases are accompanied by increases in system costs. Aircraft sensor systems have limited computing, storage, and bandwidth resources, which must balance monetary, computational, and throughput costs, sensor fusion performance, aircraft safety, data security, robustness, and modularity system objectives while meeting strict timing requirements. Performing trade studies of these system objectives should come before incorporating new ML models into the sensor fusion software. A scalable and automated solution is needed to quickly analyze the effects on the system's objectives of providing additional resources to the new inference models. Given that model-based systems engineering (MBSE) is a focus of the majority of the aerospace industry for designing aircraft mission systems, it follows that leveraging these system models can provide scalability to the system analyses needed. This paper proposes adding empirically derived sensor fusion RNN performance and cost measurement data to machine-readable Model Cards. Furthermore, this paper proposes a scalable and automated sensor fusion system analysis process for ingesting SysML system model information and RNN Model Cards for system analyses. The value of this process is the integration of data analysis and system design that enables rapid enhancements of sensor system development.

Keywords: MBSE, SysML, Machine Learning, Model Card, Sensor Fusion, Resource, datasheets, factsheets

1. INTRODUCTION

Sensor fusion and sensor fusion architectures have been researched for many decades[1–4] and continuous advancements have been made as sensors, computational power, fusion algorithms, and the collection of data improve in performance and capability. Currently, one of the fastest growing segments in sensor fusion is in Machine Learning (ML) and Deep Learning (DL) algorithms. DL is becoming the leading approach compared to previous classical methods in time-series prediction algorithms as shown by the performance of Recurrent Neural Network (RNN) based approaches in the M5 competition.[5] These large ensemble DL algorithms consume large amounts of computational resources when forecasting. This is not a problem with scalable compute resources, but the ability to run these models on a resource-limited system, such as aircraft or automobiles, may not meet system timing requirements. When considering sensor fusion solutions for resource-limited systems, choosing the best sensor fusion method is more than selecting the model with highest accuracy and precision.[6] Sensor system software designs need to estimate the resource needs of new sensor fusion models as they are developed and before they are implemented. The speed of developing software and new sensor fusion models will continue to outpace hardware upgrades. Therefore, system analysis and design of these limited-resource systems must keep pace with rapid releases of new DL sensor fusion architectures, algorithms, and models.

Safety of flight, system robustness, security, openness, and system modularity objectives are considerations for system analysis and design. Model-Based Systems Engineering (MBSE) is a practice that is focused on

Further author information: (Send correspondence to Tom Booth or Sudipto Ghosh)
Tom Booth: E-mail: tom.booth@NexusDE.com
Sudipto Ghosh: E-mail: sudipto.ghosh@colostate.edu

building model-based designs capable of complex trade studies. MBSE has been shown to provide significant advantages to project performance[7,8] for complex systems. The system development, qualification, verification and validation, and trade studies were more efficient while using MBSE. The Systems Modeling Language (SysML) has become the ubiquitous MBSE language in the aerospace industry and if these SysML system models become the Authoritative Source of Truth (ASoT) for aircraft system specifications, then it is imperative that models contain the information needed for sensor system analyses. This information needs to delivered in a standardized machine-readable format or API for future analysis scalability. In addition to the need for a standardized SysML system model API, DL models need to have machine-readable data, or Model Cards,[9] to enable the automation of and to integrate sensor fusion DL algorithm development with system analysis and design (i.e. close the feedback loop). These DL Model Cards need to contain quantitative data of the sensor fusion performance (accuracy, precision, reliability, latency, etc) and the resource costs (CPU cycles/speed, volatile and non-volatile memory, data size and rates, etc).

The goal of this paper is to evaluate a scalable and automated process we developed that easily passes sensor fusion model information back to system analysis and design through the use of SysML system models and DL Model Cards. We refer to this as the sensor fusion system analysis process in this paper. This process is executed on a case study to evaluate its usefulness. The goal of the case study is to choose the best sensor fusion model for a limited-resource system using pre-trained RNN-based sensor fusion models. An objective function scores the performance of four models as a function of network throughput, model inference latency, data pre-processing latency, accuracy, generalization, and volatile and non-volatile memory consumed. A SysML sensor system model provides system specifications, while the pre-trained DL models provide the performance and cost measurements for the Model Cards. The DL model will be chosen based on the input from the SysML model and the Model Cards and the output of the objective function. The novelty here is the scalable and automated process that uses the machine-readable RNN Model Cards and system input from a SysML model for system analysis and design.

Section 2 summarizes previous work applying Machine Learning (ML) to sensor fusion and the application of Model Cards to ML. Section 3 describes the proposed extension to Model Cards for RNN-based sensor fusion models on resource-limited systems. Section 4 describes the use case and the application of the sensor fusion system analysis process. We then present the results and discuss them in Sections 5 and 6, respectively. Lastly, we present our conclusions and outline directions for future work in Section 7.

2. RELATED WORK

This section reviews several applications of ML to sensor fusion and time-series forecasting. Specifically Section 2.1 reviews three different methods used for sensor fusion and time-series forecasting and highlights the need to share quantitative model performance and cost information with the sensor fusion industry for analysis on limited-resource systems. Section 2.2 discusses the current methods for delivering AI/ML model, data, and service information to the consumers of these products in a responsible manner. This section notes the lack of standardization in the delivery of this information and lack of machine-readable formats being proposed.

2.1 ML in Sensor Fusion

The first application of ML to sensor fusion applications reviewed in this paper is for an autonomous driving vehicle which should be a resource-limited system. Howard and Lewicki[10] utilized RNNs for sensor fusion in an autonomous driving vehicle, which increased certainty in the recognition, localization, and prediction of surrounding objects. This application used model performance to compare four different RNN-based approaches for sensor data fusion. However, this paper did not mention the resource cost.

Ensemble methods of layered RNNs and Convolutional Neural Networks (CNNs) are increasing the performance and accuracy over basic ML algorithms. The DeepSense framework[11] uses layers of CNNs and RNNs together for sensor data fusion on embedded devices and also measure its performance and system costs. Yao et al.[11] noted an acceptable system trade-off of accuracy for energy on their battery-powered systems, where an acceptable degradation in accuracy consumed less energy. The results of this study provided latency and energy consumption for their inference model on two resource-limited devices, however it is not mentioned if these specifications are provided with the models in something similar to Model Cards. The absence of Model

Cards makes it difficult for anyone to use or analyze the applicability of these pre-trained models on their system without any guidance. The latency of the DeepSense[11] CNN models were on the order of 100 milliseconds on a Nexus 5 mobile phone, while the random forest model took on the order of 300 milliseconds on an Intel Edison and about 50 milliseconds on the Nexus 5.

On the other end of the spectrum from the mobile capable DeepSense model, the DeepAR[12] models are large, ensemble, time-series forecasting algorithms and predictions take on the order of tens of minutes when running on a single p2.xlarge AWS EC2 compute instance with 4 CPUs and 1 GPU. This specific RNN-based model is not used for sensor fusion, but this type of predictive inference may be valuable on aircraft or other resource-limited devices in the future. Therefore system performance, cost, and latency would be good additions to Model Cards. Additional system analysis is needed as DL techniques become more advanced and the aerospace industry starts adopting them. This highlights that the application of DL models for sensor fusion systems are missing a standardized specification for model performance and cost in a machine-readable format.

