Efficient and Dependable Deep Learning Systems

by

Salar Latifi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2023

Salar Latifi

salar@umich.edu

ORCID iD: 0000-0001-6933-615X

ACKNOWLEDGEMENTS

In this section, I would like to first acknowledge my advisor, Prof. Scott Mahlke, who has been guiding me through the PhD program since Fall 2016. Without his supports, feedbacks, and his generous time and thoughts, I wouldn't be able to reach to this point and achieve the same results. I would also like to acknowledge my dissertation committee, Prof. Ronald Dreslinski, Prof. Lingjia Tang, and Prof. Zhengya Zhang, for their valuable time and feedback throughout both my thesis proposal and defense. They have been highly friendly while helping me prepare this dissertation, which I am really appreciative of.

I would also like to acknowledge each one of my family members for their full support throughout my PhD degree. First, my mom and dad who have been patiently tolerating the distance while providing me the full strength to go through the lows and highs of this program. I also want to say special thanks to Babak, Baharak, Navid, and Aysan, my brothers and sisters for being on my side all the time and pushing me to cope with the difficulties and to try my best moving forward.

Finally, I want to say thank you to all my dear friends and labmates in Ann Arbor. We have made lots of enjoyable memories, good and bad moments, and controversial discussions, all helping me to have the necessary energy and recovery time to be able to go back and face the challenges of my thesis. Without having you, I wouldn't be able to reach my current position.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Deep learning is overhauling a plethora of applications, and we can see its impact in our day to day experiences-including voice assistants, autonomous vehicles and driving assist technologies, and e-commerce. With such a huge impact on human daily life, researchers are pushing towards better deep learning models with higher quality and better performance.

However, the trend of training bigger and higher quality models, potentially could lead to higher demands of the hardware resources to be able to process and service the daily applications in an efficient and timely manner. While there are also significant efforts going on in the hardware domain to adapt with the future and more expensive models and applications, considering the at-scale inference performance while designing novel deep learning architectures could also have a vital impact on the efficacy and practicality of these deep learning models.

On the other hand, Deep neural networks (DNNs) are also now starting to emerge in mission critical applications including autonomous vehicles and precision medicine. Therefore another important question is the dependability of DNNs and trustworthiness of their predictions. Considering the irreparable damage that can be caused by mispredictions, assessment of their potential misbehavior is necessary for safe deployment.

In this research dissertation, I am aiming to tackle both of these problems. The goal is to optimize different deep learning applications with respect to the reliability of their predictions, and improve their inference performance by reducing their latency and energy requirements. In the first two parts of this dissertation, I focus on vision models which have a wide function in mission critical applications, such as self-driving cars and precision medicine, for their efficient and safe deployment. And in the final part, I will be mainly

focusing on the performance and efficiency of the at-scale inference of recommendation systems, which are widely used in e-commerce and online advertisement, and are getting a significant attention due to their large financial impacts on the industry.

In Chapter II, I will be focusing on the dependability of image classifiers as the first vision application. First, I characterize the modern image classifiers and explore the root reasons for the misclassification cases that they exhibit. Then, I analyze the traditional confidence threshold checking as a reliability metric, and show its deficiencies. Next, traditional hardware reliability solutions such as modular redundancy are explored for their practicality in the deep learning domain. And finally, I propose a heterogeneous solution based on modular redundancy to detect up to 50% of the mispredictions while preserving the original accuracy levels of the baseline model.

In chapter III, I will be extending the learnings from image classification space to object detectors as the second vision application. I analyze the state of the art object detectors for different causes of unreliability. It is observed that nearly 40% of objects are completely missed without even being detected. Next, I modify the modular redundancy based heterogeneous system proposed in Chapter II to adapt for the low-latency and high throughput requirements of the object detectors. In addition, a new fusion algorithm is also proposed to combine the predictions of individual modules in the system with the goal of dependability and recovering the undetected objects of the baseline model. The resulting solution, which is called intra-sensor fusion, is shown to improve the average precision of the baseline by 3.45% with less than 25% overhead on the latency.

Finally in chapter IV, I will be focusing on designing hardware-aware and efficient Transformer architectures for the language modeling tasks. First, I will discuss the significant performance costs of the multi-head attentions. Then I will introduce the PLANER optimizer which takes an existing Transformer-based network and a user-defined latency target and automatically produces an optimized, sparsely-activated version of the original network that tries to meet the latency target while maintaining baseline accuracy.

CHAPTER I

Introduction

There is a diverse set of applications for deep learning in many different fields. Due to their critical role and immense functionality, substantial research is being done on deep learning in different domains. We have the deep learning researchers focusing on designing higher quality models and applying deep learning to new applications, as well as focusing on side effects of the deep learning in practice, including their execution performance or their resiliency against adversarial attacks [3, 135, 86, 87]. In hardware domain, the main focus is to design new hardware architectures/infrastructures or optimizing the existing ones to better suit for the computation demands of these models [20, 45, 65]. All of these efforts are necessary to enable efficient, reliable, and practical deployment of deep learning in real life.

In this dissertation, the main focus is to propose systematic solutions to optimize the existing deep learning solutions with two different targets: First, to improve the computational efficiency and inference performance of these models, and second, to introduce the concept of dependability in deep learning and facilitate reliable inference predictions.

Computational Efficiency and Inference Performance: Due to the significant number of applications that are deploying deep learning models, researchers are pushing towards models with higher accuracy levels by creating more complex and capable architectures. Figure 1.1 represents the accuracy improvement over the past decade on the image classifi-

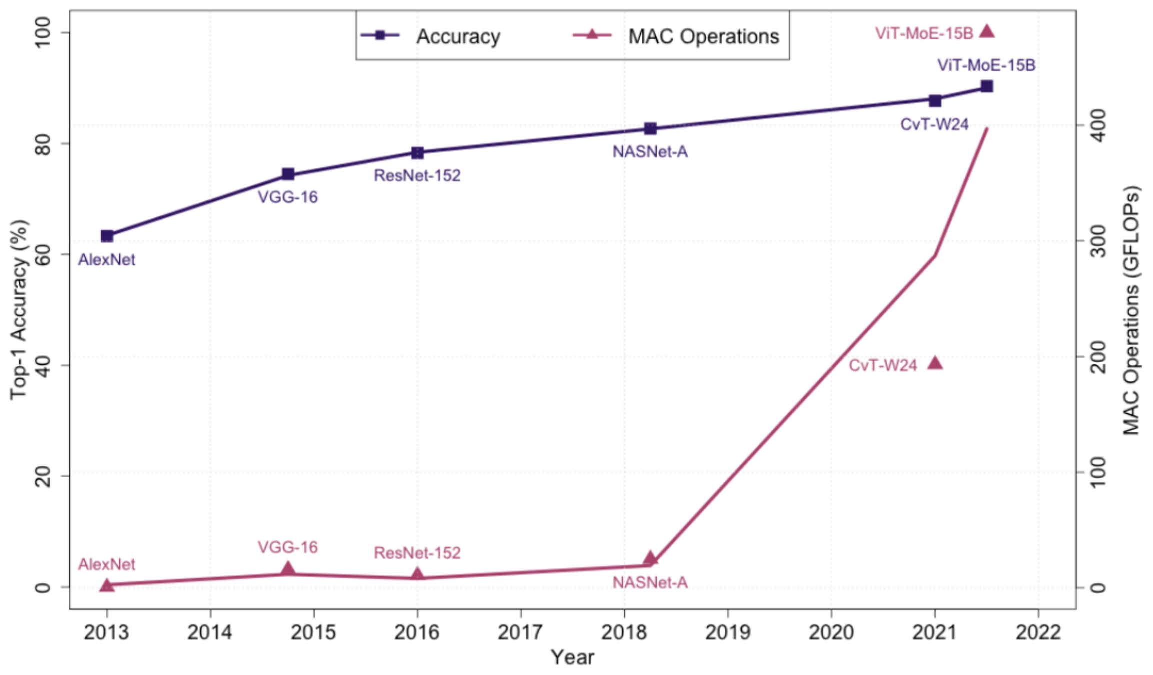
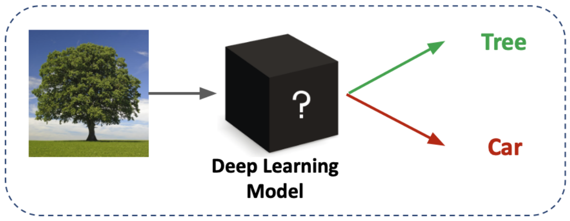


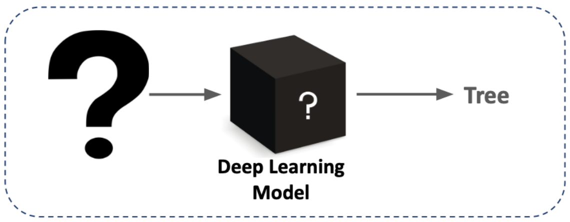
Figure 1.1: Progress of deep learning models in ILSVRC-2012 Image Classification task over the past decade.

cation task. We can see that classification accuracy on ImageNet dataset has been improved by more than 20% with the introduction of the new architectures. However, this accuracy improvement does not come free of cost, Figure 1.1 also depicts an exponential growth in the computational demands of these models. As a result, the cost of running these models for inference tasks is getting more expensive, which could also end up limiting the deployment of these models in time sensitive applications due to their extensive cost. We can see similar trends in other machine learning domains as well. For example, the number of parameters in the proposed architectures for language modeling domain has been increased by more than 300$\times$ in less than three years, e.g., 1.5B parameters in GPT-2 [92] and 450B parameters in PaLM [19].

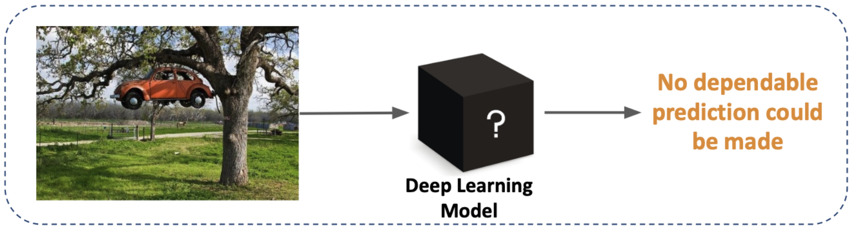
My first goal in this thesis is to optimize deep learning models with respect to their inference figures, which would make them more efficient and facilitate bigger and better models to be deployed in industry. All of the projects discussed in the next chapters share this efficiency optimization target.



(a) Training



(b) Baseline Inference



(c) Dependable Inference

Figure 1.2: Definition of dependability concept in deep learning inference.

Some of the common practices for optimizing the inference runtime of the deep learning models could be categorized in two different groups. The first group consists of the network compression techniques [34] such as pruning, quantization and tensorization. The common goal in this set of solutions is to enable architectures with lower performance and memory footprint by shrinking down the necessary computations while achieving a similar function as the baseline. The second group of the techniques is composed of the compiler optimization solutions [17]. This group includes approaches such as layer/tensor fusion, kernel configuration tuning, dynamic memory mapping/allocation. The main idea with this group is to perform the exact computations as the baseline in a more efficient manner according to the underlying hardware platform.

