Scientific Research Publishing

Journal of Computer and Communications, 2025, 13(12), 56-79
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Time Series Models for Predicting Application GPU Utilization and Power Draw

Evan Coleman[1], Dorothy X. Parry[2], Masha Sosonkina[2]

[1]Department of Computer Science, University of Mary Washington, Fredericksburg, USA
[2]Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, USA
Email: ecolema4@umw.edu, dparr005@odu.edu, msosonki@odu.edu

How to cite this paper: Coleman, E., Parry, D.X. and Sosonkina, M. (2025) Time Series Models for Predicting Application GPU Utilization and Power Draw. *Journal of Computer and Communications*, **13**, 56-79.
https://doi.org/10.4236/jcc.2025.1312004

Received: October 4, 2025
Accepted: December 16, 2025
Published: December 19, 2025

Abstract

This paper explores the application of various time series prediction models to forecast graphical processing unit (GPU) utilization and power draw for machine learning applications using data sets arising from two distinct but representative deep learning inference workloads: the architecturally diverse Inception-v3 and the industry-standard MLPerf ResNet-50. We investigate the use of statistical models, Recurrent Neural Networks (RNNs), and Transformer Neural Networks (TNNs). Our results show that RNNs outperform other models for GPU utilization prediction, while TNNs excel in power draw prediction. These findings may contribute to the development of energy-efficient strategies in high-performance computing environments equipped with GPUs.

Keywords

Time Series Prediction, Performance Modeling, GPU Utilization, Power Draw, Machine Learning, High-Performance Computing

1. Introduction

The increasing demand for computationally expensive and memory-intensive applications has led to a growing need for efficient utilization of Graphics Processing Units (GPUs). Predicting GPU performance metrics, such as utilization and power draw, can provide valuable insights for implementing energy-saving strategies in high-performance computing environments. This paper investigates various time series prediction models to forecast these metrics for two canonical convolutional neural network (CNN) models for image classification: Inception-v3 [1], and MLperf Inference ResNet-50 [2]; inference-time forecasts of GPU utilization and power enable proactive scheduling and power-capping that reduce tail-

latency and energy cost without sacrificing accuracy. In particular, this work makes progress towards utilizing accurate predictions on GPU utilization and power draw to provide energy savings. The two categories of prediction that succeeded best were: Recurrent Neural Networks (RNNs) and Randomized Transformer Neural Networks (TNNs).

Time series predictive algorithms assume that the data is input in a sequential temporal order. This temporal order lists data at regular intervals of time. Time series predictive algorithms use a certain number of past values to predict future values.

This study is designed to explore various time series prediction algorithms and study the accuracy of their prediction results on GPU utilization and power draw traces of parallel GPU machine learning applications. To evaluate the GPU's power and utilization characteristics, two widely recognized deep learning workloads for image classification were employed. These workloads were selected to represent different computational profiles:

- Inception-v3 [1]: A prominent CNN model known for its architectural complexity and computationally efficient "Inception modules", which involve multiple parallel convolutions and pooling operations within the same block. This serves as a test of the GPU's ability to handle diverse and concurrent kernel execution. Inception-v3 is implemented in parallel using TensorFlow 1.15.
- MLPerf Inference ResNet-50 [2]: A standardized benchmark workload using the ResNet-50 v1.5 model. As part of the MLPerf suite, this workload is designed to provide a reproducible and industry-accepted measure of system performance for a canonical deep learning task, emphasizing a more uniform computational graph dominated by sequential 3×3 convolutions.

The selected algorithms were strategically selected to test hardware performance against architectural diversity (Inception-v3) and an industry-standard performance baseline (MLPerf ResNet). The trace data is captured using nvidia-smi [3] and consists of two application performance metrics, GPU utilization and power draw, which are recorded during multiple runs of the selected algorithms. This work studied and implemented various time series predictive models, with a focus on applying neural networks such as a Recurrent Neural Networks (RNN) and Transformer Neural Networks (TNN) to the forecasting problem. The objective is to use the model's short-horizon forecasts to drive power-capping and scheduling decisions that improve energy efficiency, and the findings indicate a clear specialization, with RNNs proving most effective for utilization forecasting and Transformer-based models excelling at the more complex task of power draw prediction, providing a road map for hybrid, energy-aware runtime systems.

2. Background

There are two main classes of power consumption for GPUs, leakage of power and dynamic power. Leakage of power occurs when a GPU is powered (but not actively being used). Dynamic power is consumed based on runtime and it is a func-

tion of circuit technology and operating temperature [4]. A megawatt of power costs about one million dollars per year. To put that in perspective, the top-ranking supercomputer for November 2022's Green500 was Henri, which showed an energy efficiency of 65.09 GFlops/Watts [5]. This same report took note of Frontier, a supercomputer with a good balance between performance and energy. Its energy efficiency was 52.2 GFlops/Watts, with a Rmax (realistic maximum performance) value of 1102 PFlops/second. This was accomplished due in part to four purpose-built AMD Instinct 250X GPUs [5] [6]. Energy efficiency is important to both decrease costs and decrease the impact on the climate. Applying energy efficiencies comes with trade-offs in performance. However, these trade-offs can be minimized. One way to apply model efficiency, and thus energy efficiency, is by training on smaller representative data. However, this does require specialized domain knowledge [7].

2.1. Performance Measuring

Collecting application performance information is the first step towards investigating efficient execution. After understanding how an algorithm executes, metrics can track the improvements, or lack thereof, of the proposed optimizations implemented. nvidia-smi [3] is a light-weight utility that collects performance measures on parallel GPUs, such as memory, GPU utilization, power draw, and GPU temperature [3]. When picking an appropriate performance measuring tool, it is important to consider any implications on the performance of the application. nvidia-smi provides a basic overview of the performance statistics for each parallel GPU while minimizing effects on the performance [3]. Other performance profilers such as DLProf, PyProf, and NVIDIA Nsight Systems profiler provide more detailed statistics, such as wall GPU idle percentage and tensor core kernel efficiency that are outside the scope of this work, and thus, have more impact on the memory and GPU utilization [8] than nvidia-smi does so. With an already memory-intensive algorithm, using a profiler that requires memory would increase and introduce the possibility of a bottleneck that would skew the true performance values. Hence, collecting GPU utilization and power draw data herein employed nvidia-smi.

Evaluation Metrics

Comparing evaluation metrics provides insights into which model predicts the most accurately. Accuracy equates to how close a model's predictions are to the actual value. Evaluation metrics were collected on the different prediction models. The main two evaluation metrics collected were mean absolute error (*MAE*) and mean square error (*MSE*):

$$MAE = \frac{1}{N}\sum_{n=1}^{N}\left|p_n - \hat{p}_n\right|,\ MSE = \frac{1}{N}\sum_{n=1}^{N}\left(p_n - \hat{p}_n\right)^2,$$

where p_n is the ground truth (*i.e.*, actual) and $\hat{p}_n$ is the predictions [9]. These two commonly used metrics were deemed sufficient for the evaluation because

they enabled us to distinguish among the models and correlate with the prediction accuracy across the entire trace data.

2.2. Inception-v3 Algorithm

The Inception architecture, developed by Szegedy *et al.* [10], represents a significant advancement in the design of efficient Convolutional Neural Networks (CNNs). The selected iteration, Inception-v3, refines this architecture to further improve accuracy while minimizing computational cost. Its core innovation is the "Inception module", a network block that performs multiple convolution and pooling operations with different filter sizes (1 × 1, 3 × 3, 5 × 5) in parallel. The outputs from these parallel branches are then concatenated into a single feature map, allowing the model to capture information at multiple spatial scales simultaneously. Inception-v3 enhances this design by introducing factorized convolutions, which decompose larger filters into smaller, sequential ones (e.g., replacing a single 5 × 5 convolution with two consecutive 3 × 3 convolutions), and by extensively using batch normalization. These modifications reduce the number of parameters and computational complexity, making Inception-v3 a powerful and efficient model for image classification tasks [10].

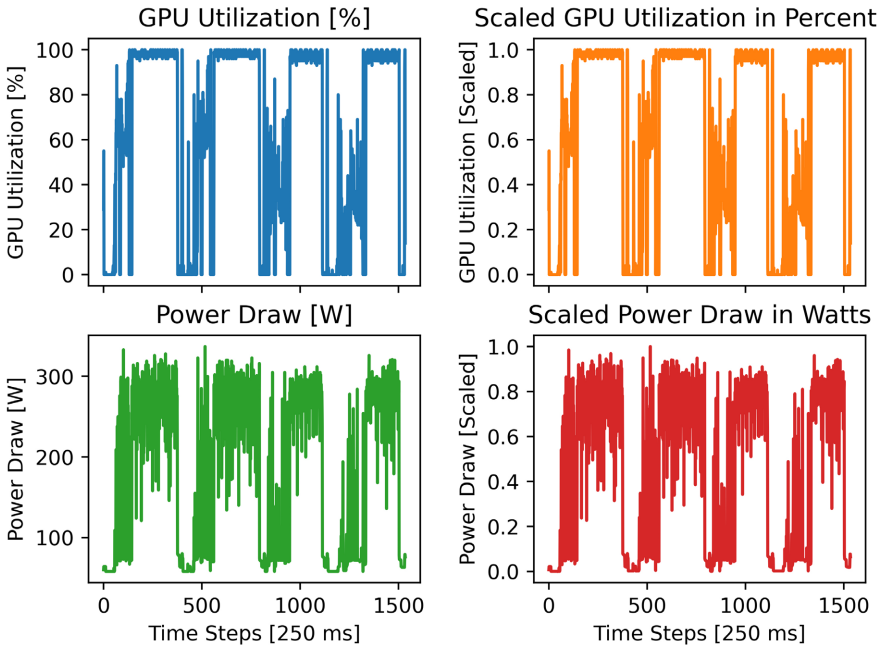


Figure 1. Typical output for the GPU utilization and power draw for GPU 1 both in the original units (on the left side) and after using MinMaxScaler [11] (on the right side).