2.2 Model Cards

This section compiles the previous work done toward the responsible use of AI/ML models in the commercial and government industries. There are many papers that comment on the need for a standardized set of meta-data to ship with any pre-trained AI/ML model to engender trust. As stated in Richards et al.,[13] recent work has outlined the need for increased transparency in AI for data sets,[14–16] models,[9] and services.[13,17] Furthermore, Arnold et al.[17] envision this meta-data in *FactSheets* to contain purpose, performance, safety, security, and provenance information to be completed by AI service providers. Additionally, Richards et al.[13] describe a general methodology for creating FactSheets however, they fail to take the next step to provide a machine-readable format or schema to help automate ingest of the meta-data created by software applications.

Mitchell et al.[9] developed *Model Cards* as a step towards the responsible democratization of ML and related AI technology and increased transparency. They recommend a benchmark evaluation in a variety of conditions, such as different cultural, demographic, or phenotype groups that are relevant to the intended application domains. Mitchell et al.[9] discuss that Model Cards should be standardized and we intend to start this standardization process for RNN-based ML models.

Gebru et al.[15] developed *Datasheets for Datasets* that focus on the characteristics of ML data sets since these fundamentally influence the model's behavior. These Datasheets aim to provide data provenance for the ML community and increase the standardized processes for documenting ML data sets. The *Dataset Nutrition Label*[16] also focuses on data provenance to mitigate concerns of bias, safety, and security. Data provenance has significant safety impacts as it pertains to sensor fusion models used in aircraft. Mitchell et al.[9] also recommends using Datasheets as a complement to Model Cards which is also our goal in this paper.

Blasch et al.[18] provides guidance for ranking/scoring AI/ML models for intended system evaluations in the *Multisource AI Scorecard Table* (MAST) for System Evaluation, however, like the other papers, it does not directly address ML costs for resource-limited systems.

Model Cards and Factsheets overlap and focus on purpose, performance, safety, security, transparency, and responsible use. In this paper, we extend these concepts to include system costs to analyze ML model's impact on resource-limited systems.

3. MODEL CARD EXTENSION

This section describes the current work that builds on previous work accomplished on Model Cards, Datasheets, Nutrition Labels, FactSheets, and MAST by extending them to include quantitative meta-data for system-level analysis of resource-limited systems. We also begin the process of combining these efforts into machine-readable Model Card formats in an effort toward standardization and automation.

3.1 RNN Meta-Data

The Model Card extension in this paper will focus on performance and system cost measurements of RNN-based ML models. This is work done toward the goal of generalizing these quantitative measurements for all ML algorithms eventually. The concepts developed here will be applicable to other ML Model Card extensions, such as other Neural Networks (NN), regression, Random Forest, Bayesian, K-means, Ensemble methods, etc. Table 1 includes the performance and cost measurements suggested for this extension and other key quantities from the previous work we're building on. These quantities are broken into six main categories: RNN properties, data pre-processing properties, model Datasheet, training metrics, bootstrap aggregation (or bagging) metrics, and benchmark metrics.

Table 1: RNN Quantitative Model Card Extension Properties

NAME	DESCRIPTION
RNN algorithm	Name of the RNN algorithm implemented
Model Name	Custom field to name the approach/method
	RNN Properties
Number of Inputs	Number of inputs for RNN
N Outputs	Number of outputs of RNN
Hidden Units	Number of RNN hidden units
RNN Layers	Number of RNN layers
Sequence Length	RNN sequence length (Number of previous time steps used in prediction)
Learning Rate	RNN optimizer learning rate
Optimizer	RNN optimizer
	Data Pre-Processing Properties
Preprocessor Name	Data preprocessor file name
Preprocessor Location	URL or network location available to users
Input Data Frequency	Frequency of input for RNN
Input Variables	List of input variable names used to train the model
Output Variables	List of output variable names used to train the model
	Model Datasheet
Data Set Location	URL or network location available to users
Train Files	List of file names used to train the model
Test Files	List of file names used to test the model
Validation Files	List of file names used to test the model
	Training Metrics
Test MSE$_n$	Normalized Mean Squared Error on test data set
Test MSE	Mean Squared Error on test data set (non-normalize)
Test Error Max	Max error on test data set (w/Units)
Test Error Min	Min error on test data set (w/Units)
Test Error Mean	Mean error on test data set (w/Units)
Test Error Std	Standard deviation of error on test data set (non-normalized)
Error Units	Non-normalized value units
Training MSE	Normalized Mean Squared Error on training data set
Validation MSE	Normalized Mean Squared Error on validation data set
	Bagging Metrics
N Bootstrap Loops	Number of Bootstrap loops for aggregation
File Replacement	Boolean to indicate there was file replacement or not
Test mean MSE	Mean of the Normalized Mean Squared Error on bootstrap test set
Bagging Model Cards	List of model card files used in bootstrap loops
	Benchmark Metrics
Device Name	Custom name for devices used on benchmark
Device Specifications	URL or network location of benchmark device specifications
Benchmark Files	List of files used to generate performance and cost
Inference Latency	Time in microseconds to run inference on benchmark data set
Preprocessor Latency	Preprocessor latency on the benchmark data set
Volatile Memory Usage	Volatile memory (MB) allocated to preprocessor and model
Non-Volatile Memory Usage	Size of Non-Volatile memory (MB) of preprocessor and model
Data Throughput	Amount of data flowing across networks for this model's input
Validation MSE$_n$	Normalized Mean Squared Error on test data set
Validation MSE	Mean Squared Error on the test data set (w/Units)
Validation Error Max	Max error on test data set (w/Units)
Validation Error Min	Min error on test data set (w/Units)
Validation Error Mean	Mean error on test data set (w/Units)
Validation Error Std	Standard deviation of error on the test data set (w/Units)

The Datasheet, pre-processing, training, and bagging categories provide details of the provenance of the raw data as it is processed, trained on, and tested with. This also creates the provenance of the model. These categories together capture the equivalent of what Booth and Ghosh refer to as a DL approach.[19] A DL approach is the end-to-end process from raw data, through data processing, to model training and selection. This DL approach replicates what must happen in real-time sensor fusion models from raw data to model prediction. Providing training, testing, and validation accuracy provides an indication of the model's ability to generalize well which adds additional transparency to these models.

The RNN properties are the hyperparameters and other basic properties used to train the RNN model. The bagging metrics represent best-practices in ML training to remove the stochastic error from noisy data sets when measuring performance.[20] The benchmark section quantifies the model performance and cost on benchmark devices toward the estimation of these values on the targeted system. Ideally, this data would be measured on a device representative of the targeted system.