Most of these common practices rely on post-design/train optimization of the system. However, a question that I want to explore during this thesis is: *Can we further improve the efficiency by also considering hardware-awareness in our design/train process?*

Dependability in Deep Learning and Inference Prediction Reliability: With the emergence of the deep learning solutions in the mission critical applications such as autonomous vehicles or medical imaging, a key question is faced which is how to ensure the

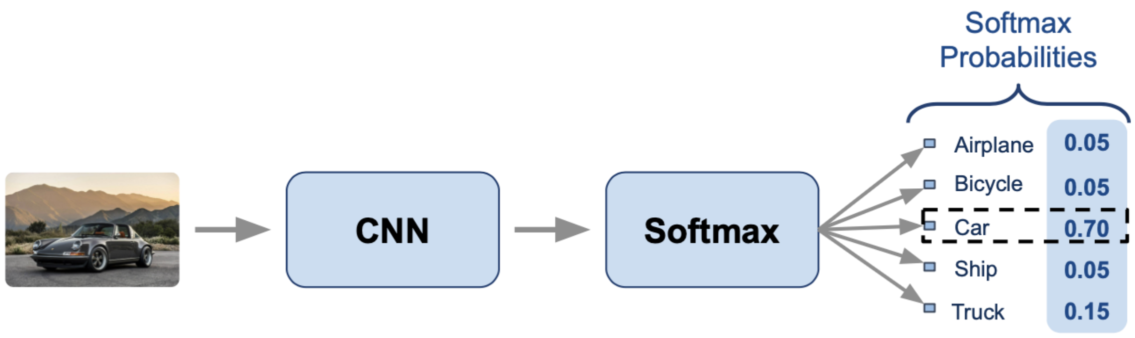


Figure 1.3: Confidence checking with Softmax probability outputs for accepting a reliable prediction.

reliability of the predictions made by these probabilistic algorithms. To be able to safely deploy deep learning models in the mission critical systems, we need to introduce the concept of dependability in the inference pipeline of these algorithms. Figure 1.2 illustrates the concept of dependability: during the training phase of a deep learning model(Figure 1.2a), we know the ground truth labels associated with the training samples fed to the system, therefore, we can easily evaluate the correctness of the predictions and optimize the model towards higher accuracy levels. However in a baseline inference pipeline(Figure 1.2b), both input samples as well as the ground truth outputs will be unknown. As a result, the deep learning model is treated as a black box, where the prediction outputs are blindly used without validating their correctness status. This introduces the dependability problem considering that deep learning models are a probabilistic approaches with limited accuracy levels, as a result, it is guaranteed for these algorithms to output incorrect predictions. What we need in a dependable inference pipeline, is the functionality for the deep learning system to prohibit making a blind prediction when it faces challenging inputs, or when a guaranteed reliable and correct prediction can not be made depending on the current status of the system and environment(Figure 1.2c).

A prominent example of introducing dependability in a deep learning inference pipeline, is to use the probability output of the Softmax layer as a metric of network confidence [35, 36]. Figure 1.3 presents the formulation of this approach, in which the probability output of the final prediction is associated with the confidence level of the network in the correctness of its prediction. And to validate the reliability of a prediction, a scalar threshold value could be used in which the output is assumed correct and reliable if the corresponding confidence value is higher than the threshold, and is considered an unreliable prediction vice versa. I will discuss the deficiency of this solution in detail in Section 2.2.2.

Another common practice to improve the dependability, is to introduce redundancy in the system. A perfect example of this approach is sensor fusion in the AV systems, in which multiple physical sensors are used in order to capture and process the surrounding environment. However, a notable problem with the redundancy is the runtime and energy cost of the system, which could easily surpass the limits if not designed carefully.

In this dissertation, my second target is to answer the following question: *How can we efficiently improve the reliability of predictions made by the probabilistic deep learning systems?*. The goal is to come up with systematic solutions to tackle the inherent inaccuracy of the deep learning models, and to design deep learning systems in which we have the ability to efficiently tell apart when the networks are making incorrect predictions, rather than treating them as black boxes and blindly relying on their raw results. The first two projects in this thesis are mainly aligned with this goal, each optimizing different deep learning solution in computer vision, i.e., image classification and object detection.

1.0.1 Enhancing the Reliability of Image Classifiers

In the first part of the thesis, the main focus is going to be on the reliability and dependability of the image classifiers. In this project, I first show the deficiency of current confidence-based methods as reliability measurement, and assess the effectiveness of traditional architecture reliability methods such as modular redundancy (MR). Then, I propose PolygraphMR and show that the combination of input preprocessing, smarter decision policies, and inclusion of prediction confidences can substantially improve the effectiveness of MR for DNNs. Next, I show how to prohibit explosive growth in the cost of MR by the help of reduced-precision designs and staged activations. Across six benchmarks, PolygraphMR detects an average of 33.5% of the baseline mispredictions with less than $2\times$ overhead.

1.0.2 Enhancing the Reliability of Object Detectors

Object detection is another important deep learning solution that is being deployed in mission critical applications such as autonomous vehicles. Ensuring the reliability of detections made by these models play an essential role in safety and robustness of these applications. By analyzing the available object detectors, I observe a significant number of objects being missed by the detector, which is a sign of unreliability. Being inspired by the sensor-fusion systems in AVs, I propose SoftFusion in this paper. SoftFusion introduces the concept of intra-sensor fusion wherein a diversity of inputs is efficiently synthesized during runtime to be evaluated by the original object detector. The predictions are then intelligently combined to realize more robust detections compared to the baseline detector. SoftFusion is evaluated across 7 different benchmarks over 2 different datasets, and is shown to achieve 3.45% gain in average precision with less than 24% latency overhead in comparison to the baseline.

1.0.3 Enhancing the Inference Efficiency of Transformer Architectures

In my last project, I will be focusing on optimizing the inference time and quality of Transformer architectures. These models are the backbone of many different tasks in natural language processing and computer vision. Recent developments in transformer architectures are scaling them towards higher accuracy levels [105, 14]. However, the result of this parameter scaling could be seen in significant increases in resource demands, both in memory footprint as well as computation, of Transformers. For instance, both Megatron-LM and GPT-3 deploy parameters in the scale of billion, and require 100s of GPUs for both training and inference.

In this project, I will present PLANER, a systematic search methodology to design latency-aware transformer models. The inputs to the system consist of a baseline transformer-based architecture, as well as the user's latency target, e.g., 60% latency of the baseline model. The output of the system includes the proposed optimized architecture that meets the runtime goals of the user, while maximizing the achievable accuracy levels. PLANER is evaluated across three different tasks in language modeling domains, and an speedup of over $2\times$ is achieved across the board while still maintaining the baseline accuracy levels.

1.0.4 Road Map

Throughout the remainder of this dissertation, I first elaborate on my major contributions towards efficiently improving the reliability and trustworthiness of computer vision domain in Chapters II and III. I then discuss my two-step optimization solution for decreasing the inference latency of the transformer architectures in Chapter IV. Finally, I summarize this dissertation in Chapter V and describe how we can extend the learnings in this dissertation to other tasks and deep learning domains.

CHAPTER II

PolygraphMR: Enhancing the Reliability and Dependability of CNNs [*]

2.1 Introduction

There is no doubt that Deep Neural Networks (DNN) have revolutionized many domains of applications including computer vision [96, 47], natural language processing [133], speech recognition [4] and handwriting recognition [122]. More and more products and services such as smart speakers, mobile photography, and social networks are integrating these algorithms to facilitate and enhance their usability and functionality. Many different types of DNNs are proposed each targeting different kinds of tasks, for instance, recurrent neural networks (RNNs) are available for tasks like speech recognition, or convolutional neural networks (CNNs) are well-established for image classification problems. Our target in this project are CNNs for image classification tasks.

CNNs are now moving from non-critical tasks such as gaming and labeling personal photos to mission critical tasks such as pedestrian identification for autonomous vehicles [129], steering commands generation for self-driving cars [12], and patient diagnoses with precision medicine [74, 94]. With mission critical tasks, incorrect answers can be disastrous. While CNN accuracies will continue to rise, robust and reliable CNNs must be realized

[*]Published in the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'20) [61]

with imprecise networks. Furthermore, CNNs may never achieve acceptable accuracies for mission critical tasks due to inherent limitations of their mathematical models. Constraints on computation, storage, and energy consumption may also place practical limits on growing CNN sizes, thereby limiting accuracy particularly for energy-constrained environments.

Despite the widespread use of deep learning, there are few metrics to determine the reliability of predictions made by CNNs. When a CNN assigns a label for an input image, there is not any established solution to determine correctness of the prediction and tell apart the correct ones from unreliable wrong answers. This phenomenon leads to an uncertain and unreliable environment when deploying CNN algorithms.

The most apparent solution to this problem is to design networks with higher accuracy. Considering the recent trends in the ImageNet image classification task [98], deeper models with more layers and parameters greatly improve the accuracy of these CNNs [56, 34, 40]. But, unless we have networks with 100% accuracy, which may not be possible to achieve, this solution is not sufficient by itself. Considering the vital demand for a practical solution, alternative approaches are necessary to assure reliable operation of CNNs.

In prior works [35, 36, 32], it is argued that the output of the last layer (Softmax) in CNNs can be used as a confidence meter for the network result. The output of the Softmax layer is a vector with size equal to the number of classes in the classification problem, which computes exponentially normalized values of the final fully-connected layer outputs. The final network prediction is the class number with the maximum value in this vector. Therefore, it is possible to interpret the vector values as the probability of assigning the input to the corresponding class, and if the probability value of predicted class is higher, we can make the statement that network is more confident about the generated result. We investigated the use of network confidence for reliability and show that DNNs produce substantial numbers of high-confident wrong answers, thus this metric does not solve the reliability problem by itself (see Section 2.2.2).

The machine learning community proposed network calibration [83] to improve confidence metric credibility. According to network calibration, the reason for high-confidence wrong answers comes from two sources: miscalibration of the CNNs; and, confidence values are not well-correlated with the actual accuracy of predictions [32]. In other words, if a CNN generates an output which has a confidence of 90%, we cannot make the statement that the probability of answer being correct is also 90%. Network calibration tries to solve this problem by correlating the confidence values with accuracy. Unfortunately, we demonstrate that even with well-calibrated networks, using confidence as a reliability metric still results in considerable mispredictions with high confidence (see Section 2.4.5).

In this project, our goal is to develop a practical method to design and realize systems of CNNs that can increase robustness and reliability of classification results with imprecise and currently available CNNs as the building blocks. The goal is not to increase accuracy but rather focus on the dependability of results. PolygraphMR uses input preprocessing techniques to develop different variations of a CNN and combines them as a modular redundant (MR) system of heterogeneous CNNs. It also employs a tunable decision policy engine that uses outputs of the networks and user's reliability demands to make decisions on the trustworthiness of each prediction. However, MR systems are notoriously expensive in terms of area and energy consumption. Thus, the footprint of each CNN variant in the MR system is reduced by scaling down the data precision and intelligently staging activations of individual CNNs so that most of the time only a subset of the MR system is active.

PolygraphMR enhances reliability by leveraging variations in behavior of each CNN that is trained in different circumstances and environments. It takes advantage of behavior diversity to detect symptoms of unreliability from the predictions of different CNN variants. A systematic approach is used to develop an efficient PolygraphMR system for any benchmark using the initial CNN as the basic building block. The functionality of PolygraphMR is orthogonal to the accuracy and topology of baseline CNN and can be applied to the future networks with higher accuracy levels for further reliability enhancement.

This project makes the following contributions:

- We demonstrate that high confidence, but wrong answers are a problem for most CNNs. We also show that current solutions such as using a confidence threshold or calibrating the network confidence do not satisfactorily solve the reliability problem.

- We introduce PolygraphMR, a heterogeneous MR system of CNNs that detects unreliable predictions by recognizing the symptoms of unreliability in the behavior variation among the constituent CNNs. Behavior diversity is synthesized by using a set of simple image preprocessing techniques for training/inference.

- We eliminate a large fraction of traditional MR overheads by scaling down the data precision of individual CNNs and deploying a resource-aware decision engine to activate only a subset of CNNs for each inference. We show that more aggressive data precision scaling without sacrificing prediction accuracy is possible with a PolygraphMR system than for a standalone CNN.

- We evaluate PolygraphMR system across three well-known image classification datasets and six CNNs and show that it is capable of detecting an average of 40.8% of mispredictions in a non-constrained resource environment, or 33.5% of mispredictions with less than 86.5% overhead on energy and latency.