The parallelization of Inception-v3 is handled by tf_cnn_benchmarks, which defines how variables and gradients are implemented across multiple GPUs [10]. The trace for this application consists of the performance measures-GPU utilization, and power draw-recorded with respect to time (see **Figure 1** (left column)). Scaling was performed on the input trace, except as otherwise specified, using

MinMaxScaler, which shifts and scales the data so that the data ranges from zero to one while retaining the original shape of the data set [11] (see **Figure 1** (right column)). For the trace-data collection, Inception-v3 was run four times: The first two represented the 1568 batches completed in 0.12 epochs, and the final two represented the 3072 batches completed in 0.24 epochs. The first two resulted in a rate of around 2200 images per second and the last two processed around 6200 images per second [12].

2.3. MLPerf/ResNet Inference Benchmark

The Residual Network (ResNet) architecture, introduced by He *et al.* [13], was a transformative development that enabled the training of substantially deeper neural networks than was previously practical. It directly addresses the degradation problem, where accuracy saturates and then degrades as networks grow deeper, by introducing "residual blocks". The central element of a residual block is a "skip connection" (or shortcut) that allows the input of a block to bypass one or more layers and be added to the block's output. This formulation enables the network to learn residual functions and facilitates unimpeded gradient flow during backpropagation, mitigating the vanishing gradient problem in very deep models. The 50-layer variant, ResNet-50, has become a canonical model in computer vision due to its excellent balance of depth, accuracy, and performance. Its widespread adoption and representative computational profile have led to its inclusion as a standard workload in the MLPerf benchmark suite, where it serves as a key yardstick for evaluating the performance of machine learning systems.

The GPU implementation of ResNet-50 is handled directly by MLperf [2]. As with Inception-v3, the trace for this application consists of the performance measures, GPU utilization, and power draw. Scaling was performed on the input trace, except as otherwise specified, using MinMaxScaler, which shifts and scales the data so that the data ranges from zero to one while retaining the original shape of the data set [11]. The MLPerf Inference benchmark is a suite of tests developed by the industry consortium MLCommons to provide a standardized and objective evaluation of machine learning system performance. The image classification benchmark within this suite utilizes the ResNet-50 v1.5 model and the ImageNet 2012 validation dataset, which consists of 50,000 images across 1000 classes.

2.4. Time Series Prediction Models

Forecasting models attempt to predict future values based on learned historical data. For time series prediction models, the original time series data ("trace") is split into different subsets. For statistical models, these data sets are split into data to fit on and data to compare against the model's prediction. For machine learning models, these data sets are split into train, validation, and test sets.

2.4.1. Recurrent Neural Networks (RNNs)

RNNs, including Simple RNN and Long Short-Term Memory (LSTM) variants, are neural networks specifically designed to handle sequential data and capture

temporal dependencies [14]. Unlike feedforward networks, RNNs feature a recurrent loop that allows information to persist, creating a form of memory. At each time step t, the RNN processes an input x_t and its hidden state from the previous step, h_{t-1}, to produce an output and update its hidden state to h_t. This internal state enables the network to retain a summary of past observations, making it naturally suited for time series forecasting, where future values are contingent on historical patterns. However, Simple RNNs are often hindered by the vanishing gradient problem, which makes it difficult for them to learn dependencies over long-time intervals.

To address the limitations of Simple RNNs, more sophisticated architectures like the Long Short-Term Memory (LSTM) network were developed [15]. LSTMs introduce a more complex internal structure consisting of a dedicated cell state and a series of "gates": the *forget*, *input*, and *output* gates. These gating mechanisms regulate the flow of information, allowing the network to selectively add, remove, or retain information in its cell state over long periods. The forget gate determines what information to discard from the cell state, while the input gate decides what new information to store. Finally, the output gate controls which parts of the cell state are used to produce the output at the current time step. This design enables LSTMs to effectively capture long-range dependencies, making them a powerful and widely adopted tool for complex time series modeling tasks.

2.4.2. TNN Models

A Transformer Neural Network (*TNN*) is another type of machine learning algorithm used to perform time series prediction. TNNs rely on attention mechanisms implemented within the encoder and decoder layers. RNNs struggle with longer sequences and learning dependencies between elements from the input sequence that are temporarily farther apart [16]. Attention mechanisms (the mapping of queries, keys, and values to an output) solve this by creating global dependencies between inputs and outputs. [17] provides a probabilistic TNN time series forecasting model, which trains a "global" probabilistic model using all available trace data (assumed to be time series data) and outputs predictions with an amount of uncertainty (+/− one standard deviation). Inference utilizes a Greedy Sampling/Search to convert the output of the model back into predictions. A prediction distribution is created and utilizes an autoregressive sampler that is used to return the prediction outputs.

3. Related Work

Significant decreases in temperature and power draw can be accomplished by implementing power capping on GPUs at a supercomputing center with minimal effects on job performance [7]. Prior work by [7] studied the effects of reducing peak power to 60% for a pre-trained BERT model, resulting in an increased computational time (by 8.5%) but with a 12.5% energy savings [18].

In their more recent paper, BERT training under a 200W power cap resulted in

a 15% reduction with little degradation in training speed [7]. Reducing power consumption by implementing a frequency cap is promising, but results in unconventional and unpredictable execution time [19]. Experimental results aiming to minimize the training time by utilizing a cooperative distributed GPU power capping system for GPU-based clusters found that it improves power capping accuracy to have a mean absolute error of less than 1% [20].

A related field for forecasting energy is energy consumption by devices. An enhanced 3 Neural Basis Expansion Analysis for Interpretable Time Series (N-BEATS) saw success in energy consumption predictions for smart grids and smart homes [21]. A study on forecasting energy consumption of buildings revealed that a multivariate Transformer Neural Network performed 3.2 percentage points better than RNN models in terms of MSE and MAPE (mean absolute percentage error) evaluation metrics [22].

Reference [23] surveyed predicting power, and the following works were discussed. Song *et al.* used a neural network with an input layer size of 10 and two hidden layers of size 4 in conjunction with a counter-based performance model. This model was more accurate than linear regression models. "Performance counters" utilize hardware actions, such as average time per operation. Chen *et al.* in the "GPU Power Model built on GPGPU-Sim Simulated Events" compared linear regression, regressive tree, and random forest models, showing that random forest performed best for estimating kernel energy. Ma *et al.* found that a support vector regression model outperformed a support linear regression model.

In reference [24], experimental transformer neural network (TNN) results on influenza-like illness time series data outperformed the LSTM model. The transformer model resulted in the root-mean-square error (RMSE) relative decrease of 27% [24]. A study into power profiling of TNNs for text translation tasks, revealed that PyTorch (as opposed to NumPy) implementation resulted in shorter runtime, and thus lower cumulative energy consumption [25].

One other flavor of TNNs that is related to the work here are Annotated TNNs [26]. This model is implemented similarly to a typical transformer neural network and is optimized for classification, where a single label is returned as opposed to a probability distribution.

Reference [27] studies power consumption with different GPU machine configurations. In this work, the best base model, with respect to MAE, was SMOReg (a model that finds a curve to map inputs to outputs). Comparing this model with the most optimal ensemble model, which combined the results of three different models, reduced the MAE from 4.5% to 3.5%. Research into energy efficiency through power limiting focuses mostly on CPUs rather than GPUs. Predicting GPU utilization was previously studied by [28] for deep learning frameworks in the Cloud. The research utilized a predictive engine that extracted performance metrics from its model computation graph. It achieved up to a 61.5% improvement in GPU cluster utilization.

In [29], an algorithm was proposed that would predict and, based on those pre-

diction values, change the maximum allotted power at runtime. The paper reported the highest average energy savings of 17.7% with the lowest average performance loss of 5.1%. In [30], the continuance of [29], a similar approach was taken, GPU utilization was modeled as a time series and an Auto Regressive Integrated Moving Average (ARIMA) process was used to dynamically predict the GPU utilization in the next time slice to be executed. This paper utilized the predictions in each GPU to allocate power in accordance with the predicted utilization.

4. Model Architectures and Implementations

In this work, several model architectures were implemented, varying model type and the associated hyperparameters. The most successful models came from two neural forecasting families tailored to the GPU traces in this study: a stacked recurrent neural network (RNN), and a Transformer Neural Network (TNN) implemented in both deterministic and probabilistic variants. All models operate in a direct multistep (sequence-to-sequence) mode, producing a horizon of H future points from the most recent C context points.