3.2 Model Card Format

There are numerous machine-readable formats and we will not attempt to cover even a fraction in this paper. However, when making a choice about the format, we recommend that the Model Card format needs to be a common and open standard with a broad range of well-supported libraries in multiple languages. We only explored 2 different machine-readable Model Card formats during this effort for conciseness. The first format embedded the Model Card into the `PyTorch` class object itself. The standard `PyTorch` library is widely available, common, open-source, and `Python` is currently the most used software language.[21] Embedding the Model Card in the object class guarantees the information doesn't get separated from the RNN model itself, however as `Python` libraries and `PyTorch`, specifically, is updated it may break backward compatibility with previously created object classes and render these model cards unreadable. For this reason, we explored defining a common schema file for model cards as a backup and for storage in future data catalogs for ML models. The YAML format is the second machine-readable format used in this study. YAML files are more human-readable than XML, have a wide variety of libraries in different languages, and are commonly used. More specifically, for this study, `Python` will be used as the analysis tool and provides at least two YAML libraries, 1 of which is currently supported. An example Model Card extension YAML file is located in Appendix A.

4. APPROACH

This section describes the sensor fusion system analysis process and how it integrates ML model development into system analysis and design. Section 4.1 describes the case study used to evaluate the implementation of this analysis process. Section 4.2 will describe the SysML model built to provide data for the analysis. Section 4.3 describes the objective function and analyses developed for this case study.

Figure 1 shows the sensor fusion system analysis process being evaluated in this paper. For clarity, this is the SysML process model and is separate from the sensor system SysML model (i.e. SysML system model). This figure shows a *SysML Query* that reads *SysML Model Information* from the *Sensor System SysML Model* and writes the system specifications and data to a SysML system Model Card YAML file (e.g. Appendix B) that will be read by the *Sensor Fusion Model Evaluation* and *Sensor Fusion Model Development* applications. The *Sensor Fusion Model Development* application represents the model selection process of the previous study[19] that generated the RNN models used in this study. For this case study, the Sensor Fusion Model Development reads the SysML Model Card, loads the previously trained RNN models, evaluates the models on a benchmark PC, and creates an RNN Model Card (e.g., Appendix A) for each of the models. After the RNN Model Cards are created, the *Model Card Ingest* application reads in each RNN Model Card and provides the information to the *Sensor Fusion Model Evaluation* analysis application. This application executes an Analysis of Alternatives (AoA) application based on the system objective function, then executes a Monte Carlo analysis application to return the distribution of possible objective function values given a range of random objective variable values. The results of these analyses provide information back to the system engineers that will modify the SysML system model, as the ASoT, which will inform the next iteration of sensor fusion software development.

In a production environment the SysML system model would also provide hardware, software and data specifications to a team of system developers that would build or update the physical sensor system, as shown

in the grey boxes in Fig. 3. Once the physical sensor system has been built and generates data, the data will be loaded into a Data Warehouse where the Sensor Fusion Model Development could begin or continue to iterate on a solution.

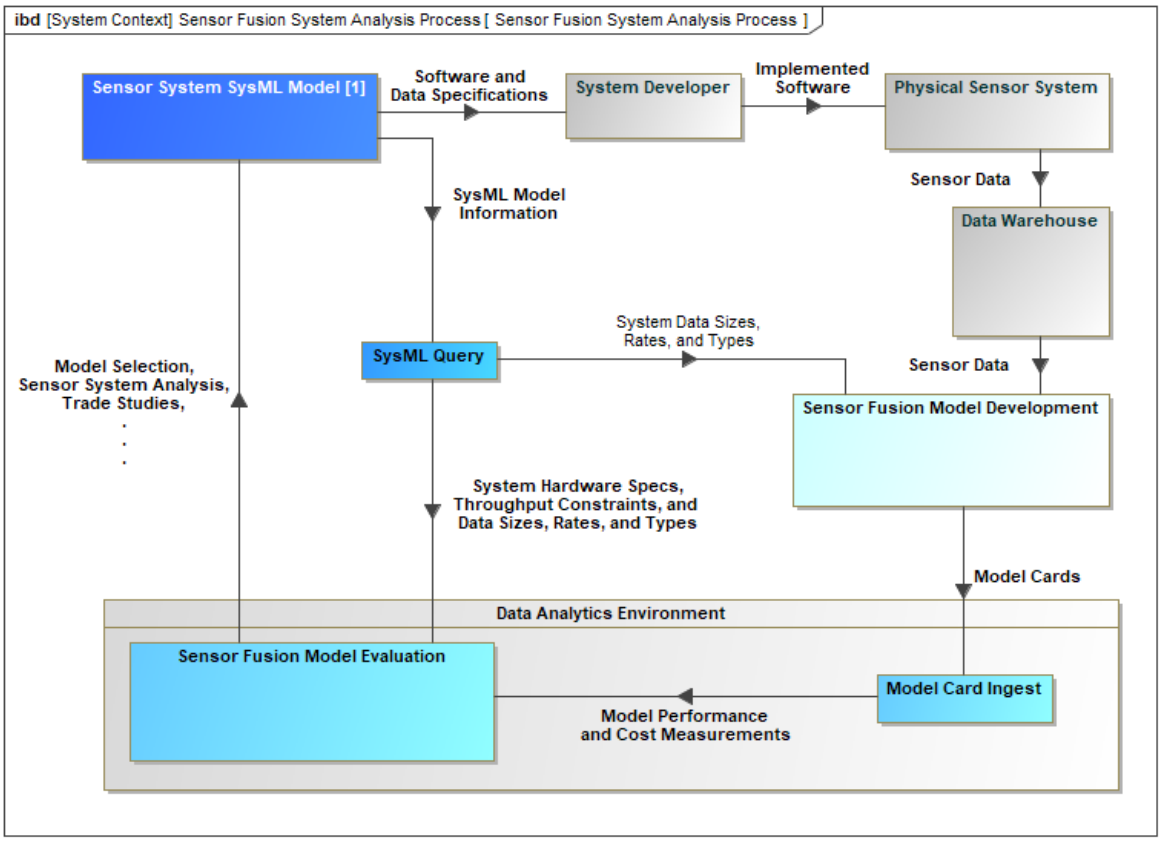


Figure 1: SysML internal block diagram (ibd) of the system analysis process

4.1 Case Study

In this case study an RNN sensor fusion model is used for predicting the aircraft's altitude from a set of 26 multi-modal and multi-sensor inputs. A SysML system model was built to provide the data specifications of the sensor system. SysML system models generally provide more than this, however, this model was limited in scope for the purpose of this case study. We use RNN models pre-trained on a publicly available NASA data set,[22] from a previous sensor fusion study,[19] to evaluate the sensor fusion system analysis process. The NASA data was recorded onboard multiple aircraft with the same sensor suite of a single type of regional jet operating in commercial service. The data files contain detailed aircraft dynamics, system performance and other engineering parameters captured on the data recorder.

The SysML system model provides the data size (bits) and update rate (Hz) of each variable in the system. The RNN Model Card of each pre-trained model lists the input and output variables used. Each pre-trained model has different RNN variables that affect the system cost while running. Table 2 shows the RNN hyperparameters for each model that could also contribute to additional resource costs. For instance, a sequence length of 16 requires 16 of the previous time steps to be stored and used to predict the altitude at the next time step. This could increase the volatile memory resources needed and possibly affect the inference latency. Also, larger RNN hidden units could possibly increase the volatile memory used during model predictions. More RNN model inputs could directly increase the amount of throughput resources consumed by the model. However, throughput is a function of variable sizes and update rates, so the impact to system resources depend on variable size and rate.

