# Evaluation of NVIDIA Jetson System for Vibration HUMS

Christopher J. Valant[1], Sean P. McConky[2], Michael G. Thurston[3], Patrick Horney[4], Allen Jones[5], and Nenad G. Nenadic[6]

[1,2,3,6] *Rochester Institute of Technology, Rochester, NY, 14623, USA*
*christopher.valant@gmail.com spm9605@rit.edu mgtasp@rit.edu nxnasp@rit.edu*

[4] *NAVAIR, Cherry Point, NC, 28533, USA*
*patrick.horney@navy.mil*
[5] *NAWCAD, Patuxent River, MD, 20670, USA*
*allen.jones2@navy.mil*

**ABSTRACT**

An NVIDIA Jetson graphical processing unit (GPU) was evaluated for utilization in a health and usage monitoring system by computing vibration-based condition indicators (CIs) and evaluating autoencoders for anomaly detectors. The GPU performance was subsequently compared to a central processing unit (CPU) performing the the same computations, included signal preprocessing. Two distinct cases of interest were considered with neural network autoencoders: model evaluation and model adaptation with limited training. The experiments found that computations associated with signal preprocessing and condition indicators performed faster on the CPU, but neural network model evaluation and adaptation were faster on the GPU. Utilizing the GPU capability of the Jetson Nano, it was estimated that 42 accelerometer signals could be evaluated through autoencoders in real time, when data was processed in one second batches.

## 1. INTRODUCTION

Emergence of lower cost *micro electromechanical systems* (MEMS) accelerometers (specifically Analog Devices' ADXL 1001/ADXL 1002) made vibration monitoring more affordable by reducing not only the sensor cost, compared to traditional piezoelectric accelerometers, but also by reducing the overall cost of the data acquisition channel because their low impedance voltage output obviates the charge amplifier and its accompanied cost. The reduced cost of vibration sensing opens the potential to expand their deployment to *health and usage monitoring systems* (HUMS) of ground vehicles. With sampling rates at 10 kHz or more, the vibration data volumes are considerably larger than that of pressure or temperature sensors, thus requiring more on-board computational power (or computational power on the *edge*). The NVIDIA Jetson family of hardware, specifically designed for efficiency and edge computing, has been adopted into multiple commercial products, ranging from tablets and phones to vehicle vision systems. Computational performance is measured in floating point operations per second (FLOPS). Current versions of the Jetson have stated computational performance of 472 gigaFLOPS ($10^9$) for the 128 CUDA core Jetson Nano to approximately 5.3 teraFLOPS ($10^{12}$) for the Jetson AGX Orin 64GB.

NVIDIA has put significant effort into developing hardware specifically for AI-powered self-driving vehicles, developing the NVIDIA Drive Platform for this specific market segment, even partnering with major automotive parts supplier Bosch. Self-driving vehicles are typically equipped with a set of sensors that include multiple cameras, LIDAR, acoustic/ultrasonic sensors, and radar systems. The sensor data is fused to provide the vehicle with a complete, detailed representation of the surrounding environment. Computer vision algorithms process video frames to identify key objects and structures. This picture of the environment is also merged with GPS data, requiring significant processing power on a continuous basis, to allow the vehicle to follow a predetermined path to its destination.

As significant research efforts exist in applying neural networks and deep learning to CBM, the next logical step is to apply mobile GPU enabled SoC computing platforms into CBM applications. The use case considered here was vibration monitoring, included computing vibration-based *condition indicators* (CIs) (Lebold, McClintic, Campbell, Byington, & Maynard, 2000; Samuel & Pines, 2005; Sait & Sharaf-Eldeen, 2011; Sharma & Parey, 2016). Specifically, in this study the following *classical* CIs were computed: RMS, Kurtosis, Crest factor (Swansson, 1980),Energy ratio, FM4 (Stewart, 1977), M6A, M8A (Martin, 1989), NA4 (Zakrajsek, Townsend, & Decker, 1993), NA4*, and NP4 (Polyshchuk,

Choy, & Braun, 2002).

In addition to these *classical* CIs, autoencoder neural networks were trained. With the emergence of deep learning, with powerful open-source frameworks TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2019), the autoencoders have become the most efficient anomaly detectors (Eklund, 2018). However, a successful implementation and deployment of an autoencoder in *prognostics health monitoring* (PHM) predates the emergence of *deep learning* (Japkowicz, Myers, & Gluck, 1995). More recently, autoencoder-based anomaly detectors have been shown to have considerable promise because, unlike classical classifiers that demand balanced datasets, their training can be based on data associated with normal operation, which comes in abundance, as opposed to data associated with failures, which is difficult to come by (Yan & Yu, 2015). Anomaly detection is a first and lowest level of PHM capability, followed by diagnostics and prognostics (Vachtsevanos, Lewis, Roemer, Hess, & Wu, 2006; Goebel et al., 2017). The addition of low cost MEMS sensors and autoencoders may open the door to lower cost HUMS capabilities that have been limited in ground vehicle applications (Heine & Barker, 2007; Rajesh & Francis, 2012).

