[1]Laila Nassef , Rana A. Tarabishi [1,2], Sara A. Abo Alnasor [1,2]

Benchmarking NLP and Computer Vision Models on Domain-Specific Architectures: Standard vs. TensorRT-Optimized Performance

Journal of Electrical Systems

Abstract: **-** In this work we systemically study performance of deep learning models for Natural Language Processing (NLP) and Computer Vision (CV) tasks using two popular representative architectures, ResNet-50 and BERT across two configurations: a standard ONNX Runtime configuration and an optimized TensorRT configuration. The main goal is to measure and compare inference time, throughput, CPU and GPU utilization as well as memory usage of all models on Domain-Specific Architectures (DSAs), in this case an NVIDIA GeForce RTX 3060 GPU. We experimentally demonstrate these trade-offs of latency-focused and throughput-focused optimizations and implications for at-scale deployment in realistic resource-constrained environments. The main findings show that the TensorRT-Optimized configuration yields a much higher throughput (up to 432 inferences per second with ResNet-50), and that the Standard configuration shows lower inference time, which is more appropriate for latency-sensitive applications. It should be noted that due to its dense transformer structure and large number of parameters, BERT has a much higher resource demand than ResNet-50, highlighting how model choices need to match performance with the constraints imposed by their deployment. Analysis of CPU and GPU utilization further illustrates the efficiency gains and potential bottlenecks associated with each configuration. Along with the benchmarking results, we also describe optimizations for serving the model: dynamic batching, mixed-precision training and memory management techniques to improve throughput as well as inference time. This study gives practitioners rich information to choose between model configurations and optimization strategies for an effective deployment of NLP and CV models on DSAs.

Keywords: Deep Learning Benchmarking, Inference Time, Throughput, Resource Utilization, TensorRT Optimizations, ONNX Runtime, ResNet-50, BERT, Domain-Specific Architectures (DSAs)

1. INTRODUCTION

Over the past decade, we have witnessed substantial progress in artificial intelligence (AI) fields powered by deep learning models used in many areas specifically Natural Language Processing (NLP) and Computer Vision (CV). They utilize large neural networks to tackle hard tasks, sentiment analysis in NLP and image classification in CV, which are intractable due to their high computational requirements. With the need to deploy these in production, driving demand for their optimization goals such as throughput, latency (inference time), and resource utilization. This is even more daunting when deploying models on Domain-Specific Architectures (DSAs) with strict limitations on computation and memory resources.

In response to these requests, exploration of techniques such as model compression, quantization, and domain-specific hardware optimizations has become active research. However, the effectiveness of such techniques is highly architecture and application dependent. As an example, convolutional neural networks (CNNs) like ResNet-50, that are often used for image classification, and transformer models such as BERT, that are often used for different NLP tasks, both of which present very different computations and memory patterns. These differences make it critical to understand the tradeoff between latency-oriented and throughput-oriented configurations, especially when deploying on DSAs like GPUs with TensorRT optimizations.

In this paper, a benchmarking study is performed to analyze the performance of ResNet-50 and BERT models under two configurations: Standard configuration using ONNX runtime and TensorRT-Optimized configuration. The Standard configuration describes a baseline for throughput, inference time, as well as resource utilization, while the optimized configuration uses TensorRT with dynamic batching along with half-precision (FP16) operations to improve throughput. Focusing on important parameters like inference time, throughput, CPU/GPU utilization and size

[1] Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia; lmohamed@kau.edu.sa
[2]Department of Computer Science, College of Computer Science and Engineering, Taibah University, Madinah 42353, Saudi Arabia; rtarabishi@taibahu.edu.sa; saboalnasor@taibahu.edu.sa

of the model, this comparison gives an overall view via different aspects for deployment with specialized hardware for models of deep learning.

The contributions of this paper are as follows:

1. **Benchmarking Analysis**: We perform a comprehensive performance analysis of ResNet-50 and BERT models with various configurations to showcase the throughput and the inference time trade-offs present in NLP and CV models.
2. **Evaluation of Resource Utilization**: The study illustrates how CPU, GPU and memory are consumed per configuration setting to better enable practitioners to use resource-constrained devices for deployment.
3. **Proposed Optimizations**: Depending on the results, we propose possible optimizations to maximize the efficiency of the model and ideas for future work in model deployment for DSA-based.

The structure of this paper is as follows: Section 2 discusses related work, specifically, the methods of performance enhancement for NLP and CV models of deep learning. Section 3 presents the details of the proposed experiments including the specific model, software and specification of hardware, the model used, and the datasets used. Section 4 explains the benchmarking configurations, metrics and procedures. Section 5 presents the results and explores the effects of each configuration on performance. Section 6 provides the summary of the results and their applications in relation to future work. Apart from these sections, recommendations on how the optimizations highlighted herein can be incorporated into real-life settings for practitioners will also be provided.

2. RELATED WORK

In the last few years, it has emerged as a primary concern in enhancing deep learning models for resource-constrained platforms for real-time implementation. For achieving low-latency and high throughput inferences, researchers have tried several methods like model quantization and model pruning apart from using accelerators like GPU, TPU or TensorRT. In addition to these approaches, there has also been growing demand in deploying models of deep learning on an edge device, leading to the emergence of these frameworks characterized by energy efficiency while still achieving a good level of performance. Later in the year, other researchers demonstrated that fine-tuning BERT and ResNet models for edge platforms can produce big gains in both edge throughput and power efficiency on the NVIDIA Jetson series platforms. Furthermore, as a larger number of IoT applications demands inference services, the need to serve multiple DNN inferences that aim to reduce latency and at the same time maintain accuracy becomes even more complicated. Challenges as diverse as these call for comprehensive solutions that will positively enhance these models' performance and make sure that their application is sustainable in many developing contexts classified by resource-constrained platforms. This section presents a review of the prior studies to enhance performance measures, such as inference time, throughput, as well as resource utilization in the term of models of deep learning which are particularly designed for DSAs with targeted NLP and CV tasks.

Domain-Specific Architectures (DSAs) for Deep Learning

All these DSAs such as Google TPU, NVIDIA Jetson series, and Intel Movidius are launched uniquely for the purpose of enhancing deep learning model execution. This type of architecture takes advantage of hardware special characteristics for optimizing the performance parameters like inference time, achieved throughput, and energy consumption.

It has been observed that the incorporation of GPU, TPU, and related accelerators (NVIDIA Jetson; Intel Movidius) leads to a significantly enhanced training result of the model. For example, recent work by (Hernández et al., 2024) focused on examining the performance and energy efficiency for convolutional networks on embedded platforms and demonstrated that these features are tightly dependent on the hardware to be used.

Furthermore, (Al Ghadani et al., 2020) also proved that by utilizing TensorRT and CUDA optimization on hardware level it is possible to improve inference time when using edge devices to permit the deployment of a large model. The authors of (Shishvan & Soyata, 2018) also proved that deep learning using GPUs with cuDNN library was faster than implementation on CPUs. Similar works of (Ho et al., 2022) have also extended that deep learning

throughput on GPU can be enhanced up to 19% by partially transferring part of the general matrix multiplication (GEMM) in Tensor cores into CUDA.

- **NLP on DSAs**: Multi-head attention used in transformer-based models such as BERT and GPT makes them computationally expensive. These models have been researched and developed to be run on DSAs for lower latency while delivering reasonable accuracy. There are already works such as study by (Lin et al., 2020) that has focused on the investigation of how TPUs and GPUs can enhance BERT's throughput. To perform efficient and real-time BERT inference on the edge devices, in (Tambe et al. 2020), the authors have employed early exit prediction, dynamic voltage and frequency scaling and hardware acceleration with help of CUDA modifications. From having obtained good results of speedup and efficiency enhancement for the purpose of hardware customization, we can therefore claim that the real-time deployment of NLP models in resource-constrained environments is feasible.