Table 2: Top 4 models from the GD-MAGS example[19]

DL Approach	N Inputs	Seq. Length	Hidden Units
Base+RemoveVar	23	16	20
Baseline	26	8	25
Base+Outlier	26	4	25
Base+RemoveVar+Model	24	8	15

The RNN pre-trained models were trained by an LSTM algorithm using an ADAM optimizer implemented in *PyTorch*. This is the baseline for each of the DL approaches originally created to test GD-MAGS methodology[19] on this same NASA data set. Fig. 2 shows the results of the GD-MAGS evaluation. This figure show the bagging average Mean Squared Error (MSE) for the top four pre-trained models selected from the previous study.[19]

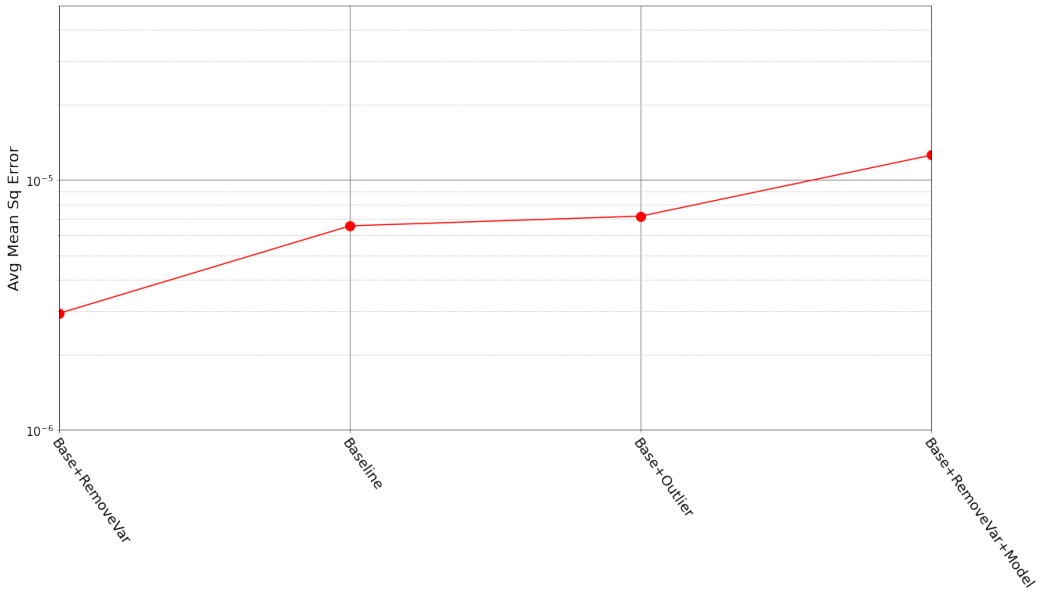


Figure 2: Top four DL approaches from GD-MAGS study[19]

4.2 SysML System Model

We built a generic SysML system model to represent a possible sensor architecture for the NASA data set. A SysML internal block diagram (ibd) of this system is shown in Fig. 3. This system contains two serial buses (BUS1 and BUS2), five sources of sensor and flight data, one pilot display, and one mission computer. The mission computer contains the sensor fusion software and the pilot display shows the pilot the predicted altitude from the sensor fusion model. This figure also labels the data flowing between the computers, sensors and display across the two buses. This SysML system model also defines the composite data signals flowing across these buses shown in Fig. 4. Fig. 4a shows a SysML block definition diagram (bdd) of the decomposition of the BUS 1 and BUS 2 sensor data. Fig. 4b shows a bdd of the data further decomposed into individual data elements from the NASA data set. The data elements are defined by signal frequency and element size in bits. The data information was extracted from this SysML system model to build the SysML Model Card YAML file shown in Appendix B. This SysML Model Card YAML file is used to calculate the RNN Model Card's throughput cost.

4.3 Analysis

Two simple, but representative, system analysis applications were created and executed on this case study to evaluate the effectiveness of our sensor fusion system analysis process. The first analysis is an AoA developed

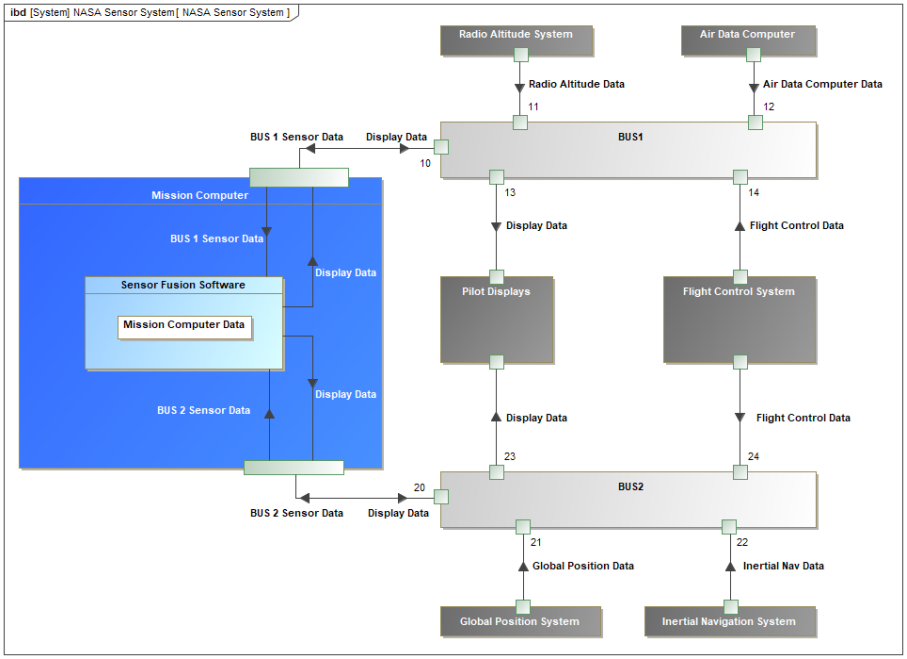


Figure 3: SysML internal block diagram (ibd) of the sensor system

for this case study which presents five different alternative scenarios for an aircraft sensor system. The second is a Monte Carlo analysis that looks at a random distribution of 10,000 different scenarios to see the range of objective function performance for each model on the system.

We created the objective function defined in Eq. (1) for this case study to score each model's system performance and cost. Lower values of this objective function indicate a better score. $BenchMSE$ is the Mean Squared Error of the model on the benchmark data, $GenMSE$ is a measure of the model's ability to generalize well and is calculated by Eq. (2), where MSE_{avg} is the bagging MSE average during the bootstrap testing phase. $Infer$ is the model inference latency, $Preproc$ is the data pre-processing latency, $Throughput$ is the system data throughput required for the sensor fusion model, $vMem$ is the volatile memory consumed during inference, $nvMem$ is the non-volatile memory needed to store the static model. Each of these quantities are multiplied by coefficients C_1 - C_7 to modify the weight each quantity has on the objective function. These 7 coefficients must always add up to 1.

