Fast prediction of job execution times in the ALICE Grid through GPU-Based Inference with Quantization and Sparsity Techniques

Tomasz Lelek[1,⋆][0000−0001−7268−6484], Szymon Mazurek[2,3,⋆][0009−0006−7557−0157], Maciej Wielgosz[2,3][0000−0002−4401−2957], and Bartosz Balis[1][0000−0002−3082−4209]

[1] Faculty of Computer Science, AGH University of Krakow, al. Mickiewicza 30, 30-059 Krakow, Poland
{tlelek, balis}@agh.edu.pl

[2] Faculty of Computer Science, Electronics and Telecommunications, AGH University of Krakow, al. Mickiewicza 30, 30-059 Krakow, Poland
{smazurek, wielgosz}@agh.edu.pl

[3] ACC Cyfronet AGH, Nawojki 11, 30-072 Krakow, Poland

Abstract. We propose a latency-optimized neural network model to dynamically predict job execution times for the ALICE experiment at CERN, replacing static Time-To-Live (TTL) allocations. Utilizing Nvidia A100 GPUs, we optimize inference latency via FP16 and INT8 quantization, 2:4 sparsity, quantization-aware training, and graph compilation. Results show that FP16 and sparsity reduce latency for larger batches, while INT8 is optimal for single-sample predictions. For single-sample online inference, static INT8 quantization achieves a median 0.38 ms prediction time, a 1.8x improvement over the 0.71 ms baseline. The model achieves a 1.9-hour RMSE, improving on the 14.23-hour RMSE of current TTL assignments. With sub-40ms inference latency on GPU hardware, this work demonstrates how NN optimization can help achieve performance demands of large-scale distributed computing systems.

Keywords: Deep learning · ALICE · latency optimization · high-energy physics · real-time inference · job scheduling

1 Introduction

AI is increasingly important in large-scale computing environments, where vast amounts of system data are continuously generated. Effectively analyzing this data is crucial for efficient administration, maintenance, and resource management [7]. Large-scale computing is essential in high-energy physics experiments, such as ALICE at CERN. The ALICE Grid, with 60 global computing clusters, runs 500k daily jobs running simulations and analyzing data. With up to 200k concurrent jobs, efficient scheduling is crucial. Currently the job scheduling algorithm uses Time-To-Live (TTL) values (expected execution time), but these

⋆ Equal contribution.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_12

are often overestimated, leading to suboptimal resource allocation. While accurate execution time estimation could improve job scheduling, a prediction service needs to be very fast due to the high job throughput. Neural network inference and training acceleration through pruning, quantization, and other optimizations is well-studied [9, 13]. However, research often emphasizes memory footprint and performance degradation resistance [3]. Inference speed is frequently evaluated with fixed network architectures, data modalities, and batch sizes, limiting insight into technique effectiveness [11, 3]. Evaluations also tend to focus on complex CNNs or transformers with many parameters [9].

In this work, we develop a neural network model to predict execution times of ALICE Grid jobs, based on their input parameters and machine characteristics. We then analyze inference acceleration techniques, including quantization, semi-structured sparsity, and graph-based model compilation to reduce the inference latency, critical for optimizing job scheduling in the ALICE large-scale computing infrastructure. Our key contributions include demonstrating that INT8 quantization achieves a 1.8× speedup for single-sample inference, while semi-structured sparsity improves large-batch processing by up to 4×. We also highlight the trade-offs in quantization strategies, noting that weight quantization is highly effective, whereas activation quantization requires careful application. Finally, we present a scalable framework which can be considered as a blueprint for deploying deep learning optimizations, enhancing efficiency at large scale. These contributions provide a guidance for future large-scale scientific projects, outlining how deep learning optimizations can be integrated into mission-critical workflows where latency, scalability, and accuracy are essential.

The paper is organized as follows. Section 2 outlines research context and methods. Section 3 presents experimental results. Section 4 contains discussion and concluding remarks.