2.2 Motivation

2.2.1 High-Confidence Wrong Answers

In order to better understand the reliability problem of current CNNs, we analyze errors of the six well-known networks for ImageNet dataset [23]. Benchmarks and their top-1 accuracies for the validation set are presented in Figure 2.1. The output of Softmax layer is used as the probability/confidence values of labels as suggested by previous works [35, 32]. Figure 2.1 presents the distribution of wrong predictions made by CNNs across the entire validation set. To ease the analysis, wrong answers are grouped into four categories based

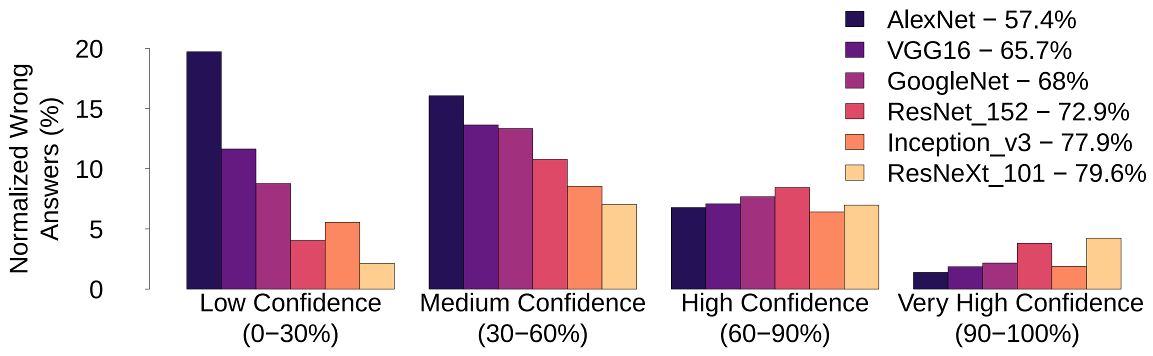


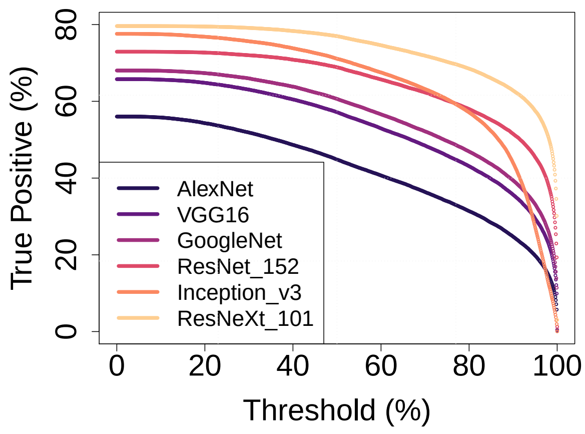
Figure 2.1: Histogram of normalized wrong answers generated by AlexNet [56], VGG16 [108], GoogleNet [110], ResNet_152 [34], Inception_V3 [111], and ResNeXt_101 [128].

on their prediction probabilities: low $(0-30\%)$, medium $(30-60\%)$, high $(60-90\%)$ and very high $(90-100\%)$ confidence. All bars are normalized by the total number of samples in the validation set, so the distributions can be compared to each other.

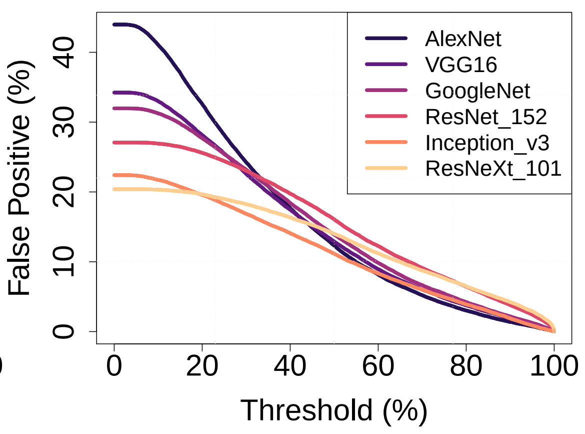
It is clear that the low and medium confidence wrong answers are the largest for most CNNs, which is intuitive. But, nearly 10% of the answers are wrong with high or very high confidence for each CNN. From a reliability perspective, 10% high confidence wrong answers is quite large. Figure 2.1 also exposes a more subtle, but concerning trend. As the CNNs become more accurate, severity of problem increases and there are a higher fraction of high confidence mispredictions. This shows that the higher accuracy is mainly coming from correctly predicting low confidence wrong answers of less accurate CNNs, and overall, confidence values for majority of predictions, whether correct or wrong, are shifted higher which makes them less reliable.

2.2.2 Limitations of the Confidence Metric

To explore the use of confidence as a reliability metric more deeply, one approach is to choose a minimum threshold for the prediction probability in order to consider it reliable. When the prediction probability falls below the threshold, the CNN output is unreliable. Figure 2.2 demonstrates the rate of undetected mispredictions, or False positives (FP), and correct predictions, or True positives (TP), as a function of the confidence threshold. At a



(a) True Positives



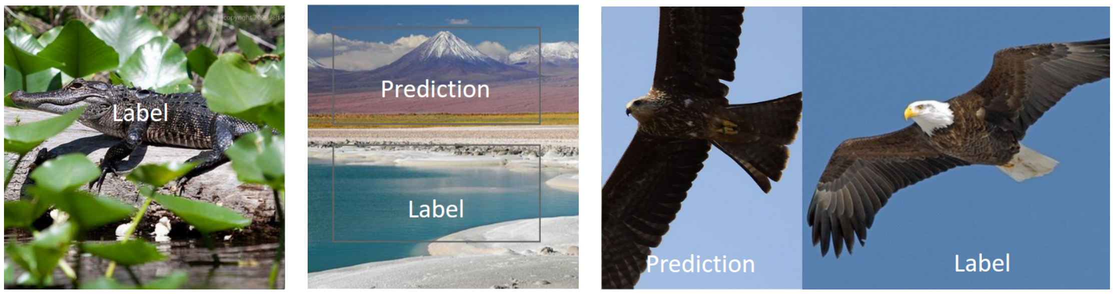
(b) False Positives

Figure 2.2: Effect of the probability threshold on the network predictions. (a) Distribution of true positives over threshold value. (b) Distribution of false positives over threshold value.

threshold of 0, none of the predictions are affected and the rate of FP and TP matches the original accuracy. As the threshold increases, the FP rate is decreased but at the same time, TP rate goes down as a portion of correct predictions are lost due to their low confidence. As depicted in Figure 2.2a, the change in TP rates for different CNNs tends to be similar, thereby maintaining a relatively constant difference in TP values regardless of threshold. Conversely, Figure 2.2b demonstrates the higher vulnerability of more accurate CNNs to the high confident wrong answers. Although the FP rate is initially lower in more accurate CNNs, the curves cross and the less accurate CNNs get lower FP rates at higher thresholds. This result again reinforces a somewhat counter-intuitive result that as accuracies go up, it is more difficult to eliminate the FPs and overall we have more high-confident wrong answers.

2.2.3 Misclassification Analysis

To get a better understanding of CNN behavior, we analyzed the wrong predictions of AlexNet over the validation set of ImageNet. The highest confidence wrong answers, i.e, mispredictions with confidence of 90% or more, which corresponds to about the top 5% of wrong answers, were manually examined to determine if any trends were apparent. Figure 2.3 summarizes the top 3 characteristics. The first characteristic is having poor image



(a) Image Details (b) Multiple Objects (c) Class Similarity

Figure 2.3: Misclassification analysis on ImageNet dataset.

detail which includes obstruction, obfuscation, blur, etc. Figure 2.3a presents an example where the crocodile is obstructed by leaves in the foreground. The second characteristic is to have multiple objects in the image as shown by Figure 2.3b. This picture contains two objects, a seashore in the front and a mountain in the back. In this example, network incorrectly predicted the mountain whereas seashore was the correct label. Finally, the third characteristic is the similarity between classes as shown in Figure 2.3c. The right image is labeled as bald eagle in the dataset, while it is mispredicted as a kite, and the left image is a sample of kite class, which shows its similarity.

The critical problem is not that these images are mispredicted, but rather the wrong prediction is made with very high confidence (e.g., over confident predictions). Humans may also classify these images incorrectly, but in contrast would likely have lower confidence due to the image characteristics. This points out one of the limitations of machine learning that the underlying mathematical models do not differentiate hard versus easy to classify images nor utilize confidence as an input to the training process. Rather, training designates a ground truth class for each sample. During training, weights of the network are continuously updated until the probability outputs are converged toward the ground truths. So, the network is forced to get the probability of the output corresponding to the label class number to 100%, making it more sensitive and reducing the generality of the trained model [87]. And as a result, the network ends up making over confident predictions.

We hypothesize that this limitation can be alleviated by creating a system of networks

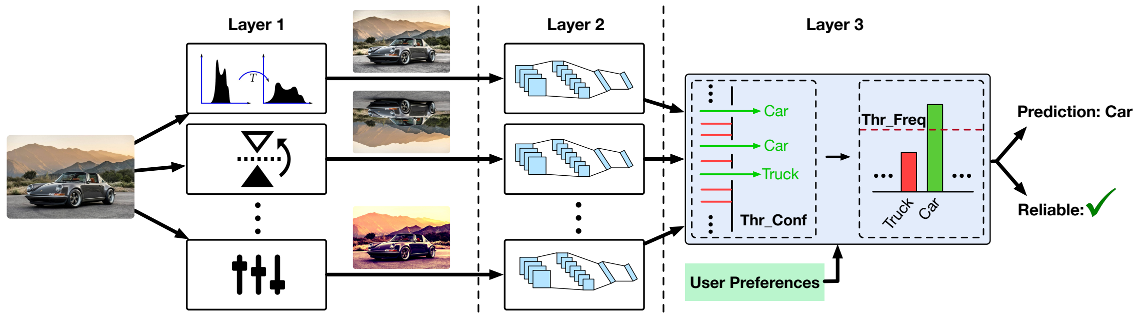


Figure 2.4: General design of a PolygraphMR system: Layer 1 is responsible to preprocess input image and inject diversity to system, layer 2 provides redundancy by making prediction on individual inputs, and layer 3 generates final prediction and decide on output reliability.

that provides both multiplicity and diversity. Multiplicity enables multiple independent predictions on the same input and can be used to adjust confidence based on agreement of answers. And diversity further enhances multiplicity by differentiating individual learners, thus increasing overall comprehensiveness of prediction space. We believe combination of these two characteristics can help our system to perform better at approaching and resolving more demanding inputs. The challenges are then to systematically create heterogeneous systems of multiple networks so the user is not burdened with this task and mitigate the multiplicative factors of energy/cost that deploying multiple networks will seemingly require. Evaluating the merits of this hypothesis while overcoming these challenges is the focus of PolygraphMR and the rest of this project.

2.3 PolygraphMR

2.3.1 Overall Design

PolygraphMR (**PGMR**) uses available imprecise CNNs as building blocks and leverages behavior diversity between them as symptoms of unreliability and likelihood of unreliable predictions. Figure 2.4 shows an overview of the system for single discrete inputs, still images in the illustration. The design consists of three layers: preprocessing units (Section 2.3.2), modular networks (Section 2.3.3), and decision making engine (Section 2.3.5).

For inference, the system operates as a group of heterogeneous DNNs (Layer 2) that are driven by a diverse set of real and synthetic inputs crafted from the original input by the available preprocessors (Layer 1). The goal of creating such a wide input diversity is to provide more information in order to render more confident predictions. Outputs of the heterogeneous group are analyzed to determine the final prediction and also whether or not the answer is reliable (Layer 3). The decision making portion is configurable and users can specify the target reliability for the system. The final output of this system could land in one of the following three domains: True Positives (TP) which are correct and reliable answers, False Positives (FP) which are undetected mispredictions, and Unreliable answers which are composed of detected wrong answers (false negatives) and correct answers that are undesirably marked as unreliable (true negatives). Our reliability goal in this project is to reduce FPs as much as possible by marking them as unreliable predictions, while keeping the desired TPs unchanged.

2.3.2 Layer 1: Pool of Preprocessors

Traditionally, diversity in CNNs is created by random initialization of weights in the training phase. However, as expected, the amount of diversity is very limited as will be shown in Section 2.3.3. Instead, we turn to prior work that has studied image preprocessors and shown they are helpful in improving the overall accuracy of CNNs [21, 53, 126]. Our goal of preprocessing is instead to create a group of CNNs with diversity in behavior. We hypothesize that diversity will help us identify inputs where the prediction should be treated as unreliable due to behavior variation across the CNNs.

Each CNN in layer 2 will be fed by transformed images generated by one of the preprocessors. We can use simple linear transformations like flipping, or more complex nonlinear functions like histogram equalization or contrast normalization as preprocessors. Each of these preprocessors introduces a different level of diversity to the system depending on the dataset or functionality of the preprocessor itself. We examine the effectiveness of each

Table 2.1: Image preprocessors and their functionality.