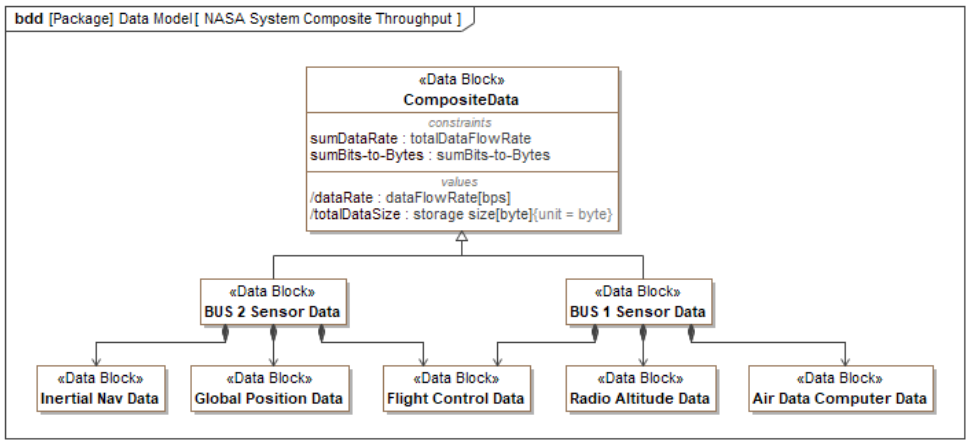
$$f(x) = \sum(C_1 BenchMSE_n, C_2 GenMSE_n, C_3 Infer_n, C_4 Preproc_n, C_5 Throughput_n, C_6 vMem_n, C_7 nvMem_n) \tag{1}$$

$$GenMSE = |MSE_{avg} - BenchMSE| \tag{2}$$

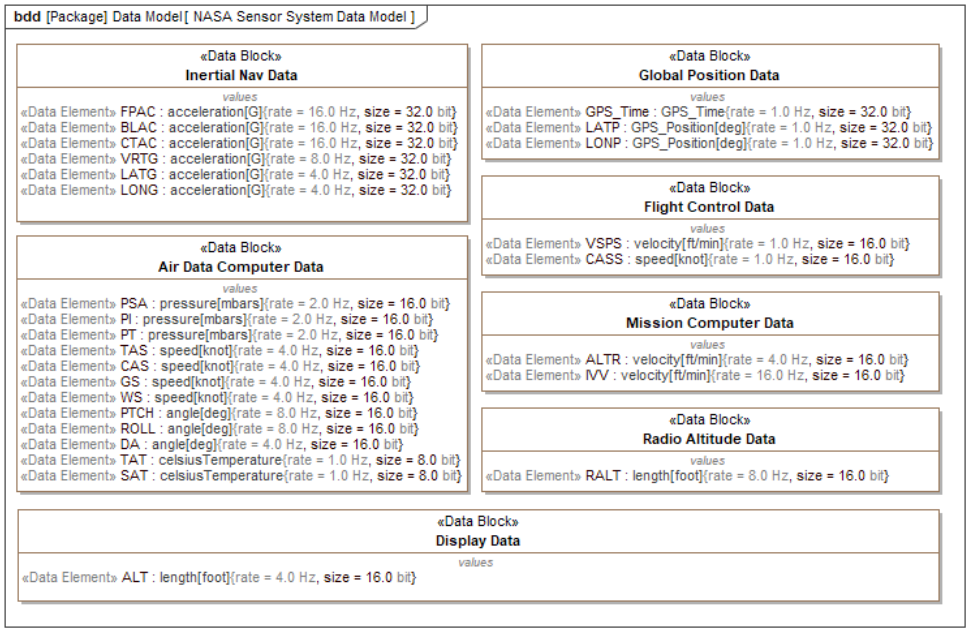
All performance and cost variables were normalized in Eq. (1), as indicated by the $_n$, using the Min-Max Scaling method shown in Eq. (3). Where $min(x)$ is the minimum value across each of the four models, $max(x)$ is the maximum across the four models, x is the original model value and x' is the normalized value on a scale of [0,1]. Normalization reduces number biases of very large or very small objective function variables.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{3}$$

We developed five different scenarios for the AoA and estimated the seven weighted coefficients for each of them. Fig. 5 shows a plot each of the seven weighted coefficients values for each of the five different scenarios. The



(a) Composite data



(b) Data elements

Figure 4: SysML block definition diagrams (bdd) of the data model

first scenario is one in which the aircraft has practically unlimited resources and therefore the MSE and GenMSE are weighted the most heavily, inference and pre-processing latency are still important, but weighted less than MSE. Resource costs were weighted at almost zero. The second scenario is one in which all of the current sensor fusion models meet or exceed the overall system tolerance for accuracy. This means that additional accuracy in the model is shadowed by poor accuracy elsewhere in the system. This scenario weighs the MSE and GenMSE low while weighing inference and pre-processing speed higher since there are still timing requirements to meet. The third, fourth, and fifth scenarios are ones in which the throughput, volatile memory, and non-volatile memory, respectively, are near their constrained limits and are each weighted high.

The Monte Carlo analysis for this case study operates on the same principle as the AoA. However, for each Monte Carlo run a random weight was assigned to each of the seven coefficients and they were normalized to sum to one. The objective function was calculated for this run and this was repeated for 10,000 runs to capture the distribution of performance scores possible for each model.