2 Methods

2.1 Architecture of the ALICE system

The architecture of the ALICE system, extended with the *Prediction Service* which is the subject of this research, is shown in Fig. 1. This service should efficiently predict the job runtime (TTL) based on its submission parameters specified by the user, and the target machine. In addition to the obvious requirement of minimizing the TTL prediction error, the model deployed within the prediction service must perform the predictions as fast as possible. In our scenario, the high job arrival rate requires that the total time from job submission to receiving the TTL prediction from the service be under 40 ms.. While the latency will be influenced by each component of the new service, here we focus solely on the deep learning model inference time.

2.2 Hardware accelerators in AI

In modern DL models, most computations fall into the matrix multiplication and addition category (MMA). The nature of these operations allows for mas-

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_12

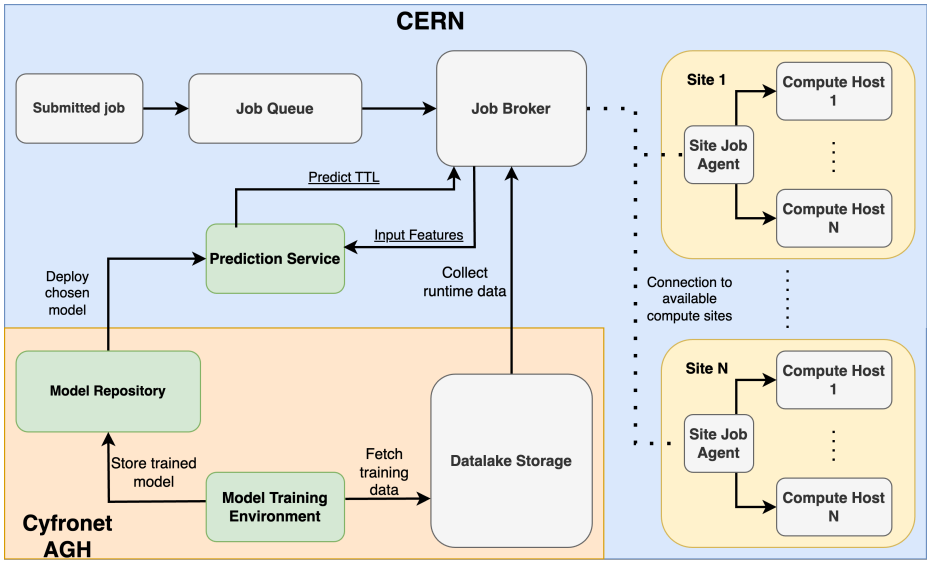

Fig. 1. The architecture of the ALICE computing framework, integrating CERN's infrastructure and the proposed AI-based prediction service. Elements of the architecture discussed in this paper are marked in green.

sively parallel execution, as most atomic operations on separate elements of the matrices can be done independently. This fact led to the development of specialized hardware accelerators focusing on parallel-vector operations. Among those, graphics processing units (GPUs) are the most widespread. They are rapidly evolving, with each generation introducing new, faster architectures with support for additional features, such as operations on floating-point values with reduced precision or higher I/O bandwidth.

The Nvidia Ampere GPU series are one of the most recent developments in this domain. They introduce numerous improvements compared to the previous ones, such as the new generation of tensor cores; extended support for numeric precision formats, and cache memory expansion [2]. Our experiments utilize this type of GPU along with the following techniques: *semistructured sparsity* [11], *quantization* [4], and *model compilation* with the Pytorch native compiler, Inductor [1].

2.3 Neural network architecture and experiment design

To solve the TTL prediction problem posed, we created a simple feedforward neural network. The architecture consisted of blocks of linear and batch normalization layers followed by ReLU activation [8, 12]. The last block was an exception as it contained only the linear layer.

In the sparse runs, pruning was applied to all linear layers except the last one, as the single output neuron did not meet the divisibility criterion for 2:4

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_12

sparsity patterns. The number of input neurons was set to 19840 for sparse runs and 19809 for dense ones. The difference arises from the requirement that 2:4 sparse tensors must have dimensions divisible by 32 or 64 for the FP32 and BF16 precisions, respectively. To meet these criteria, zero-padding was applied to the input feature vector in sparse runs.

