Scaled FP32 and Quantization-aware Training of PatchTST for Efficient Time Series Forecasting

Lorson Blair
Rensselaer Polytechnic Institute
Troy, New York, USA
blairl@rpi.edu

Jeremy Buhler
Washington University in St. Louis
St. Louis, Missouri, USA
jbuhler@wustl.edu

Kaoutar El Maghraoui
IBM T. J. Watson Research Center
Yorktown Heights, New York, USA
kelmaghr@us.ibm.com

Christopher D. Carothers
Rensselaer Polytechnic Institute
Troy, New York, USA
chrisc@cs.rpi.edu

Naigang Wang
IBM T. J. Watson Research Center
Yorktown Heights, New York, USA
nwang@us.ibm.com

Jordan Murray
IBM Research AI Hardware Center
Albany, New York, USA
kelmaghr@us.ibm.com

Abstract

Transformers have gained prominence in the time series forecasting domain, with the patch time series Transformer (PatchTST) achieving state-of-the-art forecasting performance on a variety of tasks, outperforming older LSTM and RNN models and remaining competitive with the newer MLP and LLM based models. With the vast amount of times series data being generated daily, time series forecasting is becoming more resource intensive and less energy efficient. To improve its efficiency, we combine parallel training and quantization. By applying learning rate scaling and gradient clipping, we achieved full precision floating-point 32-bit (FP32) training of PatchTST across 128 GPUs, leading to an 81× speedup in training latency compared to single-GPU training, with no MSE degradation. We then quantize PatchTST to 8 bits (INT8) and 4 bits (INT4) per parameter and introduce the *Square Root 2 (Sqrt2) Scaling Rule*, which, in conjunction with gradient clipping, allowed us to effectively scale INT8 and INT4 quantization-aware training (QAT) across 128 and 80 GPUs respectively, achieving a 96× and a 65× reduction in training latency, with the INT8 model maintaining MSE and the INT4 model experiencing a 2.6% degradation in MSE compared to the baseline. *Sqrt2 Scaling Rule* also led to additional training stability and more consistent MSE performance when used to train our FP32 model. Our INT8 and INT4 models theoretically achieve a 74% and 87% reduction in model size, respectively, translating to a 3.9× and 7.5× compression compared to the FP32 model.

CCS Concepts

• **Computing methodologies**; • **Computer systems organization**;

Keywords

Multivariate time series forecasting, Time series transformer, time series transformer scaling, quantization-aware training, quantized PatchTST, scaled quantization-aware training.

ACM Reference Format:
Lorson Blair, Jeremy Buhler, Kaoutar El Maghraoui, Christopher D. Carothers, Naigang Wang, and Jordan Murray. 2025. Scaled FP32 and Quantization-aware Training of PatchTST for Efficient Time Series Forecasting. In *Proceedings of KDD MILETS Workshop 2025.* ACM, Toronto, ON, Canada, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Pre-trained transformer models have grown exponentially in size, posing challenges for deployment in resource-constrained environments. Recent research focuses on reducing model size and computational requirements to enable efficient use on resource-limited hardware. For instance, the Artificial Intelligence Unit (AIU) family of accelerators [24], developed by IBM, showcases hardware designed for high-performance AI workloads with an emphasis on energy efficiency. Similarly, companies such as NVIDIA, Google (TPU), Intel (Habana Labs), Graphcore, and Cerebras Systems are developing custom AI chips tailored for specific use cases. These innovations focus not only on improving performance but also on scalability and energy efficiency through techniques such as low-precision computation, hardware-software co-design, architectural specialization, and advanced cooling solutions. These advancements are pivotal for sustainable AI scaling and deploying models in real-world applications.

Recently, the Transformer architecture has expanded its reach into the time series forecasting domain [15, 27, 31, 34]. However, Zeng et al. [30] revealed several limitations of transformers when applied to time series data and showed that a simple linear model outperformed existing transformer models. Inspired by the patching techniques used in Vision Transformers (ViT) [8] and other works such as BEiT [2], Nie et al. [19] developed the Patch Time Series Transformer (PatchTST). Their simple patching technique successfully addressed the limitations of time series transformers described in [30], achieving SOTA performance. PatchTST has proven effective for many time series forecasting tasks, including predicting electricity consumption, electricity transformer temperatures, weather and traffic conditions, and disease infection rates [19]. The model can also be used for anomaly detection, predicting simulation completion time in high-performance (HPC) environments, and routing decisions in HPC networks. In edge scenarios, PatchTST can be utilized for predictive maintenance in IoT-connected devices, real-time monitoring of industrial equipment, energy optimization in smart grids, and adaptive control in autonomous systems. Compressing PatchTST will enable these predictive capabilities to be deployed on the resource-constrained hardware and in complex HPC environments, aligning with our ultimate goal.

With the vast amount of times series data being generated daily, time series forecasting is becoming more resource intensive and

less energy efficient. Quantization and reduced precision are increasingly popular techniques for compressing transformer models. These methods are widely used to speed up deep neural networks on GPUs and other AI hardware platforms [1] and [25]. This work explores the nascent fields of time series transformer quantization and parallelizing both full precision floating-point 32-bit (FP32) and quantization-aware (QAT) training of a time series transformer for efficient time series forecasting. Specifically, we combine scaling and quantization of PatchTST, quantizing it down to 4 bits per parameter, using common quantization techniques, and scaling QAT over multiple GPUs, enabling substantial model compression and improved training while maintaining competitive mean-squared error (MSE) performance. Our study provides insights into the effects of quantization on both model size and inference accuracy, advancing the understanding of quantization in time series forecasting transformers. A key challenge we addressed is the computational cost of training large PatchTST models, which necessitated parallelization across multiple GPUs. However, we discovered that standard data-parallel training led to a significant degradation in model accuracy. To overcome this, we used a combination of techniques including gradient clipping and learning rate scaling. These techniques not only mitigate the accuracy degradation in parallel training but also enhance the accuracy of the FP32 and quantized PatchTST (QPatchTST) models. To the best of our knowledge, this work provides the first comprehensive scaling analysis of a time series transformer and its quantized variants. We make the following contributions:

(1) We first analyze the performance of FP32 PatchTST to multi-GPU training, analyzing the effects of the linear, square root, and no learning rate scaling on inference accuracy. Initially, we observed varying rates of MSE degradation as we increased the number of GPUs. Utilizing gradient clipping and learning rate scaling, we stabilize training which allowed successful scaling over 128 GPUs with no degradation in MSE, yielding a speedup of 81× over single-GPU training.
(2) We quantize PatchTST down to 8 bits (INT8) and 4 bits (INT4). When trained on a single GPU, INT8 QPatchTST maintained ISO MSE compared to the FP32 model, while INT4 QPatchTST suffered a marginal 1.7% degradation in MSE.
(3) We devise a new learning rate scaling rule, *Square Root 2 (Sqrt2) Scaling Rule*, by modifying the *Square Root (Sqrt) Scaling Rule*, to aid QAT scaling. Using *Sqrt2 Scaling Rule* we successfully scale training of INT8 QPatchTST over 128 GPUs, resulting in a 96× speedup with no degradation in MSE compared to both the single-GPU trained FP32 and INT8 models. Additionally, we achieve successful INT4 QAT scaling over 80 GPUs, resulting in a 65× speedup in training latency, with a minimal 2.6% and 0.9% degradation in MSE compared to the single-GPU trained FP32 and INT4 QPatchTST models, respectively. *Sqrt2 Scaling Rule* also led to additional training stability and more consistent MSE performance when used to train our FP32 model.
(4) Our QPatchTST models demonstrate significant memory efficiency, with the INT8 variant reducing model size by 74% (3.9× compression) and the INT4 variant achieving an 87% reduction (7.5× compression).

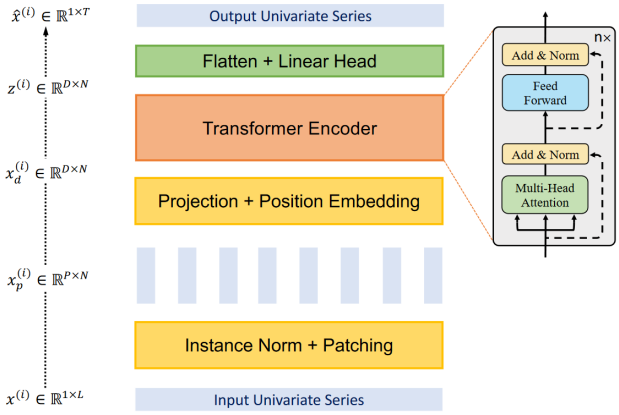


Figure 1: Transformer Backbone (Supervised). Each univariate series is passed through instance normalization operator and segmented into patches, which are used as Transformer input tokens.

2 Background

2.1 PatchTST Model

PatchTST is a state-of-the-art (SOTA) transformer-based model for multivariate time series forecasting and representation learning [19]. Given a collection of multivariate time series samples with look-back window $L : (x_1, ..., x_L)$, where each x_t at time step t is a vector of dimension M, we attempt to forecast T future values $(x_{L+1}, ..., x_{L+T})$. Figure 1 gives an overview of PatchTST, where the vanilla Transformer encoder forms its core architecture. As illustrated in Figure 1, the model makes use of the vanilla Transformer encoder as its core architecture. A multivariate time series is first divided into a set of univariate time series which share weights and embedding. The univariate time series are then divided into (possibly overlapping) patches, which are fed into an embedding layer followed by a transformer encoder. Patching reduces the number of input tokens from L to approximately L/S, where S is the stride, i.e., the non-overlapping regions between consecutive patches. Patching retains local semantic information in the embedding, yet quadratically reduces computation and memory usage of the attention maps given the same look-back window, which enables the use of longer history at the same computational cost.

We study PatchTST/64 (Supervised) [19] (referred to as PatchTST throughout the remainder of this study), which uses 64 patches with a look-back window $L = 512$. We set patch length $P = 16$ and stride $S = 8$. We use the default architecture for the three largest datasets studied in [19], consisting of three encoder layers, each with $H = 16$ heads and latent space of dimension $D = 128$. In the feed-forward network, one linear layer projects the hidden representation $D = 128$ to a new dimension $F = 256$. Another linear layer then projects it back to $D = 128$. PatchTST also achieved SOTA performance in time series representation learning. We focus

exclusively on the model's *supervised learning* capabilities given that our target applications will rely on supervised learning.

3 Data Distributed Parallel for Efficient Training

In order to improve both FP32 and QAT training efficiency, we parallelize our models and train over multiple GPUs using Pytorch's Data Distributed Parallel (DDP) library [16]. DDP splits the training data into batches that are distributed across GPUs. After each forward pass of training, DDP gathers the losses and gradients from each GPU onto a single GPU, which is used to perform the backward pass. During the backward pass, DDP synchronizes gradients across the GPUs (processes) to ensure consistent model updates. Using the synchronized gradients, each GPU/process independently updates its local model parameters, during the optimization step. The effective batch size (*ebs*) for the entire GPU ensemble is given by $ebs = b \times g$, where b is the per GPU batch size and g is the number of GPUs. Because PatchTST is small enough to fit comfortably on one of our GPUs, DDP is a convenient way to parallelize training, and more aggressive model-parallel approaches are not required.

3.1 Learning Rate Scaling

As mentioned above, in DDP training, each GPU processes a portion of the overall batch (*ebs*). As such, it requires scaling of the learning rate to achieve convergence similar to that expected for training on a single GPU. Goyal et. al. [10] suggested a hyperparameter free *Linear Scaling Rule*, which allowed effective data parallel training without reducing the per GPU workload. This rule scales the learning rate by:

$$\eta_{new} = \eta \cdot g, \tag{1}$$

where η and η_{new} are the initial and new (scaled) learning rates, respectively, and g is the total number of GPUs. This scaling rule is suitable for our work, as the goal is to improve training efficiency, and maintaining per GPU workload results in reduced training latency. With larger *ebs*, data parallelism is also efficient, since gradient synchronization occurs once.

Another learning rate scaling rule is the *Square Root Scaling Rule* suggested by Krizhevsky et. al. [14] and updates the learning rate by:

$$\eta_{new} = \eta \cdot \sqrt{g}. \tag{2}$$

This rule preserves the variance of the gradients computed on each GPU. We compare the effectiveness of these scaling rules on model training in Section 5.