Preprocessor	Functionality
AdHist	Locally adjusts image intensities to enhance contrast
ConNorm	Locally normalizes image contrast
FlipX	Flips image in the horizontal axis
FlipY	Flips image in the vertical axis
Gamma	Gamma correction, controls the overall brightness
Hist	Adjusts image intensities to enhance contrast
ImAdj	Maps image intensity values to a new range

preprocessor and compare them together in Section 2.3.7. Table 2.1 presents the preprocessors that are used across our benchmarks in Section 2.9. Among these preprocessors, FlipX and FlipY are used more frequently. On the other hand, ImAdj is used only by one of the benchmarks. Overall, we observed that the preprocessors which preserve the vital features of the inputs while providing sufficient diversity to the system, are more frequently used across different datasets. Whereas a preprocessor like ImAdj, which heavily modifies the input features, like pixel colors, is less useful.

2.3.3 Layer 2: Heterogeneous Modular Redundancy

The base of layer 2 is Modular Redundancy (MR), which is well recognized as a standard solution for building mission critical computer systems [58]. With this approach, multiple copies of the unreliable module are instantiated and activated with a majority vote taken to decide the final answer. If the modules operate correctly most of the time, then the majority are likely correct for any single input. For probabilistic models including CNNs, the goal of using MR is separating reliable and unreliable answers rather than increasing the accuracy. Therefore, when the CNNs agree, the output is labeled as reliable and when they disagree then unreliable. To assess the success of MR, FP rates are measured.

We extend the traditional MR design with two modifications to account for the probabilistic behavior of CNNs:

1. We change the decision policy from majority voting to require a specific number of

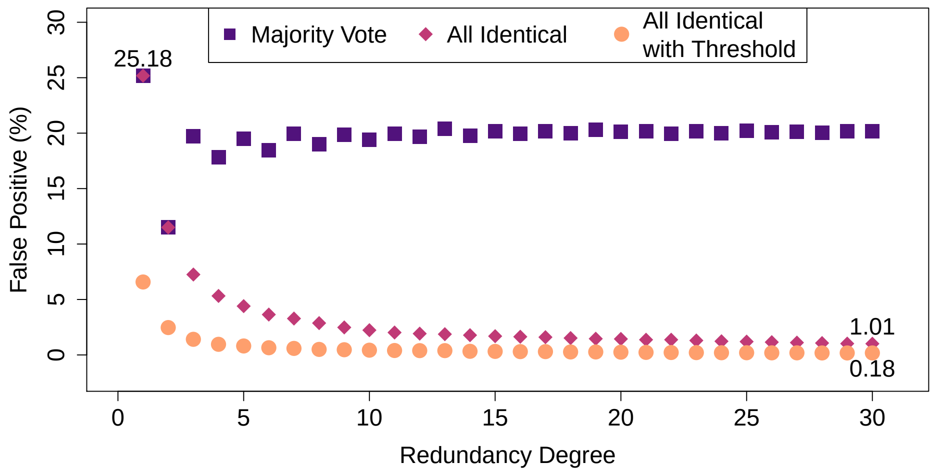


Figure 2.5: Different modular redundancy methods applied to ConvNet on CIFAR10.

networks to agree on the final prediction. We call this number the *frequency threshold* (Thr_Freq).

2. We include a confidence threshold for accepting the prediction result from each network. If the confidence of a specific prediction is below the threshold, it will be neglected. For the rest of the chapter, we call this value the *confidence threshold* (Thr_Conf).

To evaluate traditional MR, we run an experiment using ConvNet on the CIFAR-10 dataset [55]. This dataset contains 10k images which are classified into ten categories (1k images per category) with a baseline accuracy of 74.7%. The degree of MR varies from 2 to 30. MR networks are created by instantiating *n* copies of the baseline CNN, randomizing the starting weights, and training each on the original dataset. Each CNN ends with different weights and thus behaves differently. Figure 2.5 shows the impact of redundancy degree on the number of wrong answers predicted by the MR system with different decision policies: 1) Traditional MR with majority voting (Majority Vote), 2) MR with a Thr_Freq equal to the number of CNNs, which will require all networks to predict the same label and is the most restrictive threshold (All identical), 3) the previous MR design plus a Thr_Conf of 75% (All identical with Threshold). The Thr_Conf is chosen somewhat arbitrarily, but which corresponds to a relatively high confidence threshold. Note that for the design with majority voting if two classes share the same highest frequency, the result is considered unreliable.

From Figure 2.5, majority voting does not provide a significant decrease in FPs regardless of the redundancy degree. The FP rate flattens at about 20%, after starting at 25.2% with a single CNN. MR with the Thr_Freq is much more successful, reducing the FP rate down to 1%. MR with both Thr_Freq and Thr_Conf decreases the FP rate even further to 0.18%. However, the latter two solutions have an undesirable side effect of substantially decreasing the number of TPs. For example with the All Identical method, to achieve a 1% FP rate, TPs reduce from 74.7% with a single network to 40.4% because a large number of correct predictions are considered unreliable.

We make three conclusions based on these results:

First, traditional MR is incapable of effectively reducing the FP rate regardless of the redundancy degree, whereas they are shown to be practical while applied to other computer reliability problems such as transient faults [106, 7]. The reason is that with traditional computer systems, execution of applications is flawless in a fault-free setting and faults are relatively rare. However, CNNs are inherently faulty regardless of the hardware, and errors are much more common due to the inherent inaccuracy of the mathematical models. This results in a significant disagreement between individual CNNs and poor results.

Second, from the majority voting results, the effect of behavior diversity reduces the FP rate by 5% without any loss of TPs. Conversely, the all identical voting has much larger drops in FPs, but loses too many TPs. Thus, it is important to have less restrictive voting mechanisms like majority while at the same time injecting larger amounts of diversity than simply random initial weights, which led to our decision to use input preprocessing.

And finally, single CNNs are expensive in terms of computation, storage, and energy consumption. Thus, the multiplicative cost increase of CNNs in an MR system could easily become infeasible. Thus, we need to deploy an resource-aware implementation of our MR system.

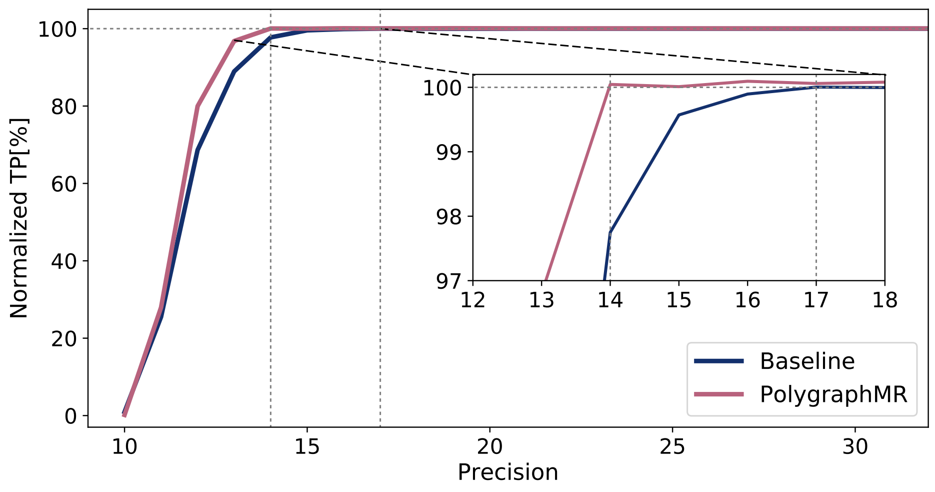


Figure 2.6: Effect of precision reduction on original and PolygraphMR system using AlexNet.

2.3.4 Resource-aware MR (RAMR)

Due to inherent redundancy in Layer 2, computation of PolygraphMR introduces new energy and latency overheads per inference. To mitigate a portion of these overheads, we explore narrow-precision floating-point representation for each CNN in the system.

We follow a similar implementation introduced in [46, 38] and reduce the overhead of data transfer by precision reduction. We assume all weights and intermediate values are using a lower precision. Hence, it is possible to pack them together during both on-chip and off-chip data transfer. As a result, the reduced traffic on memory hierarchy leads to higher utilization of compute units and higher performance.

Although there is an opportunity to reduce the cost of PolygraphMR by using narrow-precision computation, we should acknowledge that this solution is not unique to our system and can be applied to any CNN for energy saving purposes. To analyze the effect of lower precision, we run an experiment using AlexNet [55] on the ImageNet [23] dataset. The goal is to compare the degree of precision reduction on a PolygraphMR system with a baseline of an individual AlexNet. We hope that the combination of diverse CNNs in our system would be more resilient against the negative effects of lower precision on accuracy. Therefore, we would be able to further reduce the precision of each individual CNN in comparison to the baseline.

Figure 2.6 presents the accuracy of both PolygraphMR and the baseline CNN with respect to the precision they are using. We focus on maintaining the accuracy level of the baseline CNN, which is 57.4% for AlexNet, and investigate the possibility of achieving this accuracy at each precision level. At the first glance, both designs seem to be responding equally to the reduction of precision. But in a closer look, we can see that the baseline AlexNet starts to lose its accuracy passing 17 bits precision level. On the other hand, PolygraphMR can still tolerate lower precision levels of up to 14 bits. In fact, the accuracy of each individual CNN in PolygraphMR also follows a similar trend to the baseline AlexNet, but combining their predictions and then making decision performs similar to ensembles and compensates for the individual accuracy drop. As a result, we can further reduce the precision on PolygraphMR system and mitigate a portion of the multiplicative energy and latency overhead.

2.3.5 Layer 3: Decision Engine

The last layer of PolygraphMR is responsible for collecting the outputs from layer 2, generating the final prediction of the system, and determining whether or not the prediction is reliable. This happens in two steps during the system inference phase. First, the decision engine compares the probability vectors generated by the softmax layer of each network, with the designated Thr_Conf value and forms a histogram of acceptable votes for each class. As a result, all labels in individual output vectors, which meet threshold restrictions, are recorded in the histogram. Next, the decision engine reports the class label with the highest frequency as the final prediction of the system and reports the reliability of the label by comparing the respective frequency with the preselected Thr_Freq.

The appropriate values for Thr_Freq and Thr_Conf are determined after training the MR networks and during an offline profiling stage. First, the value space for the set of thresholds is swept, and the respective TP and FP rates of the design points over the validation dataset is recorded. This process is not time consuming and has a negligible overhead compared

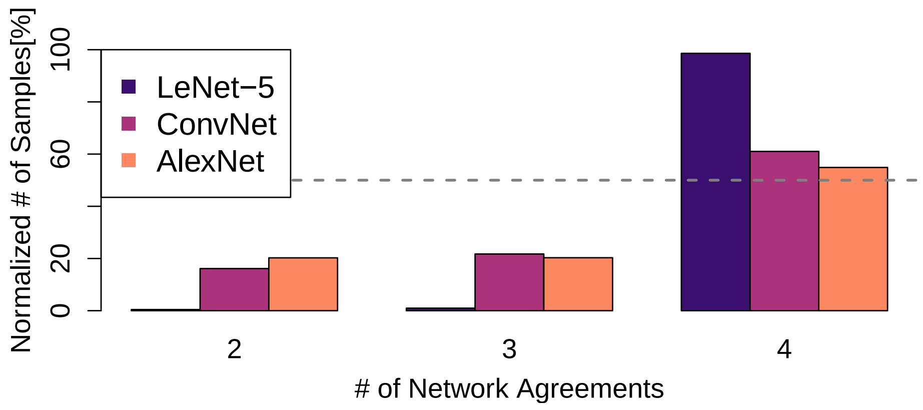


Figure 2.7: Histogram of prediction agreements in a system with 4 CNNs, profiled on LeNet-5, ConvNet, and AlexNet.

to the actual training of the MR networks. Next, a Pareto frontier for the threshold values, which maximizes the TPs and minimizes the respective FPs, is formed. And finally, a set of Thr_Conf and Thr_Freq values is selected from the Pareto frontier based on the user demands, which might be a specific TP or FP limit.