Models were run on two computing platforms, the Wahab Computing Cluster at Old Dominion University and the ACES Computing Cluster at Texas A&M University, such that the RNN model for the Inception-v3 and ResNet-50 ran on Wahab Comput and ACES, respectively, while the TNN model was used on ACES for both data sets. All training is offline on identical CSV-format traces with the same preprocessing and windowing. The same PyTorch version, mixed precision (AMP), and optimizer settings across platforms; since the data is identical and evaluation is performed on the same held-out windows, hardware differences affect only wall-clock training time, not the predictive results reported.

4.1. Data Acquisition and Preprocessing

GPU utilization and power draw measurements were collected using nvidia-smi while running the Inception-v3 and ResNet-50 benchmarks on A DGX-1 node equipped with eight NVIDIA V100 GPUs on the Wahab Computing Cluster at Old Dominion University. Measurements were recorded at 250-millisecond (ms) intervals, providing high-resolution temporal data for our analysis. The 250 ms interval was chosen because it is the default NVIDIA Management Library (NVML) sampling cadence, and it fits typical scheduler and dynamic voltage and frequency scaling (DVFS) dynamics without incurring excessive overhead.

The RNN baselines are trained and evaluated on single long traces: Inception-v3 has 34,408 samples (≈143.37 minutes) which contains 7 distinct executions of the workload (referred to as 7 *episodes*), and ResNet-50 has 78,109 samples (≈325.45 minutes) which contains 20 distinct executions of the workload (20 episodes). In contrast, for the Transformer (TNN) analyses, multiple shorter runs are aggregated per workload: Inception-v3 uses 24 runs × 1660 samples each (415 s ≈6.92 minutes per run; 166.0 minutes total), and ResNet-50 uses 25 runs × 4057

samples each (1014.25 s ≈16.90 minutes per run; 422.60 minutes total). Single- and multi-run regimes are kept separate in reporting because they feature different aspects of data variability. The former captures long-range temporal continuity and rare transitions within one run, while the latter stresses generalization across repeated executions. Four statistics per target metric (GPU utilization or power draw) are reported to demonstrate data variability:

- Std is in-run (in-episode) standard deviation, which captures typical short-horizon volatility;
- Range reflects the amplitude of swings the model must track;
- CV is the coefficient of variation σ/μ, which normalizes volatility by the mean value to enable fair cross-workload comparison;
- T/min counts the step changes (transitions) above a small threshold (2 percentage points for the GPU utilization and 10 W for the power draw) indicating how often regimes shift.

Each statistic is reported as mean ± std across runs (episodes), explicitly exposing between-run (between-episode) spread rather than only central tendency. By reporting CV (mean ± std) and T/min (mean ± std), in addition to Std and Range, both the scale-normalized variability and the rate of regime shifts are reported, allowing visibility into how much these quantities themselves vary from run to run (or episode to episode).

Table 1. Data-centric variability (250 ms). TNN rows aggregate per-run dispersion while RNN rows aggregate per-episode dispersion for GPU utilization Z and power-draw P target metrics (Column 3). The statistics Std, Range, CV, and T/min are reported $\mu \pm \sigma$ across the workload runs (episodes).

Workload	Model	Target	Std	Range	CV	T/min
Inception-v3	TNN	Z	41.77 ± 0.67	100.00 ± 0.00	0.67 ± 0.03	101.30 ± 3.44
Inception-v3	TNN	P	99.79 ± 1.02	272.13 ± 6.12	0.54 ± 0.02	129.19 ± 3.19
ResNet-50	TNN	Z	45.40 ± 0.24	100.00 ± 0.00	0.69 ± 0.01	3.71 ± 0.23
ResNet-50	TNN	P	107.88 ± 0.67	263.31 ± 1.69	0.53 ± 0.01	91.35 ± 4.71
Inception-v3	RNN	Z	39.38 ± 10.07	99.14 ± 1.22	1.12 ± 0.15	77.32 ± 52.75
Inception-v3	RNN	P	79.04 ± 26.40	270.46 ± 11.02	0.59 ± 0.07	94.60 ± 15.61
ResNet-50	RNN	Z	10.62 ± 0.42	100.00 ± 0.00	0.11 ± 0.00	3.37 ± 3.83
ResNet-50	RNN	P	28.00 ± 0.92	246.22 ± 4.01	0.10 ± 0.00	137.24 ± 11.09

Table 1 summarizes dispersion of the targets at 250 ms, aggregated across runs (for TNN) or episodes (for RNN). In the TNN setting, Inception-v3 exhibits pronounced short-horizon choppiness, with GPU utilization Z and power draw P transition rates of ≈101 and 129 transitions per minute T/min, having mean CVs of 0.67 and 0.54, respectively. By contrast, TNN ResNet-50 shows far fewer Z tran-

sitions (≈3.7/min) yet remains active on P (≈91/min), with CVs of 0.69 and 0.52, respectively. The RNN trace episodes reinforce this pattern: Inception-v3 episodes are highly volatile, exhibiting CVs of 1.12 and 0.59 with approximately 77 and 95 T/min for Z and P, respectively. Conversely, ResNet-50 episodes are smoother in level (CVs of 0.11 and 0.10 for Z and P) but still feature frequent power draw transitions (≈137/min).

The Inception-v3 traces show sizable run-to-run dispersion in both CV and transition rates, indicating heterogeneous execution phases and ramp/dip frequencies. ResNet-50 displays a narrower spread for GPU utilization but a broader spread for power-draw transitions, consistent with power-side activity bursts. Because these across-run (across-episode) dispersions capture the dominant sources of forecasting difficulty, they provide a conservative, data-centric proxy for the variability typically probed by additional random-seed restarts, without altering the qualitative ordering of models.

4.1.1. Series Preprocessing

The preprocessing pipeline addresses several challenges inherent in GPU performance data. For multi-GPU systems, we identify and extract the most representative single GPU trace by selecting the GPU with the longest continuous active period, defined as sustained utilization above a threshold (1% for utilization, 45W for power draw). This approach ensures we capture meaningful operational patterns rather than idle behavior.

The raw time series undergoes several additional preprocessing steps:

- Active region extraction: When enabled, the longest continuous active segment is identified and extracted, with padding of $2n_{\text{steps}}$ on each side to preserve temporal context.
- Length capping: Series can be truncated to the most recent N samples to focus on recent behavior patterns.
- Downsampling: Every k^{th} sample can be retained to reduce computational complexity while preserving overall trends.
- Normalization: All series are normalized using training set statistics (μ, σ) to ensure stable gradient flow during training.

4.1.2. Data Splitting

Two distinct data splitting strategies were employed depending on the model architecture:

- For RNN/LSTM models, we use a temporal split with 70% training, 15% validation, and 15% test data. To ensure proper temporal context at split boundaries, we include an overlap of n_{steps} samples between consecutive splits, allowing the model to make predictions immediately at the beginning of each partition.
- For Transformer models, we implement a more sophisticated approach where the split occurs at the series level rather than temporally. Each time series is divided such that the test set contains the final prediction_length samples, the

validation set contains the preceding prediction_length samples, and the training set contains all earlier data. This ensures that the model learns from complete historical patterns before making future predictions.

4.2. Recurrent Neural Network Architectures

4.2.1. Stacked Architecture Design

The chosen RNN implementation employs a deep, stacked architecture with carefully designed layer tapering. Rather than using uniform layer sizes, we implement a progressive reduction in hidden units across layers, following the pattern $[d_1, d_2, d_3]$, where typically $d_1 > d_2 > d_3$. This tapered design serves multiple purposes: it creates a natural information bottleneck that encourages the network to learn increasingly abstract representations, reduces the parameter count in deeper layers where overfitting risk is higher, and improves gradient flow by preventing the vanishing gradient problem common in uniform deep architectures.

The architecture consists of three recurrent layers, where the first two layers are configured with return_sequences = True to pass their complete hidden state sequences to subsequent layers, while the final layer returns only the last time step's output. This design enables the network to build hierarchical temporal representations, with each layer capturing patterns at different levels of abstraction.

4.2.2. Training Strategy and Regularization

Training employs several sophisticated strategies to ensure robust convergence. These include: sliding window generation, regularization, and adaptive learning. For sliding window generation, training samples are created using a sliding window approach with configurable stride *S*. For a series of length *N* and window size n_{steps}, $\left(N - n_{\text{steps}}\right)/S$ samples are generated, significantly augmenting the effective training set size. Standard regularization is used to apply both standard dropout (probability 0.2) between layers and recurrent dropout (configurable, typically 0.0 - 0.2) within recurrent connections. This dual approach prevents overfitting in both the feedforward and recurrent pathways. Lastly, for adaptive learning, gradient clipping is leveraged along with two callback mechanisms: 1) early stopping with patience of 20 epochs monitors validation loss and restores best weights, and 2) learning rate reduction on plateau with factor 0.5 and patience of 8 epochs reduces to a minimum of 10^{-6}.

4.2.3. Quantile Regression for Power Prediction

For power draw prediction, we implement pinball loss to perform quantile regression, targeting the 65$^{\text{th}}$ percentile rather than the mean:

$$L_q\left(y, \hat{y}\right) = \text{mean}\left(\max\left(q\left(y - \hat{y}\right), \left(q - 1\right)\left(y - \hat{y}\right)\right)\right), \tag{1}$$

where $q = 0.65$. This approach is motivated by the asymmetric nature of power draw distributions, where underestimating power consumption has more severe consequences than overestimating. The quantile target provides more robust predictions against outliers and better captures the typical operating range.

