MxV RAIL
Technology Digest
JUNE 2025
TD25-014

Edge Computing Using Ultrasonic A-Scans for Rail Inspection

Anish Poudel, Ryan Alishio, and Survesh Shrestha

Key Findings

- MxV Rail is exploring the potential to utilize UT A-scan data to develop a machine learning system to accurately identify internal defects in rail.
- Phase 1 demonstrated the capability of a CNN model to utilize UT A-scan data to accurately discern different types of rail defects. Phase 2 focused on identifying hardware, software, and configuration methods to provide the necessary speed for real world deployment.
- The CNN model implemented in the edge device achieved an overall accuracy of 89 percent for the 10-class problem.
- Through the optimization of multiple parameters, the team was able to increase the processing rate from 200 FPS to 2,500 FPS. This translates to an approximate processing time of 0.4 ms per A-scan and would allow a scan to be collected and analyzed approximately every 3/16 inch (4.5 mm) at a detector car speed of 25 mph.

MxV Rail, a subsidiary of the Association of American Railroads, performed the research described in this publication.

In 2024, MxV Rail continued collaborating with the Project JeZero team to deploy the convolutional neural networks (CNN) model developed in Phase 1 on an edge computing device for rail ultrasonic testing (UT) using A-scans. The intent of this collaboration was to increase the accuracy and speed of ultrasonic rail flaw inspection and open up the possibility of using ultrasonic amplitude scans (A-scans) for future defect trending and data fusion work. This *Technology Digest* highlights the technical approach and implementation of Phase 2 work, which includes edge hardware selection, optimization of CNN model for edge deployment, data processing pipelines, and evaluation of the deployed system's performance. MxV Rail's work is not aimed at developing new products; rather, it is centered on demonstrating the feasibility of refining and optimizing existing rail UT technology using edge processing with A-scans.

The edge computing device chosen for this test was the NVIDIA® Jetson Orin™ NX platform and various optimization methods, such as TensorRT™ integration, batching, and quantization, were employed. These methods led to an impressive 12.5x boost in the CNN inference speed, compared to a standard laptop setup. The optimized CNN model on the edge device achieved a processing speed of 2,500 A-scans or frames per second (FPS), allowing for real-time A-scan analysis and setting the stage for a portable rail defect detection system. A processing rate of 2,500 FPS would allow a UT scan resolution of approximately 3/16 inch (4.5 mm) at a rail inspection detector car speed of 25 mph. A client-server architecture that used the edge device as the detection server and a lab PC as the client was also established for lab testing. Although the data processing pipeline was limited due to the A-scan file reading speed, network latencies, and data pre-processing needs, the client-server architecture still shows promise for achieving real-time defect detection at speeds exceeding 1,550 FPS. The practical implementation and demonstration of the CNN model for rail UT A-scan signals on an edge device marks a significant achievement toward creating a fully autonomous, field-ready ultrasonic rail inspection system. With the ability to detect defects in real-time, this approach could transform rail UT inspection, significantly improving safety, reliability, and operational efficiency.

BACKGROUND

Rail detector cars, specifically used for non-stop operations, collect ultrasonic B-scan data during the inspection and transfer the collected data to the cloud for processing and further analysis. The cloud computing approach leverages more powerful computational resources in the cloud, where larger and more complex machine-learning (ML) models

can be used for deep analysis. While a cloud-based analysis offers advantages in terms of computational power, scalability, and advanced ML capabilities, some key challenges include latency, connectivity issues, security risks, and high operational costs. In some environments, especially where real-time analysis is critical or network infrastructure is limited, these disadvantages might outweigh the benefits of using the cloud for ultrasonic rail flaw detection. As such, hybrid approaches, such as those that combine edge computing or processing with cloud support, potentially offer solutions to mitigate these challenges.

A-scans are typically not retained or stored for further analysis in rail detector car UT systems due to the generation of a significant amount of raw data, especially when conducting inspections over long distances. Also, storing and managing this large volume of A-scan data can be impractical and resource-intensive. A-scans contain many features not easily discerned by human eyes, and they provide opportunities for data fusion to other inspection modalities.