The threshold values selected in the profiling stage, will be fixed during the inference phase of the system. But, if the user demands are updated at any point, a new set of threshold values can be selected from the Pareto frontier to meet the new requirements.

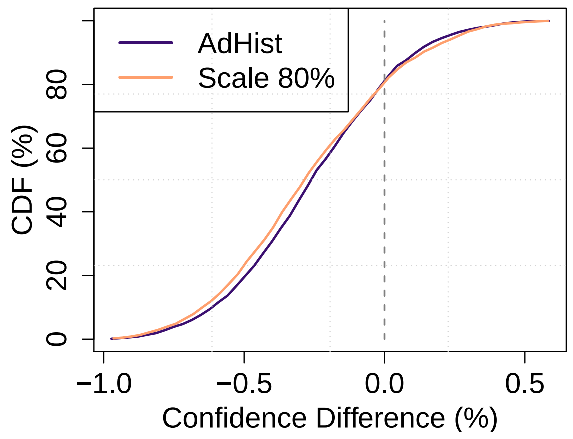
2.3.6 Resource-aware Decision Engine (RADE)

To further reduce the performance overhead of Layer 2, we deploy a resource-aware decision policy. Unlike the traditional MR with majority voting which requires all prediction outputs from every network to be present in order to make the final decision, PolygraphMR can decide on the reliability of the prediction if a subset of the networks provide the same label with a minimal confidence.

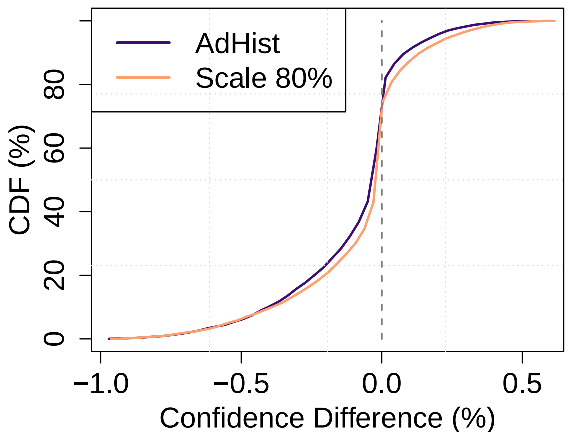
To analyze the number of networks that need to be activated, we run an experiment using a PolygraphMR with four networks on three benchmarks: LeNet5 [63] on MNIST [64], ConvNet [54] on CIFAR-10 [55], and AlexNet [55] on ImageNet [23] dataset. Our goal from this experiment is to gain a general idea about how often we need to activate all networks to get a reliable prediction. We collect the prediction results of each network to see how often they are in agreement with each other. To simplify the experiment, we do not use any Thr_Conf and just gather the top prediction from each network regardless of its confidence. Figure 2.7 presents histogram of network agreements for our benchmark. The x-axis shows the number of agreements among the four CNNs, and the y-axis represents the corresponding normalized frequency over test samples. We can see that in more than 50% of times, we don't need to activate all the CNNs since their prediction results are in harmony with each other. This gives us the opportunity to activate just a portion of CNNs and reduce the performance overhead. But, the key point is to decide on which network(s) to activate, since there is a possibility that we would face an input where predictions from the selected networks might conflict with each other. An oracle decision engine would be the one which activates the single reliable CNN per input sample. But even if it is possible to design such an engine, it is likely complex and would consume considerable energy.

In order to design a resource efficient decision engine, we use a priority scheme on CNN activations. In this case, we can stage activation by activating a batch of high priority CNNs first to check their predictions, and only continue to execute other CNNs if we are not able to determine the final answer after the first round of invocations. This approach can give us the opportunity to have early detection of TP or unreliable answers, and result in lower energy consumption.

To come up with a priority scheme, we statistically analyze the contribution of each CNN in the system. In other words, we record the frequency of instances that each network provides a correct label to the decision engine over a specific number of test cases during training. Next, we use the measured frequency numbers to give priority to each network. In the inference phase, we first execute the top Thr_Freq networks to see if we can determine the final answer. Next, we move to other networks based on their contribution until we're ready to generate the output. Energy saving results and latency improvement of this decision engine are discussed in Section 2.4.3.



(a) Wrong predictions.



(b) Correct predictions.

Figure 2.8: Comparing effects of AdHist and Scale 80% on confidence changes with respect to original CNN.

2.3.7 PolygraphMR System Design

We use a two-step procedure to select the best preprocessors and form a PolygraphMR system for each individual application and dataset. First, we compare the relative performance of preprocessors regarding the potential behavior diversity each can introduce. Next, a set of candidate preprocessors is used in a greedy approach to select the final preprocessed networks. As discussed in Section 2.3.2, a wide range of linear and non-linear preprocessors are available to use. But, not all of them provide sufficient behavior diversity to justify their energy overhead. Hence, the first step is to compare preprocessors and select the best ones.

For each input instance, we profile the difference between prediction confidence of the baseline CNN and each preprocessed CNN, called *delta*. The delta values are then used to compare pairs of preprocessors. Figure 2.8 presents a comparison between AdHist (in which image intensities are locally adjusted to enhance contrast) and Scale 80% (where input image is scaled down and up by 20% to soften noise levels) on ConvNet. The x-axis presents *delta* values, and y-axis projects the corresponding cumulative distribution. Figure 2.8a displays the distribution of *delta* values for inputs that are initially mispredicted by the baseline CNN. As shown, AdHist has a higher probability in negative *deltas*. Even though the difference

Dataset	CNN	Accuracy	# of Layers	# of Classes
MNIST	LeNet-5 [63]	99.01%	5	10
CIFAR10	ConvNet [54]	74.70%	4	10
	ResNet20 [34]	91.50%	20	10
	DenseNet40 [41]	93.07%	40	10
ImageNet	AlexNet [55]	57.40%	8	1000
	ResNet34 [34]	71.46%	34	1000

Table 2.2: Benchmark set used to evaluate PolygraphMR.

is relatively small, the likelihood of having lower confidence for mispredicted results with the AdHist is higher. Thus, the probability of getting the same misprediction with AdHist is lower than Scale 80%, and AdHist would be a better preprocessor to introduce behavior diversity for this network. In addition to Figure 2.8b, Scale 80% has higher probability in negative *deltas*, which means there is a higher chance of getting a lower confidence for samples that are correctly predicted by the baseline. Hence, the probability of predicting the same correct answer is lower compared to AdHist. This comparison shows that the AdHist has a higher potential for introducing behavior diversity to our system. In our experiments, preprocessors listed in Table 2.1 are the most frequently selected ones.

The second step is to run an iterative greedy approach on candidate preprocessors to select the final networks for PolygraphMR. First, it starts by selecting a baseline CNN as the first network in layer 2. It also gathers the respective TP and FP rates to be used as baseline. Next, each of the preprocessed CNNs is separately selected and added to the current configuration and reduction in FP rate is recorded. Then, the best preprocessor for the current iteration is added to the final design. We keep iterating through this algorithm until a predefined maximum number of CNNs is reached.

2.4 Evaluation

We evaluate PolygraphMR in two stages. First, we focus solely on reliability improvements without considering any performance optimizations in Section 2.4.2. Next, we apply RAMR and RADE to reduce the overheads in Section 2.4.3.

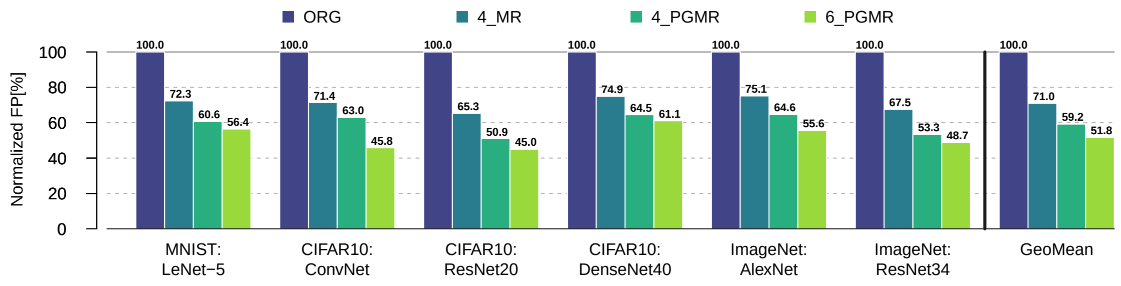


Figure 2.9: Comparison of normalized FP rate for each design on different benchmarks (TP rate of all design points are matching baseline accuracy level).

2.4.1 Methodology

Benchmarks: Three datasets are selected for evaluation: MNIST [64], CIFAR-10 [55], and ImageNet [23] dataset. As for the CNNs, we choose six networks with different ranges of accuracies and topologies to show that our solution is independent from the original network correctness level and architecture. Table 2.2 presents the accuracies and specifications of each benchmark.

Comparisons: To evaluate reliability of optimal configurations proposed for each benchmark, two comparisons are made. We compare PolygraphMR with N networks (N_PGMR) with the original baseline network (ORG), and with an MR system composed of N networks (N_MR).

Evaluation Metrics: For the reliability comparison metric, we use FP rate of design points where no desirable correct predictions are lost. Therefore, the FP rates included in the rest of the results section correspond to design points with normalized TP of 100% of the baseline network. FP rates are also normalized with respect to the ORG FP rate.

We also use latency and energy of original CNNs for each benchmark as our performance measurement baseline.

Preprosseing: We use OpenCV library and MATLAB for preprocessing of the datasets during training and testing.

Reliability Modeling: We use Caffe [44] framework to implement, train and test our CNNs. To evaluate the accuracy of low-precision networks used in RAMR, we modify the

Dataset	CNN	Configuration
MNIST	LeNet-5	ORG, ConNorm, FlipX, Gamma ($\gamma = 2$)
CIFAR10	ConvNet	ORG, AdHist, FlipX, FlipY ($\gamma = 2$)
	ResNet20	ORG, FlipX, FlipY, Gamma ($\gamma = 1.5$)
	DenseNet40	ORG, ImAdj, Gamma ($\gamma = 1.5$), Gamma ($\gamma = 2$)
ImageNet	AlexNet	ORG, FlipX, FlipY, Gamma ($\gamma = 2$)
	ResNet34	ORG, FlipX, FlipY, Gamma ($\gamma = 2$)

Table 2.3: 4_PGMR configuration selected for each benchmark (ORG stands for original baseline network).

Caffe library by replacing default cuDNN framework with our custom CUDA kernels that support variable precision. This enables changing inference precision by truncating values of load and store instructions to the desired settings. We use a unified precision throughout the network and for all layers.

Performance Modeling: Energy and latency of PolygraphMR are measured on a machine with Intel Core i7-5930K CPU and an NVIDIA TITAN X (Pascal) GPU. Inference of each CNN in PolygraphMR is done sequentially, and at the end, decision policy is deployed to get the final result. In this pipeline, preprocessing and CNN inference are executed on GPU, whereas decision policy is based on CPU.

To measure the performance of low-precision CNNs, we model data packing and unpacking in software which is then integrated in our kernels. Next, GPGPUsim v4.0 [6] and GPUWattch v1.0 [66] are used with TITAN X configuration [50] to run the simulation on individual benchmarks.

2.4.2 Reliability Results

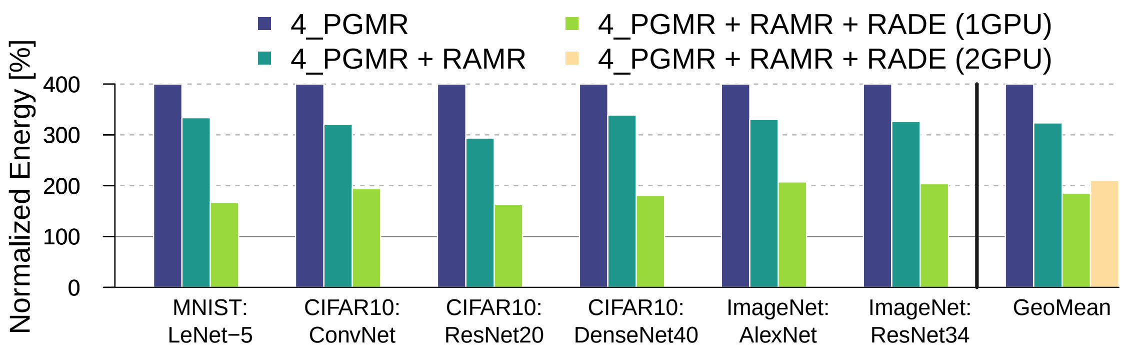
In this section, we assess the reliability results of PolygraphMR without including any side effects of performance optimization. We evaluate two configurations of PolygraphMR, one with four networks (*4_PGMR*) and another with six networks(*6_PGMR*) to show the scalability.