4.3. Transformer Neural Network Architecture

The TNN model implemented is a Time Series Transformer, which utilizes an encoder-decoder architecture specifically adapted for time series forecasting. The configuration used specifies a deep architecture with four encoder layers and four decoder layers, with the dimensionality of the model's hidden states saved as a variable d. A key aspect of this model is its ability to integrate multiple types of features. It uses a context length of past observations that is twice the length of the desired prediction horizon and incorporates lagged values of the target variable as input.

Beyond historical values, the model is enriched with both dynamic and static features. Dynamic features include time-based attributes derived from the data frequency (e.g., second, minute, hour) and a continuous "age" feature that represents the position within the time series. A static categorical feature is also used to represent the unique ID of each time series, for which the model learns a distinct embedding. This allows the model to capture series-specific patterns. Instead of producing a single point forecast, this architecture performs probabilistic forecasting by outputting the parameters of a probability distribution for each time step in the forecast horizon. The default configuration is set to use a Student's t-distribution, enabling the model to quantify the uncertainty in its predictions.

4.3.1. Time Series Transformer Configuration

The main Transformer implementation leverages the Hugging Face Time Series Transformer for Prediction model, specifically adapted for time series forecasting. This architecture employs a symmetric encoder-decoder structure with four layers each, providing sufficient depth to capture complex temporal patterns while maintaining computational efficiency.

Key architectural parameters include a model dimension (d_{model}) of 64, determining the size of all internal representations. The context length is set to twice the prediction length, providing substantial historical context. Configurable lag sequences (1, 2, 3, 4, 6, 8, 16, 32, 64, 120, 240) are automatically filtered to ensure they remain within the context window. The distribution output uses Student's t-distribution for probabilistic forecasting, providing both point estimates and uncertainty quantification.

4.3.2. Rolling Window Training Strategy

Unlike traditional approaches that use fixed train/validation/test splits, the TNN implementation used in this study employs a rolling window training strategy that dramatically increases data efficiency. For each time series and at each epoch, a configurable number of multiple random forecast windows are sampled (typically 32 windows per series per epoch).

The window sampling process works as follows. First, for a series of length T, identify all valid forecast start positions in the range from context_length+max_lag to T-prediction_length. Second, randomly sample from these positions to create diverse training examples. Third, each sample includes past values, future values (for

training), and associated temporal features. This approach ensures the model sees diverse temporal contexts during training, improving its ability to generalize across different regimes and reducing overfitting to specific temporal positions.

4.3.3. Probabilistic Forecasting

The TNN model performs probabilistic forecasting by outputting parameters of a Student's t-distribution at each forecast horizon. This approach provides several advantages: uncertainty quantification through confidence intervals alongside point predictions, robustness to outliers through the Student's t-distribution's heavy tails that better accommodate extreme values, and flexible prediction through the model's ability to generate multiple forecast trajectories through sampling. During inference, greedy sampling is used to generate the median forecast from the learned distribution, providing stable point estimates while maintaining access to the full predictive distribution for uncertainty analysis.

4.4. Training Infrastructure and Optimization

The main neural network implementations showcased in this analysis leverage the following deep learning models, each chosen for its strengths in handling specific model architectures. The RNN and LSTM models are implemented using TensorFlow 2.x with the Keras functional API. This framework provides excellent support for recurrent architectures through its optimized CuDNN-backed implementations when available, significantly accelerating training on NVIDIA GPUs. The Keras API enables clean expression of complex architectures, including our residual connections and stacked layers with varying dimensions. The Probabilistic TNN, in contrast, is implemented using PyTorch through the Hugging Face Transformers library. Training is optimized using the Accelerate library, which provides automatic mixed precision training and efficient multi-GPU utilization. This combination is particularly well-suited for Transformer architectures, offering memory-efficient attention mechanisms and seamless integration with the probabilistic forecasting capabilities of the TimeSeriesTransformerForPrediction model.

Hyperparameter Configuration

Hyperparameters were selected via iterative, literature-informed exploration with validation-driven adjustments. The RNN/LSTM baselines, experiments varied history length, architecture (RNN vs. LSTM), loss (MAE, MSE, Huber), learning rate ($[1 \times 10^{-5}, 1 \times 10^{-2}]$), context lengths ($C \in \{24, 48, 65, 100, 160, 185\}$), and training focus weights via command-line flags. For the Transformer (TNN), experiments varied capacity (width d_model, layers), temporal shape (prediction length, context multiplier, lags), context ($C \in \{2H, 6H\}$), dropout ([0.05, 0.3]), slope-consistency weight ([0, 0.6]), horizon-weighting mode/alpha (exp/front, $\alpha \in [0.03, 0.15]$), and optional focused sampling and elapsed-time features, again via command-line flags. Exploration was conducted through repeated runs of jobs, where comparisons reflect manual hyperparameter trials selected by valida-

tion MAE.

For RNN/LSTM models, hidden units [128, 84, 64] were used for most inferences except for the univariate GPU utilization where [200, 128, 64]. was used. The input context C is chosen on the target metric basis from $C \in \{24, 48, 65, 100, 160, 185\}$. Batch size is set to 256, with training for up to 600 epochs with early stopping.

For TNN, a decoder-only Transformer with $L = 6$ layers, $h = 8$ heads, model width $d_{\text{model}} = 256$, feed-forward width 1024, and dropout 0.1 was used. The forecast horizon was set to $H = 60$ steps and context as a multiple of the horizon, $C \in \{4H, 6H\}$ (*i.e.*, 240 or 360 steps); training uses AdamW (learning rate set to 3×10^{-4}, weight decay 10^{-4}), batch size 256, 200 batches/epoch, and 300 epochs with early stopping. Horizon-weighted loss (exp, $\alpha = 0.05$) and an optional slope-consistency term ($\lambda \leq 0.4$) were enabled to improve responsiveness to dips.

4.5. Baseline Methods

The work in [31] showed that the performance of various statistical models on the Inception-v3 dataset fell short of the performance of the RNN and TNN for time series forecasting of GPU trace data. However, to help contextualize our neural network results, results for two simple statistical methods are included: a Naive Baseline and Ridge Regression models. The Naive Baseline predicts the last observed value for all future time steps, representing the simplest possible forecast. The Ridge Regression model fits a linear model with L2 regularization ($\alpha = 1.0$ for utilization, $\alpha = 3.0$ for power) on flattened window features, providing a simple yet often effective linear baseline. Both baselines use the same train/validation/test splits and evaluation protocols as the neural models, ensuring fair comparison.

5. Results

The best-performing RNN architectures for the Inception-v3 benchmark are compared on both univariate and multivariate utilization- and power-trace data in Table 2. The corresponding traces are featured in Figure 2, Figure 3 along with their predicted values.

Table 2. Best-performing RNN model architecture and the corresponding MAE metric value on the GPU utilization Z and power P traces (Column 1) in the univariate U and multivariate M testing (Column 2). The number of past values (window length) w is shown in Column 3. The hyperparameter set includes optimizer Opt, and the number of units as described in Section 0.0.12.

Workload	Trace	Test	w	Opt	MAE
Inception-v3	Z	U	100	RMSprop	3.72
Inception-v3	P	U	65	RMSprop	17.87
Inception-v3	Z	M	160	RMSprop	3.74
Inception-v3	P	M	65	Adam	17.25

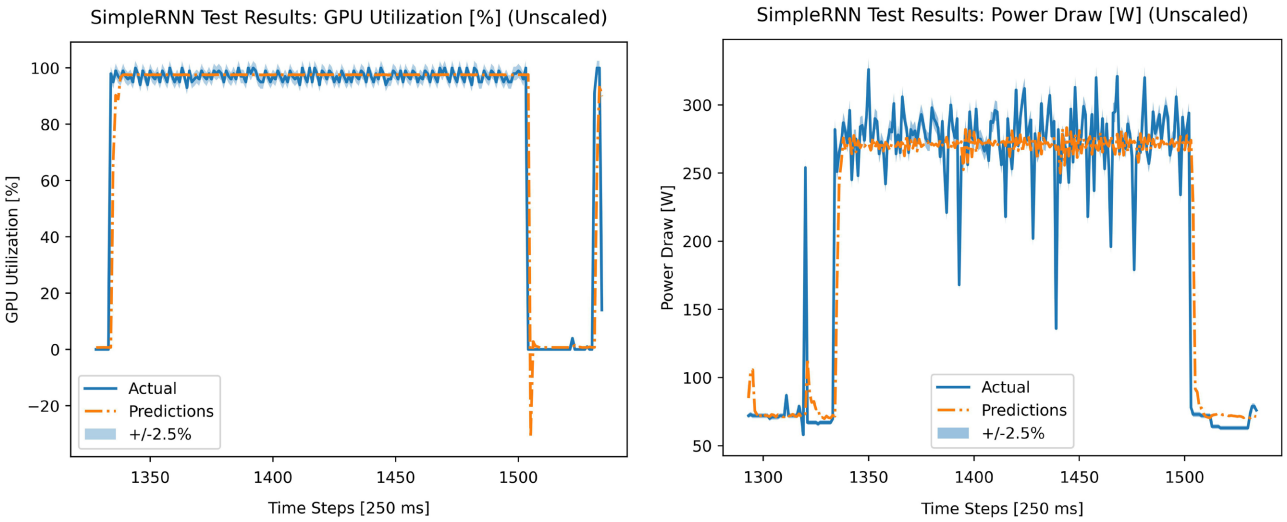


Figure 2. Univariate RNN model prediction results with the hyperparameters and MAE as shown **Table 2**, for GPU utilization (left, details in line 1) and power draw (right, details in line 2).