## 2. USE CASE AND TEST SYSTEM SETUP

As stated in Section 1, lower-cost accelerometers enable expansion of vibration monitoring to a wider-range of HUMS, including HUMS for ground vehicles. High sampling rates of accelerometer data immediately introduces the question of where computations should take place, at the edge or in the centralized location, with trade-offs between the cost of edge computations and cost of bandwidth associated with transfer of large volume of data.

These considerations promote a lower cost vibration monitoring system as an important and highly-relevant use case for PHM systems. Figure 1 shows a block diagram of a legacy HUMS, which employs vibration data.

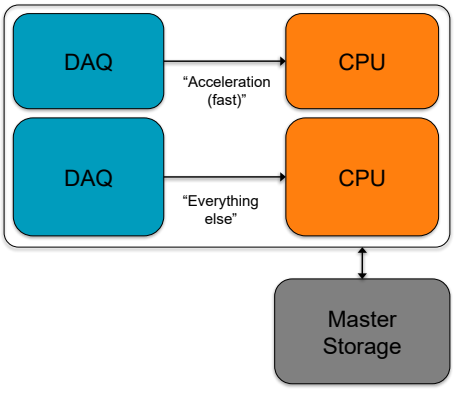



Figure 1. Legacy HUMS

It features two *data acquisition system* (DAQ)-CPU pairs: one for low sampling rate contextual data (e.g. temperature, speed, torque) and the other for high sampling rate vibration data. The master storage exists at the processing units, as the bandwidth requirements of high sample vibration data cannot be met by off-board communication systems. In this investigation we used a modified HUMS, illustrated with the block-diagram in Figure 2. The addition of a GPU allows for complex computations, e.g. autoencoders, to be performed at the edge; decreasing the vehicle based storage and bandwidth requirements.

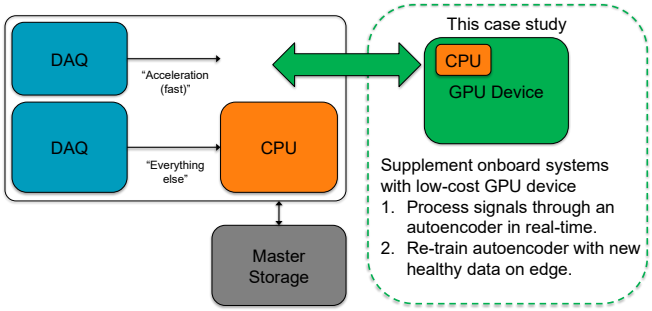



Figure 2. Modified HUMS, equipped with GPU

While this modified HUMS is still a prototype (it has not been implemented on an asset), the envisioned main outputs, that would be stored in the master storage, are: CIs, computed once per second, including data-driven CIs such as *mean average error* (MAE) of the autoencoder's error; a single TSA sample (4,096 points) for each instance of a predefined steady-state operation (specified torque and speed), and an associated 10 seconds of raw vibration data, with decimation of $10\times$, to yield 100,000 points for 10 seconds.

Two types of computations were employed: computing classical vibration-based CIs and data-driven CIs. Classical CIs were computed using a Python toolbox for gearbox computations[1]. Data-driven CIs were computed as *mean-squared error* (MSE) of the autoencoder, applied to *time synchronous averaged* (TSA) vibration data. The TSA data was computed using a tachometer to average over multiple shaft rotations, effectively converting time-domain data into angle domain data in the range $0 \leq \theta < 2\pi$. An accelerometer vector, $\boldsymbol{a}(t)$, with length of $10^5$, associated with one second of operation, was compressed into the TSA vector $\boldsymbol{x}_{TSA}(\theta)$ with a length of $2^{12}$ points. Twenty-four revolutions were used for computing the average, where the remaining points ($10^5$-$24\times2^{12} \simeq 1.7 \times 10^3$ per second) were discarded. More details on TSA in general can be found in (Bechhoefer & Kingsley, 2009).

Table 1 lists three GPU cards, with their specifications in the descending order of their computational power. The V100 and Titan V cards have an additional 640 *Tensor Cores* which

[1] manuscript in preparation

Table 1. GPU specifications for three cards.

| Hardware | GPU Card | | Memory | | Power |
|---|---|---|---|---|---|
| **Model** | **Cores** | **Arch.** | **(GB)** | **(GB/s)** | **(W)** |
| Tesla V100 | 5120 | Volta | 16 | 900 | 250 |
| GTX 1070 | 1920 | Pascal | 8 | 256 | 150 |
| Jetson Nano | 128 | Maxwell | 4 | 25.6 | 10 |

handle 4x4 matrices, which are not available on the low cost, entry level Jetson Nano.