EDGE HARDWARE SELECTION

Edge computing is a computing framework that involves processing, analyzing, and storing data closer to where it is generated. This framework offers several advantages, including improved real-time response, enhanced privacy by reducing data transmission to external servers, and reduced dependency on network connectivity. Selecting suitable edge hardware for deploying the ML model is paramount to achieving real-time rail defect classification in the laboratory or field settings. Three edge hardware candidates were initially considered to accelerate ML tasks on edge devices. A comparison was performed based on each component's computing capability, power consumption, developer ecosystem, cost, and size, as shown in Table 1. TOPS stands for "trillions of operations per second" and is a generic way of comparing the computation capacity of various devices.

Table 1. Comparison of three edge hardware candidates.

Criteria	Nvidia Jetson Orin	Google Coral™	Intel® Movidius™
Computation capability [TOPS]	70	4	1
CPU	6 Core, 610 MHz	N/A	N/A
GPU	1024 CUDA Core	Google TPU	Intel VPU
Power	20 W	2 W	1W
Ecosystem	Robust	Limited	Limited/ Niche
Cost	$500-$2,000	$60	$60
Size	100×80×20 mm	60×30 mm	73×27 mm

The Nvidia Jetson Orin NX 8GB was the best choice due to its strong computation power and developer ecosystem. Although the Nvidia consumes more power than the other options, it can run for 4 to 8 hours on a standard battery pack. While the Google Coral Edge TPU and Intel Movidius have low power consumption and compact sizes, their computing power and limited ecosystems did not meet the project's needs and therefore, neither option was considered further.

CNN IMPLEMENTATION ON EDGE DEVICE AND OPTIMIZATION

This section discusses the optimization process for implementing the CNN model developed during Phase I and achieving maximum inference performance on the edge computing platform. First, baseline performance tests were conducted on a standard laptop (Intel® Core™ i5-9300H, 4 cores at 2.4 GHz, 16 GB RAM) to simulate a field scenario. Using the PyTorch framework, the initial inference speed determined using this setup was 200 FPS. The baseline test was used as a reference for measuring performance improvements after further optimization. While PyTorch (used in Phase 1) offers a versatile and dynamic execution environment well-suited for training models, it also adds computational overhead during inference on edge computing devices. The optimization process included TensorRT integration, batching, and quantization.

TensorRT Integration

TensorRT ML framework was considered and implemented to address PyTorch limitations in the Nvidia edge device. TensorRT is a high-performance inference library designed specifically by Nvidia to accelerate deep learning models on its edge devices. The library achieves this acceleration through several key optimizations, including computational graph analysis and runtime engine optimization. The combined effect of these optimizations is a significant reduction in computational overhead and improved overall inference efficiency. Converting the PyTorch model to TensorRT and deploying it on the edge device boosted the processing rate to 600 FPS, a 3x increase in speed compared to the baseline, demonstrating the significant performance gains from using TensorRT.

Batching

The FPS corresponds to processing a single A-scan at a time.[1] While this approach yielded significant performance improvements compared to the baseline processing, it cannot fully utilize edge device parallel computation capability. With over 1,024 cores, the Jetson GPU is designed to handle multiple tasks simultaneously, and processing A-scans one by one underutilizes its inherent parallelism. To address this limitations of a single-scan processing

approach, MxV Rail tested processing multiple A-scans simultaneously (batching) using TensorRT and found that a batch size between 20 and 30 A-scans worked best, boosting the processing speed to 1550 FPS. A 2.5x improvement over processing A-scans individually, this boost shows significant performance gains from batch processing on the edge platform. Figure 1 shows the impact of batch size on inference speed.