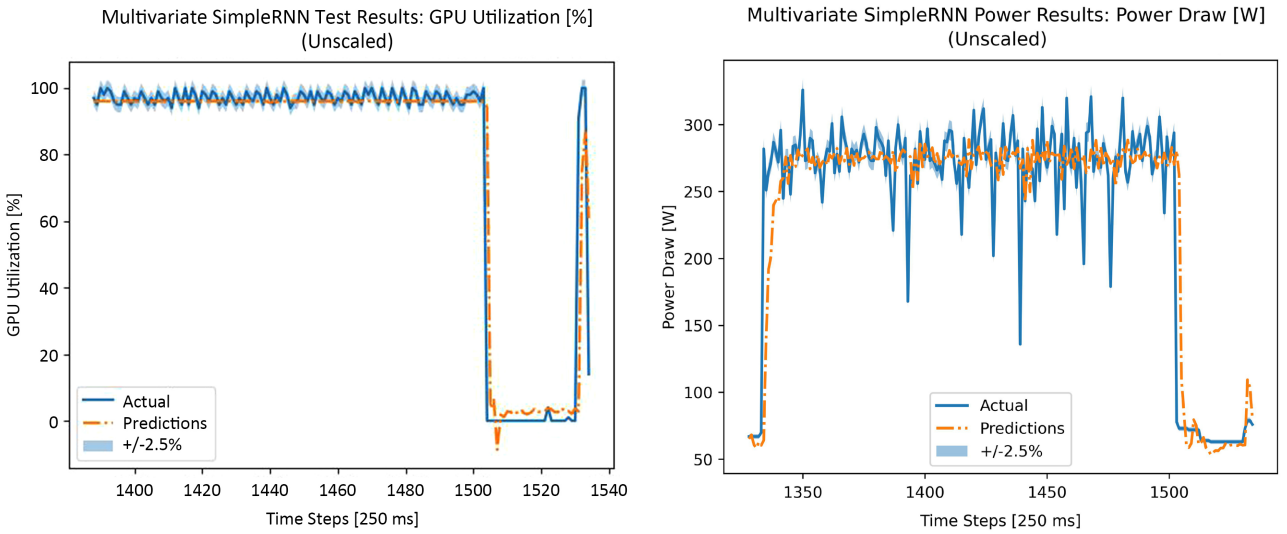


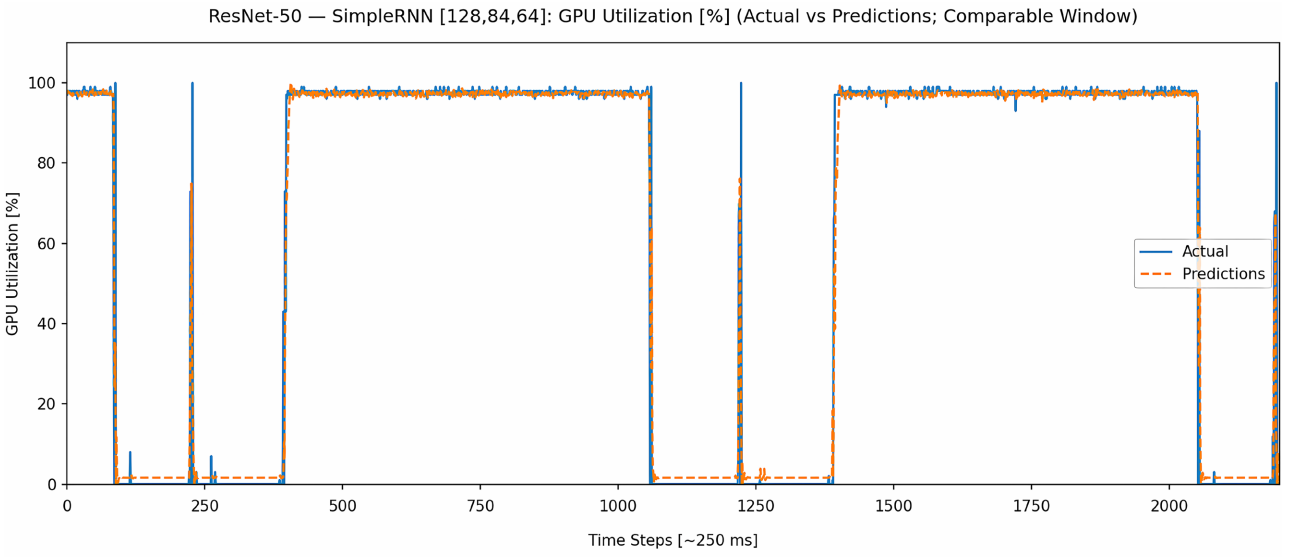
Figure 3. Multivariate RNN model prediction results with the hyperparameters and MAE as shown in **Table 2**, for GPU utilization (left, details in line 3) and power draw (right, details in line 4).

The best-performing RNN and LSTM architectures for the ResNet-50 benchmark for univariate datasets are detailed in **Table 3**. The corresponding traces are featured in **Figure 4**, showing the RNN performance for GPU utilization (a) and power draw (b) as well as **Figure 5** showing the LSTM performance for GPU utilization (a) and power draw (b).

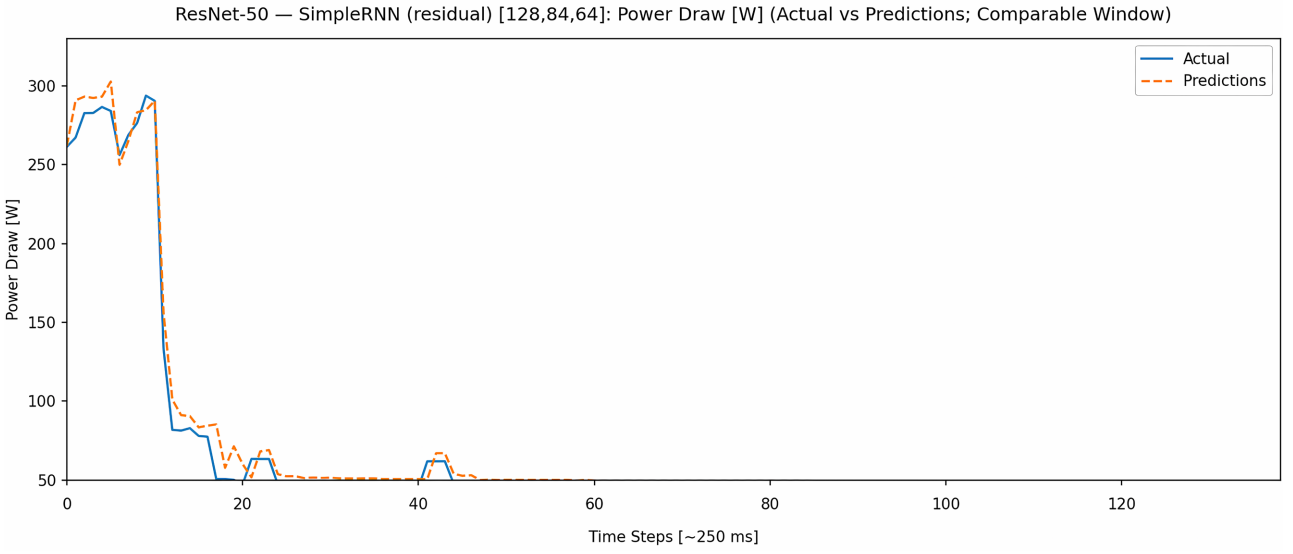
Figure 6 and **Figure 7** illustrate representative forecasts for the Inception-v3 and ResNet-50 data sets, respectively, using a probabilistic TNN. Both **Figure 6** and **Figure 7** show the data for GPU utilization on the left and for power draw on the right. The probabilistic TNN produces calibrated uncertainty bands that widen during transitions and contract on plateaus. The data for the best-performing runs for both data sets is captured in **Table 4**.

Table 3. Best-performing RNN model architecture and the corresponding MAE metric value on the GPU utilization Z and power P traces (Column 1) showing the use of both an RNN and an LSTM. The number of past values (window length) w is shown in Column 3. The hyperparameter set includes optimizer Opt and the rest of the parameters as in Section 0.0.12.