A Jetson Nano Developer Kit was selected for analysis as it is a fully functional system that includes a CPU, GPU and memory. Additionally, it is the lowest cost option and therefore, if successful would be the entry level of computational performance. On the other hand, the V100 and GTX 1070 are more likely to be utilized in server or desktop applications, especially given their power requirements. For the purpose of evaluating GPU performance, the data acquisition was simulated with previously collected data from a helicopter gear test fixture. Each data file consists of 1 second of vibration data collected at 104.167 kHz. The data is read in and processed computationally according to Figure 3.


Figure 3. The flow of process computations

Two types of autoencoder models were considered: one based on fully-connected layers (Figure 4) and one based on *convolutional neural network* (CNN) layers (Figure 5).

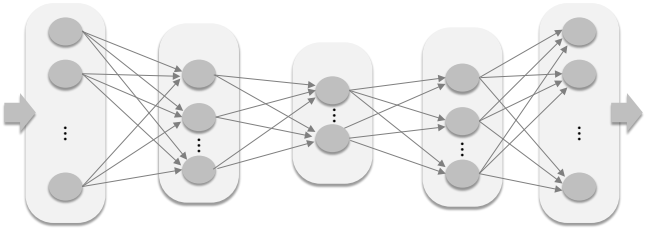


Figure 4. Autoencoder with fully-connected layers

## 3. CPU/GPU CI AND AUTOENCODER COMPUTATIONAL PERFORMANCE

The performance testing was broken down into two simulated runs of real-time edge processing. First, the computational performance for calculating the 16 selected vibration CIs, including the preprocessing step of computing TSA from raw vibration, and second, the evaluation performance of the two autoencoders. The first part was run for two lengths of time; 900 files representing 15 minutes of data and 54,000 files representing 15 hours of data. The shorter run was chosen to represent a short but realistic time period of data acquisition. The longer processing run was performed to ensure that any impacts from background processes of the system/OS are included in the results. Several detailed comparisons are provided in this paper including CI computation on the CPU vs. GPU, autoencoder evaluation on the CPU vs. GPU, and several pipeline figures comparing the end-to-end process with and without the utilization of the GPU.

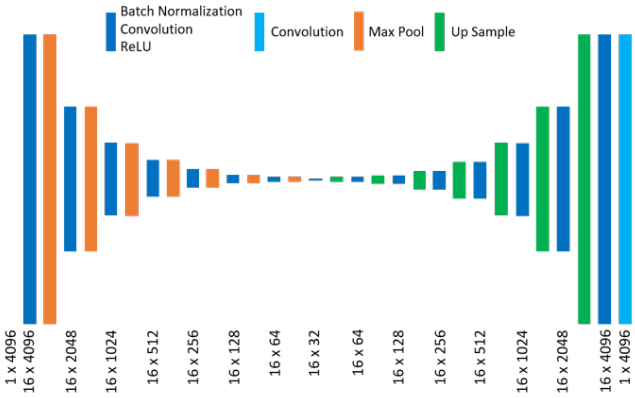



Figure 5. Autoencoder with CNN layers

Table 2. CPU vs. GPU computations for different hardware

| Card | No. Files | CPU (s) | GPU (s) |
|---|---|---|---|
| Tesla V100 | 900 | 3.95 | 19.81 |
| Tesla V100 | 54000 | 212.43 | 1150.27 |
| GeForce GTX 1070 | 900 | 3.90 | 16.61 |
| GeForce GTX 1070 | 54000 | 238.28 | 870.11 |
| Jeston Nano | 900 | 14.08 | 93.15 |
| Jetson Nano | 54000 | 813.87 | 5043.95 |

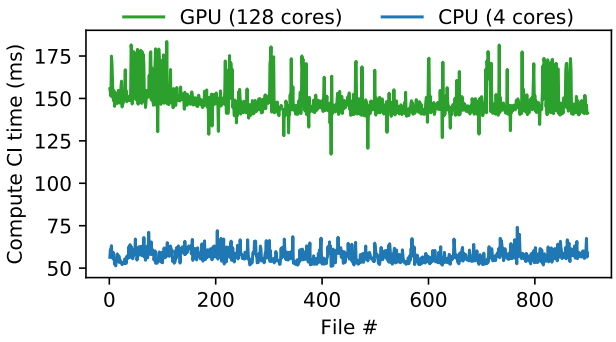



Figure 6. Computational run time for a simulation of 900 seconds worth of accelerometer data processed into TSA on the Jetson Nano.