- **CV on DSAs**: As of now, deep learning architectures such as CNNs like ResNet have been adapted for DSAs in target application domain of fields including image classification. The related work by (Duggan et al., 2023) focused on the evaluation of multiple pre-trained CNN models, including ResNet, on the NVIDIA Jetson platform, with reference to quantization to favorize power behavior and inference speed. Among the works that speed up object detection based on images on the Nvidia Jetson Nano platform using CUDA and TensorRT, one of the works was published by (Myronyuk & Blagitko, 2023). In general, inferring deep neural networks on an edge device can expand the number of applications like autonomous vehicles and smart surveillance systems.

2.1 Model Optimization Techniques

The variations are essential for the model inference optimization that is very significant in real-time applications of deep learning models. In the time it has led to different strategies for optimization of deep learning models like TensorRT, mixed precision and dynamic batching etc. In so far as possible, to maximize the throughput and minimize the inference time, different strategies have been considered. Optimizations such as these other than enhancing execution help in the optimal utilization of the limited computational resource in such environments, many studies come up with sophisticated artificial intelligence solutions across these resource-constrained environments.

- **Model Quantization and Pruning**: Similarly, to improve the inference speed work has been carried out efficiently in many areas such as quantizing the models with low precision (INT8 or FP16, etc.) or pruning the weights which are not required. Prior work of (Hawks et al., 2021), (Wang et al., 2020), (Chowdhury et al., 2021) and (Jiang et al., 2023) has shown that quantization and pruning reduce the post deployment inference time on hardware accelerators while incurring minimal accuracy loss. (Kumar, et al, 2023) revealed that, weight pruning technique which includes the removal of unnecessary weights applied would improve the inference time without significant effect on the accuracy. In another study, (Sakr et al., 2022) extended this concept to quantization aware training to design models that are trainable specifically to integer only hardware accelerators such as ResNet and BERT. Such development also proves the effectiveness of these techniques in model size reduction as well as optimizing the use of resources which makes them important for effective deployment on edge devices as well in cloud.

- **Leveraging TensorRT for Optimization:** TensorRT is becoming one of the most popular frameworks for optimizing neural network inference on NVIDIA GPUs especially for real-time applications. In (Zhdanovskiy et al., 2024), work has defined some techniques for single and multi-DNN TensorRT including the dynamic batching and FP16 precision for improving DNNs inference, hence improving throughput. Estimation of DNN models on NVIDIA GPU based TensorRT in real time inference for DNN models, (Liao et al., 2020). (Nikoghosyan et al., 2023) stated that new transformer models can be improved in terms of inference speed and memory usage by applying TensorRT to Jetson Xavier NX edge device. Similarly, (Jeong et al., 2022) showed the opportunity to speed up the CNN inference on NVIDIA GPUs with TensorRT to enable real-time computer vision for deep learning across various domains such as autonomous driving robotics.

Other studies that have used similar solutions have also shown that TensorRT on GPUs has a substantial performance boost for NLP models such as BERT (Niu, Zhou, and Xie, 2020) as well as the CV models like ResNet

(Zhang et al., 2022). Likewise, (Nikoghosyan et al., 2023) and (Zhou & Yang, 2022) shown up to one order of magnitude acceleration improvement in inference time and higher throughput when using TensorRT optimization with BERT and ResNet-50 models. These observations demonstrate how TensorRT can be applied both to CNN-based architectures and the transformer-based scalable models which in turn will allow the faster and more efficient development of AI applications in the fields of NLP and CV.

- **Dynamic Batching and Precision Tuning:** Dynamic batching is required for a model to be able to accept more than one input at a time as well as to control more than just the throughput acceptable latency. With Transformer models, dynamic batching started helping gradually becoming helpful in (Tyagi & Sharma, 2020), several networks are efficient when used with GPUs. In addition, (Wang et al., 2022) explains that high throughput can be achieved when using dynamic batching in transformer models while does not affect the latency especially on GPUs. Furthermore, some related work such as (Fadge et al., 2023) and (Fang et al., 2020) have been given for using dynamic batching with transformers to improve throughputs. That showed that dynamic batching sizes presented the possibility of addressing more workload without having recourse to more latency. This is particularly good when used with a transformer-based model where the input sizes are tremendously large.

There has also been a demonstration of how to use less memory, train with lower precision–FP16 and how to make models faster (Hayford et al., 2024). In another similar study, while (Gu and Becchi, 2020) tuned the precision of the model, (Wang and Rubio-González, 2024) the picture shifted slightly, while mixed precision (FP16) was considered as the best middle ground in terms of efficiency and usage of resources. In these two studies, actual FP16 precision was used to optimize memory as much as possible and to maximize processing time. This approach is especially valuable for the fine-tuning of NLP models because, in general, models of this type are computationally intensive. They also added that throughput performance increases with dynamic batching and mixed-precision training, and also the big models can be used in practice conditions.

2.2 Benchmarking Techniques for Deep Learning Models

Some previous works are comparing various types of setups in an effort to establish the most optimal hardware configuration to use for certain models (Mahmoud, et al., 2019), (Li, et al., 2019), (Hao, et al., 2023), (Krak, et al., 2023) and (Russo, et al., 2022). The authors also compare the throughput, inference time, and memory of the deep learning models with the DSAs. They demonstrate that inference costs may be lower considerably while preserving the accuracy with the correct hardware. Some of the latest findings have pointed that design of models can impact on latency and throughput, especially where transformers are involved (Zhuang et al. 2024). Certain layers operation result in transformer structures being heavier than convolutional networks, which is mainly seen as a challenge of deploying transformers (Kim et al., 2023). The kind of differentiation exhibited in the two scenarios helps the argument that one can pick the right architecture if factors of deployment are considered. This includes the architecture of a concurrent search model that takes the hardware into account (Anthony et al., 2024), spatial-sequential hybrid architecture for better latency-throughput trade-offs than (Zhuang et al., 2024), and autoregressive architecture search for vision transformers (Wong et al., 2023). All these methods have been confirmed to yield the order of improvement in the throughput, energy efficiency and latency in the different forms of hardware platforms. For example, SSR architecture had up to 35.71x throughputs improvement over some GPUs as well as FPGAs (Zhuang et al., 2024), and TurboViT demonstrated approximately 3x reduction in latency and also optimizing throughputs to other efficient transform models (Wong et al., 2023). These enhancements do not only improve different aspects or make the learning of a machine faster but at the same time also enhance the feasibility of other applications of real-time systems as well as resource-constrained environments.

2.2.1 Comparative Studies on Inference Time vs. Throughput in Deep Learning

Recent research addresses the tradeoff between latency and throughput with transformer model inference. A spatial sequential hybrid architecture proposed by (Zhuang et al., 2024) combines sequential execution model and spatial execution model to achieve both high throughput with low latency. With these approaches, improvements by

6.4x in latency and 1.5x in throughput over previous state-of-the-art method were observed (Aminabadi et al., 2022), while TurboTransformers achieves state-of-the-art performance on GPU platforms (Fang et al., 2020).