3.2 Gradient Clipping

By itself, learning rate scaling is not sufficient to maintain baseline performance. As such, we also use gradient clipping [32]. Gradient clipping prevents the exploding gradients we would expect during scaling by limiting the maximum magnitude of the gradients. We use *Clip-by-norm*, which scales the entire gradient vector if its *L2-norm* exceeds a given threshold θ. As we show in Section 5, when combined with learning rate scaling, gradient clipping significantly improves the generalizability of the multi-GPU trained model. In DDP, gradient clipping occurs after gradient synchronization process. During the optimization step, the optimizer uses the clipped gradients to update the model parameters.

3.3 Modified Square Root Scaling

Our experiments revealed that *Linear Scaling* is not suited for INT4 QAT scaling (see Figure 6), and our INT8 QPatchTST model showed some instability at higher GPU counts, $g \geq 112$, while the FP32 model exhibited marginal variability. On the other hand, we observed that FP32 models trained using *Square Root* scaling maintained relatively consistent inference MSE with increasing GPUs for all gradient clipping levels θ studied, although general inference MSE increased with increasing GPUs and remained worse than with *Linear* scaling. Based on this observation, we hypothesized that the aggressive learning rate increase produced by *Linear* scaling is essential for low MSE, while *Square Root* scaling is essential to limit variability. We thus modified the *Square Root Scaling Rule* to allow it to aggressively scale similar to the *Linear Scaling Rule* but maintain its ability to limit variability. Our new scaling rule, *Square Root 2 Scaling Rule*, scales the learning rate using the following:

$$\eta_{new} = \eta \cdot \sqrt{\gamma \cdot b \cdot g}, \tag{3}$$

where $\gamma \geq 1$ is a scaling factor that controls the rate of growth. We show the effectiveness of this new scaling rule in Section 5.4.

4 Quantization

4.1 Background

Quantization is a popular model compression technique that converts weights and activation from high-precision, usually 32-bit floating-point (FP32) or 16-bit floating-point (FP16), to a lower precision, like 8-bit (INT8) or 4-bits (INT4). In simple terms, quantization maps a range of floating-point values to fixed-point values One of the main benefits of quantization is the significant reduction in model size. Quantized models require less memory and computational resources and may exhibit higher throughput/lower latency for inference if implemented on hardware that supports efficient arithmetic on the quantized numerical representation. Arithmetic on integer formats are also more power efficient. However, quantization typically reduces a model's inference accuracy relative to the full-precision representation due to rounding errors introduced during quantization.

4.1.1 Linear Quantization. In this work, we focus exclusively on linear (uniform) quantization and its variants. Linear quantization can be symmetric or asymmetric and implies a fixed step size, Δ. In symmetric linear quantization, the quantization range is symmetric around 0. Given a floating-point tensor x, the quantized tensor is given by

$$\hat{x} = round(x/\Delta) \cdot \Delta, \tag{4}$$

where $\Delta = 2 \cdot max|x| / 2^k - 1$. k is the number of bits. With asymmetric quantization, the distribution of quantized values is not symmetric around 0. The step size is given by $\Delta = x_{max} - x_{min} / 2^k - 1$, and the quantized version of x is given by

$$\hat{x} = round(x/\Delta - z) \cdot \Delta + z, \tag{5}$$

where z is the zero point, i.e., the integer value that corresponds to the floating-point zero.

4.1.2 Quantization-Aware Training (QAT). Quantization can be done by post-training quantization (PTQ) or quantization-aware training (QAT). With PTQ, a trained floating-point model is quantized to a lower-precision fixed-point integer model for inference without retraining or fine-tuning. Since PTQ is applied to a fully trained model, only a limited amount of training data is required for calibration, making PTQ fast and convenient. However, PTQ leads to accuracy degradation, which can be mitigated by further refining of the quantized model using QAT.

When performing QAT, quantized weights and activations are incorporated during the training process to *simulate* low-precision training in the forward pass, while the backward pass remains in full-precision. This allows the optimization process to minimize the loss and quantization error. Usually, QAT is initialized from an existing trained full-precision model for a small number of epochs to allow the model to adjust to the quantized weights. This method improves the accuracy of the quantized model. A more challenging quantization initialization method is *from scratch*—QAT is performed on an untrained full-precision model. QAT *from scratch* is the main focus of this study, as we aim to show that a lower-precision PatchTST model can be used as a surrogate model for many HPC tasks without the need for a full-precision checkpoint. As a result, developers can focus exclusively on training their low-precision models, thus reducing resource and energy requirements. We discuss our quantization methods in the next section.

4.2 Quantized PatchTST (QPatchTST)

We quantize the weights and activations of the linear layers in the Transformer Encoder and Flatten + Linear Head, using the methods described in Sections 4.2.1 and 4.2.2. Figure 2 shows the layers and operations that are quantized. When quantizing ML models, it is customary to leave the first and last layers in a higher precision, i.e., FP32 or FP16, because of their high impact on accuracy and negligible contribution to the overall model size. However, we only ignore PatchTST's first layer, the *Projection-Position Embedding*. Due to PatchTST's design, its last layer, the *Flatten+Linear Head*, contributes the majority of the total model parameters. We quantize this layer to achieve further memory savings. We show our quantization results in Section 5.3 and give more details on theoretical memory and computation savings in Section 5.5.

4.2.1 Weight Quantization. We use **SAWB+** for weight quantization [26]. SAWB+ is an enhancement of the SAWB (statistics-aware weight binning) quantizer [5], which exploits the first and second moments of the weight distribution to minimize the quantization error. SAWB clamps sufficiently large weights, which can cause instability in training because their gradient information is lost; SAWB+ allows the gradients of the clipped weights to propagate through the backward pass, while the forward pass remains unchanged. For **SAWB+**, we used the coefficients from [26].