Figure 2.9 shows the evaluation results for all six benchmarks. We compare the normalized FP rate of the PolygraphMR system with other designs. The y-axis displays FP rate which is normalized to the FP rate of the corresponding baseline CNN. All design points also have a normalized TP of 100%. One can see that on average, 4_*PGMR* can reduce the FP rate of the baseline CNN by 40.8%. This value is also 16.6% lower compared to MR configuration with the same number of CNNs. Table 2.3 summarizes the preprocessors selected for each benchmark in 4_*PGMR* configuration.

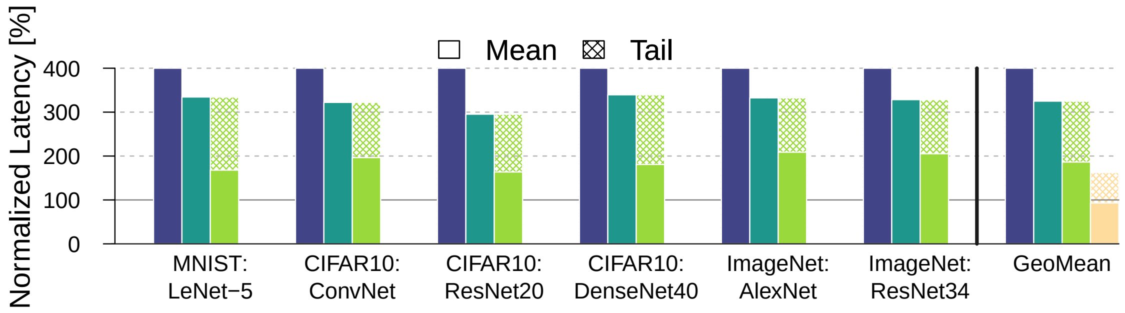
Figure 2.9 also shows that PolygraphMR designs are orthogonal from the baseline accuracy. For example in CIFAR10, three benchmarks with different accuracy and complexity levels are evaluated. We can see that 4_*PGMR* is successfully reducing FPs in all three benchmarks. The same observation can be made on the ImageNet based benchmarks. In conclusion, PolygraphMR can work in harmony with future more accurate networks to enhance their reliability.

Another interesting point is the selection of FlipX preprocessor in the benchmarks that are evaluated on ImageNet. During the training of AlexNet and ResNet34, samples are randomly flipped and fed to the network to introduce robustness towards the rotation of objects in the image. Hence, there was not any gain expectation by addition of this preprocessor, since we believed that the intended diversity was already introduced to the system during the training. But as the result of final configuration depicts, we still get benefits by deploying FlipX. This shows the sensitivity of the network to minor changes in the input and the explanation for this phenomenon is discussed in Section 2.3.

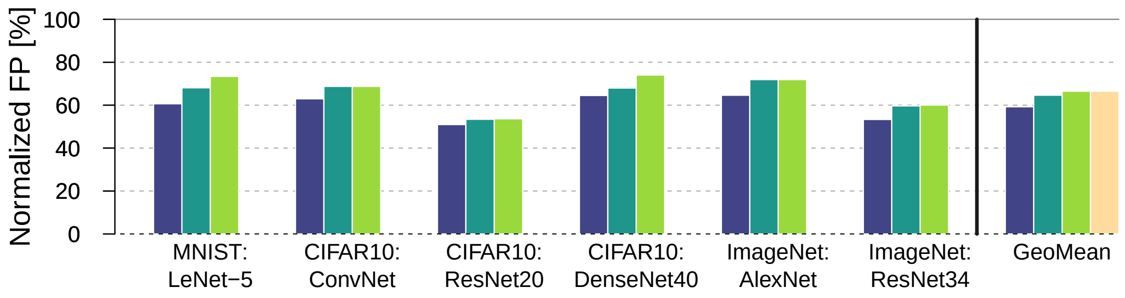
Figure 2.9 also presents FP rates for 6_*PGMR* configurations. On average, 6_*PGMR* designs can detect 48.2% of baseline FPs, which is 12.5% improvement over 4_*PGMR*. Among all benchmarks, ConvNet on CIFAR10 and AlexNet on ImageNet benefit the most from increased diversity by, respectively, detecting 27.3% and 14% more FPs over their 4_*PGMR* counterparts. But, we observe that for the majority of benchmarks, 4_*PGMR* provides the sweet spot in reliability and cost trade-off. For the rest of chapter, 4_*PGMR* designs are used for further analysis.



(a) Energy change



(b) Latency change



(c) FP change

Figure 2.10: Energy, latency, and FP rate trend during cost-oriented optimization.

2.4.3 Energy/Latency Optimizations

To alleviate performance overhead of PolygraphMR, we deploy a two-step optimization procedure. First, we reduce the precision of each individual CNN in the 4_*PGMR* as discussed in Section 2.3.4. Next, we replace the decision engine of the new system with a resource-aware version.

In our evaluation, we assume that the proposed design is executed on an average hardware setup including only one GPU. This is the worst case scenario which requires PolygraphMR

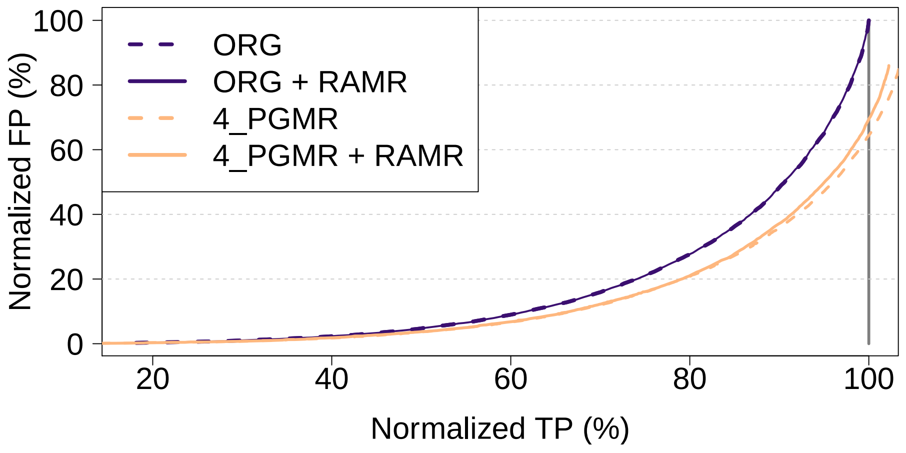


Figure 2.11: Pareto frontier comparison of precision reduced AlexNet on ImageNet dataset.

to execute individual networks sequentially. Therefore, the latency and energy overhead will grow linearly with the number of networks. If a more advanced hardware setup with multiple GPUs is available, such as the NVIDIA DRIVE AGX self-driving compute platform equipped with two TensorCore GPUs [85], latency overhead can be scaled correspondingly down. The latencies of the preprocessing and decision engine are also measured, but their overhead is negligible compared to CNN computation, e.g., 2.5% for AlexNet and 0.6% for ResNet34.

4_PGMR + RAMR: For each benchmark, we reduce the precision of both the baseline CNN as well as all CNNs in the 4_*PGMR*. As shown in Section 2.3.4, the PolygraphMR system is more resilient to precision reduction than a single CNN. Therefore, we can reduce the precision of individual CNNs in the 4_*PGMR* more aggressively. Figure 2.11 presents the precision reduction results on AlexNet benchmark. We compare the Pareto frontier of baseline and 4_*PGMR* in full precision and reduced precision settings. To get the Pareto frontier of *ORG*, it is coupled with a confidence threshold. In this figure, the x-axis represents the TP rate and y-axis stands for FP rate, both normalized to corresponding baseline rates. In this experiment, the precision of *ORG* is reduced to 17bits without any accuracy loss. As for 4_*PGMR*, precision is further decreased to 14bits again with no accuracy loss. But, as shown in Figure 2.11, the FP rate of 4_*PGMR* system is little changed with RAMR, and still

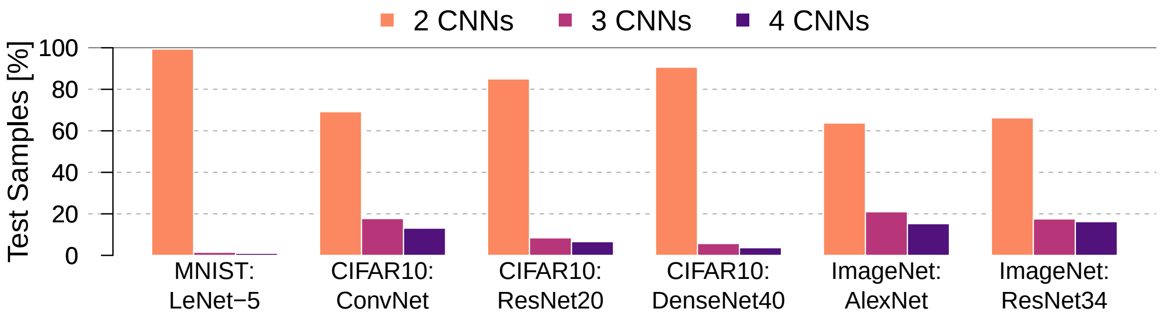


Figure 2.12: Distribution of number of networks activated in 4_*PGMR* system over test set of individual benchmarks.

offers 28.1% FP detection rate. We observe a similar behavior for other benchmarks and precision of each CNN is further-decreased by two to four bits.

The second bars in Figure 2.10 summarize the effect of RAMR on energy, latency, and normalized FP rate. On average, we can reduce the energy consumption and latency overhead by 76.5% and 75.0%, respectively. Whereas, FP rate is modestly increased by 5.4%.

4_PGMR + RAMR + RADE: Next, we deploy the resource-aware decision engine described in Section 2.3.6. Figure 2.12 presents distribution of the number of networks activated by RADE for individual benchmarks over test set. We observe that the majority of samples only require two CNNs to get the prediction. We only need to activate more networks for more demanding and complicated inputs. Figure 2.12 also shows that benchmarks with a higher accuracy baseline, less frequently require the activation of extra networks.

The third bars in Figure 2.10 present the energy and latency reductions as well as FP rate changes. By applying both performance optimization, we reduce average energy overhead to 185.5% and normalized average latency to 186.3%. On the other hand, normalized FP rate is increased by 7.2%.

Although RADE is effective on reducing the average latency, the tail latency is left unaffected. This might be problematic on real-time applications that have a latency budget per input. As an important example, self-driving car systems are required to have a tail latency threshold of 100*ms* [71]. But, considering the overall low latency of baseline networks, it is still possible to satisfy the latency requirements while providing predictions with higher reliability. For example, ResNet34 as the most demanding network in our benchmark suite, requires less than 17*ms* on TITAN X (Pascal) to do a forward pass on single input.

Optimized 4_*PGMR* is also evaluated on a hardware with two GPUs similar to NVIDIA DRIVE AGX platform. In this scenario, CNNs are activated in a batch of two over available GPUs. As shown in Figure 2.12, the majority of inputs only require two networks to get the expected reliability. Therefore as shown in Figure 2.10b, the average latency can be reduced to baseline levels.

Discussion: To give perspective on the significance of these results, we consider an example of reliability improvement achieved by using networks with higher accuracy and complexity. We compare accuracy and cost of ResNet20 and DenseNet40 on the CIFAR10. Based on Table 2.2, DenseNet40 reduces FP rate of ResNet20 by 18%. Whereas, MAC operations are increased from 41 MFLOPs to 267 MFLOPs, more than $6\times$ extra computation. This shows the inevitable trade-off between reliability and cost. In comparison, 4_*PGMR* on ResNet20 reduces FPs by 49.0% with $4\times$ cost, or by 46.3% with $1.6\times$ cost after optimizations. We observe that in this case, 4_*PGMR* is more cost effective to get higher reliability. It is good to mention that we are not suggesting to use PolygraphMR as a replacement for more accurate networks. Instead, our preference is to deploy PolygraphMR on top of them to achieve even a higher reliability as discussed in Section 2.4.2.