The nature of the use-case determines whether models of deep learning are optimized for inference time or throughput. As an example, (Lin et al., 2023) investigated the design space tradeoff under low-latency autonomous vehicle systems as well as throughput-optimized large-scale language processing. This study highlighted that lower inference time is essential for real-time systems, while throughput optimizations may be beneficial in batch processing scenarios like the case of text generation or video analytics applications.

Decide whether to use inference time or throughput to enhance deep learning models depending on the application in question. For instance, (Lin et al., 2023) compared the low latency of autonomous vehicle systems with the high throughput required for language processing with large scale. They argued that though shorter inference time is crucial for real-time systems, throughput optimizations could be useful where many inputs are considered to be processed at once, for example, as in video analysis or as in text generation applications. Throughput targeting dynamic precision (FP16 vs FP32); CPU offloading can provide adaptable dynamic between resource usage and throughput and so can enable machine vision applications at resource-constrained edge devices.

Survey study of the investigation of inference suggesting the tool for latency and throughput that could be useful for selecting the primary combination of hardware and deep learning (Ulker et al., 2020). Such points might throw light to the fact that system architecture in addition to selection of the model should be dependent on the nature of the variety of applications for getting the maximum mean efficiency and performance for the AI undertakings such as the NLP or the CV.

- **Latency-Focused Approaches**: In (Totlani, 2024) states, regarding issues for performance, the authors present a complete method to reduce the inference latency of language models through model partitioning, edge caching, and light model deployment. Early pruning strategies and layer reduction that have been practiced helps in performing the inference at low latency in the edge devices for NLP (Tambe et al., 2020). Similar work by (Ghosh et al., 2023) introduced a DNN partitioning strategy for the platform-independent techniques for reducing the amount of time it takes to perform inference of NLP and CV at the edges of devices. Other related works include a distributed framework of edge systems (Hu et al., 2021) that also employs pipeline parallelism to improve inference performance and allows NLP and CV large models to run on devices with resource constrained.
- **Throughput-Focused Optimizations**: Dynamic batching coupled with a kernel fusion technique is used frequently to enhance the throughput, (Jeon et al., 2024) and (Nabavinejad et al., 2022). Such methods are even more efficient in the environment where the usage of GPUs is possible, for instance, in parallel computations. The use of dynamic batching and kernel fusion in deep learning accelerator NVIDIA TensorRT framework could boost the throughput to about fifteen times (Lijun et al., 2020).

However, the extra computational throughput in the deep learning does not mean it has the least inference latency due to the issues from deep learning frameworks (Fernandez et al., 2023). These bottlenecks mean there is always a need to fit high throughput and acceptable latency for efficient operation, and this is where we are noticing a combination of both two approaches above. These are characterized by diverse routes for optimization such as applying pruning and quantization besides aggregating throughputs in order to form efficient system performance. In inference for model of large language, the additional of both the CPU and GPU serve as a plus to throughput and latency (Park et al., 2023).

Recent research also indicates that when latency and throughput are both optimized, the best performance is attained when both are used, in order to get higher efficiency in natural language applications and at the same time maintain the accuracy (Antonius et al., 2023).

2.2.2 Resource Utilization and Energy Efficiency

The majority of works focus on resource utilization productivity by deep learning models, especially with DSAs and regarding edge AI. To deploy a model to use an optimal amount of CPU and GPU while using reasonable energy consumption, which is very important if you want to deploy models on the go in areas with little or no resource support. In light of this, the following work determines the distributed deep learning resource consumption wherein computing resources and energy saving on training time were implemented from (Frey et al., 2022).

- **Energy Efficiency on Edge Devices**: Therefore, (Tran & Tran, 2023) and (Zein et al., 2022) introduced energy efficient optimization for BERT and ResNet models on the edge devices. Moreover, models with TensorRT optimization have extended power consumption without affecting in throughput (Zhou et al., 2023), while TensorRT optimization increases the rate of transformer models inference on an edge device with reasonable accuracy (Nikoghosyan et al., 2023). Indeed, the extension of such models for training of large neural networks like ResNet and BERT on memory-constrained edge devices and energy optimization, (Patil et al., 2022). This optimization is efficient to provide real time services in various fields like autonomous systems, smart city, IoT gadgets etc.

New work also shows how the improved use of hardware through TensorRT and CUDA translated to how the models decreased power consumption without a compromise in accuracy. Regarding the efficiency, TensorRT outperforms CUDA each time (Alizadeh & Castor, 2024), and energy consumption increases to 1.53x compared to TensorFlow (Yao et al., 2020). In the case of NVIDIA Jetson platforms, TensorRT has acquired the optimization of energy consumption, cutting down to 55% of the GPU-only inference, according to (Jeong et al., 2022). The paper by (Suo et al., 2021) shows how the power consumption of AI applications on NVIDIA Jetson devices can be minimized using lightweight models and TensorRT optimizers. Based on the outcomes summarized in this paper, it is further concluded that deeper energy saving via hardware optimizations for deep learning inference use cases is possible.

- **CPU/GPU Utilization**: Some of the recent work has shown that mere consumption of the CPU and GPU utilization can lead to the drastic enhancement of energy efficiency. It also becomes possible to employ mixed precision like the extended precision of the fixed-point numbers and half precision that results in the simultaneous use of the GPU along with the achievement of higher speed in the application of machine learning (Gallouedec, 2021). These allow a computational rate that is higher in the throughput GPUs. Further on, (Dhakal et al., 2020) examined different scenarios of CPU and GPU utilization when deploying the introduced model in embedded devices; again, focusing on effective utilization of the hardware. The optimization of the GPU use in such platforms such as the Jetson series from NVIDIA is discussed in (Carvalho et al., 2022) on a variety of models including ResNet and BERT. Here, a new work by (Iyer et al 2024) provides a solution which is an allocation algorithm of the CPU and GPU backends to enhance the multiple models' AI inference throughput rate in resource-constrained devices. This new technique increases speed and reduces latency to the point where it is feasible to execute such complex programs in real-world applications in which computational processing is limited.

Using edge devices for AI models highlight the need to manage the CPU and the GPU in addition to the power management (Fanariotis et al., 2023) and (Dutt et al., 2023). Furthermore, (Wang et al., 2020) reveals in this paper that transformer-based NLP models can make the use of GPU devices when deployed with TensorRT get a boost and at the same time reduce the summation of the energy consumption. Most related works on the field basically have catered for the concerns pertaining to optimization from the machine resource standpoint which is the CPU or the GPU and energy utilization. They found out that optimizations made to TensorRT and which take into consideration the underlying hardware architecture, can help reduce power consumption while also not impacting performance; and enable the deployment of complex models in power-sensitive environments.

2.3 Comparative Analysis of Model Performance on DSAs

The above studies collectively reveal that there is an increasing application of optimization for models of deep learning based on the trade-off between inference time, throughput rate and resource utilization. Despite the fact that a lot of work has been published in the context of improving the performance of deep learning models for NLP and

CV tasks, we are able to locate little work done in the context of jointly optimizing for both inference time as well as throughput. Even so, there are limited attempts made in providing overall assessment of these optimizations for both latency and resource utilization, especially in the context of DSAs such as the NVIDIA RTX series.

This work contributes to the current literature by providing an extensive performance comparison by varying ONNX Runtime configuration with TensorRT-Optimized configurations on various deep learning tasks in NLP and CV categories. To address this knowledge gap, we benchmark both configurations in terms of inference time, throughput, and resource usage to establish how these metrics are related to DSAs and, more importantly, to provide pragmatic advice on how to implement AI models on DSAs because it has an immediate impact on the user experience and system performance in application scenarios.