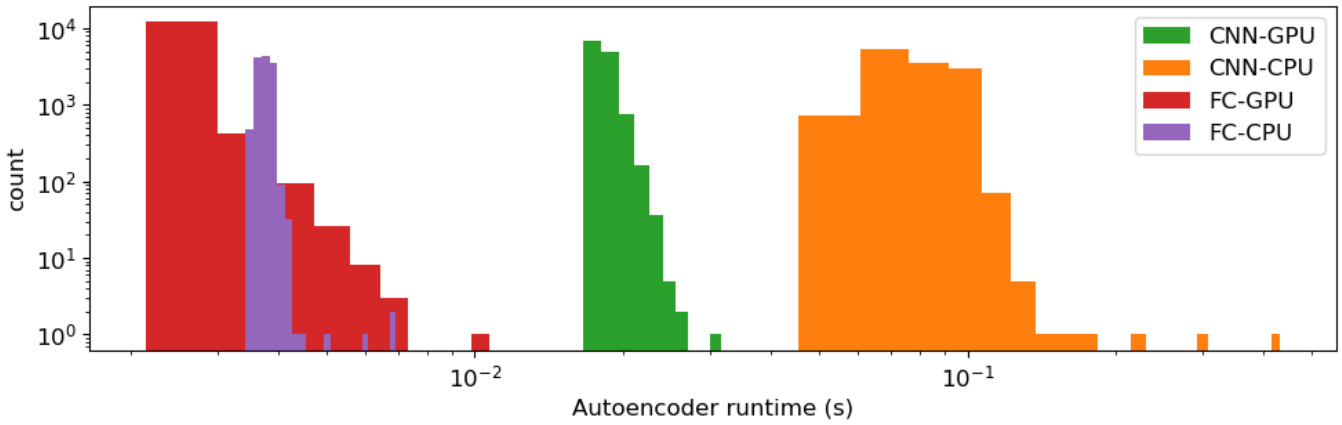



Figure 7. Summary of AE performance computations

Table 2 shows the computational time necessary for calculating the vibration CIs on the three different GPU devices. This is largely attributed to the extra time needed for data transfer between CPU and GPU memory outweighing any benefits from computing on the GPU. Figure 6 shows the difference in computational time (in milliseconds) for each file processed on the Jetson Nano CPU vs. the GPU.

However, evaluation of autoencoder models were considerably faster when computed on the GPU as summarized in Figure 7, which shows four distributions, based on computation location and type of autoencoder, from the 54,000 second data runs. It should be noted that all of these distributions appear to be right skewed. Therefore, in our calculations of processing time, we utilized average processing times to avoid outliers. Although outliers may slow performance during a single processing cycle, over the duration, the average time is more representative of actual performance.

The performance across the end-to-end pipeline, that includes TSA computations, computing classical CIs, and CNN autoencoder is summarized in Figure 8. The entire pipeline requires 43.5 ms to compute the 16 CIs on the CPU and the CNN on the GPU per 1 second of accelerometer data. Therefore, a single Jetson board is capable of processing 22 accelerometers of data. Whereas, a similar pipeline utilizing only the CPU would require 101.4 ms, allowing for only 10 accelerometers. In the 22 accelerometer case, they are processed in sequence with total run-time reaching just under 1 second (0.957). Processing multiple accelerometer signals simultaneously was not part of this study. This would require splitting already parallel GPU operations among different running accelerometer signals and would likely increase the processing time due to overhead on the GPU memory transfers. This conceptual design did not have specific requirements for the number of accelerometers; instead, we wanted to determine how many could be processed with a single unit within the required buffer time interval, in this case, 1 second between data acquisitions. Note, however, that some existing helicopter HUMS systems can collect data from a large number of accelerometers: e.g. the IMD-HUM main processing units are capable of collecting up to 46 accelerometers, although only 32 accelerometers and index sensors are utilized for CH-53E IMD-HUMS, and 22 accelerometers for SH-60B (2 engines). (Duke, Bailer, Thitchener, & Gebauer, 2001). However, legacy systems do not include autoencoder processing.

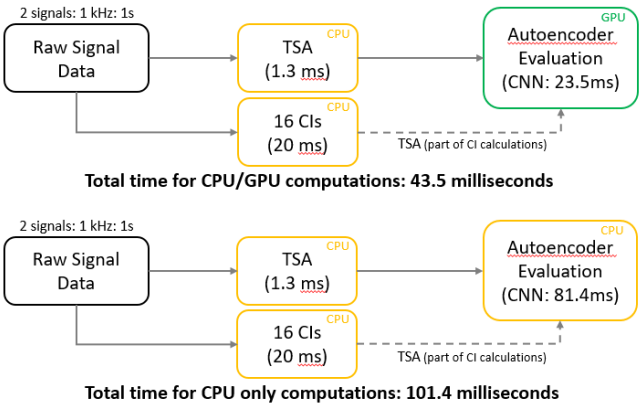
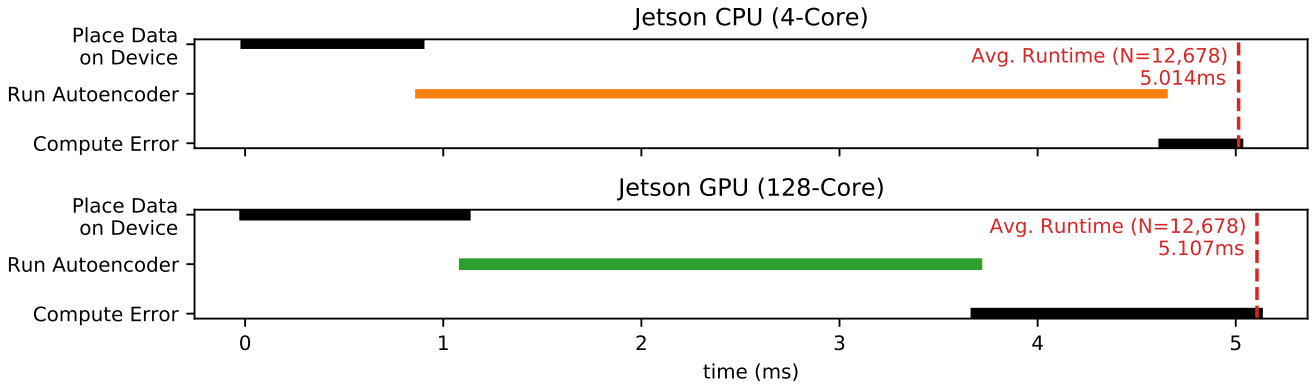