To assess model latency, forward pass execution times were measured after applying the optimization techniques discussed earlier. The measurements included median and P99 values across 1000 trials, with each trial preceded by a GPU warm-up period not included in the recorded results. Latency measurements were conducted for batch sizes ranging from 1 to 16384, each being a consecutive power of 2.

For final model evaluation, the optimizations yielding the highest inference time reduction for given batch size ranges were selected. These models were trained and validated using the 10-fold cross-validation method, assessing the mean squared error of the average root (RMSE), the maximum prediction error and the median error across all folds. The network training process continued for up to 100 epochs, with early stopping implemented when no validation loss improvement occurred for 5 consecutive epochs. The AdamW optimizer [10] was used with a learning rate and weight decay set at 10^{-3}. The validation subset was extracted as 10% from the training data for each fold, ensuring robust model evaluation.

2.4 Experiments setup and used hardware

The experiments were carried out within the HPC cluster, using a single Nvidia A100 GPU, up to 16 cores of AMD EPYC 7742 CPU and up to 200 GB of RAM. The experimental code was created with the Pytorch 2.5.1 framework [1], using the CUDA 12.1 toolkit for GPU integration. For sparse matrix operations, we used CUTLASS [6] 3.4.0 as a backend for sparse matrix operations.

3 Results

3.1 Analysis of available features and job execution times in the past ALICE workloads

In the existing ALICE scheduling system, the job execution times are assigned static TTL values, determined manually by the operators. These values are often highly overestimated to ensure that jobs are completed successfully, even on the slowest CPUs in the grid. Although this conservative approach prevents job failures due to insufficient time, it is obviously suboptimal from the perspective of effective resource utilization.

To understand the extent of the inefficiencies, we calculated the RMSE between operator-defined TTLs and actual job execution time. Analysis has shown that the mean RMSE was equal to 14.23 hours, the median to 15.12 hours, and the maximal RMSE observed was 23.89 hours. Most jobs are assigned TTLs that

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_12

exceed 20 hours, while their actual execution time in most cases is considerably shorter. Thus, it is clearly visible that there exists vast room for improvement to achieve more accurate execution time predictions.

Based on the analysis of historical job data, we have identified 20 categorical and numerical features that describe the job when it is submitted. Categorical features were encoded into one-hot vectors, with the missing categorical values being treated as a separate category. For the numerical features, missing data was filled with the median value calculated across all jobs in the data set. Next, each numerical feature was independently z-score standardized. Lastly, we combined categorical and numerical features into a single one-dimensional vector that was used as input for the network.

3.2 Evaluating inference latency for chosen optimization techniques

We first evaluated the forward pass time of the constructed network for a given batch size. The median and P99 latency are shown in Fig. 2. The results show some visible trends. It can be seen that using BF16 precision leads to a lower inference latency in all scenarios. Even when weights or activations were quantized with lower precision and BF16 was used only to represent activations, it still provided performance advantages. Interestingly, in runs using low-precision quantization, the latency was usually drastically higher than in the unoptimized baseline model. An exception is a single sample performance of a model with statically quantized INT8 weights. These two phenomena can be explained by the overhead introduced with the quantization techniques. In the case of static weight quantization, the activations of each layer are computed in the input precision (either FP32 or BF16 in our case). To perform such operation, weights have to be casted into the corresponding precision to compute activations, thus introducing additional overhead. However, when the forward pass is bound by memory transfer between GPU vRAM and streaming multiprocessor cache, copying of data in lower precision is faster than in higher ones. As the number of samples is small, the overhead caused by additional computation does not yet impact the processing time in a significant way. When the batch size increases, the overhead of quantization starts to manifest, showing lower inference speeds. Similar phenomena occur in dynamic quantization. As activations are in INT8 precision and weights are INT4, casting is also required. Furthermore, dynamic quantization of activations requires computing quantization parameters on the fly, adding another source of delay. Sparse runs perform worse than baseline in small batch sizes; however, they outperform all other methods when the batch size starts to grow. It is caused by additional operations required to properly process compressed sparse tensors [5]. Lastly, we observe that graph compilation leads to slight speed-ups in each case. This is in contrast to what could be expected, although we speculate that the cause lies in the lack of utilization of CUDA graphs, which we purposely disabled.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_12