Workload	Trace	Model	w	Opt	MAE
ResNet-50	Z	RNN	165	Adam	2.42
ResNet-50	P	RNN	165	Adam	13.84
ResNet-50	Z	LSTM	165	Adam	1.18
ResNet-50	P	LSTM	165	Adam	9.07

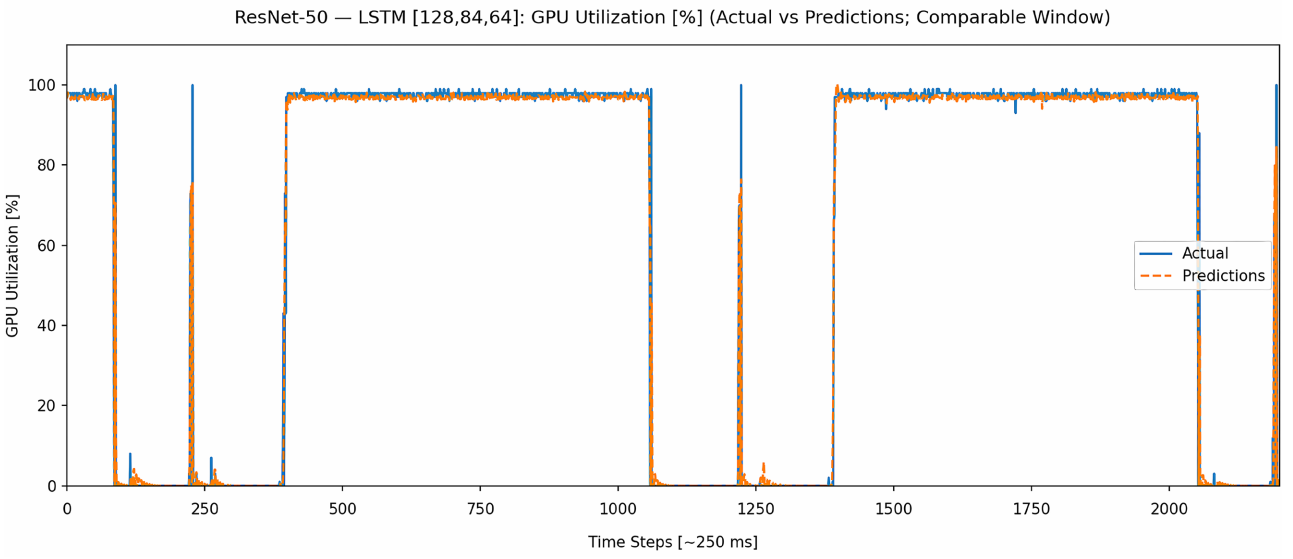


(a)

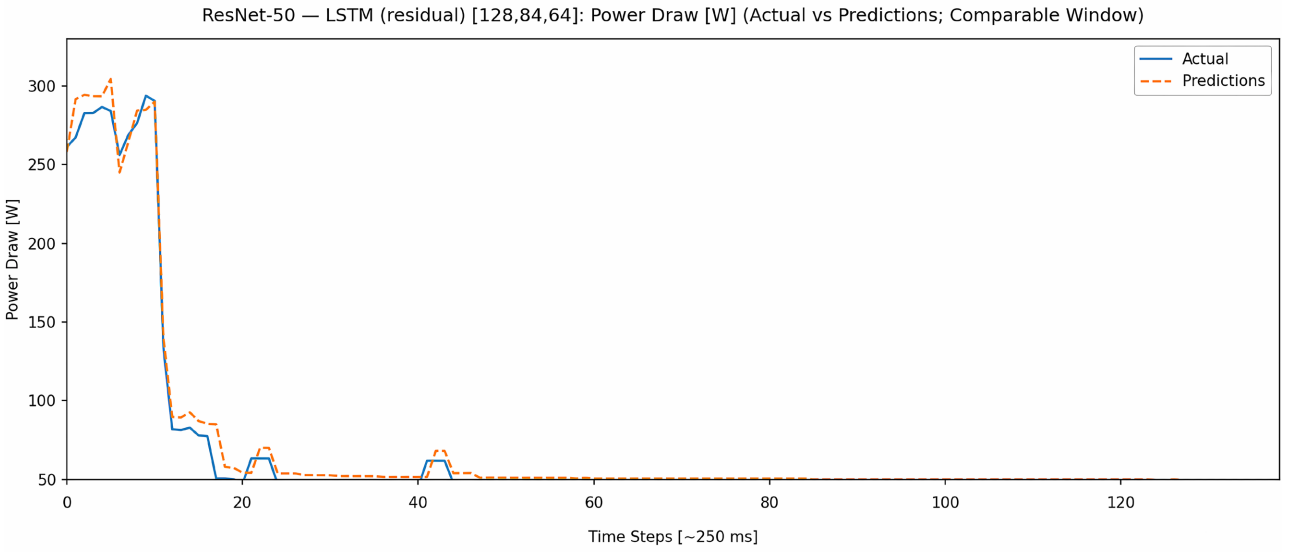


(b)

Figure 4. RNN model predictions for the ResNet-50 data set for GPU utilization (a) and power draw (b).



(a)



(b)

Figure 5. LSTM model predictions for the ResNet-50 data set for GPU utilization (a) and power draw (b).

Table 4. Best-performing TNN model architecture, having six layers, model width (d_{model}) of 256, the loss parameter set as MSE, and using unit-scaled sequences, on the univariate GPU utilization Z and power P traces (Column 2). MAE and MSE metrics on both the Inception-v3 and the ResNet-50 data sets are shown.

Workload	Trace	MAE	MSE
ResNet-50	Z	2.84	9.62
ResNet-50	P	3.22	16.16
Inception-v3	Z	2.05	7.16
Inception-v3	P	5.29	36.10

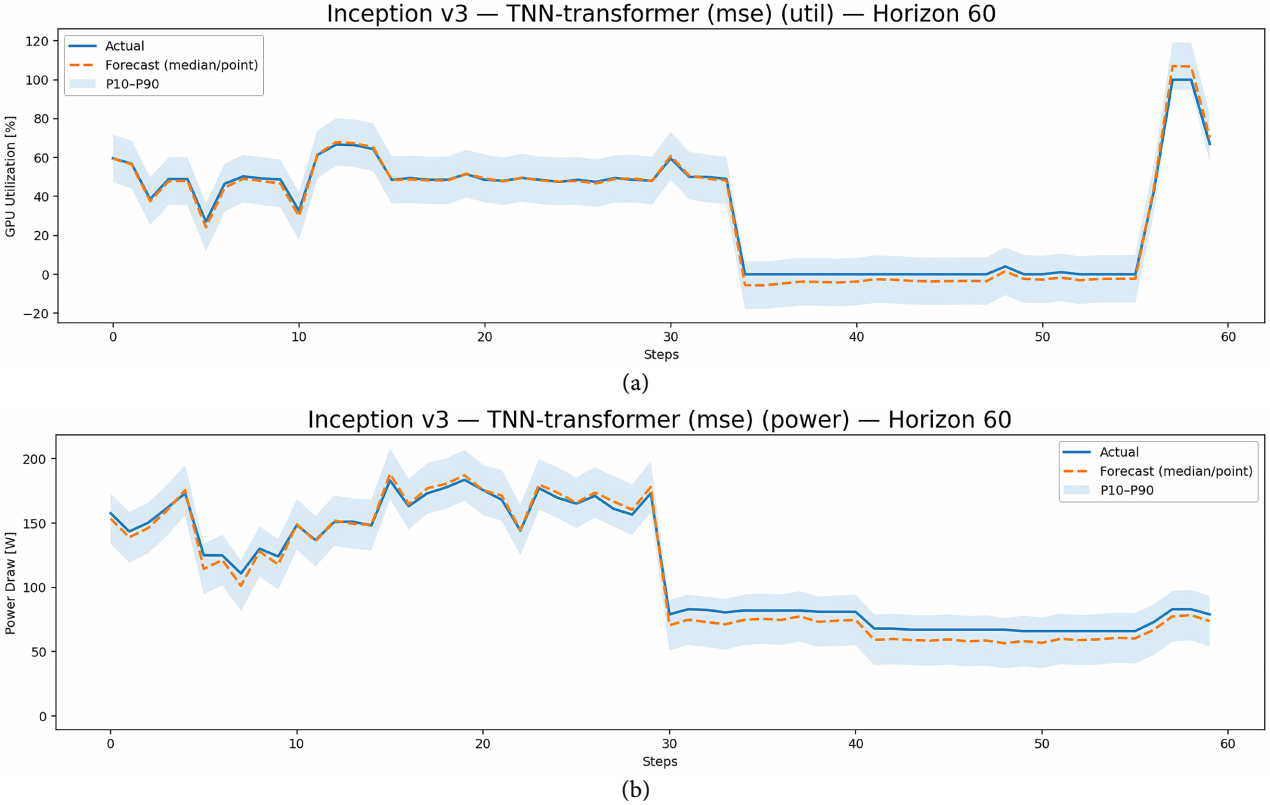


Figure 6. TNN model prediction for Inception-v3 for GPU utilization (a) and power draw (b).