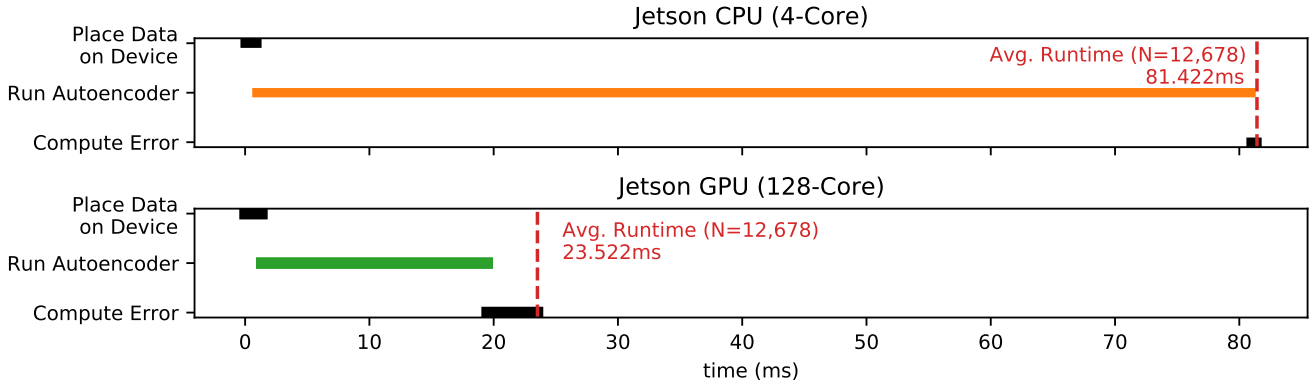

Figure 8. Computation performance summary across the two pipeline options

A more detailed view on timing of specific operations on NVIDIA's Jetson CPU and GPU are provided in Figure 9. Three main steps were identified: 1) placing data on the device, 2) running autoencoder, and 3) reading data from the device and computing the error metric associated with the autoencoder's output. Figure 9a and Figure 9b compare 4-core CPU to 128-core GPU computations for the case of fully-connected and CNN autoencoders, respectively.

In the case of fully-connected autoencoder (Figure 9a), autoencoder evaluation is faster on the GPU than on the CPU, but overall process actually is computed slightly slower on




(a)




(b)

Figure 9. Detailed CPU vs. GPU comparison for (a) AE with fully-connected layers (b) AE with CNN layers

the GPU than on the CPU, because both placing of the data on the device and reading data from the device during error computation were slower on the GPU.

By contrast, the CNN autoencoder (Figure 9b) model evaluation is vastly faster on the GPU than on the CPU, resulting in overall $\simeq$3.5$\times$ speed-up on the GPU. Comparing the time scales of Figure 9a and Figure 9b, we see that convolutions of the CNN autoencoder were considerably slower than matrix computations of the fully-connected autoencoder: $\simeq$6$\times$ slower on the GPU, and more than 20$\times$ slower on the CPU.

## 4. MODEL ADAPTATION AT THE EDGE

Most edge computing is limited to model evaluation for two reasons: 1) model training is generally much more computationally intense than model evaluation and 2) there are difficulties associated with arranging and automating on-board training. However, there are some use cases where on-board training may be very useful. One such use case is *model adaptation* where a pre-trained model is adapted to a specific asset; weight updates to the model based on new data, where parameters associated with training are already known. After a significant maintenance event, such as replacing fuel injectors in an engine with vibration monitoring, the model can re-learn the new healthy state which may be represented differently than the previously known healthy state.

To demonstrate model adaptation utilizing the previously sampled vibration data, a second data set was collected from a different gear on the helicopter gear test fixture. Instead of training a new model, the previous model (Model A) is retrained, or adapted, with each new data file. The adaptation process continues across 7,200 files of data, effectively re-training on 2 hours of data from the new gear. The average training time per step (1 second of data) was 70.16 ms, as shown in Figure 10. Similar to the evaluation pipeline, retraining requires the calculation of the TSA by the CPU, which, as stated above, averages 24 rotations to compute $2^1 2$ points from $10^5$ points of raw vibration for every second of operation. The entire pipeline for retraining (see Figure 11) is approximately 71.4 ms, allowing for retraining of up to 13 models at a time. The adapted model will be referred to as Model B.