As in [26], we use **Zero alignment** to ensure that floating-point zeros from inputs are represented by fixed-point zeros. This is done by utilizing one less than the full precision range, i.e., $2^k - 1$ levels instead of the full 2^k levels. Zero alignment keeps the distribution

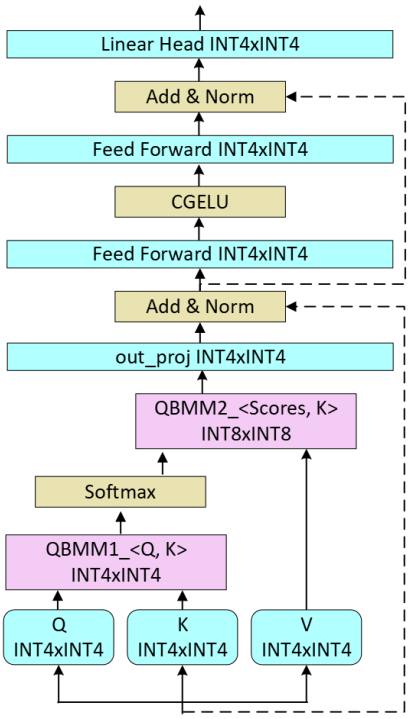


Figure 2: Quantized PatchTST model, showing INT4 quantization. The Transformer encoder block consists of linear and MatMul modules. All linear modules—QKV layers, two feed forward layers, and an output projection layer—and MatMul (BMM) modules are quantized. The 2nd MatMul module, MatMul2 (BMM2) is quantized to 8-bits for both INT8 and INT4 quantization.

symmetric around zero. Zero alignment is desirable for eventual deployment of the quantized model because it permits easy hardware implementation with simplified multiply-accumulate (MAC) design and efficient data-flow [13].

4.2.2 Activation Quantization. For activation quantization, we use the **PACT+** quantizer [11]. PACT+ is an extension of the PACT2 quantizer [26]. We use **PACT2** to quantize BMM operations, and the **MinMax** quantizer to quantize the positively bounded BMM2 operation. For the PACT quantizers, α and α_n define the dynamic range. They are parameterized and dynamically adjusted by gradient descent-based training, and the clamped activation output is uniformly quantized to k bits for dot-product computation. They are initialized by sampling 5 batches of the dataset of each layer based on a given percentile of the observed activation distribution [5]. PACT+ differs from PACT2 in that it scales the gradients by a factor $\eta = (z - round(z))/(2^k - 1)$, where $z = x - \alpha_n/\Delta$ is the zero point, Δ is the step size, and k is the number of bits. For the MinMax quantizer,α and α_n take the min and max of the tensors. We also use zero alignment to preserve zero-value activations; however, unlike for weights, we utilize the full 2^k levels as in [26].

5 Experimental Setup and Results

5.1 Overview

5.1.1 Dataset. We evaluate the scaling performance of FP32 training and QAT of PatchTST on the Traffic[1] dataset. This dataset is one of the multivariate datasets commonly used for benchmarking time series forecasting tasks [27]. It provides the road occupancy rates from different sensors on San Francisco freeways and was the largest studied in [19]. With 862 features and over 17544 time steps, it is less prone to overfitting and requires significant computational resources to train, thus making it suitable for our study. We use a {70, 10, 20} % split for training, validation, and test sets for all experiments.

5.1.2 Training Settings and Baselines. We perform multivariate long-term forecasting over prediction length $T = 96$ and calculate the mean-squared error (MSE) as our evaluation metric. A decrease in MSE indicates an improvement in accuracy, while an increase indicates a degradation in accuracy. We evaluate the performance of our FP32 training of PatchTST over multiple GPUs, comparing the learning rate scaling rules described in Section 3 and the default case where no LR scaling is used. Additionally, we combine LR scaling with gradient clipping and compare the performance with different norm clip values, θ. We compare these results to the inference (test) MSE of a single-GPU trained model. For quantization and QAT scaling, we compare performance to the FP32 model and the single-GPU trained quantized models.

For FP32 training, we use the default training settings from [19]. We use the Adam optimizer with learning rate of 1×10^{-4} and a One-cycle learning rate (OCLR) schedule with a percentage start of 0.2[2]. We train for a total of 100 epochs, which was sufficient to achieve convergence, and a dropout rate of 0.2. For QAT, we also use Adam optimizer with learning rate of 1×10^{-4} but with a Cosine annealing learning rate (CALR) schedule, and all quantized models—INT8 and INT4—were initialized from "scratch". For the PACT+ quantizer, we use a learning rate (α_lr) of 1×10^{-4}, with L2 decay (α_{decay}) of 5×10^{-5}, and initialize α and α_n from the 99.9 percentile. Because our NVIDIA V100 GPUs lack low-precision capabilities, we must perform QAT and quantized inference in emulation. This adds additional parameters during training and inference, thus limiting the maximum batch size, b. To get a fair comparison between our FP32 and QPatchTST models, we use the highest allowable batch size, $b = 6$, for FP32 and quantized training and inference.

5.1.3 Implementation Details. We adapted and modified the code base from [19], which is publicly available on GitHub, to support DDP training on our computing platform, Artificial Intelligence Multiprocessing Optimized System (AiMOS[3]). We implemented QPatchTST using IBM's Foundation Model Stack package [11]. All experiments were performed on AiMOS' NPL cluster which consists of 40 compute nodes interconnected with dual 100Gb/s Infiniband links configured in a Fat-Tree network topology. Each compute node has 768 GB of DRAM, 2 20-core, 2.5 GHz Intel Xeon Gold 6248 processors, and 8 NVIDIA Tesla V100 GPUs with 32 GiB High Bandwidth Memory (HBM). Both FP32 training and QAT were performed on one or more GPUs, while inference is performed on one GPU. All experiments were the average of three runs, unless otherwise stated.

[1]https://pems.dot.ca.gov/
[2]The percentage of total epochs the scheduler spends increasing the learning rate.
[3]https://cci.rpi.edu/aimos

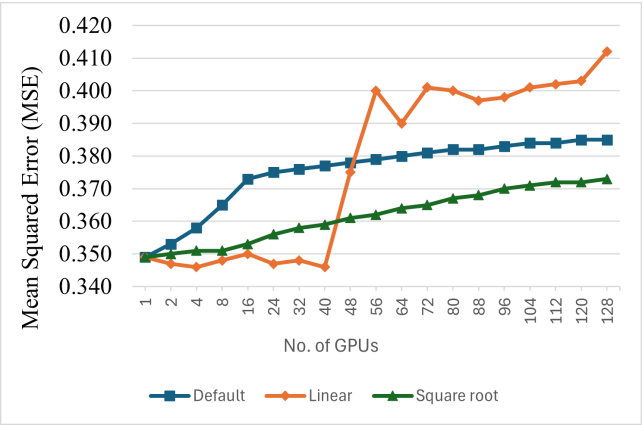
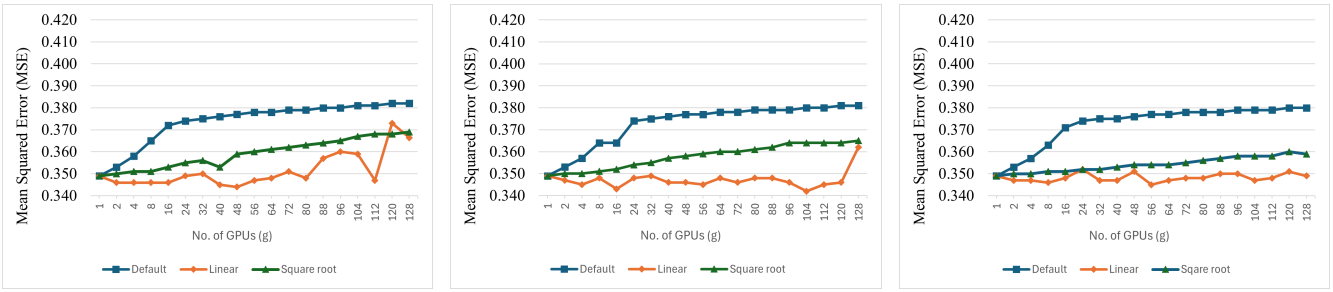