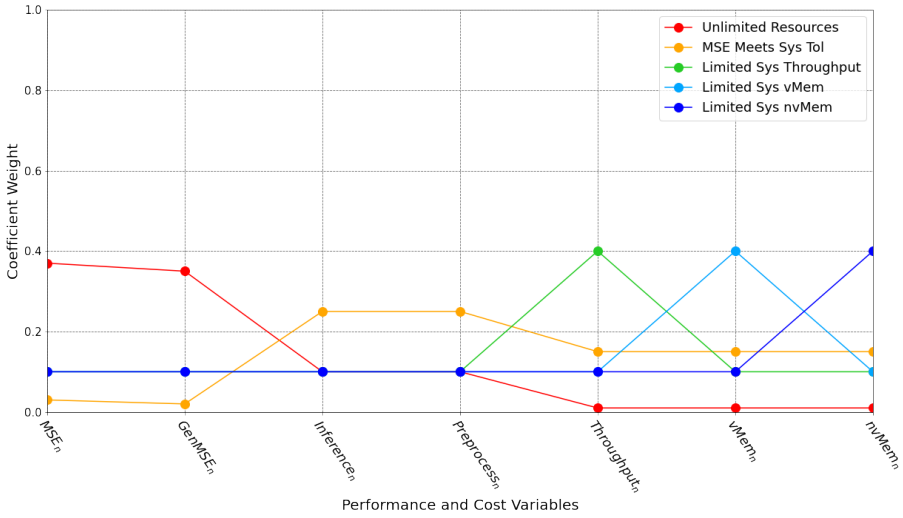


Figure 5: Objective function coefficient weights for AoA

5. RESULTS

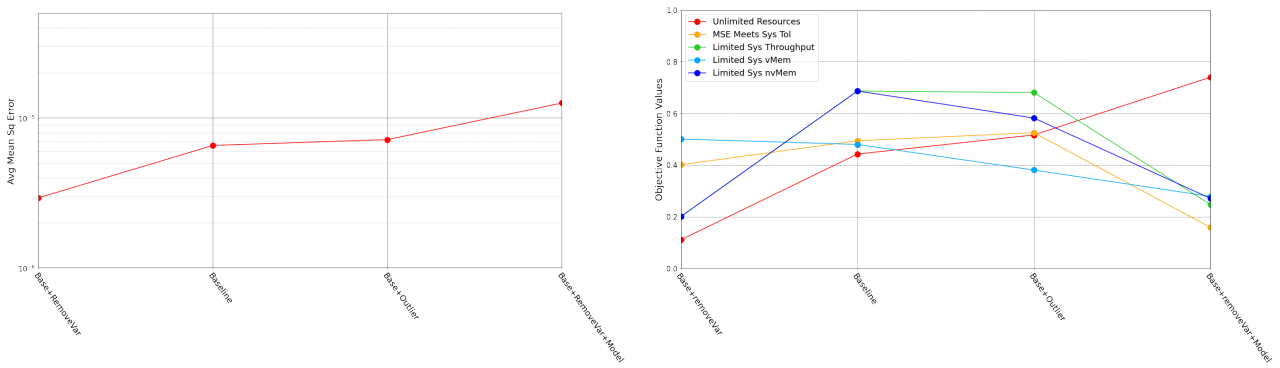
The sensor fusion system analysis process described in Section 4 was implemented and executed in *Python*. First, an example NASA data file was chosen that was not part of the training, testing or validation that happened in the original study[19] that produced the four DL approaches used in this case study. Next, a SysML sensor system model was created and all of the data variables used in each of the DL approaches were modeled and allocated to the appropriate sensors, computers, displays, and serial bus traffic. This SysML data model informed the creation of the SysML Model Card yaml file. Unfortunately, this was done manually since the SysML v2 (beta) API query tool implementation was not complete by the time this study was done.

After the SysML Model Card was defined and the DL approaches were prepped, each RNN sensor fusion model was run through a series of inference test loops to measure and record the quantities needed for the RNN Model Card. We measured each inference event and repeated this 3 times to get average performance and cost measurements for the RNN Model Cards. The previous performance metrics and data used to train, test and validate these DL approaches initially was captured and recorded in the model object class by the previous study.[19] This data was extracted from these object classes and recorded in the RNN Model Card yaml files. Unfortunately, not all the data listed in the current RNN Model Cards were captured in this previous study so some values are blank/*null* in the example file. The lack of data did not affect this study, but a completed Model Card is recommended for future and more complex studies.

The next two sections summarize the results from the AoA and the Monte Carlo analysis using these RNN Model Cards and the SysML Model Card as inputs. These analyses are automated and can produce new results anytime changes to either of these two input files are made.

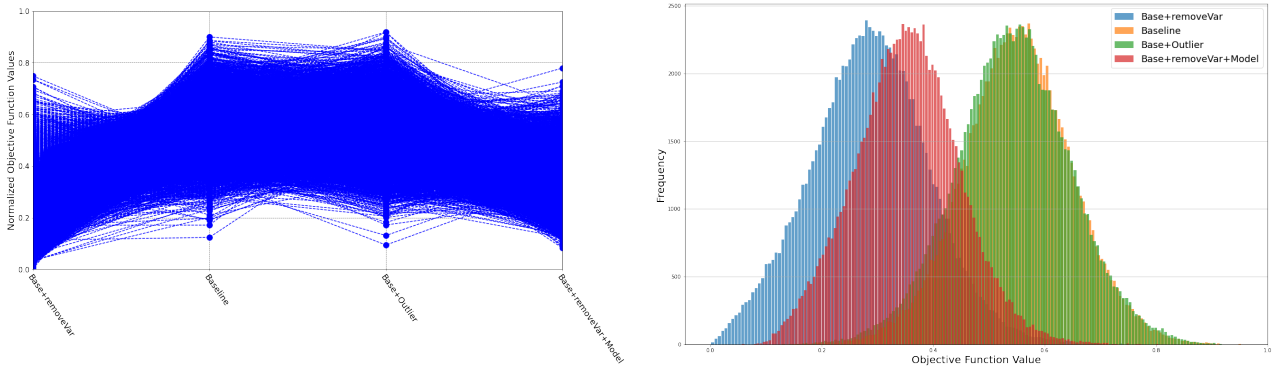
5.1 Analysis of Alternatives

Once the SysML and RNN Model Cards were created they were read in and each performance and cost measurement was normalized using the Min-Max Scaling method listed in Eq. (3). This normalized each value on a scale of 0-1. The set of coefficients for each AoA scenario shown in Fig. 5 and described in section 4.3 were used as inputs to the objective function (Eq. (1)) and the results are shown in Fig. 6b. The objective function is a discrete minimization function, meaning lower values are better. These values represent system costs which should be minimized and also sensor fusion error which should also be minimized. Fig. 6a is the same data as Fig. 2, but was placed next to Fig. 6b to highlight the differences in relative performance when additional performance and cost measurements are considered. In Fig. 6b the *Unlimited Resources* scenario is plotted in red and closely resembles the plot in Fig. 6a that shows the relative performance based solely on MSE. This indicates that the *Base+RemoveVar* DL approach is still the best model for this scenario. Fig. 6b also shows that the



(a) MSE (b) AoA

Figure 6: MSE performance vs AoA relative objective function performance



(a) DL approach vs objective function (b) Monte Carlo objective function distributions

Figure 7: Monte Carlo analysis of objective function coefficients for 10,000 runs

Base+RemoveVar+Model model is the best model for the second scenario where all models meet or surpass the system MSE tolerance. This indicates that when system costs are the driving factors and other areas of your system limit your accuracy, then models that require fewer resources are the best choice. The other scenarios indicate a similar trend that shifts the focus away from performance measures alone for selecting DL approaches for sensor fusion. Fig. 6b also shows a trend that each model has its own distribution of objective function values with the spread of different coefficients used in each scenario. This observation leads to a Monte Carlo analysis to quantify the relative performance distribution of these four models in any random scenario.

5.2 Monte Carlo Analysis

The Monte Carlo analysis begins where the AoA left off. The AoA only considered five different, but specific, scenarios, while this analysis is focused on the statistical quantities of 10,000 different random scenarios. This Monte Carlo analysis varied the weight coefficients randomly for each of these different scenarios and plotted each scenario in Fig. 7a. This figure shows a similar view as Fig. 6b except includes all 10,000 scenarios. Fig. 7b is a plot of the distributions of the objective function values for each of the four models. It is easy to see that the *Base+RemoveVar* model outperforms the other models on average when measured by the objective function. This happens to line up with comparing these four models on MSE performance alone as well. However, the *Base+RemoveVar+Model* model is a close second place on average which does not line up with MSE measurements alone. The other two models have almost identical distributions and may result in very little performance difference between the two if selected for a sensor fusion system.

6. DISCUSSION

The objective function used in this case study may not necessarily be the most ideal equation. The results shown in the Monte Carlo distribution plots show that this equation and distribution of coefficients drastically favors

the *RemoveVar* function out of the four models. This may indicate a bias towards these DL approaches with fewer variables, which end up affecting multiple other objective variables such as throughput, pre-processing, and volatile memory. However, the system analyst must decide what objective variables are important to their specific system and write the objective function accordingly.