The performance of Model A on baseline and failure prop-

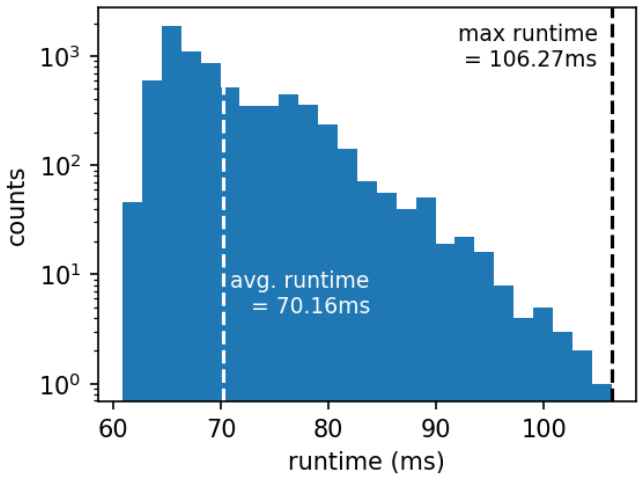



Figure 10. GPU processing time for retraining the autoencoder on 1 second of new gear data


Figure 11. Computation pipeline for retraining the autoencoder on 1 second of new gear data

agation data of the original gear (Gear A) can be seen in Figure 12. Evaluation of the adapted autoencoder no longer provided the ability to differentiate the original gear's baseline and failure propagation; they completely overlapped each other, however, the adapted autoencoder model showed good separation on the new gear (Gear B), see Figure 13.

A similar adaptation process was performed utilizing 5 second steps of data in the hopes that it would improve data processing times. The idea was to collect a small batch of data to be processed during the time the next batch was being collected. The processing time for each retraining step on 5 seconds of data only increased by 11.23 ms. This is equivalent to a 5% decrease in time to read input data per second, which allows for larger batches. however, the performance of the autoencoder model was significantly degraded. This trade-off between faster processing of larger batches and better outcomes of smaller batches is very common in training neural networks (Geron, 2019). A noticeable overlap in the mean absolute error of the baseline and propagation data is shown for the 5-second case in Figure 14.

## 5. CONCLUSION

We compared computational performance on CPU to the associated performance on GPU, using vibration-based engineered and data driven CIs. We simulated a modified HUMS to demonstrate a real-time edge processing scenario that computed TSA from raw accelerometer data, produced 16 traditional CIs, and evaluated a neural network anomaly detector.

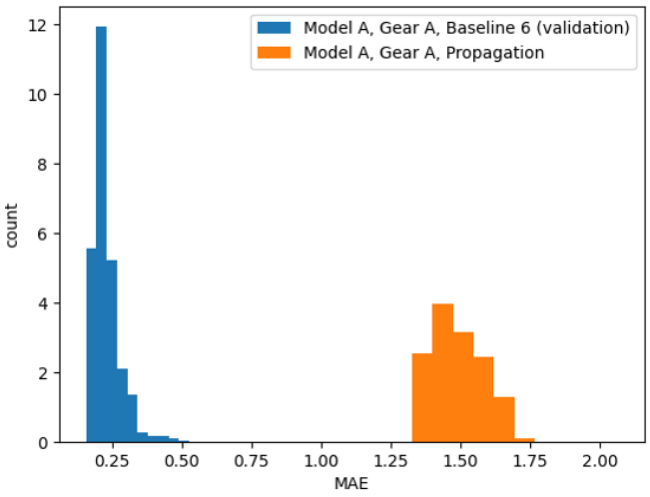



Figure 12. Mean absolute error of the original autoencoder model under baseline and crack propagation conditions for Gear A

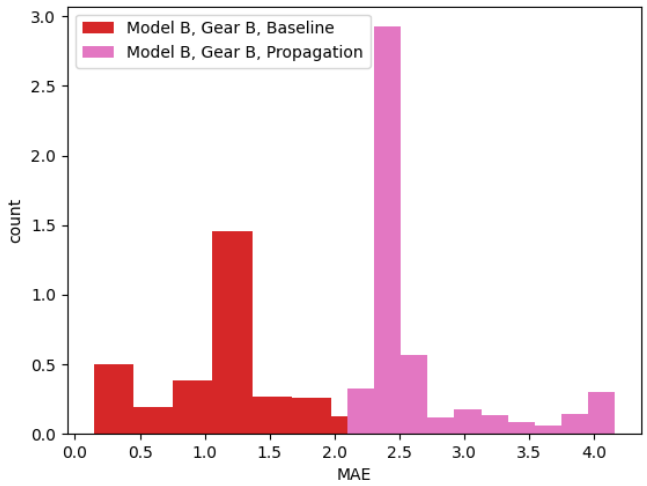



Figure 13. Mean absolute error of the adapted autoencoder model under baseline and crack propagation conditions for Gear B