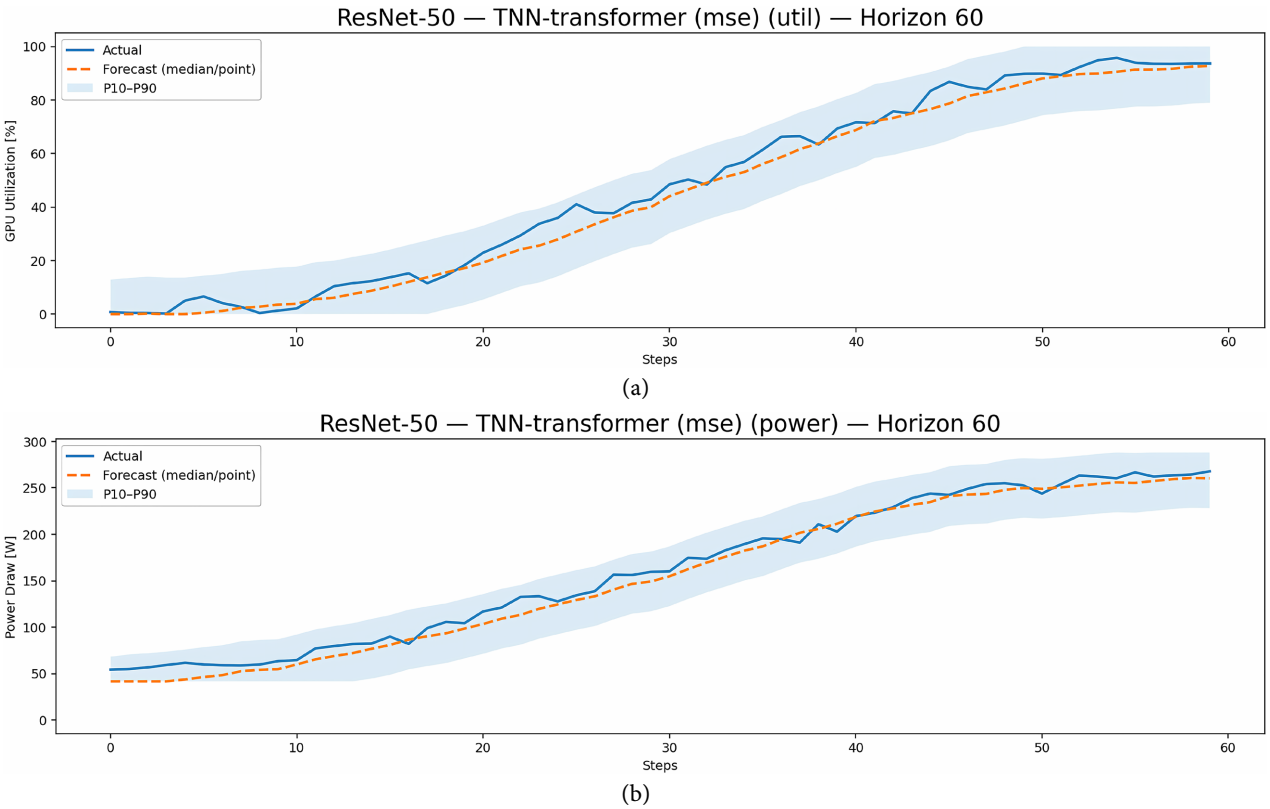


Figure 7. TNN model prediction for the ResNet-50 data set for GPU utilization (a) and power draw (b).

Lastly, the performance of both models using the naive baseline and the ridge regression model is provided in Table 5.

Table 5. Baseline statistical model comparison and the corresponding MAE and MSE metrics on the GPU utilization Z and power P traces.

Dataset	Trace	Naive	Naive (MSE)	Ridge	Ridge (MSE)
ResNet-50	Z	46.44	3321.45	46.90	3365.85
ResNet-50	P	108.27	17439.41	115.96	19131.41
Inception-v3	Z	29.12	1407.95	25.93	1145.42
Inception-v3	P	48.72	3522.67	59.09	4991.14

6. Discussion

The experimental results from this study reveal a clear and compelling specialization among the neural network architectures for predicting different GPU performance metrics. The choice of the optimal model is not only dependent on the target metric (GPU utilization or power draw in this work) but is also influenced by the underlying computational characteristics of the deep learning workload.

6.1. Statistical Models

The quantitative results obtained with statistical models are not shown in this paper because these models poorly captured sudden changes in trace data values. However, the statistical model analysis helped in qualitative characterization of the traces. A more detailed analysis is available in [31]. In particular, time series analysis revealed that neither original GPU utilization nor power draw traces were stationary. The GPU utilization trace required a differencing and the power draw required a detrending to produce a stationary time series. Experimental results on the statistical-based models AR(p), MA(q), ARMA(p,q), ARIMA(p,d,q), SARIMA(p,d,q)(P,D,Q)[m], and models selected from auto-ARIMA, revealed that the autoregressive models with lags equating to the last significant lag from the partial auto-correlation graphs of the stationary series (for both GPU utilization and power draw) outperformed the other statistical models considered in terms of the prediction evaluation metrics of MSE, MAE, and RMSE. The moving average models did not perform well in terms of the evaluation metrics. For both data sets, using the stationary GPU utilization and power draw traces, the selected seasonal models outperformed the selected non-seasonal models, but neither performed better than the AR(p) models, in terms of the prediction evaluation metrics MSE and MAE. It is worth, however, to continue testing statistical models, as baseline, on the performance trace data sets.

6.2. RNN and LSTM Models

RNN and LSTM variants compress *local* temporal context and respond quickly to

sharp mode switches. That bias is well-matched to utilization, which often flips between small sets of operating regimes over short horizons. Two design choices help in our runs: stacked recurrence (hierarchical temporal features) and regularization that tempers overfitting to bursty segments. Multivariate inputs sometimes reduce large deviations (helping MSE) even when the median error (MAE) is similar to univariate runs; this is most visible when utilization and power are weakly offset in time.

The sequential, stateful nature of RNNs is exceptionally well-suited to modeling GPU utilization. Utilization traces are often characterized by sharp, distinct transitions between idle and active states. An RNN's ability to maintain a "memory" of the immediately preceding state allows it to precisely capture the timing of these step-changes. The slightly better performance on ResNet-50 (relative to Inception-v3) may be attributed to its more regular and predictable computational graph, which consists of sequential convolutions, making the on/off utilization pattern even easier for a sequential model to learn.

6.3. Probabilistic TNN Models

The Transformer (TNN) architecture developed here conditions on long histories and outputs a full predictive distribution at each step (these experiments use a heavy-tailed choice, which is robust to occasional spikes). Empirically, its predictive bands tighten on plateaus and widen at regime changes—exactly the behavior a downstream controller can exploit.

Power draw is an inherently noisier and more complex signal than utilization. It fluctuates based on the specific kernel being executed, memory access patterns, and thermal conditions. The TNN's self-attention mechanism, which allows it to weigh the influence of all points in the context window rather than just recent steps, is better equipped to identify the complex, non-sequential patterns within this noisy data. Furthermore, the model's probabilistic output is a key advantage. As seen in **Figure 6** and **Figure 7**, the uncertainty bands appropriately widen during volatile periods and contract during stable ones. This ability to quantify uncertainty is invaluable for a stochastic signal like power draw.

6.4. Other Considerations

Conditioning on the "other" signal (e.g., forecasting power with utilization as an input) helps most when the objective penalizes *spikes*. Across our experiments, multivariate models typically reduce tail error for power (improving MSE and tightening high-quantile bands), while utilization benefits are modest unless there is strong cross-signal coupling.

Transformer performance is sensitive to the context/prediction horizon. Short horizons tend to miss sharp high to low transitions; very long horizons can dilute attention on the most informative recent context. We found mid-to-long horizons to be the most reliable for power, and mid-range (but not too short) horizons for utilization. A simple operational heuristic emerges: 1) sweep several horizons; 2)

inspect whether predictive bands widen at the right moments; and 3) prefer the shortest horizon that consistently anticipates regime changes.

Because the TNN outputs a full predictive distribution, controllers can tie aggressiveness to predictive variance: tight bands suggest stricter power caps or earlier prefetch/migration, while wide bands suggest deferring aggressive actions, widen guardrails, or request more look-ahead. Given the order-of-magnitude margins over naive/linear baselines in typical error, coupling the TNN's uncertainty with guardrail logic should translate into fewer cap violations and safer use of thermal/power headroom without impinging on performance guarantees.

Comparing the results between the two workloads provides insight into model selection for different applications.

- Inception-v3, with its "Inception modules" featuring multiple parallel operations of different sizes, creates a more complex and varied GPU trace. This architectural complexity likely contributes to the higher variance in its power draw, making it a strong test case for the TNN's global attention capabilities.
- ResNet-50, as an industry-standard benchmark, has a more uniform computational graph dominated by sequential 3×3 convolutions. This leads to a more regular, repeating pattern in its performance traces, which may explain why the simpler, sequential RNN models perform so exceptionally well on its utilization data.

6.5. Validity and Robustness

Internal validity. Traces were sampled via nvidia-smi at fixed intervals, preprocessed with MinMax scaling, and long idle segments (sustained zero-utilization) were removed to focus on active regions. We reused the same splits/evaluation across all model families to avoid protocol drift.

External validity. Results were collected on a specific hardware platform (NVIDIA V100 GPUs) and two workloads (Inception-v3 training; MLPerf ResNet-50 inference). Generalization to newer accelerators, different sampling intervals, and attention-heavy/decoder workloads is promising but not guaranteed. In practice, measurement semantics and aggregation differ by GPU generation [32]. Moreover, cross-architecture power models often require recharacterization, or they fail under new partitioning modes (e.g., Multi-Instance GPU, Virtual GPU, time slicing), and transformer inference exhibits distinct system bottlenecks relative to CNNs [33].

Robustness. To mitigate over-interpretation, we 1) reported both MAE and MSE (typical vs. tail error), 2) compared across two architecturally distinct workloads, and 3) assessed univariate/multivariate variants. A natural next step-left as a future work, is to add a different GPU generation and a transformer-style training workload.