Even then, energy efficiency is one of the most important factors for deploying AI models, especially in a resource-constrained environment, but apparently our work only focuses on inference time as well as throughput, while there is more concentration on achieving maximum utilization of GPU for high performance applications. Further work on this can be extended by finding energy optimization to go along with the inference time benchmark and throughput benchmark which is highlighted in this research.

3. METHODOLOGY

This section provides information on the benchmarking approach that has been adopted in this research study. This section also outlines the process of assessing the performance of deep learning models to NLP and CV tasks on DSAs. We consider more than one model configuration with respect to metrics such as inference time, throughput, CPU and GPU consumption, memory consumption and the size of the model itself. It was done with two setups, Standard configuration and TensorRT-Optimized configuration.

3.1 Experimental Setup

All experiments are conducted on a system with an NVIDIA GeForce RTX 3060 GPU on Windows 11 with CUDA enabled. The first category of the software was:

- Python 3.10.11
- ONNX Runtime 1.15.1
- PyTorch 2.5.1+cu118
- TensorRT 8.6 for hardware optimizations.

3.2 Models and Datasets

In order to perform a thorough analysis of the performance on diverse deep learning tasks, we chose representative models from two fields:

- **Computer Vision (CV)**: ResNet-50 model which is one of the classic deep learning of convolution neural network models for image classification problems. The experiments were conducted on CIFAR-10 dataset which consist of 60 000 32×32 color images in 10 classes. We have used sample test inputs for the given NLP task which consists of sentiment analysis tasks.
- **Natural Language Processing (NLP)**: A transformer-based text classifier, BERT (Base). To keep it simple, for testing purposes, we had the input of sample text input that mimics sentiment analysis (specific datasets such as IMDB or SST- 2). The evaluation itself was set entirely in terms of the time it took to infer, the throughput and how resources were used rather than in terms of the accuracy, in an attempt to measure how consistent and efficient the model was.

3.3 Configurations

In the evaluation, we used two different setups in order to see how optimizations impact the results in terms of inference time as well as throughput.

A. Standard Configuration

In this configuration, we used ONNX Runtime without any configuration that is optimized for batch size or hardware acceleration for both ResNet-50 and BERT models. The ONNX models were loaded using the following providers:

- TensorrtExecutionProvider
- CUDAExecutionProvider
- CPUExecutionProvider

The main goal of the work was to define a starting point for comparisons in terms of throughput, inference time, and resource utilization.

B. TensorRT-Optimized Configuration

The optimized configuration utilized TensorRT to utilize low-level optimization from the hardware, which includes dynamic batching and lower precision (FP16). We applied the following optimizations:

- **Dynamic Batching:** ResNet-50 model was exported to ONNX with dynamic shapes enabled, this feature allows the model to work with different sizes of batch and get better throughout.
- **TensorRT Precision:** As an example, in BERT only, the FP16 TensorRT engine was applied and the FP16 precision was used in order to boost the throughput.

The models in this configuration were loaded using the following providers:

- **TensorrtExecutionProvider** with FP16 support and engine caching enabled.
- **CUDAExecutionProvider** for GPU acceleration.

3.4 Performance Metrics

To evaluate the performances of each configuration, we measured the following metrics:

- **Inference Time:** For each of them, the total time it took to perform inference on the input/batch till the end results is in milliseconds (ms). Lower values show faster processing.
- **Throughput:** The number of inferences per second completed, it is measured in inferences/second. Higher throughput is favorable in contexts where there is a need to make fast predictions on a big number of instances.
- **CPU Utilization:** Average CPU usage in terms of the percentage (%), while executing the inference.
- **GPU Utilization:** Average GPU usage in terms of the percentage (%) during inference, particularly relevant when the configuration employs CUDA and TensorRT optimizations.
- **Memory Usage:** The peak memory usage at inference phase, in MB, which is important when the application is to be deployed on low memory devices.
- **Model Size:** The size of the model in terms of storage, that is the number of megabytes (MB).

3.5 Benchmarking Procedure

The steps involved in the benchmarking process were as follows:

1. **Model Conversion:** Specifically for the purpose of this paper, we used PyTorch to export the ResNet-50 and BERT models to format of ONNX. As for the TensorRT-Optimized configuration, the ONNX models are then optimized with TensorRT based on dynamic batching and using the 16-bit float point number precision.

2. **Data Preprocessing:**

- Images belonging to CIFAR-10 dataset were preprocessed through resizing, center cropping and normalization for ResNet-50 model.
- Also, text inputs were preprocessed then converted into tensors that are ready for the inference for BERT model.

3. **Inference and Measurements:** For each configuration, we ran the inference time, throughput, CPU and GPU utilization measurements ten times, once with 1 batch size (for the inference time analysis) and then with increasing batch sizes (for the analyses of throughput).

4. **Result Analysis**: To maintain consistency, all collected metrics were averaged over runs and then exported to CSV files for further analysis. To make it easier for readers to compare the performance of both configurations, we prepared charts.

3.6 Evaluation Tools and Libraries

The experiments were performed using a range of Python libraries to run, including:

- **ONNX Runtime** for model inference.
- **PyTorch** for conversion and model training.
- **TensorRT** for hardware-specific optimizations.
- **Matplotlib** for the results visualization.
- **Psutil and subprocess** for system resource monitoring.

We expect to demonstrate the performance trading between acquiring high throughput and low inference time by comparing results from the Standard configuration vs TensorRT-Optimized configuration, thereby demonstrating the implications for resource utilization in DSAs.

4. EXPERIMENTAL RESULTS

In this section, we present benchmarking results of deep learning models deployed on DSAs from both configurations: Standard configuration and TensorRT-Optimized configuration. Each configuration's performance is measured across key metrics, including inference time, throughput, memory usage, CPU and GPU utilization, and model size. Table 1 below summarizes the benchmarking results with each metric reported as an average of 10 inference runs for each configuration. In addition, Figure 1 and Figure 2 provide visual comparisons of the metrics for each configuration to facilitate side-by-side comparisons, which provides a clear view of how each configuration impacts performance for both ResNet-50 (CV) and BERT (NLP) models.

Table 1: Benchmarking Results Across Configurations

Configuration	Model	Inference Time (ms)	Throughput (inferences/sec)	CPU Usage (%)	GPU Usage (%)	Memory Usage (MB)	Model Size (MB)
Standard Configuration	ResNet-50	19.09	52.39	0.92	6.1	0.04	97.76
	BERT	13.19	75.84	0.6	6.3	9.56	417.65
TensorRT-Optimized Configuration	ResNet-50	18.50	432.56	1.0	5.6	0.04	97.76
	BERT	31.08	257.38	0.35	42.0	47.79	417.65

4.1 Standard Configuration (Latency-Focused)

The Standard configuration serves as a baseline to measure performance without advanced optimizations. Here, ONNX Runtime is used for inference without modifications for dynamic batching or precision reduction. The metrics obtained as summarized in Table 1 and as shown in Figure 1 are as follows:

- **Inference Time**: ResNet-50 and BERT models achieved inference times of approximately 19.1 ms and 13.2 ms, respectively.
- **Throughput**: Throughput values were measured at 52.4 inferences/second for ResNet-50 and 75.8 inferences/second for BERT.
- **CPU Utilization:** The average of using CPU for ResNet-50 and BERT models was around 0.92% and 0.6%, respectively.
- **GPU Utilization:** The average of using GPU for ResNet-50 and BERT models was around 6.1% and 6.3%, respectively.
- **Memory Usage**: Memory usage was also minimal, with ResNet-50 using approximately 0.04 MB and BERT around 9.56 MB during inference.
- **Model Size**: The storage size of ResNet-50 model was 97.8 MB, while BERT occupied a significantly larger 417.6 MB.