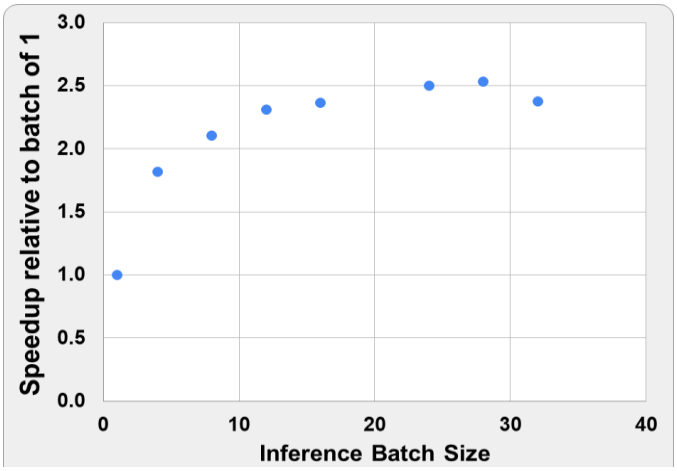

Figure 1. Impact of batch size on inference speedup

The graph illustrates the relative speedup in inference FPS compared to the inference speed when handling one batch at a time related specifically to the time taken by the edge device GPU and excluding any factors related to data transfer or additional processing. Increasing the batch size beyond 28 did not improve performance and led to a slight decrease, suggesting a saturation point where memory bandwidth or other system resources become limiting factors. If the CNN architecture is updated in the future, the optimal batch size should be re-evaluated.

Quantization

The CNN model developed using the PyTorch processes data in a 32-bit floating-point format, offering high precision (i.e., up to 10^38 significant digits). However, A-scan values generally fall within a range of zero to a few hundred, where the less significant decimal places likely contribute little to the neural network's classification accuracy. As a result, substantial computing resources are spent on high-precision calculations that have minimal effect on accuracy, leading to reduced inference speed.

MxV Rail used quantization with 16-bit floating-point numbers in the TensorRT framework to improve efficiency. Compared to the default 32-bit format, the 16-bit format reduced the memory and computational demands, significantly accelerating neural network execution with minimal accuracy loss. Using 16-bit quantization also boosted the inference rate to 2,500 FPS, a 1.6x improvement with negligible accuracy loss (less than 0.1 percent). Figure 2 shows the efficiency gains relative to baseline through TensorRT adoption, batching, and quantization.

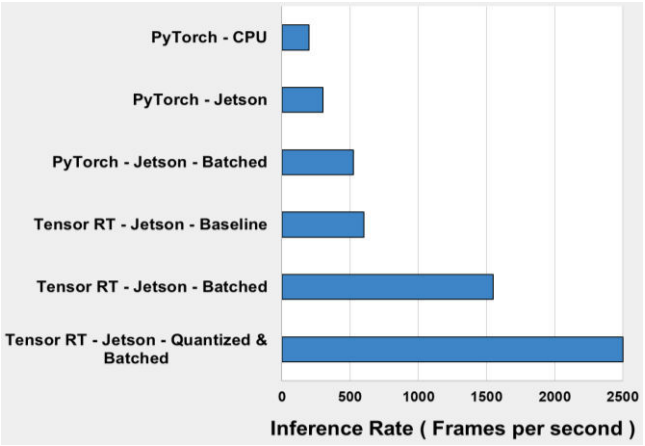

Figure 2. Efficiency gains using TensorRT framework

"Post-training quantization" (PTQ) is quantization applied to a pre-trained model without retraining, making it simple and suitable for this project phase. While PTQ can sometimes lead to notable accuracy loss, MxV Rail did not observe any loss of accuracy. Quantization-aware training (QAT), an alternative to PTQ, incorporates quantization during training, enabling higher levels, like 8-bit integers, for faster inference and potentially better accuracy. While PTQ met current performance needs, exploring QAT during the next retraining phase could further enhance speed and efficiency with minimal impact on accuracy.

RESULTS

The implemented CNN model on the edge device was first assessed using the 208,000 labelled A-scans used in Phase 1. The performance of the optimized model was within 0.02 percent of the baseline model. Next, the CNN model on the edge device was assessed using 23,150 A-scans that were never exposed to the model.

The classification report indicates that the CNN model performs with an overall accuracy of 89 percent. The precision, recall, and F1-scores for each class range from 0.75 to 1.00, indicating varying levels of performance across the different classes. The macro average F1-score is 0.81, suggesting a balanced performance across the classes. Taking into account the class imbalance, the weighted average F1-score is 0.89, indicating the model performs consistently across the dataset. Figure 3 shows the confusion matrix of the model's performance, with entries representing true positive, true negative, false positive, and false negative instances, providing insights into model accuracy and errors.