We showed the number of accelerometer signals that could be concurrently processed within each second of data input. We found GPU utilization was most effective for CNN architectures. The modified HUMS with the entry level Jetson Nano was capable of processing autoencoders for 2.2X as many accelerometer signals concurrently on the GPU as opposed to running them on the CPU. We showed that classical engineered CIs do not benefit significantly from GPU computations, but data-driven autoencoder MSE can be computed significantly faster on a portable GPU. We found model adaptation on the edge to be useful in the cases where data, model, and error metric were fixed for detection of near future anomalies on different assets. The case of updating a model's

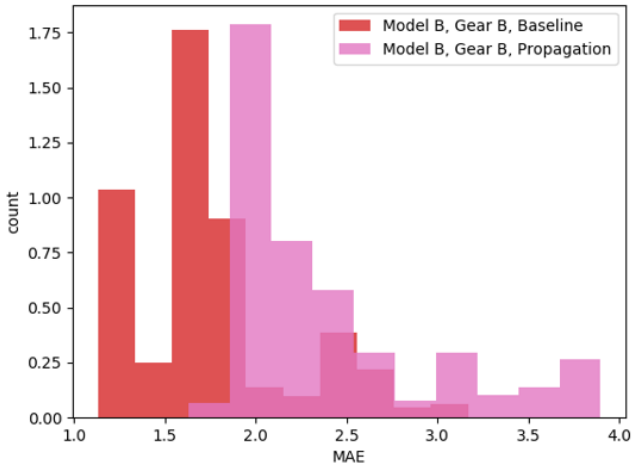



Figure 14. Mean absolute error of the 5 second training step adapted autoencoder model under baseline and crack propagation conditions for Gear B

weights to provide the ability to differentiate healthy state operation from failure was accomplished with an adapted model to a new gear. The benefit of an immediately transferable anomaly detector was considered for large fleets on similar vehicles, where the cost and labor associated with physically transfer data from these vehicles was infeasible. While the updated model lost the ability to differentiate its original gear's failure, it was successful in identifying anomalies in its new gear. The usefulness of this computation adaption is promising for computing on the edge in applications where model training is fixed. Fleets with hundreds or thousands of assets can significantly benefit from edge computations like that demonstrated in this research. Edge devices are a rapidly growing area of focus and will continue to improve at an increasing rate. This research provides a starting point and baseline for low-cost edge computing in vibration-based anomaly detection.

## ACKNOWLEDGMENT

This material is based upon research sponsored by the Department of the Navy, Office of Naval Research under ONR Award No. N00014-18-1-2339.

## DISCLAIMER

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.

## REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... others (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)* (pp. 265–283).

Bechhoefer, E., & Kingsley, M. (2009). A review of time synchronous average algorithms. In *Annual conference of the phm society* (Vol. 1).

Duke, A., Bailer, M. E., Thitchener, L. P., & Gebauer, U. (2001, February). *The u.s. navy/b.f.goodrich integrated mechanical diagnostic cossi program - an update* (Vol. 5-16-2). `https://humsconference.com.au/Papers2001/5-16-2.pdf`. Melbourne, AU.

Eklund, N. (2018). Anomaly detection tutorial. In *Proceedings of the annual conference of the prognostics and health management society.*

Geron, A. (2019). Hands-on machine learning with scikit-learn, keras tensorflow. In (2nd ed., p. 326). O'Reilly.

Goebel, K., Daigle, M. J., Saxena, A., Roychoudhury, I., Sankararaman, S., & Celaya, J. R. (2017). *Prognostics: The science of making predictions*. CreateSpace Independent Publishing Platform.

Heine, R., & Barker, D. (2007). Simplified terrain identification and component fatigue damage estimation model for use in a health and usage monitoring system. *Microelectronics Reliability*, *47*(12), 1882–1888.

Japkowicz, N., Myers, C., & Gluck, M. (1995). A novelty detection approach to classification. In *Ijcai* (Vol. 1, pp. 518–523).

Lebold, M., McClintic, K., Campbell, R., Byington, C., & Maynard, K. (2000). Review of vibration analysis methods for gearbox diagnostics and prognostics. In *Proceedings of the 54th meeting of the society for machinery failure prevention technology* (Vol. 634, p. 16).

Martin, H. (1989). Statistical moment analysis as a means of surface damage detection. In *Proceedings of the 7th international modal analysis conference* (Vol. 1, pp. 1016–1021).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... others (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*, 8026–8037.

Polyshchuk, V., Choy, F., & Braun, M. (2002). Gear fault detection with time-frequency based parameter np4. *International Journal of Rotating Machinery*, *8*(1), 57–70.

Rajesh, S., & Francis, B. A. (2012). *A study of condition based maintenance for land force vehicles*. DSTO.