4.2 TensorRT-Optimized Configuration (Throughput-Focused)

The TensorRT-Optimized configuration incorporates hardware-specific optimizations such as FP16 precision and dynamic batching. These optimizations lead to the following metrics as they are shown in Table 1 and as shown in Figure 2:

- **Inference Time**: Despite optimizations, inference time increased slightly for BERT (31.1 ms) due to the added computational overhead from batch processing, while ResNet-50 achieved inference time at similar level of Standard configuration (18.5 ms).
- **Throughput**: The optimized configuration significantly improved throughput, achieving 432.6 inferences/second for ResNet-50 and 257.4 inferences/second for BERT.
- **CPU Utilization:** CPU utilization remained relatively low, with an average of 1.0% for ResNet-50 and 0.35% for BERT.
- **GPU Utilization:** GPU utilization increased notably in this configuration for BERT with the average of 42.0%, while the average of using GPU for ResNet-50 remained relatively low with around 5.6%.
- **Memory Usage**: Memory usage was higher in this configuration, particularly for BERT with around 47.8 MB, due to increased batch processing demands. While stay at the same level for ResNet-50 at 0.04 MB.
- **Model Size**: While using the same models, the storage size will not change, ResNet-50 model was 97.8 MB, while BERT occupied a significantly larger 417.6 MB.

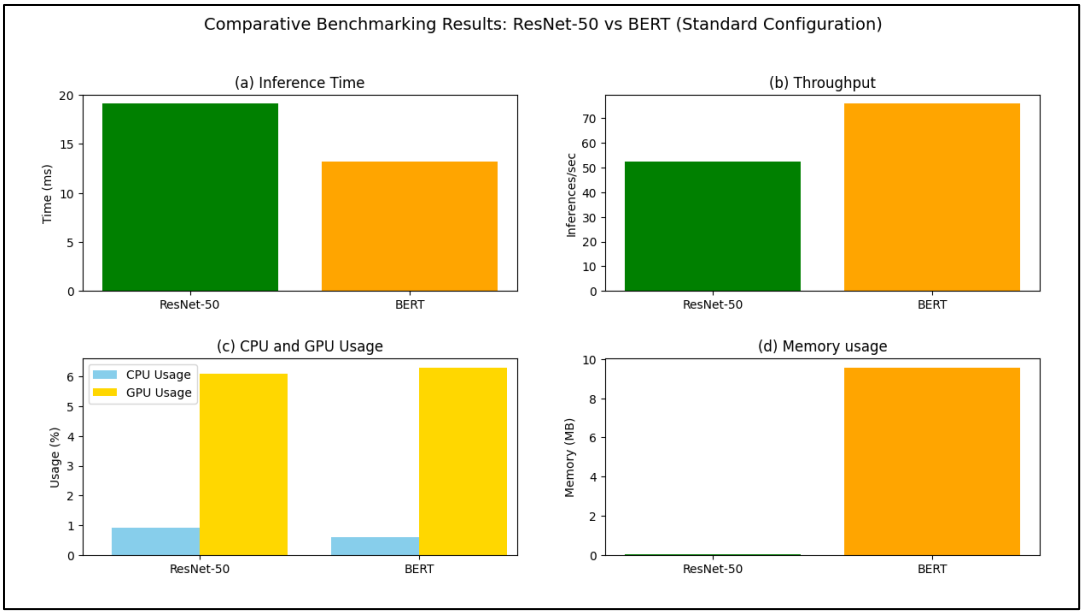

Figure 1: Comparative Benchmarking for Standard Configuration: ResNet-50 vs BERT

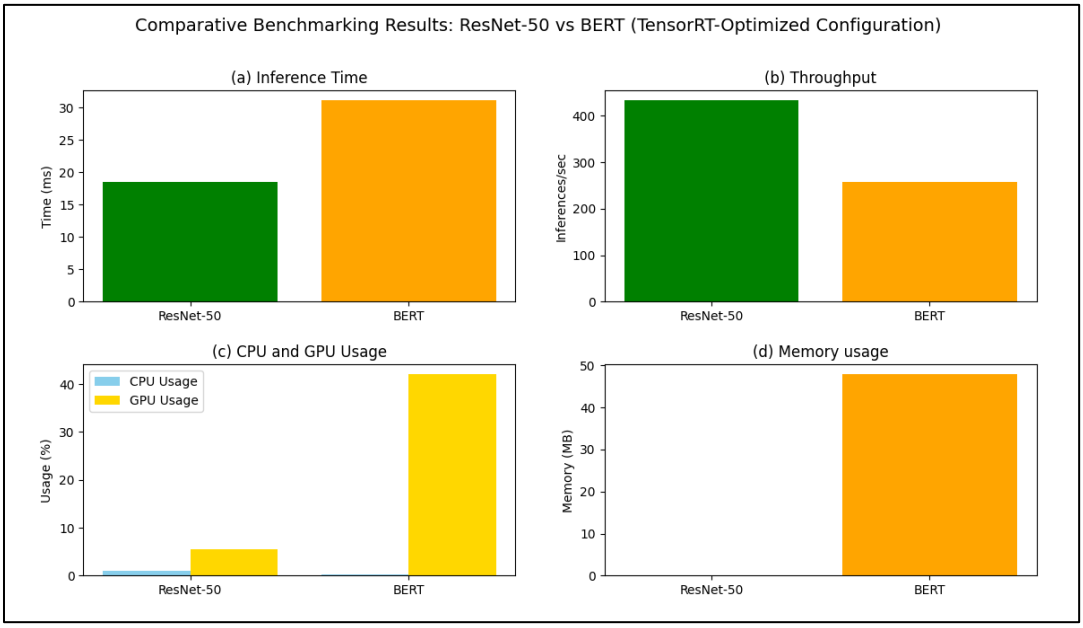

Figure 2: Comparative Benchmarking for TensorRT-Optimized Configuration: ResNet-50 vs BERT

5. ANALYSIS AND DISCUSSION

This section provides a general synthesis of the results presented in the experimental results section above. We study the performance of different aspects such as inference time against throughput, computational resource consumption against model size with focus on real-world application.

5.1 Trade-Offs in Inference Time vs. Throughput

If we analyze the TensorRT-Optimized configuration, we can observe that ResNet-50 has gained a tremendous improvement in throughput of 8.25x going from 52.4 to 432.6 inference/sec. Batch processing over large chunks of

data and FP16 precision are also useful for inference speed up is also pointed out. But this configuration does a higher inference time for BERT which is raised from around 13.19 to 31.1 ms.

- **Latency Implications**: The improvement in throughput during inference by the use of batch processing would be good for operations like image classification, but the latency implication may be marred in real time NLP applications where latency is a key factor.
- **Optimal Use Cases**: Standard configuration is ideal for applications where low latency predictions are desired in real-time, and the TensorRT-Optimized configuration will likely be ideal for applications that focus more on large volumes of data.

5.2 Resource Utilization Analysis

Resource consumption is another major deployment component as this affects the operation cost and system utility. Here, it is illustrated with additional details about how CPU and GPU load and memory consumption look like.

Comparing TensorRT-Optimized configuration, it lowered the GPU utilization to 6.3% in BERT from 42%, while even with ResNet-50 it would remain at a low rate of 5.6%-6.1%. GPU utilization was generally low for Standard configuration, indicating limited hardware acceleration. Additionally, Overhead of CPU was comparatively less for both the configurations.