Figure 3: Multivariate long-term forecasting results for FP32 training of PatchTST scaled over up to 128 GPUs using *Linear Scaling Rule* (1), *Square Root Scaling Rule* (2), and no Scaling Rule (Default) without gradient clipping. We evaluate on the Traffic dataset for prediction length $T = 96$.

5.2 Scaling FP32 Training of PatchTST

We first analyze the robustness of FP32 PatchTST trained over multiple GPUs. Figure 3 compares the scaling performance of PatchTST with the learning rate scaling rules described in 3, and no LR scaling (Default) without gradient clipping. As shown, training with *Linear Scaling Rule* performed best up to 40 GPUs, after which our model became unstable and showed poor inference performance. To remedy this, we incorporate gradient clipping. Figure 4 show the performance of our model trained over multiple GPUs using learning rate scaling combined with gradient clipping with $\theta \in \{1.0, 0.5, 0.1\}$. These results show that gradient clipping with *Linear* scaling, (1), significantly improves MSE performance of the FP32 model trained on increasing number of GPUs and that decreasing θ stabilizes training. It is also worth noting that models trained with *Square Root* scaling and gradient clipping showed minimal variability with changing number of GPUs. We show the inference MSEs of multi-GPU training of the learning rate scaling rules and gradient clipping with $\theta = 0.1$ in Table 1.

With $\theta = 0.1$, we successfully scale FP32 training over 128 GPUs with no degradation in MSE compared to the single-GPU trained model, leading to an $\approx$ 81x speedup. We show the change in training latencies with increasing GPUs in Figure 5 and Table 3. We measure training latency as the end-to-end latency including training and validation data loading over 100 epochs. We average the training latencies for training with linear LR scaling with no gradient clipping and gradient clipping with $\theta \in \{1.0, 0.5, 0.1\}$. As is evident, training latency decreases exponentially with increasing number of GPUs, with the largest rate of decrease occurring moving from 1 to 2 GPUs. The rate of decrease drops with increasing number



(a) Gradient clip-by-norm $\theta = 1.0$ (b) Gradient clip-by-norm $\theta = 0.5$ (c) Gradient clip-by-norm $\theta = 0.1$

Figure 4: Multivariate long-term forecasting results for full precision training of PatchTST scaled over up to 128 GPUs using *Linear Scaling Rule* (1) and gradient clipping with (a) $\theta = 1.0$, (b) $\theta = 0.5$, and (c) $\theta = 0.1$. We evaluate on the Traffic dataset for prediction length $T = 96$.

Table 1: Multivariate long-term forecasting results of FP32 PatchTST trained on multiple GPUs with *Linear* (1) and *Square Root* (2) Scaling Rules, and no learning rate scaling (Default) and gradient clipping with $\theta = 0.1$.

Nodes	GPUs/ node	Total GPUs	Default	Linear	Square Root
1	1	1	0.349	0.349	0.349
1	2	2	0.353	0.347	0.350
1	4	4	0.357	0.347	0.350
1	8	8	0.363	0.346	0.351
2	8	16	0.371	0.348	0.351
3	8	24	0.374	0.352	0.352
4	8	32	0.375	0.347	0.352
5	8	40	0.375	0.347	0.353
6	8	48	0.376	0.351	0.354
7	8	56	0.377	0.345	0.354
8	8	64	0.377	0.347	0.354
9	8	72	0.378	0.348	0.355
10	8	80	0.378	0.348	0.356
11	8	88	0.378	0.350	0.357
12	8	96	0.379	0.350	0.358
13	8	104	0.379	0.347	0.358
14	8	112	0.379	0.348	0.358
15	8	120	0.380	0.351	0.360
16	8	128	0.380	0.349	0.359

of GPUs. This can be attributed to the increase in communication overheads, especially with increasing compute nodes. We will study the cause of the bottlenecks in future studies.

5.3 Quantized PatchTST and QAT Scaling

We evaluate the viability of PatchTST under 8- and 4-bit quantization. We quantize PatchTST using **SAWB+** for weights, **PACT+** for activations, **PACT2** for BMM operations, and **MinMax** for the positively bounded BMM2 operation (see Section 4.2). We compare inference MSE to the FP32 model. We also scale QAT of our QPatchTST models over multiple GPUs and compare inference MSE to the single-GPU trained FP32 and quantized models. For scaled

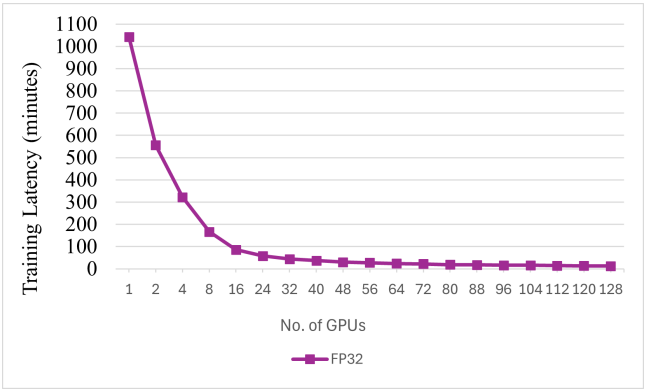


Figure 5: Training latency with increasing GPUs of (a) FP32 and (b) INT8 QAT. We measure end-to-end training latency, including training and validation dataset loading, over 100 epochs.