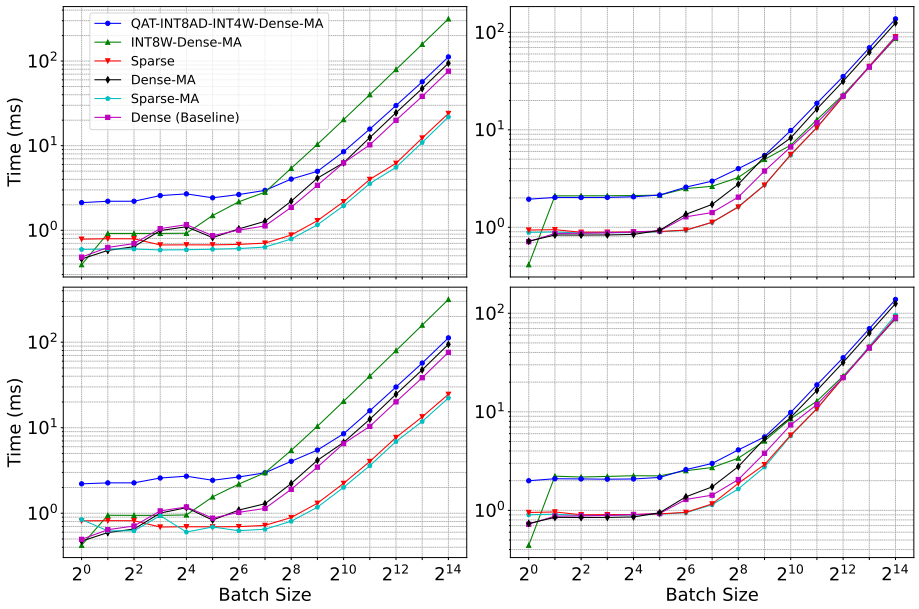

Fig. 2. Forward pass latency with different optimization techniques applied depending on an input batch size. Charts in the top show median forward pass time, while bottom ones present P99 values. The left and right colums refer to measurements in BF16 and FP32 activation precision, respectively. *QAT* denotes quantization aware training; *AD* refers to dynamic activation quantization in corresponding precision, and *W* refers to static weight quantization with corresponding precision. In both BF16 and FP32 activation precisions, we refer to dense run with no optimizations as a baseline.

3.3 Evaluating the performance of the obtained optimized models

We chose to evaluate the performance of only specific models based on the previous latency measurements, each showing an advantage for specific batch size ranges. The optimization techniques chosen were as follows: BF16 activation precision, BF16 activation precision combined with semistructured weight pruning, and static weight quantization to INT8 format, with activations remaining in BF16. The FP32 dense network served as a baseline. In each case (including baseline), the compilation of the model was added, as it was safe to assume that it will not affect the qualitative performance of the model. In previous experiments, it was also shown that it consistently provided a speed-up of inference. The results, along with brief descriptions of advantages in latency reduction, are shown in Tab. 1.

4 Discussion and conclusions

In this study, we have presented a latency-optimized neural network to predict workload execution time in the ALICE computing infrastructure. We have exam-

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_12

Metric \ Method	FP32	BF16	BF16-Sparse	BF16-INT8W
RMSE	**1.93** ±0.01	1.92 ±0.01	1.91 ±0.01	1.91 ±0.1
Avg. max error	19.11 ±0.67	19.14 ±0.71	**18.92** ±0.06	19.1 ±0.67
Avg. median error	0.51 ±0.01	0.48 ±0.01	**0.47** ±0.01	0.48 ±0.01
Advantage in latency reduction	Baseline	Simple to use, moderate speed-up	Significant speed-up for larger batches	Speed-up when processing single sample

Table 1. Average metrics summarizing 10-fold cross validation evaluation of the networks with different latency reduction techniques applied, ± SEM. Each run included model compilation and dense weights, unless stated otherwise. Best results are highlighted in **bold**. We also note the advantages of each technique for inference speedup.