In addition, TensorRT-Optimized configuration also enhances memory size of BERT, so the memory request rises from 9.56 MB to 47.79 MB. However, in the case of ResNet-50 the low usage of 0.04 MB remains stable independently from the configuration.

- **CPU Utilization**: CPU utilization was still low for each configuration; thus, it is safe to conclude that the workload is most reliant on the GPU, while CPU is not a limitation to this system.
- **GPU Utilization**: For BERT, the optimized configuration gets better utilization out of the GPUs by maximizing the compute for more batch tasks. It also shows the overhead in computation due to optimizations, and also the much better utilization of the hardware. We observed that GPU usage had a relatively low ratio in Standard configuration; therefore, the amount of hardware acceleration was also low. This indicates a more resource-progressive configuration suitable for high batch processes, which employ higher levels of input read. With regards to ResNet-50, GPU usage again remains constant at a low level for both configurations.
- **Memory Usage:** As could be seen from the BERT with TensorRT-Optimized configuration is more memory extensive, the reason is to the use of FP16 optimization and dynamic batching. Due to intermediate computations and caching that require higher memory, this is so. The memory utilization with ResNet-50 remains stable because its convolutional architecture and TensorRT optimizations are computationally slight, highlighting its appropriateness for production in low memory contexts.

5.3 Impact of Model Structure on Resource Requirements

After comparing model sizes of ResNet-50 and BERT, it reveals that BERT model consumes more space in memory of 417.6 MB, whereas ResNet-50 only requires 97.8 MB. That difference comes from the structures of transformer and convolutional models:

- **Convolutional Efficiency**: With deep convolutional architecture, ResNet-50 model used for the image classification involves parameter-projection in spatial dimensions, hence has a relatively small model size.
- **Parameter-Heavy Transformers**: Large parameters and layers in BERT transformer model to process text have resulted in heightened model size.

This difference suggests that we have to decide on a deployment model; for instance, BERT is a density representation of challenges in memory-constrained settings, and ResNet-50 is fully good for resource-constrained devices.

5.4 Potential Optimizations

According to our current findings, several areas are available to improve:

- **Fine-Tuning Batch Sizes**: Exploring different sizes of batch in the TensorRT-Optimized configuration might be useful to achieve a reasonable combination between the throughput and the inference time, especially for latency-critical NLP tasks.
- **Mixed Precision in CV Tasks**: With ResNet-50 and TensorRT using FP16 precision, there could be a further increase in throughput without affecting model accuracy.
- **Enhanced Resource Management**: Dynamic use of GPU resources with advanced scheduling techniques that respond to real time requirements is another level in performance improvement.

5.5 Summary of Findings

Now, it can be summarized that the throughput for both the models was more for the TensorRT-Optimized configuration compared to the Standard configuration. Yet using the optimized configuration meant we had to give up some of the optimizations that led to better resource usage.

Every configuration has a preferable deployment scenario:

- **Standard Configuration:** Low memory usage and latency sensitive applications should use it.
- **TensorRT-Optimized Configuration:** For high throughput workloads, if you can manage more GPUs and memory, it can be useful.

The visual summary of our findings from these tests on ResNet-50 and BERT are shown in Figure 1 and Figure 2 and give us an idea of the trade-offs and benefits of each configuration. Combining the results above provides optimization guidelines for DSAs in different deep learning applications.

6. RECOMMENDATIONS FOR PRACTITIONERS IMPLEMENTING OPTIMIZATIONS IN REAL-WORLD SCENARIOS

1. Using Tensor RT optimizations, it's a good approach to maximize throughputs when dealing with CV models such as ResNet-50. Dynamic batching can be efficient for real-time applications, if you have very different workloads.
2. If you need a fast inference (for example, if you are involved with autonomous cars or NLP Chatbot), set batch size to 1 and focus on minimizing inference time. If you are doing a lot of data processing (e.g. offline processing or batch jobs), then on the other hand, increasing batch size gives you better throughput.
3. Small models like MobileNet or distilled versions of BERT should be used in good practice to save edge devices resources. There are also techniques such as model quantization and edge caching for low latency and low memory footprint.
4. However, larger models like BERT or ResNet50 can be deployed to cloud or server machines on a more powerful GPU with full precision (FP32), that will get the most out of the accuracy for inference.
5. For instance, if you have applications where data preprocessing takes a long time, and you do not have much time to perform inference (e.g. video streaming or real time language translation), you will want to optimize your applications for inference time by minimizing the amount of data preprocessing and using light weight models. For latency sensitive tasks, use optimization frameworks such as ONNX Runtime’s execution providers (e.g. CUDA and TensorRT) to accelerate edge devices inference.
6. When reduced precision (FP16), the throughput can increase for BERT models for NLP without affecting accuracy.
7. Make sure you get the latency, throughput, and accuracy tradeoffs right for CV tasks by trying things such as image resolution adjustments, channel pruning and model distillation methods.

Without sacrificing on the efficiency and scalability of these deployments, these recommendations help practitioners to choose and tune models such that they perform at their best in real-world use cases.

7. CONCLUSION AND FUTURE WORK

A. Conclusion

We performed a comprehensive benchmarking analysis of two deep learning models which are ResNet-50 for Computer Vision (CV) and BERT for Natural Language Processing (NLP) across two different configurations. In our case, this is a standard ONNX Runtime setup as well as an optimized TensorRT configuration on an NVIDIA GeForce RTX 3060 GPU. The results demonstrated that a TensorRT optimized configuration for ResNet-50 provides quite a high throughput up to 433 inferences per second, while the CPU and GPU utilizations are within acceptable levels. However, standard ONNX Runtime results in a lower inference time making it better suited for latency sensitive applications.

An important finding is how model architecture affects resource consumption. Resnet-50 has heavy convolutional architecture, but the BERT model requires a lot more memory and computational resources as it is transformer architecture. This highlights the importance of thoughtful selection of the right model and configuration to focus on based on the need of a specific deployment environment, especially in a constrained resource environment.

B. Future Work

Although this work has brought to light some utilized trade-offs between inference time and throughput optimizations, there is much work left to be done. Future work can take the following directions:

1. **Optimizations Beyond TensorRT**: Identifying other optimization techniques that shorten model sizes and consume fewer resources (quantization and pruning), while maintaining accuracy.
2. **Cross-Hardware Benchmarks**: A further analysis is conducted for all the DSAs, but the current evaluation of the analysis is performed on specific DSAs, it would be intriguing to evaluate the portability and scalability of the optimizations on other DSAs (e.g. Google TPU, Intel Movidius, and Apple Neural Engine).
3. **Real-World Applications**: Perform benchmarking on a particular application, i.e., object detection or machine translation using real datasets to validate the optimizations on realistic deployment settings.
4. **Energy Efficiency**: Also show the influence of various configurations on energy consumption an important factor for edge deployments where power efficiency is a crucial concern.
5. **Auto-Tuning Pipelines**: Designing auto-tuning frameworks that automatically adjust configurations (e.g., batch size, precision) to optimize inference time as well as throughput in an on-demand manner as the workload changes.

Future research aspects enhance deployment capability in the real-world scenario by improving deep learning models that lead to an improvement in the efficiency of the models in handling NLP and CV.

REFERENCES

[1] Al Ghadani, A. K. A., Mateen, W., & Ramaswamy, R. G. (2020).*Tensor-Based CUDA Optimization for ANN Inferencing Using Parallel Acceleration on Embedded GPU*.