QAT, we use *Linear Scaling Rule* and gradient clipping with $\theta = 0.1$. In Section 5.4, we show how our modified *Square Root 2 Scaling Rule* improves stability of our multi-GPU trained QPatchTST models.

We show our results in Figure 6. When trained on a single GPU, INT8 QPatchTST maintained ISO MSE compared to the FP32 model. For multi-GPU training, INT8 QPatchTST maintained ISO up to 104 GPUs, with a $< 2\%$ degradation in MSE compared to the FP32 model trained on the same number of GPUs (104), and a $< \%1$ degradation in MSE compared to single-GPU trained FP32 and INT8 models. With 104 GPUs, we achieved QAT speedup of $\approx 82\times$ over training on a single GPU.

When trained on a single GPU with default QAT settings, INT4 QPatchTST suffered an $\approx 5\%$ degradation in MSE compared to the single-GPU trained FP32 model and experienced poor scaling performance with increasing GPUs. By increasing the PACT+ quantizer learning rate (α_{lr}) to 6×10^{-4}, we successfully recovered MSE of the single-GPU INT4 model to $\approx 1.7\%$ of the single-GPU trained FP32 model. However, this did not improve the scaling performance of the model, with the model performing $\approx 3.1\%$ worse when trained

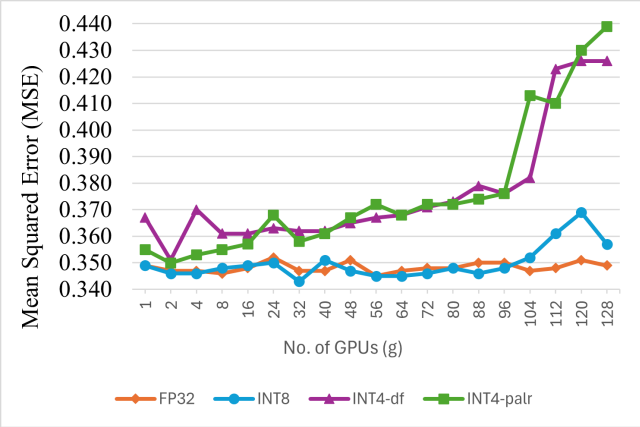


Figure 6: Multivariate long-term forecasting results for QAT scaling over up to 128 GPUs using *Linear* learning rate scaling and gradient clipping with $\theta = 0.1$. For INT4 QAT, we use $\alpha_{lr} = 1 \times 10^{-4}$ (INT4-df) and $\alpha_{lr} = 6 \times 10^{-4}$ (INT4-palr). We train on the Traffic dataset for prediction length $T = 96$.

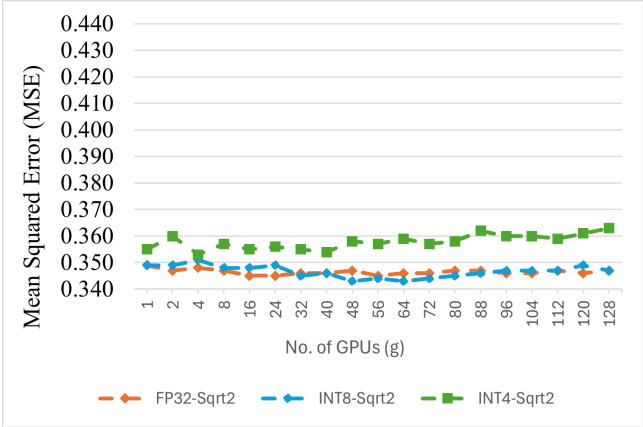


Figure 7: Multivariate long-term forecasting results for FP32 training and INT8 and INT4 QAT scaled over up to 128 GPUs using Square Root 2 Scaling Rule(3) with gradient clipping $\theta = 0.1$. We train on the Traffic dataset for prediction length $T = 96$.

on $g = 128$ GPUs compared to the default settings. We show improvements in INT4 QAT scaling using *Sqrt2 Scaling Rule* instead of *Linear Scaling Rule* for learning rate scaling in Section 5.4.

5.4 Square Root 2 Scaling Rule for Improved Multi-GPU Training

To improve scaling performance of our INT4 QPatchTST model, we perform QAT using our modified *Square Root 2 (Sqrt2) Scaling Rule* (3) with $\alpha_{lr} = 6 \times 10^{-4}$ and $\gamma = 2$. As seen in Figure 7, these adjustments allowed successful scaling of INT4 QAT over up to 80 GPUs with only a 2.6% and 0.8% degradation in MSE compared to the single-GPU trained FP32 and INT4 models, respectively. With 80 GPUs, we achieve a 65× speedup in training latency over single-GPU training. *Sqrt2* learning rate scaling also improved the stability of multi-GPU FP32 and INT8 (QAT) training, eliminating the instability in our INT8 model trained on $g \geq 112$. As a result, we achieve INT8 QAT on $g = 128$ GPUs with no degradation in MSE, leading to a 96× speedup in training latency[4]. Both the FP32 and INT8 models show more consistency with increasing GPUs, maintaining inference MSE at ≤ 0.349. Table 2 show the inference MSE of our FP32 and QPatchTST models trained on multiple GPUs using *Sqrt2* learning rate scaling and gradient clipping with $\theta = 0.1$. For INT4 QAT, $\alpha_{lr} = 6 \times 10^{-4}$.

Figure 8 and Table 3 show the change in training latency with increasing GPUs for QAT. We average the training latency of both INT8 and INT4 QAT, since both are emulated using the same loaded quantization parameters. As such, training latency for both are generally the same, with small variations. Like FP32 scaling, training latency decreases exponentially with increasing GPUs. It is worth noting the increase in training latency for QAT compared to the FP32 training. This is because our V100 GPUs do not support low-precision arithmetic, as explained in Section 5.1.2. As such, QAT and quantized inference were emulated and the additional parameters needed increased latency. We expect to see improved (reduced) training and inference latencies when experimenting on hardware with low-precision support in future iterations of this work.

[4]The speedup is due to training on increasing number of GPUs rather than quantization, since the V100s do not support low-precision arithmetic

Table 2: Multivariate long-term forecasting results of our FP32 and QPatchTST models trained on multiple GPUs using *Square Root 2 (Sqrt2) Scaling Rule* and gradient clipping with $\theta = 0.1$. INT4 QPatchTST is trained with $\alpha_{lr} = 6 \times 10^{-4}$.