Sait, A. S., & Sharaf-Eldeen, Y. I. (2011). A review of gearbox condition monitoring based on vibration analysis techniques diagnostics and prognostics. In *Rotating machinery, structural health monitoring, shock and vibration, volume 5* (pp. 307–324). Springer.

Samuel, P. D., & Pines, D. J. (2005). A review of vibration-based techniques for helicopter transmission diagnostics. *Journal of sound and vibration*, *282*(1), 475–508.

Sharma, V., & Parey, A. (2016). A review of gear fault diagnosis using various condition indicators. *Procedia Engineering*, *144*, 253–263.

Stewart, R. (1977). *Some useful data analysis techniques for gearbox diagnostics. machine health monitoring group report mhm* (Tech. Rep.). R/10/77, University of Southampton.

Swansson, N. (1980). Application of vibration signal analysis techniques to condition monitoring. In *Conference on lubrication, friction and wear in engineering 1980, melbourne: Preprints of papers* (p. 262).

Vachtsevanos, G., Lewis, F. L., Roemer, M., Hess, A., & Wu, B. (2006). Intelligent fault diagnosis and prognosis for engineering systems. In *1st ed. hoboken.*

Yan, W., & Yu, L. (2015). On accurate and reliable anomaly detection for gas turbine combustors: A deep learning approach. In *Proceedings of the annual conference of the prognostics and health management society.*

Zakrajsek, J. J., Townsend, D. P., & Decker, H. J. (1993). *An analysis of gear fault detection methods as applied to pitting fatigue failure data* (Tech. Rep.). NATIONAL AERONAUTICS AND SPACE ADMINISTRATION CLEVELAND OH LEWIS RESEARCH CENTER.

## BIOGRAPHIES

**Christopher J. Valant** received his B.S. in Physics and M.S. in Data Science from Rochester Institute of Technology (RIT). He joined the Golisano Institute of Sustainability (GIS) in 2015. His research interests include applied machine learning, in particular, artificial intelligence with computer vision and decision theory, as well as evolutionary algorithms, optimization, and programming.

**Sean P. McConky** received his B.S. in Industrial Engineering from Rochester Institute of Technology (Rochester, NY, USA) in 1999. He joined the Center for Integrated Manufacturing Studies (CIMS) at Rochester Institute of Technology in 2000, where he is currently a Senior Staff Engineer. His research interests include condition based maintenance, asset health monitoring, additive manufacturing, and remanufacturing. He holds one patent for dynamic display systems related to asset health monitoring. He was a member of the team awarded the Defense Manufacturing Excellence Award for work on lifecycle logistics support tools.

**Michael G. Thurston** received his B.S. and M.S. in Mechanical Engineering from Rochester Institute of Technology (Rochester, NY, USA) in 1988, and his Ph.D. in Mechanical and Aerospace Engineering from the University of Buffalo (Buffalo, NY, USA) in 1998. He is the Technical Director and Research Associate Professor at the Center of Integrated Manufacturing Studies at Rochester Institute of Technology. He formerly held positions in air conditioning system development at General Motor and Delphi, and as a Researcher at the Applied Research Laboratory at Penn State University. He holds 7 patents in the areas of air conditioning and asset health monitoring. His research interests include: sustainable design and production, condition based maintenance and prognostics, and asset health management. He is a member of the Society of Automotive Engineers, and was awarded the Boss Kettering Award for product innovation by Delphi.

**Patrick R. Horney** received his B.S. in Aerospace Engineering from Purdue University in 2008. Since 2012 he has worked for NAVAIR in the areas of Diagnostics, Prognostics and Condition Based Maintenance (CBM). He has established CBM/CBM+ capability for the US Navy on two aviation platforms and serves as an advisor in the areas of PHM and CBM+.

**Allen Jones** received his B.S in Mechanical Engineering from the University of Maryland and his M.S. in Engineering Management from the George Washington University. He is a mechanical engineer at the Naval Air Warfare Center Aviation Division in Patuxent River, MD, working in the Air Systems Group, Propulsion and Power Engineering, Drives and Mechanical Systems Branch. His research interests include machine learning and advanced algorithms for improved diagnostics and prognostics using Health and Usage Monitoring System data.

**Nenad G. Nenadic** received his B.S. in Electrical Engineering from University of Novi Sad (Novi Sad, Serbia) in 1996 and his MS and Ph.D. in Electrical and Computer Engineering from University of Rochester (Rochester, NY, USA) in 1998 and 2001, respectively. He joined Kionix Inc. in 2001, where he worked on development of microelectromechanical inertial sensors. Since 2005, he has been with Center for Integrated Manufacturing Studies (CIMS) at Rochester Institute of Technology, where he is currently a Research Associate Professor. His research interest include design, analysis, and monitoring of electromechanical devices and systems. He has two patents in electromechanical design. He co-authored a textbook *Electromechanics and MEMS* and is a member of IEEE.