2.4.4 Preprocessing and Decision Engine

To show the impact of preprocessors and decision engine used in PolygraphMR, two separate experiments are run on CIFAR10 dataset with ConvNet. First, 6_*PGMR* is compared to a modified version of traditional MR, which deploys the smart decision policy proposed in Section 2.3.5. We call this new system 6_*MR_DE*. By analyzing the changes in FP

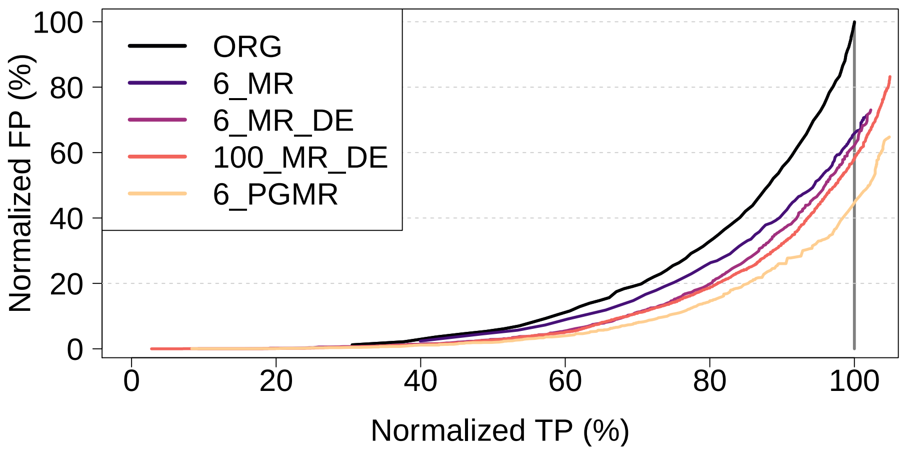
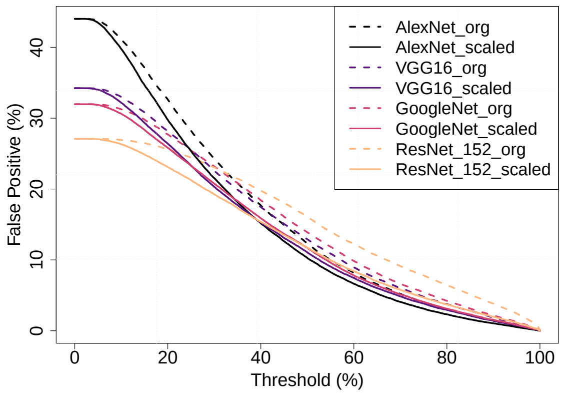


Figure 2.13: System configuration optimality analysis.

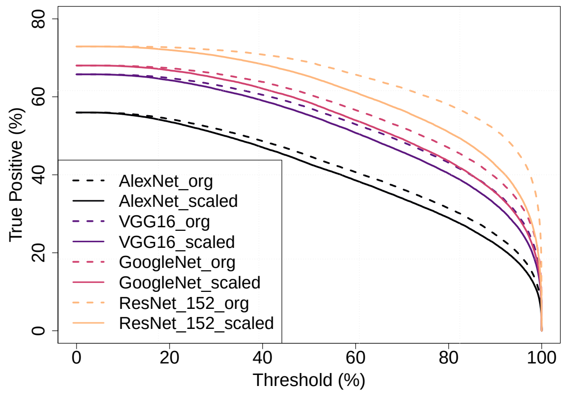
rates by moving from 6_*MR* to 6_*MR*_*DE* and from 6_*MR*_*DE* to 6_*PGMR*, we can observe the effect of decision engine and preprocessing, individually. Next, the 6_*PGMR* system is challenged by an extension of the modified MR used in the first experiment, which is assembled by training 100 copies of the baseline ConvNet.

Figure 2.13 compares the Pareto frontier of different designs. To get the Pareto frontier of baseline CNN and 6_*MR*, they are coupled with a confidence threshold(Thr_Conf). It can be seen that both modified MRs are outperformed by PolygraphMR system. By comparing 6_*PGMR* with 6_*MR*_*DE*, we can see the extra gain of 18.5% in robustness introduced by preprocessing. We can also observe the gains from decision engine by comparing 6_*MR*_*DE* and 6_*MR*. The former provides 4.1% more normalized FP detection which is the result of using a smarter decision engine instead of just taking majority vote.

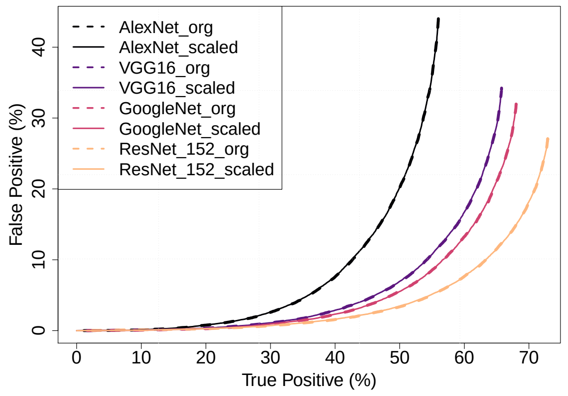
As for comparison of 6_*PGMR* and 100_*MR*_*DE*, even though the number of networks used in modified MR is 16 times more compared to 6_*PGMR*, the diversity introduced by using only 5 preprocessors is still higher. As a result, the reduction that 6_*PGMR* offers in the normalized FP rate is still 15.3% more than what 100_*MR*_*DE* can do.



(a) False Positive



(b) True Positive



(c) Pareto Frontier of Confidence Threshold

Figure 2.14: Temperature scaling results

2.4.5 Comparison with Network Calibration

Finally, we analyze network calibration as a solution to unreliability of confidence metric [130, 82, 90, 83]. As an experiment and to see the effects of calibration on the high confidence wrong answers, we implement the temperature scaling method proposed in the recent works [32]. Temperature scaling uses a scalar parameter to scale the output of softmax layer. The desired value of scaling for each benchmark is derived by solving an optimization problem [32].

The temperature scaling method is implemented and tested on 4 benchmarks over ImageNet dataset. Figure 2.14 shows the result prior to and after the temperature scaling. The dashed lines are for the original CNN and the solid lines report the results for scaled

CNNs. The effect of confidence as a reliability metric is studied through Figure 2.14. Both Figure 2.14a and 2.14b show the effect of temperature scaling on FP or TP rate, with respect to the selected confidence threshold. If we compare FP and TP rate of the scaled and original network for each threshold value, we can see a reduction in both parameters. This gives the intuition that confidence of the undesired overconfident predictions is decreased which would make the confidence metric an appropriate reliability metric. But based on Figure 2.14c, we can see that Pareto frontier of TPs and FPs is unchanged after scaling. In other words, since the scaling is done by using the same parameter for all confidence values regardless of the correctness of answers, the final Pareto chart of the TPs and FPs is kept untouched. The only change is that for accessing a specific set of TP and FP, a lower confidence threshold is required for the scaled network. Hence, the initial reliability problem of confidence is still in place.

2.5 Related Work

We focus on different areas which target the reliability or security aspects of CNNs, or target to better understand the irrational behavior of CNNs.

Model uncertainty and Bayesian NNs are a closely related area of research [30, 49, 51, 58]. The idea is to add uncertainty to the predictions of NN models, and to be able to understand when the network is not confident with the generated results. Although the concept of model uncertainty is highly promising for reliable CNN inferences, but current solutions mainly target regression tasks [49, 30], or add a very high execution overhead, e.g., $10\times$ to $100\times$ in solutions based on the ensembles [58] or dropout sampling [30, 51]. However in PolygraphMR, we target improving classification reliability in CNNs, while considering the performance overhead implications of the proposed solution.

Researchers are also focusing on corner-case behaviors of the CNNs. They try to come up with systematic approaches and testing tools to detect erroneous behaviors [113, 88]. A number of works are also attempting to detect the anomalies in CNN applications. Their goal is to make CNNs more resilient against out-of-distribution examples that have not been seen before [36, 70, 25, 37, 126].

Security researchers are also focusing on the robustness of CNNs. They seek to find new techniques of generating adversarial inputs, to fool the network and induce their desired results. They add perturbations to the inputs, which in some cases is not even visible to the naked eye, leading to mispredictions [86, 80, 112]]. In contrast, they try to make the network robust against the state of the art adversarial generation methods [31, 87, 39].

A number of works are also focusing on the understandability and interpretability of CNNs in order to add more transparency to these algorithms. Samek et al. [99] try to explain the predictions of CNNs by measuring the sensitivity of the outputs to the individual input variables. Turner [115] proposes a general model for explaining the output of the classifiers. Zeiler et al. [131] propose a visualization technique to get an insight into the functionality of the intermediate features and operations of the classifier.

There are also numerous works on the reliability of CNNs against the transient faults and soft errors [68, 9, 89]. These works study fault injection in the CNN executions and analyze the response of the network to the faults. The solutions proposed for these reliability issues include modular redundancy in the level of application, instruction, and transistors. Our work however, focuses on the internal reliability problems of the DNNs which are also very vital considering that unlike transient faults, DNN prediction faults are quite common and happen frequently regardless of the hardware system that they are executed upon.

To the best of our knowledge, reliability problem of deep learning algorithms is a new topic for research in this area. Considering the increasing utilization of these algorithms in real world and vital applications such as autonomous vehicles, assuring their reliability needs to be well studied.

2.6 Conclusion

CNNs are extensively utilized in mission critical applications such as autonomous vehicles. These applications require high levels of robustness since any error can cause irreparable damages. In this work, we demonstrated that current CNNs are failing to meet the reliability requirements of such applications by exposing a significant number of high confident mispredictions. We find that recent solutions that utilize confidence values as a metric of reliability are faulty and can lead to a significant number of false positives. We propose a new system called PolygraphMR composed of a number of CNNs as building blocks. Each network is accompanied by a specific preprocessor to provide behavior diversity among the CNNs and detect the unreliable wrong answers based on the contradictions observed due to the behavior variations. An energy efficient version of the system is also proposed that deploys precision reduction and staged activation to reduce the multiplicative costs of MR. As a result, our solution is capable to provide an average of 33.5% reduction in false positives of the original CNN, while requiring less than $2\times$ performance overhead.

CHAPTER III

SoftFusion: A Low-Cost Approach to Enhance Reliability of Object Detection Applications [*]

3.1 Introduction

Deep Neural Networks (DNNs) are now being deployed in mission-critical applications such as object detection in Autonomous Vehicles (AVs) [129, 12] and computational medicine [74, 94]. Ensuring the reliability, security, and safety of predictions made by DNNs plays an essential role in deploying these solutions in critical domains. For example, there have been numerous reported failures in object detection models deployed in AVs, which further underscores the requirement of reliability assurance in these applications [28].