[2] Alizadeh, N., & Castor, F. (2024). Green AI: A Preliminary Empirical Study on Energy Consumption in DL Models Across Different Runtime Infrastructures.*2024 IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI (CAIN)*, 134–139.

[3] Aminabadi, R. Y., Rajbhandari, S., Zhang, M., Awan, A. A., Li, C., Li, D., Zheng, E., Rasley, J., Smith, S., Ruwase, O., & He, Y. (2022). DeepSpeed- Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale.*SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–15.

[4] Anthony, Q. G., Hatef, J., Narayanan, D., Biderman, S., Bekman, S., Yin, J., Shafi, A., Subramoni, H., & Panda, D. K. (2024). The Case for Co-Designing Model Architectures with Hardware.*International Conference on Parallel Processing*.

[5] Antonius, F., Alapati, P. R., Ritonga, M., Patra, I., El-Ebiary, Y. A., Orosoo, M., & Rengarajan, M. (2023). Incorporating Natural Language Processing into Virtual Assistants: An Intelligent Assessment Strategy for Enhancing Language Comprehension.*International Journal of Advanced Computer Science and Applications*.

[6] Carvalho, H., Zaykov, P. G., & Ukaye, A. (2022). Leveraging the HW/SW Optimizations and Ecosystems that Drive the AI Revolution.*ArXiv*. abs/2208.02808

[7] Chowdhury, S. S., Garg, I., & Roy, K. (2021). Spatio-Temporal Pruning and Quantization for Low-latency Spiking Neural Networks.*2021 International Joint Conference on Neural Networks (IJCNN)*, 1–9.

[8] Dhakal, A., Cho, J., Kulkarni, S. G., Ramakrishnan, K. K., & Sharma, P. (2020). Spatial Sharing of GPU for Autotuning DNN models.*ArXiv*. abs/2008.03602

[9] Duggan, A., Scully, T., Smith, N., & Giltinan, A. (2023). Profiling Power Consumption for Deep Learning on Resource Limited Devices.*SGAI Conferences*.

[10] Dutt, A., Rachuri, S. P., Lobo, A., Shaik, N., Gandhi, A., & Liu, Z. (2023). Evaluating the energy impact of device parameters for DNN inference on edge.*Proceedings of the 14th International Green and Sustainable Computing Conference*.

[11] Fanariotis, A., Orphanoudakis, T., Kotrotsios, K., Fotopoulos, V. E., Keramidas, G., & Karkazis, P. A. (2023). Power Efficient Machine Learning Models Deployment on Edge IoT Devices.*Sensors (Basel, Switzerland)*,*23*.

[12] Fang, J., Yu, Y., Zhao, C., & Zhou, J. (2020). TurboTransformers: an efficient GPU serving system for transformer models.*Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*.

[13] Fegade, P., Chen, T., Gibbons, P. B., & Mowry, T. C. (2023). ACRoBat: Optimizing Auto-batching of Dynamic Deep Learning at Compile Time.*ArXiv*. abs/2305.10611

[14] Fernandez, J., Kahn, J., Na, C., Bisk, Y., & Strubell, E. (2023). The Framework Tax: Disparities Between Inference Efficiency in Research and Deployment.*ArXiv*, abs/2302.06117.

[15] Frey, N. C., Li, B., McDonald, J., Zhao, D., Jones, M., Bestor, D., Tiwari, D., Gadepally, V., & Samsi, S. (2022). Benchmarking Resource Usage for Efficient Distributed Deep Learning.*2022 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–8.

[16] Gallouedec, Q. (2021). Mixed precision in Graphics Processing Unit.*ArXiv*. abs/2110.12794

[17] Ghosh, S. K., Raha, A., Raghunathan, V., & Raghunathan, A. (2023). PArtNNer: Platform-Agnostic Adaptive Edge-Cloud DNN Partitioning for Minimizing End-to-End Latency.*ACM Transactions on Embedded Computing Systems*,*23*(1), 1–38.

[18] Gu, R., & Becchi, M. (2020). GPU-FPtuner: Mixed-precision Auto-tuning for Floating-point Applications on GPU.*2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 294–304.

[19] Hao, Y., Zhao, X., Bao, B., Berard, D., Constable, W., Aziz, A., & Liu, X. (2023). TorchBench: Benchmarking PyTorch with High API Surface Coverage.*ArXiv*.

[20] Hawks, B., Duarte, J., Fraser, N. J., Pappalardo, A., Tran, N., & Umuroglu, Y. (2021).*Ps and Qs: Quantization-Aware Pruning for Efficient Low Latency Neural Network Inference*.

[21] Hayford, J., Goldman-Wetzler, J., Wang, E., & Lu, L. (2024). Speeding up and reducing memory usage for scientific machine learning via mixed precision.*ArXiv*. abs/2401.16645

[22] Hernández, N., Almeida, F., & Pérez, V. B. (2024). Optimizing convolutional neural networks for IoT devices: performance and energy efficiency of quantization techniques.*J. Supercomput.*,*80*, 12686–12705.

[23] Ho, K. H., Zhao, H., Jog, A., & Mohanty, S. P. (2022). Improving GPU Throughput through Parallel Execution Using Tensor Cores and CUDA Cores.*2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 223–228.

[24] Hu, Y., Imes, C., Zhao, X., Kundu, S., Beerel, P. A., Crago, S. P., & Walters, J. P. (2021). Pipeline Parallelism for Inference on Heterogeneous Edge Computing.*ArXiv*. abs/2110.14895

[25] Iyer, V., Lee, S., Lee, S., Kim, J. J., Kim, H., & Shin, Y. (2024). Automated Backend Allocation for Multi-Model, On-Device AI Inference.*Abstracts of the 2024 ACM SIGMETRICS/IFIP PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*.

[26] Jeon, J., Koo, G., Yoon, M. K., & Oh, Y. (2024). Adaptive Kernel Merge and Fusion for Multi-Tenant Inference in Embedded GPUs.*IEEE Embedded Systems Letters*.

[27] Jeong, E., Kim, J., & Ha, S. (2022). TensorRT-Based Framework and Optimization Methodology for Deep Learning Inference on Jetson Boards.*ACM Transactions on Embedded Computing Systems (TECS)*,*21*(1), 26.

[28] Jiang, B., Chen, J., & Liu, Y. (2023). Single-shot pruning and quantization for hardware-friendly neural network acceleration.*Eng. Appl. Artif. Intell.*,*126*, 106816.

[29] Kim, S., Hooper, C., Wattanawong, T., Kang, M., Yan, R., Genç, H., Dinh, G., Huang, Q., Keutzer, K., Mahoney, M. W., Shao, Y. S., & Gholami, A. (2023). Full Stack Optimization of Transformer Inference: a Survey.*ArXiv*. abs/2302.14017

[30] Krak, I. V., Barmak, O. V., Kuznetsov, V. A., Kondratiuk, S., Stelia, O., & Kasianiuk, V. (2023). Hardware Usage Improvement for Small Data Problem Solving by Deep Learning Methods.*International Conference on Information Control Systems & Technologies*.

[31] Kumar, S. S., Nayak, R. R., Kannampuzha, J. S., Ryoo, J., Rai, S., & John, L. K. (2023). Evaluation of Pruning Techniques.*2023 IEEE International Performance, Computing, and Communications Conference (IPCCC)*.

[32] Li, C., Dakkak, A., Xiong, J., & Hwu, W. W. (2019). The Design and Implementation of a Scalable Deep Learning Benchmarking Platform.*2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, 414–425.