ined different combinations of optimization techniques, including semi-structured sparsity, mixed-precision training, and inference, static and dynamic quantization into low-bit representations of weights and activations, quantization-aware training, and model compilation algorithms. We performed our analyses on a wide range of input batch sizes, spanning far beyond our use case, to provide insight for researchers looking for different use cases in the future. We find that the optimal latency reduction technique varies depending on the input batch size. Semi-structured sparsity is suitable for larger batch sizes, while for smaller ones, it can introduce additional overhead. Quantization has to be applied with caution, as it can introduce additional overhead. This is especially true for dynamic quantization techniques, where the number of additional operations is even higher. Despite the obvious benefits of reducing the memory footprint, improper use of quantization can lead to a large decrease in processing speed.

During the final evaluation, our model greatly reduced the wall-time prediction error compared to the baseline obtained manually assigned by the infrastructure operators. The average prediction RMSE on test subsets was at the level of 1.9 hours, while for manual assignment, it was 14.23 hours across the available historical data. Importantly, the evaluated latency reduction methods did not result in the degradation of the final model performance. Based on the experimental results, we conclude that the proposed model is suitable as the backbone of the TTL estimation system within the ALICE architecture.

This work can be expanded in several directions. From the perspective of the predictive system design, extended evaluations will be made in terms of latency measurement and reduction, seeking to find further optimization targets. Secondly, a comprehensive evaluation of newer hardware can be performed. With the GPU market rapidly evolving, it is possible that using newer ones alone would yield better speedups. Other methods of latency reduction should also be explored, i.e. structured pruning or different quantization techniques. The model architecture or feature processing could be expanded to reduce the prediction errors even more. Evaluations on more extended datasets are also needed to further prove the robustness of the model.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_12

Acknowledgments. This work is co-financed by the Polish Ministry of Science and Higher Education under Agreement No. 2022/WK/01 and through the PMW program. We gratefully acknowledge Polish high-performance computing infrastructure PLGrid and the Academic Computer Centre Cyfronet AGH for providing computer facilities and support within computational grant no. PLG/2024/017775 and PLG/2024/017612. The research presented in this paper received partial financing from the funds assigned by Polish Ministry of Science and Higher Education to AGH University of Krakow. Research project supported/partly supported by program „Excellence initiative – research university" for the AGH University of Krakow.

Disclosure of Interests. Authors declare no conflicts of interest.

References

1. Ansel, J., et al.: Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (2024)
2. Choquette, J., Gandhi, W., Giroux, O., Stam, N., Krashinsky, R.: NVIDIA A100 tensor core GPU: Performance and innovation. IEEE Micro **41**(2), 29–35 (2021)
3. Danhofer, D.A.: Inducing semi-structured sparsity by masking for efficient model inference in convolutional networks. ArXiv (2024)
4. Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W., Keutzer, K.: A survey of quantization methods for efficient neural network inference. CoRR (2021)
5. Goumas, G.I., Kourtis, K., Anastopoulos, N., Karakasis, V.P., Koziris, N.: Performance evaluation of the sparse matrix-vector multiplication on modern architectures. The Journal of Supercomputing **50**, 36–77 (2009)
6. Huang, X., Zhang, X., Yang, P., Xiao, N.: Benchmarking GPU tensor cores on general matrix multiplication kernels through cutlass. Applied Sciences (2023)
7. Ilager, S., Muralidhar, R., Buyya, R.: Artificial intelligence (ai)-centric management of resources in modern distributed computing systems. In: 2020 IEEE Cloud Summit. pp. 1–10 (2020)
8. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proc. 32nd Int. Conference on International Conference on Machine Learning - Vol. 37. p. 448–456. ICML'15, JMLR.org (2015)
9. Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X.: Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing **461**, 370–403 (2021)
10. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: International Conference on Learning Representations (2017)
11. Mishra, A.K., et al.: Accelerating sparse deep neural networks. ArXiv (2021)
12. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: International Conference on Machine Learning (2010)
13. Wrobel, K., Karwatowski, M., Wielgosz, M., Pietroń, M., Wiatr, K.: Compression of convolutional neural network for natural language processing. Computer Science **21**(1) (Jan 2020). https://doi.org/10.7494/csci.2020.21.1.3375

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_12