The SysML Model Card was not the focus of this paper and the manual creation of the SysML Model Card in this study is not scalable. However, it was a useful machine-readable file that provided the ability to evaluate the rest of the sensor fusion system analysis process. The current SysML version (1.6) has many advantages, however interfaces to SysML are currently tool dependent and not standardized. The updated SysML v2 language *beta* and SysML v2 Application Programming Interface (API) *beta* specifications could be foundational this effort by providing a standard machine-readable interface for system models.[23]

In this paper our intention was to start the Model Card standardization process RNN-based ML sensor fusion models. We realize this may not be the final answer, but we recognized that progress needed to happen in this area and decided to release our progress to date. These additions to Model Cards provide additional and future possibilities toward storing Model Cards in a data library to read in quickly and iterate on analyses. This is the scalability and automation that will greatly improve sensor fusion system analysis.

A Model Card library would make searching for models with certain characteristics for specific uses relatively easy. You could analyze this Model Card library to see where there are gaps in your modeling which could drive development of new and specific models. If developed correctly and for very specific purposes, models could become analogous to software micro services that are optimized to perform very specific tasks. A Model Card catalog of these micro-service ML models would enable a designer or developer to choose from and deploy models with accurate estimates of their performances and cost metrics. These ideas, while somewhat fanciful, need enabling technologies and we believe this extension to Model Cards will provide a step in that direction.

7. CONCLUSION

The results shown in this paper indicate accuracy performance measurements provide an adequate estimation of overall model performance for systems with scalable resources. However, it is easy to see that resource-limited systems must consider both performance and cost measurements when analyzing model performance. This observation reinforces the need to perform system analyses on sensor fusion models. Sensor fusion algorithm development and performance will continue to accelerate and drive the need to provide a scalable and automated system analysis process. This paper has shown that this process benefits from standardized, machine-readable, and quantitative ML Model Cards in addition to machine-readable SysML system model specifications.

The novelty in this paper is the addition of system resource costs in machine readable RNN Model Cards for system analysis of sensor fusion models with the input of machine-readable SysML system specifications. These novel contributions are the enabling technology that supports rapid iterations and incremental improvements of sensor fusion algorithm development, system analysis and design, and sensor fusion model implementation.

Future work will focus on more advanced analyses and optimizations on this open sensor fusion system and more complex systems. This will most likely drive improvements and updates to the proposed RNN Model Card as additional analyses require changes. Future work could also add more ML algorithm specific Model Cards such as CNN or ensemble networks and lead to a generic ML Model Card schema where each ML Model Card overlaps. The current RNN Model Card yaml example and future improvements are on GitHub[24] to encourage collaboration and provide iterative updates to the community. Additional Model Card improvements will also be directed at increased statistical analysis of the training, testing, and validation data sets to quantify the differences between the training and the testing and validation data sets. At a minimum, this should include statistical quantities comparing these datasets such as Z-Test, T-Test and Analysis of Variants. SysML v2 and its API will also be the focus of future work to better automate this system analysis and design process.

ACKNOWLEDGMENTS

This work was supported in part by funding from NSF under Award Number OAC 1931363, the US Air Force STEM+M program, the AFSC/309^{th} Software Engineering Group, and Nexus Digital Engineering LLC.

REFERENCES

[1] White, F. E., "Data Fusion Lexicon," *The DATA FUSION SUBPANEL of the Joint Directors of Laboratories, Technical Panel for C3* **15**(0704) (1991).

[2] Dasarathy, B. V., "Sensor fusion potential exploitation-innovative architectures and illustrative applications," *Proceedings of the IEEE* **85**(1), 24–38 (1997).

[3] Bracio, B. R., Horn, W., and Moeller, D. P., "Sensor fusion in biomedical systems," *Annual International Conference of the IEEE Engineering in Medicine and Biology - Proceedings* **3**(C), 1387–1390 (1997).

[4] Steinberg, A. N., Bowman, C. L., and White, F. E., "Revisions to the JDL data fusion model," *Sensor Fusion: Architectures, Algorithms, and Applications III* **3719**(March 1999), 430 (1999).

[5] Makridakis, S., Spiliotis, E., and Assimakopoulos, V., "The M5 accuracy competition: Results, findings and conclusions," *Int J Forecast* (October), 1–44 (2020).

[6] Brena, R. F., Aguileta, A. A., Trejo, L. A., Molino-Minero-Re, E., and Mayora, O., "Choosing the best sensor fusion method: A machine-learning approach," *Sensors (Switzerland)* **20**(8), 1–22 (2020).

[7] Henderson, K. and Salado, A., "Value and benefits of model-based systems engineering (MBSE): Evidence from the literature," *Systems Engineering* **24**, 51–66 (jan 2021).

[8] Carroll, E. R. and Malins, R. J., "Systematic Literature Review: How is Model-Based Systems Engineering Justified?," tech. rep., Sandia National Laboratories (2016).

[9] Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., and Gebru, T., "Model cards for model reporting," *FAT* 2019 - Proceedings of the 2019 Conference on Fairness, Accountability, and Transparency* (Figure 2), 220–229 (2019).

[10] Howard, S. M. and Lewicki, M. S., *Deep Learning for Sensor Fusion*, PhD thesis, Case Western Reserve University (2017).

[11] Yao, S., Hu, S., Zhao, Y., Zhang, A., and Abdelzaher, T., "DeepSense: A unified deep learning framework for time-series mobile sensing data processing," *26th International World Wide Web Conference, WWW 2017* , 351–360 (2017).

[12] Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T., "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting* **36**(3), 1181–1191 (2020).

[13] Richards, J., Piorkowski, D., Hind, M., Houde, S., and Mojsilović, A., "A Methodology for Creating AI FactSheets," (2020).

[14] Bender, E. M., "Data Statements for Natural Language Processing : Toward Mitigating System Bias and Enabling Better Science," **6**, 587–604 (2018).

[15] Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Iii, H. D., and Crawford, K., "Datasheets for datasets," *Communications of the ACM* **64**(12), 86–92 (2021).

[16] Chmielinski, K. S., Newman, S., Taylor, M., Joseph, J., Thomas, K., Yurkofsky, J., and Qiu, Y. C., "The Dataset Nutrition Label (2nd Gen): Leveraging Context to Mitigate Harms in Artificial Intelligence," (2022).

[17] Arnold, M., Piorkowski, D., Reimer, D., Richards, J., Tsay, J., Varshney, K. R., Bellamy, R. K., Hind, M., Houde, S., Mehta, S., Mojsilovic, A., Nair, R., Ramamurthy, K. N., and Olteanu, A., "FactSheets: Increasing trust in AI services through supplier's declarations of conformity," *IBM Journal of Research and Development* **63**(4-5) (2019).

[18] Blasch, E., Sung, J., and Nguyen, T., "Multisource AI Scorecard Table for System Evaluation," *Association for the Advancement of Artificial Intelligence (AAAI)* (2021).

[19] Booth, T. M. and Ghosh, S., "A Gradient Descent Multi-Algorithm Grid Search Optimization of Deep Learning for Sensor Fusion," *IEEE System Conference* (2023).

[20] Goodfellow, I., Bengio, Y., and Courville, A., [*Deep Learning*], MIT Press (2016). Available at: http://www.deeplearningbook.org.