Nodes	GPUs/ node	Total GPUs	FP32 Sqrt2	INT8 Sqrt2	INT4 Sqrt2
1	1	1	0.349	0.349	0.355
1	2	2	0.347	0.349	0.360
1	4	4	0.348	0.351	0.353
1	8	8	0.347	0.348	0.357
2	8	16	0.345	0.348	0.355
3	8	24	0.345	0.349	0.356
4	8	32	0.346	0.345	0.355
5	8	40	0.346	0.346	0.354
6	8	48	0.347	0.343	0.358
7	8	56	0.345	0.344	0.357
8	8	64	0.346	0.343	0.359
9	8	72	0.346	0.344	0.357
10	8	80	0.347	0.345	0.358
11	8	88	0.347	0.346	0.362
12	8	96	0.346	0.347	0.360
13	8	104	0.346	0.347	0.360
14	8	112	0.347	0.347	0.359
15	8	120	0.346	0.349	0.361
16	8	128	0.347	0.347	0.363

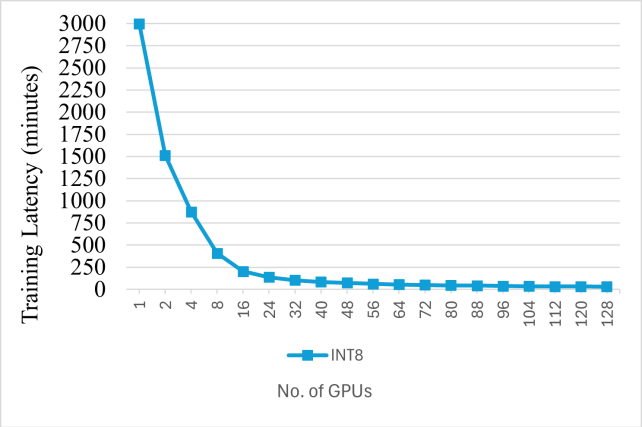


Figure 8: Training latency with increasing GPUs of QAT. We measure end-to-end training latency, including training and validation dataset loading, over 100 epochs.

Table 3: Training latency with increasing number of GPUs.

			FP32		QAT	
Nodes	GPUs/ node	Total GPUs	Latency (min)	SpUp (×)	Latency (min)	SpUp (×)
1	1	1	1042.92	-	2997.26	-
1	2	2	555.93	1.88	1513.42	1.98
1	4	4	322.66	3.23	876.31	3.42
1	8	8	166.10	6.28	407.87	7.35
2	8	16	85.64	12.18	203.27	14.75
3	8	24	58.18	17.93	137.78	21.75
4	8	32	44.24	23.57	104.64	28.64
5	8	40	37.40	27.89	85.45	35.08
6	8	48	30.17	34.57	75.59	39.65
7	8	56	27.56	37.84	62.88	47.67
8	8	64	23.86	43.71	55.9	53.62
9	8	72	22.06	47.28	50.24	59.66
10	8	80	18.88	55.24	46.06	65.07
11	8	88	17.97	58.04	42.49	70.54
12	8	96	16.53	63.09	39.22	76.42
13	8	104	15.73	66.30	36.65	81.78
14	8	112	14.36	72.63	34.63	86.55
15	8	120	13.52	77.14	32.68	91.72
16	8	128	**12.83**	**81.29**	**31.20**	**96.07**

5.5 Memory and Computation Savings of QPatchTST Models

The PatchTST model studied (PatchTST with prediction length $T = 96$) consists of ≈ 1.2 million parameters and consumes 38.22 MB of memory. The Transformer Encoder and the Flatten + Linear Head make up ≈ 99.3% of the total number of parameters. INT8 quantization reduces model size to 9.84 MB, a 74.3% reduction (3.9× compression) compared to the FP32 model. Additionally, INT4 quantization reduces model size by 86.6% to 5.11 MB, a 7.5× compression compared to the FP32 model.

In relation to computation savings, PatchTST consists of ≈ 625.85 million multiply-accumulate (MAC) operations, with the Transformer Encoder accounting for nearly 86%. Performing these operations in lower bit-width, fixed-point arithmetic would yield significant computation savings provided they are natively supported in hardware. Since we emulated quantization, our savings are theoretical. We intend to perform hardware evaluation of our QPatchTST models in future studies to confirm these savings.

6 Related Work

Transformer models have been successfully quantized using a variety of techniques, including post-training quantization (PTQ) [18], QAT [6], and quantizers, such as PACT [6], SAWB [5], and SAWB+ and PACT+ [26]. For NLP, FullyQT [21] and [3] developed an 8-bit transformer for machine translation. Q8BERT [29] quantized BERT [12] model to 8-bits. ZeroQuant [28] quantized the weights of BERT down to 4-bits, but kept activations in 8-bits. Weight quantization was pushed even lower with TernaryBERT [33] achieving 3-bit quantization, and BiBERT [22] achieving 2-bit quantization. These studies kept activations in 8-bits. Other Transformer models have also been quantized. Liu et al. [18] quantized Vision Transformers down to 8-bits using PTQ. Bie et al. [4] applied 8-bit weight quantization to a simplified encoder-decoder transformer for end-to-end ASR task. In addition, Wang et al. [26] achieved 4-bit weight and activation quantization on NLP, Vision, and Speech transformers.

Despite the abundance of work related to the quantization of LLMs, the quantization of time series transformers has been relatively unexplored. Ling et al. [17] proposed a quantization scheme that dynamically selects between symmetric and asymmetric quantization to quantize a time series transformer model for FPGAs. They performed mixed-precision quantization, with the output layer (last layer) quantized to 8-bits while the other layers were quantized to 4-bits. In addition, their work supports only single-step-ahead time series forecasting. On the contrary, we quantize all layers except the Projection + Position Embedding to 8 or 4 bits and perform multivariate time series forecasting for up to 96 future steps. Other works like Works like VQ-TR [23] use vector quantization (VQ) on time series transformers for forecasting. VQ involves mapping continuous values to discrete values and storing them in a code book represented in higher bits, e.g., 32-bits, making it a different kind of model compression than what we are pursuing. We are focused on compression that targets the architectural features (low bit-width integers and arithmetic) of the target hardware, rather than reducing the number of distinct coefficients.