Table 2. Time requirements for each stage of the UT inference

CSV load time*	Data Transfer to Jetson	A-scan preprocessing on Jetson	Neural Network Interface	Total Time	Total Inference Speed	Number of A-scans per inference request
[ms**]	[ms]	[ms]	[ms]	[ms]	FPS	Batch size
1.20	0.99	0.40	0.50	1.88	531	24
1.20	0.15	0.29	0.50	0.94	1062	48
1.20	0.11	0.29	0.38	0.78	1284	96
1.20	0.14	0.24	0.42	0.80	1250	192
1.20	0.06	0.20	0.38	0.64	1569	384

* CSV load time excluded from total time in inference FPS since A-scans are not expected to arrive in this format in field deployment, **ms = milliseconds

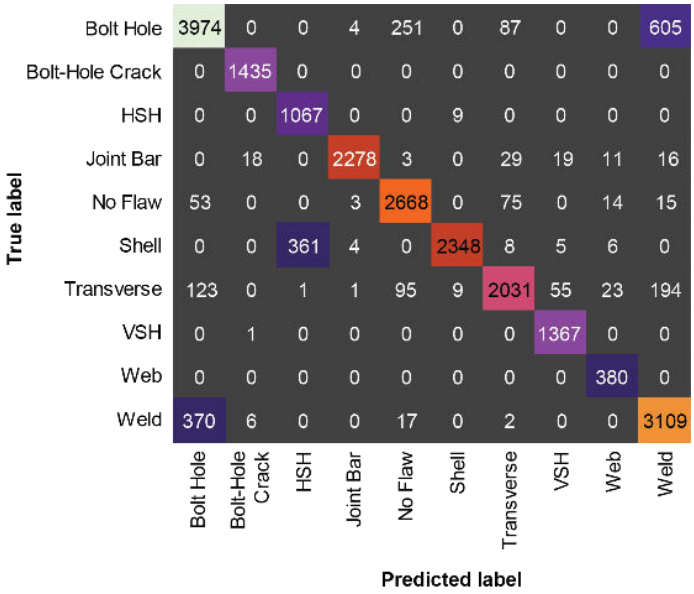

Figure 3. Confusion matrix showing the model performance

UT DATA PIPELINE ARCHITECTURE

This section describes the mechanism for transferring data from the UT scan source to the edge device for inference and the various stages of data processing. A simple client-server setup was used for the proof-of-concept laboratory demonstration and testing. The edge device served as the detection server, while a laboratory PC acted as the client, and the two were connected via Ethernet for fast data exchange. The laboratory PC sends A-scans to the edge device for processing and receives classification results facilitated by a Representational State Transfer Application Programming Interface protocol commonly used in client/server interactions. The complete data processing pipeline encompasses four steps: 1) retrieval of UT A-scan data from the sensor, 2) transfer of A-scan data to the edge device, 3) preprocessing of A-scans, and 4) the final inference using the neural network. Table 2 summarizes the time associated with each step in the MxV Rail setup.

CONCLUSIONS AND FUTURE WORK

This research demonstrated the migration of the CNN model for rail UT defect classification from a cloud-based environment to a field-ready edge device. Using the edge device and optimizations like TensorRT, batching, and quantization, MxV Rail achieved a 12x speedup compared to a laptop baseline, reaching 2,500 FPS for real-time A-scan analysis. A client-server setup for lab testing also demonstrated over 1,550 FPS but was limited by A-scan file reading, network latency, and preprocessing.

Future improvements include integrating preprocessing into the neural network, adopting 8-bit quantization, upgrading hardware, or scaling down the network architecture during retraining for a potential 5 to 10x boost in speed. Future work can focus on optimizing the results through strategies such as implementing smaller network architectures and exploring edge inference capabilities. Laboratory and field tests are necessary to see the performance of the hardware and software in more real-world conditions. Work is underway to house and install the unit with MxV Rail UT data collection hardware, and progress will be reported in future reports.

References

1. Poudel, A. and S. Shrestha. April 2024. "Classification of Rail Ultrasonic Signals Using Machine Learning." *Technology Digest* TD24-005. AAR/MxV Rail: Pueblo, CO.

For comments or questions about this publication, contact Anish Poudel, MxV Rail