[33] Liao, Y., Wang, C., Tu, C., Kao, M., Liang, W., & Hung, S. (2020). PerfNetRT: Platform-Aware Performance Modeling for Optimized Deep Neural Networks.*2020 International Computer Symposium (ICS)*, 153–158.

[34] Lijun, Z., Yu, L., Lu, B., Fei, L., & Yawei, W. (2020).*Using TensorRT for deep learning and inference applications*.

[35] Lin, J., Li, X., & Pekhimenko, G. (2020). Multi-node Bert-pretraining: Cost-efficient Approach.*ArXiv*. abs/2008.00177

[36] Lin, J., Yang, P., Zhang, N., Lyu, F., Chen, X., & Yu, L. (2023). Low-Latency Edge Video Analytics for On-Road Perception of Autonomous Ground Vehicles.*IEEE Transactions on Industrial Informatics*,*19*, 1512–1523.

[37] Mahmoud, N., Essam, Y., Shawi, R. E., & Sakr, S. (2019). DLBench: An Experimental Evaluation of Deep Learning Frameworks.*2019 IEEE International Congress on Big Data (BigDataCongress)*, 149–156.

[38] Minhas, U. I., Lee, J., Mukhanov, L., Karakonstantis, G., Vandierendonck, H., & Woods, R. F. (2022). Increased Leverage of Transprecision Computing for Machine Vision Applications at the Edge.*Journal of Signal Processing Systems*,*94*, 1101–1118.

[39] Myronyuk, D., & Blagitko, B. (2023). Nvidia Jetson NanoPlatform Using for Accelerating Image Recognition.*2023 IEEE 13th International Conference on Electronics and Information Technologies (ELIT)*, 153–156.

[40] Nabavinejad, S. M., Reda, S., & Ebrahimi, M. (2022). Coordinated Batching and DVFS for DNN Inference on GPU Accelerators.*IEEE Transactions on Parallel and Distributed Systems*,*33*, 2496–2508.

[41] Nikoghosyan, K., Khachatryan, T., Harutyunyan, E., & Galstyan, D. (2023). ACCELERATION OF TRANSFORMER ARCHITECTURES ON JETSON XAVIER USING TENSORRT.*INFORMATION TECHNOLOGIES, ELECTRONICS, RADIO ENGINEERING*.

[42] Niu, W., Kong, Z., Yuan, G., Jiang, W., Guan, J., Ding, C., Zhao, P., Liu, S., Ren, B., & Wang, Y. (2020). Real-Time Execution of Large-scale Language Models on Mobile.*arXiv: Computation and Language*.

[43] Park, D., Jo, S., & Egger, B. (2023). Improving Throughput-oriented Generative Inference with CPUs.*Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems*.

[44] Patil, S. G., Jain, P., Stoica, I., Dutta, P., & Gonzalez, J. (2022).*Training Neural Networks on Tiny Devices with Integrated Rematerialization and Paging*.

[45] Russo, E., Palesi, M., Monteleone, S., Patti, D., Mineo, A., Ascia, G., & Catania, V. (2022). DNN Model Compression for IoT Domain-Specific Hardware Accelerators.*IEEE Internet of Things Journal*,*9*, 6650–6662.

[46] Sakr, C., Dai, S., Venkatesan, R., Zimmer, B., Dally, W. J., & Khailany, B. (2022). Optimal Clipping and Magnitude-aware Differentiation for Improved Quantization-aware Training.*International Conference on Machine Learning*.

[47] Shishvan, O. R., & Soyata, T. (2018). Deep Learning Using CUDA. In *GPU Parallel Program Development Using CUDA* (pp. 425-437). Chapman and Hall/CRC.

[48] Suo, J., Zhang, X., Zhang, S., Zhou, W., & Shi, W. (2021). Feasibility Analysis of Machine Learning Optimization on GPU-based Low-cost Edges.*2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, 89–96.

[49] Tambe, T., Hooper, C., Pentecost, L., Jia, T., Yang, E., Donato, M., Sanh, V., Whatmough, P. N., Rush, A. M., Brooks, D. M., & Wei, G. (2020). EdgeBERT: Sentence-Level Energy Optimizations for Latency-Aware Multi-Task NLP Inference.*MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*.

[50] Totlani, K. (2024). Latency-Optimized Language Model Inference in Edge Computing Environments.*International Journal for Research in Applied Science and Engineering Technology*.

[51] Tran, L. D., & Tran, M. (2023). Enhancing Edge-Based Mango Pest Classification Through Model Optimization.*2023 RIVF International Conference on Computing and Communication Technologies (RIVF)*, 266–271.

[52] Tyagi, S., & Sharma, P. (2020). Taming Resource Heterogeneity In Distributed ML Training With Dynamic Batching.*2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, 188–194.

[53] Ulker, B., Stuijk, S., Corporaal, H., & Wijnhoven, R. G. (2020). Reviewing inference performance of state-of-the-art deep learning frameworks.*Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems*.

[54] Wang, Y., & Rubio-González, C. (2024). Predicting Performance and Accuracy of Mixed-Precision Programs for Precision Tuning.*2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, 152–164.

[55] Wang, Y., Lu, Y., & Blankevoort, T. (2020). Differentiable Joint Pruning and Quantization for Hardware Efficiency.*ArXiv*. abs/2007.10463

[56] Wang, Y., Wang, Q., & Chu, X. (2020). Energy-efficient Inference Service of Transformer-based Deep Learning Models on GPUs.*2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 323–331.

[57] Wang, Y., Wang, Q., & Chu, X. (2022). Energy-Efficient Online Scheduling of Transformer Inference Services on GPU Servers.*IEEE Transactions on Green Communications and Networking*,*6*, 1649–1659.

[58] Wong, A., Abbasi, S., & Nair, S. (2023). TurboViT: Generating Fast Vision Transformers via Generative Architecture Search.*ArXiv*. abs/2308.11421

[59] Yao, C., Liu, W., Tang, W. F., Guo, J., Hu, S., Lu, Y., & Jiang, W. (2020). Evaluating and analyzing the energy efficiency of CNN inference on high-performance GPU.*Concurrency and Computation: Practice and Experience*,*33*.

[60] Zein, H. A., Aoude, M., & Harkous, Y. (2022). Implementation and Optimization of Neural Networks for Tiny Hardware Devices.*2022 International Conference on Smart Systems and Power Management (IC2SPM)*, 191–196.

[61] Zhang, S., Zhang, L., Qin, M., & Guo, H. (2022). TensorRT acceleration based on deep learning photoelectric target detection.*Conference on Intelligent and Human-Computer Interaction Technology*.

[62] Zhdanovskiy, V., Teplyakov, L., & Belyaev, P. (2024). Efficient single- and multi-DNN inference using TensorRT framework.*International Conference on Machine Vision*.

[63] Zhou, Y., & Yang, K. (2022). Exploring TensorRT to Improve Real-Time Inference for Deep Learning.*2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*.

[64] Zhou, Y., Guo, Z., Dong, Z., & Yang, K. (2023). TensorRT Implementations of Model Quantization on Edge SoC.*2023 IEEE 16th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip (MCSoC)*, 486–493.

[65] Zhuang, J., Yang, Z., Ji, S., Huang, H., Jones, A. K., Hu, J., Shi, Y., & Zhou, P. (2024). SSR: Spatial Sequential Hybrid Architecture for Latency Throughput Tradeoff in Transformer Acceleration.*Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*.