7 Conclusion and Future Work

With the continuous growth of time series data, the computational and memory requirements needed to train time series forecasting models will only continue to grow. We investigated the robustness of PatchTST and quantized PatchTST (QPatchTST) to parallelization. Our work demonstrated the successful scaling of FP32 training

over 128 GPUs, leading to significant reduction in training latency. We introduce a new learning rate scaling rule, *Square Root 2 Scaling Rule*, that enables successful QAT scaling of INT8 and INT4 QPatchTST over 128 and 80 GPUs, respectively, with no MSE degradation for INT8 QPatchTST and only minimal MSE degradation for INT4 QPatchTST. Our QPatchTST models are considerably smaller in size than the FP32 model, resulting in significant memory savings. The simultaneous compression and parallelization results in improved training efficiency. We will verify these savings with future investigation into the scaling of QPatchTST QAT over multiple GPUs with low-precision capabilities, specifically the Empire AI[5] computing platform. We will identify larger and more complex datasets, for e.g., datasets generated by the DOE's Co-Design of Exascale Storage Architectures (CODES) parallel simulation framework [7] and Deep Underground Neutrino Experiment (DUNE) workflows [20], to support expanding QPatchTST training beyond 128 GPUs. This analysis will be valuable for the development of algorithms for improved QAT scaling performance where applicable. Finally, we will extend our work to the newer Tiny Time Mixers [9] time series forecasting models in the future.

Acknowledgments

This work was supported by IBM through the IBM-Rensselaer Future of Computing Research Collaboration.

References

[1] Ankur Agrawal, Sae Kyu Lee, Joel Silberman, Matthew Ziegler, Mingu Kang, Swagath Venkataramani, Nianzheng Cao, Bruce Fleischer, Michael Guillorn, Matthew Cohen, et al. 2021. 9.1 A 7nm 4-core AI chip with 25.6 TFLOPS hybrid FP8 training, 102.4 TOPS INT4 inference and workload-aware throttling. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 144–146.

[2] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. 2021. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254* (2021).

[3] Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532* (2019).

[4] Alex Bie, Bharat Venkitesh, Joao Monteiro, Md Akmal Haidar, and Mehdi Rezagholizadeh. 2019. A Simplified Fully Quantized Transformer for End-to-end Speech Recognition (2019). *URL https://arxiv. org/abs/1911.03604. Google Scholar Google Scholar Cross Ref Cross Ref* (2019).

[5] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. 2019. Accurate and efficient 2-bit quantized neural networks. *Proceedings of Machine Learning and Systems* 1 (2019), 348–359.

[6] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* (2018).

[7] J. Cope et al. 2011. CODES: Enabling co-design of multilayer exascale storage architectures. In *Proc. of the Workshop on Emerging Supercomputing Technol.*

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[9] Vijay Ekambaram, Arindam Jati, Pankaj Dayama, Sumanta Mukherjee, Nam H Nguyen, Wesley M Gifford, Chandra Reddy, and Jayant Kalagnanam. 2024. Tiny Time Mixers (TTMs): Fast Pre-trained Models for Enhanced Zero/Few-Shot Forecasting of Multivariate Time Series. *CoRR* (2024).

[10] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2019. Accurate, large minibatch sgd: Training imagenet in 1 hour. arXiv 2017. *arXiv preprint arXiv:1706.02677* (2019).

[11] IBM Research. [n. d.]. Foundation Model Stack (FMS): FMS Model Optimizer. github.com/foundation-model-stack/fms-model-optimizer

[12] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. I-bert: Integer-only bert quantization. In *International conference on machine learning*. PMLR, 5506–5518.

[13] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).

[14] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997* (2014).

[15] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems* 32 (2019).

[16] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proc. VLDB Endow.* 13, 12 (2020), 3005–3018. doi:10.14778/3415478.3415530

[17] Tianheng Ling, Chao Qian, Lukas Einhaus, and Gregor Schiele. 2023. A Study of Quantisation-aware Training on Time Series Transformer Models for Resource-constrained FPGAs. *arXiv preprint arXiv:2310.02654* (2023).

[18] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. 2021. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems* 34 (2021), 28092–28103.

[19] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *International Conference on Learning Representations*.

[20] Department of Energy. [n. d.]. Tachyon Project Summary. https://tachyon-org.github.io. Accessed: 20 Mar 2025.

[21] Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. 2019. Fully quantized transformer for improved translation. *arXiv-1910.10485v3* (2019).

[22] Haotong Qin, Yifu Ding, Mingyuan Zhang, Qinghua Yan, Aishan Liu, Qingqing Dang, Ziwei Liu, and Xianglong Liu. 2022. Bibert: Accurate fully binarized bert. *arXiv preprint arXiv:2203.06390* (2022).

[23] Kashif Rasul, Andrew Bennett, Pablo Vicente, Umang Gupta, Hena Ghonia, Anderson Schneider, and Yuriy Nevmyvaka. 2024. VQ-TR: Vector Quantized Attention for Time Series Forecasting. In *The Twelfth International Conference on Learning Representations*.

[24] IBM Research. [n. d.]. The AIU Chip Family. https://research.ibm.com/blog/aiu-chip-family-ibm-research. Accessed: 18 Nov 2024-11-18.

[25] Swagath Venkataramani, Vijayalakshmi Srinivasan, Wei Wang, Sanchari Sen, Jintao Zhang, Ankur Agrawal, Monodeep Kar, Shubham Jain, Alberto Mannari, Hoang Tran, et al. 2021. RaPiD: AI accelerator for ultra-low precision training and inference. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 153–166.

[26] Naigang Wang, Chi-Chun Charlie Liu, Swagath Venkataramani, Sanchari Sen, Chia-Yu Chen, Kaoutar El Maghraoui, Vijayalakshmi Viji Srinivasan, and Leland Chang. 2022. Deep Compression of Pre-trained Transformer Models. *Advances in Neural Information Processing Systems* 35 (2022), 14140–14154.

[27] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in neural information processing systems* 34 (2021), 22419–22430.

[28] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems* 35 (2022), 27168–27183.

[29] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 36–39.

[30] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are transformers effective for time series forecasting?. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 11121–11128.

[31] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. 2021. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 2114–2124.

[32] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. 2019. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881* (2019).

[33] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812* (2020).

[34] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 11106–11115.

[5] https://www.empireai.tech/