[21] TIOBE, "TIOBE Index 2023." Available at: www.tiobe.com.

[22] NASA, "DASH Link," (2018). Available at: https://c3.ndc.nasa.gov/dashlink/projects/85.

[23] Bajaj, M., Friedenthal, S., and Seidewitz, E., "Systems Modeling Support for Digital Engineering," *INCOSE Insight* , 19–24 (2022).

[24] Booth, T. M., "GitHub Model Card Schema," (2023). Available at https://github.com/boothtm/ML-Model-Card-Schemas.

APPENDIX A. RNN MODEL CARD EXTENSION YAML EXAMPLE

```yaml
---
rnn-based_model_card:
  name: Base+removeVar
  RNN_algorithm: LSTM
  model_metrics:
    RNN_properties: {epochs: 3500, hidden_units: 20, learning_rate: 0.0007, n_inputs: 23,
      n_outputs: 1, sequence_length: 16}
    input_data_frequency: 16
    input_variable_names: [time, RALT, PSA, PI, PT, ALTR, IVV, VSPS, FPAC, BLAC, CTAC,
      TAS, CAS, GS, CASS, WS, PTCH, ROLL, DA, TAT, SAT, LATP, LONP]
    model_datasheet:
      testing_files: [687200107241524.parquet, 687200104301119.parquet,
        687200104261527.parquet, 687200104181334.parquet, 687200107301239.parquet,
        687200104170717.parquet, 687200107170234.parquet]
      testing_files_location: /mnt/SPIE/ml-model-cards/data
      training_files: [687200107101600.parquet, 687200107060930.parquet,
        687200107192334.parquet, 687200107150546.parquet, 687200107311025.parquet,
        687200107181544.parquet, 687200104162039.parquet, 687200104202027.parquet,
        687200107251652.parquet, 687200104181127.parquet, 687200107251002.parquet,
        687200107122323.parquet, 687200107171131.parquet]

      training_files_location: /mnt/SPIE/ml-model-cards/data
      validation_files: [687200107101623.parquet, 687200107060931.parquet,
        687200107181541.parquet, 687200104162032.parquet, 687200104202023.parquet,
        687200107192330.parquet, 687200107150549.parquet]
      validation_files_location: /mnt/SPIE/ml-model-cards/data
    output_variable_names: [ALT]
    performance:
      test: {MSE: 389.1024435718944, MSE_norm: 1.0885067922572489e-06,
        mean_error: 8.00731361488409, min_error: 0.0, std_error: 29.48358127688793,
        max_error: 421.5643005371094, units: ft}
      train: {MSE: 15.48635814304236, MSE_norm: 3.140855255878705e-07,
        mean_error: 13.48635814304236, min_error: 0.00018310546875,
        std_error: 14.29056638518885, max_error: 89.2213134765625, units: ft}
      validation: {MSE: 386.1021435718944, MSE_norm: 4.5026759064453724e-07,
        mean_error: 13.48635814304236, min_error: 0.00018310546875,
        std_error: 14.29056638518885, max_error: 160.2213134765625, units: ft}
  bagging_metrics:
    model_datasheet: {dataset: null}
    n_bootstrap_loops: 3
    performance:
      testing: {max_MSE: null, mean_MSE: 2.933555492745654e-06, min_MSE: null,
        std_MSE: null}
      training: {max_MSE: null, mean_MSE: 1e-07, min_MSE: null, std_MSE: null}
      validation: {max_MSE: null, mean_MSE: null, min_MSE: null, std_MSE: null}
    properties: {N_bootstrap_datasets: 3, replacement: false}
  benchmark:
    dataset: {location: /mnt/SPIE/ml-model-cards/data, name: example_flight.parquet}
    device1:
      device_ID: 1
      name: CovidPC
```

```
      RAM:
        CAS_latency: 16
        first_word_latency: {speed: 8.889, units: ns}
        size: 32
        speed: 3600
        units: GB
        voltage: {units: V, value: 1.35}
      processor:
        L1Cache_Data: {units: kB, value: 192}
        L1Cache_Instruction: {units: kB, value: 192}
        L2Cache: {units: kB, value: 3072}
        L3Cache: {units: MB, value: 32}
        TDP: {units: W, value: 65}
        core_clock: {units: GHz, value: 3.6}
        name: AMD Ryzen 5 3600
        ncores: 6
        type: CPU
    performance:
      data_processing_latency: {units: seconds, value: 0.2591249942779541}
      model_inference_latency: {units: seconds, value: 0.401350736618042}
      nonvolatile_memory_used: {units: MB, value: 7.4619035720825195}
      throughput: {units: Kbps, value: 2.8}
      validation: {MSE: 386.1021435718944, MSE_norm: 4.5026759064453724e-07,
        mean_error: 13.48635814304236, min_error: 0.00018310546875,
        std_error: 14.29056638518885, max_error: 160.2213134765625, units: ft}
      volatile_memory_used: {units: MB, value: 2084.9192708333335}"
...
```

APPENDIX B. SYSML MODEL CARD YAML EXAMPLE

```yaml
---
sysml_model_card :
  name : 'NASA Sensor System SysML model1'
  file : 'simple_nasa_avionics_sensor_model.mdzip'
  file_location : '/mnt/SPIE/sysml-model-cards-for-dl-sensor-fusion'
  sysml_version : '1.6'
  data_model :
    data :
      ALT  : {size : 16, rate : 4}
      time : {size : 32, rate : 1}
      FPAC : {size : 32, rate : 16}
      BLAC : {size : 32, rate : 16}
      CTAC : {size : 32, rate : 16}
      VRTG : {size : 32, rate : 8}
      LATG : {size : 32, rate : 4}
      LONG : {size : 32, rate : 4}
      RALT : {size : 16, rate : 8}
      ALTR : {size : 16, rate : 4}
      IVV  : {size : 16, rate : 16}
      VSPS : {size : 16, rate : 1}
      PSA  : {size : 16, rate : 2}
      PI   : {size : 16, rate : 2}
      PT   : {size : 16, rate : 2}
      TAS  : {size : 16, rate : 4}
      CAS  : {size : 16, rate : 4}
      GS   : {size : 16, rate : 4}
      WS   : {size : 16, rate : 4}
      CASS : {size : 16, rate : 1}
      PTCH : {size : 16, rate : 8}
      ROLL : {size : 16, rate : 8}
      DA   : {size : 16, rate : 4}
      TAT  : {size : 8,  rate : 1}
      SAT  : {size : 8,  rate : 1}
      LATP : {size : 32, rate : 1}
      LONP : {size : 32, rate : 1}
    networks :
      BUS1 :
        constraints :
          throughput : {datarate : 1, units : 'Mbps'}
        data : [RALT, ALTR, IVV, VSPS, PSA, PI, PT, TAS, CAS, GS, WS, CASS, PTCH, ROLL,
          DA, TAT, SAT, ALT]
      BUS2 :
        constraints :
          throughput : {datarate : 1, units : 'Mbps'}
        data : [time, FPAC, BLAC, CTAC, VRTG, LATG, LONG, VSPS, CASS, LATP, LONP, ALT]
...
```