7. Conclusion and Future Work

Our study demonstrates that different model architectures excel at predicting dif-

ferent GPU performance metrics. RNNs showed superior performance for short-horizon GPU utilization prediction, likely due to their stateful nature, which is well-suited to capturing the sharp temporal transitions in utilization traces. Conversely, the probabilistic TNN outperformed other models for power draw prediction, leveraging its global self-attention mechanism to model the noisier, more complex patterns on a longer horizon, inherent in power consumption. Training on multivariate performance metrics helped power more than utilization prediction, since multivariate models reduced tail error for power.

These findings suggest that a hybrid approach, using different models for different performance metrics, might be optimal for comprehensive GPU performance prediction. Future work should focus on:

- Implementing power-efficiency strategies based on these predictions;
- Exploring the effectiveness of these models on different GPU architectures and applications.

However, additional avenues for future research are also inviting. For example, investigating transfer learning approaches could enable rapid adaptation to new GPU architectures or workloads with limited new training data, and exploring hierarchical prediction models that separately forecast phase transitions and within-phase behavior might better capture the bimodal nature of GPU utilization. By advancing our ability to predict GPU performance metrics accurately, there is potential to develop more energy-efficient strategies for GPU-based high-performance computing environments, ultimately reducing costs and environmental impact.

Acknowledgements

This work was supported in part by the U.S. National Science Foundation (NSF) under the grant #2320998 and by the Research Computing clusters at Old Dominion University. The Wahab cluster is supported in part by NSF under the grant #1828593. This work used the ACES Cluster at Texas A&M University under allocation CIS250436 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by NSF grants #2138259, #2138286, #2138307, #2137603, and #2138296.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Xia, X.L., Xu, C. and Nan, B. (2017) Inception-v3 for Flower Classification. 2017 2*nd International Conference on Image, Vision and Computing* (*ICIVC*), Chengdu, 2-4 June 2017, 783-787. https://doi.org/10.1109/icivc.2017.7984661

[2] Reddi, V.J., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C., *et al.* (2020) MLPerf Inference Benchmark. 2020 *ACM/IEEE* 47*th Annual International Symposium on Computer Architecture* (*ISCA*), Valencia, 30 May-3 June 2020, 446-459. https://doi.org/10.1109/isca45697.2020.00045

[3] NVIDIA Corporation (2016) Nvidia-Smi-Nvidia System Management Interface Program, 2016.

[4] Mittal, S. and Vetter, J.S. (2014) A Survey of Methods for Analyzing and Improving GPU Energy Efficiency. *ACM Computing Surveys*, **47**, 1-23. https://doi.org/10.1145/2636342

[5] Top500.org (2022) Green-500 List, November 2022.

[6] Oak Ridge National Laboratory (2022) Frontier: Direction of Discovery, 2022.

[7] Zhao, D., Samsi, S., McDonald, J., Li, B., Bestor, D., Jones, M., *et al.* (2023) Sustainable Supercomputing for AI. *Proceedings of the* 2023 *ACM Symposium on Cloud Computing*, Santa Cruz, 30 October-1 November 2023, 588-596. https://doi.org/10.1145/3620678.3624793

[8] Can, E., Arora, R. and Chitale, P. (2020) Profiling and Optimizing Deep Neural Networks with DLPROF and PYPROF. NVIDIA Developer.

[9] Steurer, M., Hill, R.J. and Pfeifer, N. (2021) Metrics for Evaluating the Performance of Machine Learning Based Automated Valuation Models. *Journal of Property Research*, **38**, 99-129. https://doi.org/10.1080/09599916.2020.1858937

[10] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2016) Rethinking the Inception Architecture for Computer Vision. 2016 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Las Vegas, 27-30 June 2016, 770-778. https://doi.org/10.1109/cvpr.2016.308

[11] Scikit-Learn Developers (2024) Sklearn.preprocessing.minmaxscaler, 2024.

[12] Lambda Labs (2024) Lambda Tensorflow Benchmark, 2021.

[13] He, K., Zhang, X., Ren, S. and Sun, J. (2016) Deep Residual Learning for Image Recognition. 2016 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Las Vegas, 27-30 June 2016, 770-778. https://doi.org/10.1109/cvpr.2016.90

[14] Francois Chollet (2021) Deep Learning with Python. Manning Publications.

[15] Hochreiter, S. and Schmidhuber, J. (1997) Long Short-Term Memory. *Neural Computation*, **9**, 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

[16] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I. (2017) Attention Is All You Need. *Advances in Neural Information Processing Systems*, **30**, 5998-6008.

[17] Rogge, N. and Rasul, K. (2022) Probabilistic Time Series Forecasting with Transformers, December 2022. https://huggingface.co/blog/time-series-transformers

[18] McDonald, J., Li, B., Frey, N., Tiwari, D., Gadepally, V. and Samsi, S. (2022) Great Power, Great Responsibility: Recommendations for Reducing Energy for Training Language Models. *Findings of the Association for Computational Linguistics*: *NAACL* 2022, Seattle, 10-15 July 2022, 1962-1970. https://doi.org/10.18653/v1/2022.findings-naacl.151

[19] Drechsler, R., Metz, C.A. and Plump, C. (2024) Energy-Efficient CNN Inferencing on Gpus with Dynamic Frequency Scaling. In: Bhattacharya, A., Dutta, S., Dutta, P. and Samanta, D., Eds., *Innovations in Data Analytics*, Springer, 375-389. https://doi.org/10.1007/978-981-97-3466-5_28

[20] Kang, D., Ha, Y., Peng, L. and Youn, C. (2022) Cooperative Distributed GPU Power Capping for Deep Learning Clusters. *IEEE Transactions on Industrial Electronics*, **69**, 7244-7254. https://doi.org/10.1109/tie.2021.3095790

[21] Shaikh, A.K., Nazir, A., Khan, I. and Shah, A.S. (2022) Short Term Energy Consumption Forecasting Using Neural Basis Expansion Analysis for Interpretable Time Se-

ries. *Scientific Reports*, **12**, Article No. 22562. https://doi.org/10.1038/s41598-022-26499-y

[22] Oliveira, H.S. and Oliveira, H.P. (2023) Transformers for Energy Forecast. *Sensors*, **23**, Article 6840. https://doi.org/10.3390/s23156840

[23] Bridges, R.A., Imam, N. and Mintz, T.M. (2016) Understanding GPU Power: A Survey of Profiling, Modeling, and Simulation Methods. *ACM Computing Surveys*, **49**, 1-27. https://doi.org/10.1145/2962131

[24] Wu, N., Green, B., Ben, X. and O'Banion, S. (2020) Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case. arXiv: 2001.08317.

[25] Verma, A. (2023) Performance Engineering of Transformers on CPU vs GPU. Master's Thesis, Friedrich-Alexander-Universitat Erlangen-Nurnberg.

[26] Rush, A.M. (2018) The Annotated Transformer. *Proceedings of workshop for NLP Open Source Software* (*NLP-OSS*), Melbourne, 20 July 2018, 52-60.

[27] Dutta, B., Adhinarayanan, V. and Feng, W. (2018) GPU Power Prediction via Ensemble Machine Learning for DVFS Space Exploration. *Proceedings of the* 15*th ACM International Conference on Computing Frontiers*, Ischia, 8-10 May 2018, 240-243. https://doi.org/10.1145/3203217.3203273

[28] Yeung, G.F., Borowiec, D., Friday, A., Harper, R. and Garraghan, P. (2020) Towards GPU Utilization Prediction for Cloud Deep Learning. 12*th USENIX Workshop on Hot Topics in Cloud Computing* (*HotCloud* 20), Boston, 13-14 July 2020.

[29] Sundriyal, V. and Sosonkina, M. (2022) Runtime Energy Savings Based on Machine Learning Models for Multicore Applications. *Journal of Computer and Communications*, **10**, 63-80. https://doi.org/10.4236/jcc.2022.106006

[30] Sosonkina, M., Sundriyal, V. and Vallejo, J.L.G. (2022) Runtime Power Allocation Based on Multi-GPU Utilization in Gamess. *Journal of Computer and Communications*, **10**, 66-80. https://doi.org/10.4236/jcc.2022.109005

[31] Parry, D.X. (2024) Time Series Models for Predicting Application GPU Utilization and Power Draw Based on Trace Data. Master's Thesis, Old Dominion University.

[32] Abe, Y., Sasaki, H., Kato, S., Inoue, K., Edahiro, M. and Peres, M. (2014) Power and Performance Characterization and Modeling of GPU-Accelerated Systems. 2014 *IEEE* 28*th International Parallel and Distributed Processing Symposium*, Phoenix, 19-23 May 2014, 113-122. https://doi.org/10.1109/ipdps.2014.23

[33] Tschand, A., Rajan, A.T.R., Idgunji, S., Ghosh, A., Holleman, J., Kiraly, C., *et al.* (2025) MLPerF Power: Benchmarking the Energy Efficiency of Machine Learning Systems from μWatts to MWatts for Sustainable AI. 2025 *IEEE International Symposium on High Performance Computer Architecture* (*HPCA*), Las Vegas, 1-5 March 2025, 1201-1216. https://doi.org/10.1109/hpca61900.2025.00092