A common solution to develop more accurate object detection models used in AVs is sensor fusion, which is based on the traditional modular redundancy methods. Sensor fusion combines the strengths of individual physical sensors for more reliable perception [101], and provides resilience against sensor failures [81]. Figure 3.1 depicts an example of fusion which deploys multiple physical sensors like cameras, LiDAR, and radar. Each sensor has its own positives and negatives. For example, cameras are capable of capturing high-resolution RGB data, but in contrast, they are not as powerful to capture depth information. On the other hand, LiDAR can effectively receive depth information from long range, but it can get

[*]Published in the 40th IEEE International Conference on Computer Design (ICCD'22) [62]

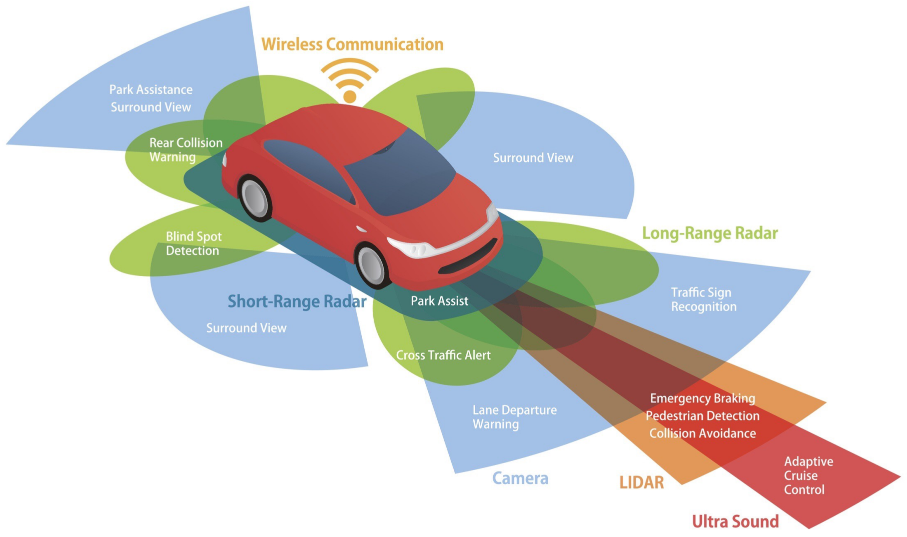


Figure 3.1: Multi-sensor fusion setup in modern self driving cars [95]

degrade in bad weather conditions, in which radar comes into play. This inter-sensor fusion scheme helps by gathering input data from individual sensors and processing them through individual models for more robust object detection. Multiplicative cost is the primary negative of this approach that includes additional physical sensors as well as the computing and memory requirements for the DNNs associated with each sensor.

In this paper, we present intra-sensor fusion, referred to as SoftFusion. A complementary approach to traditional sensor fusion is to examine intra-sensor fusion wherein a diversity of inputs is synthesized at runtime, e.g., image/video filtering, to feed the original object detector. Intelligent combination of the output data from individual inputs can identify unreliable answers as well as potential adversarial attacks. Intra-sensor fusion can be applied to individual physical sensors available in an AV, and can deploy any off-the-shelf object detector. This could increase the reliability of predictions from individual physical sensors, leading to overall more robust object detection systems, or reduce the need for traditional fusion, lowering the cost of providing a target level of robustness. Moreover, as the newer and more accurate detection architectures are designed, they could be swapped in for further robustness.

The goal of SoftFusion is to achieve higher detection accuracy and reliability, with the least amount of overhead introduced in latency and cost. With careful design of synthesized inputs, it is possible to run inference in a batched fashion on GPU systems and eliminate the multiplicative cost of multiple inferences as in traditional modular redundancy solutions. Furthermore, with intelligently processing the data from individual inputs, we can achieve more precise bounding box proposals with fewer objects left undetected.

This paper makes the following contributions:

- We demonstrate that with the current object detection models, there are still a large number of objects that are left undetected, which are clear safety concerns.
- We propose SoftFusion based on the concept of intra-sensor fusion, which synthesizes a group of images from the original input at runtime. Moreover, by effectively combining the bounding boxes from individual predictions, it composes more precise and reliable outputs.
- With the systematic selection of online augmentation techniques, we can minimize the runtime overhead of the system by deploying the underutilized hardware resources, prohibiting the multiplicative costs of multi-sensors.
- A proposal engine is introduced to combine the generated bounding boxes based on the confidence levels of each individual prediction to increase the precision and recall of the detector.
- SoftFusion is evaluated across a wide range of benchmarks with different baseline accuracy levels, and is shown to provide 3.45% gains in the mAP with less than 23% overhead on baseline latency. For context comparison, prior research on hard sensor fusion of camera and LiDAR achieves on average 4.02% gain in mAP in similar object detection tasks while leading to an approximately up to $2\times$ computation overhead [69, 119].

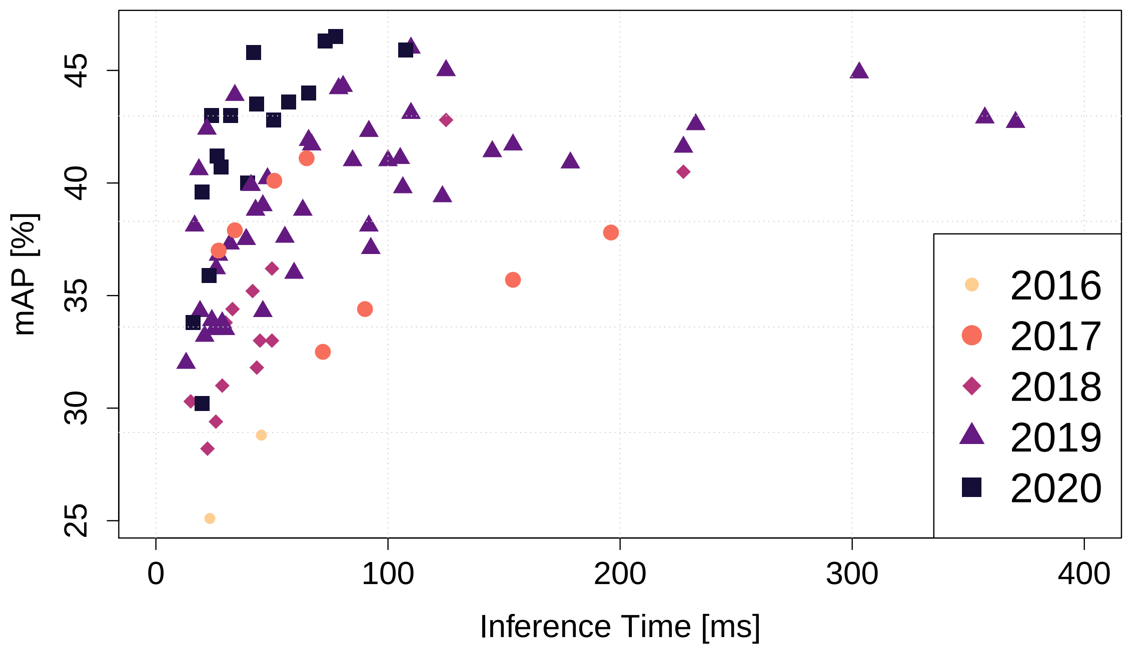


Figure 3.2: A performance overview of object detection models proposed for COCO dataset from 2016 to 2020

3.2 Motivation

3.2.1 Design Trends of Object Detectors

To get familiarized with the design space of object detectors and see what has been the focus on creating novel architectures, we analyzed the recent object detectors emerged in the recent years. We gathered the reported data for their inference time and accuracy values on COCO test-dev 2017 [73] from multiple sources [11, 2, 120].

Figure 3.2 represents the summary of design points for 87 different object detectors from 2016 to 2020. The x-axis represents the inference time for a single image, and the y-axis shows the mean Average Precision (mAP). By analyzing this graph, two trends can be observed. The first trend is the push for more accurate network architectures, which is absolutely expected. The second trend, on the other hand, is more interesting. We can see a higher emphasis on the latency of models. In other words, we can find a design point in 2020 which has a similar accuracy to the earlier models, but with a lower latency. In fact, the absolute accuracy improvement over models from 2019 is marginal in 2020. This pattern could be reasoned by considering that object detection models are mainly deployed in time-sensitive domains such as self-driving cars, which require making decisions in a timely manner.

This sets our first goal in designing SoftFusion which should induce the least amount of latency overhead over the baseline.

3.2.2 Reliability Analysis of Available Object Detectors

In another experiment, we analyze the reliability of predictions made by the object detection models. We choose 5 different models from Figure 3.2 with mAP values varying from 31.4% to 47.2% in order to cover the full spectrum of the accuracy space. Next, we evaluate these models over the 2017 validation set of the COCO dataset.

We analyzed individual objects in each image to see if there was a matching Bounding Box (BB) generated for it by the detector. We calculated the Intersection over the Union (IoU) of the ground truth BB with respect to all bounding boxes proposed by the detector for that specific image. An acceptance threshold of 75% was used for IoU result to mark an object as detected (the IoU threshold used in literature varies from 50% to 95%, so we used 75% as a middle ground). As a result, we could have three possible outcomes for each ground truth object: First, no matching BB is discovered (Wrong BB). Second, a matching BB is detected, but the predicted label for the output BB is different from ground truth (correct BB, wrong label). In the final case, there is a matching BB and label in the detector outputs.

Figure 3.3 presents the distribution of all objects in the validation set across the three possible output categories. The key observation is the large share of the objects that are missed by each detector. On average, 44.4% of the objects are missed across the 5 evaluated models. Therefore, we set our second goal to increase the reliability of detections in SoftFusion by reducing the number of undetected objects. Our primary goal is to generate

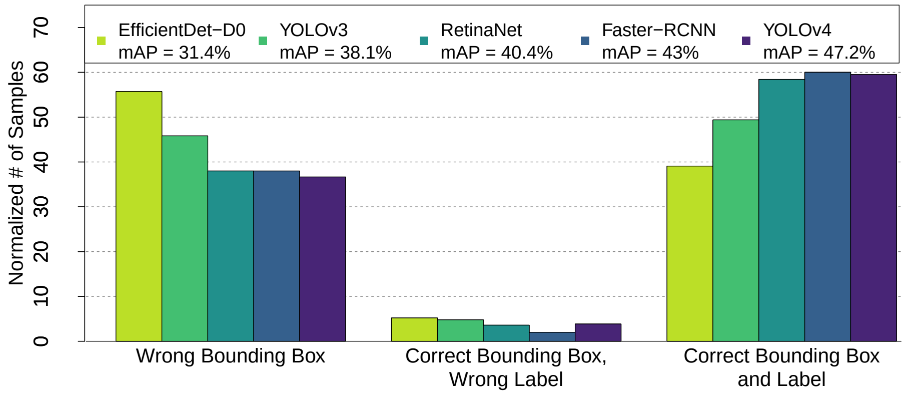


Figure 3.3: Evaluation of the reliability of prediction made by 5 different benchmarks on COCO dataset with diverse accuracy levels

correct BB and label for the originally missed objects. However, even by having a correct BB detected for an object and incorrectly classifying its label, we would have a more robust design compared to completely missing it.

3.3 Proposed Work

3.3.1 Overall Design

We propose SoftFusion based on the concept of intra-sensor fusion. Figure 3.4 represents the general inference pipeline of SoftFusion. At step 1, the input image is passed through a hierarchical augmenter to generate different variants of the input image. Next, the baseline detector is deployed to run a batched inference on the set of augmented inputs. In the next step, the proposed bounding boxes for each augmented input is post-processed to invert the effect of respective augmentation, and map the location of generated proposals to the original input . At the last step, all bounding boxes are gathered and passed through the proposal engine for analysis. During this step, overlapping bounding boxes are merged according to their confidence values into a more accurate proposal. Finally, each proposed bounding box is examined based on the number of received votes in order to decide on its reliability.

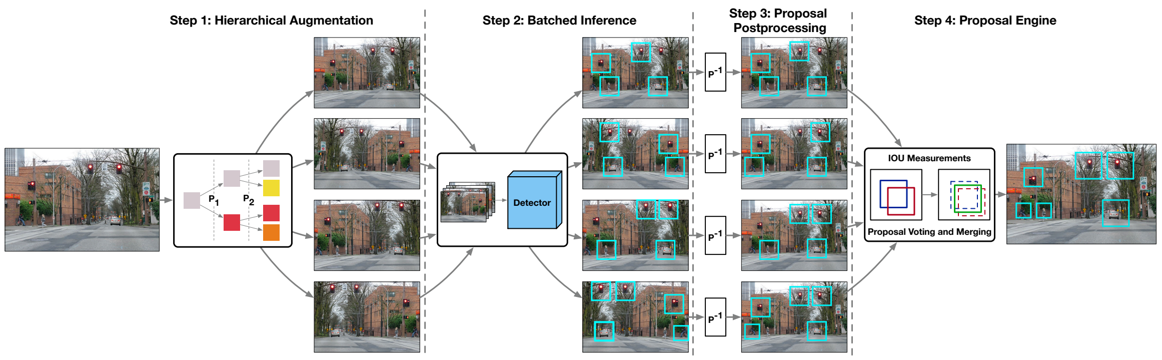

Step 1: Hierarchical Augmentation
P_1
P_2
Step 2: Batched Inference
Detector
Step 3: Proposal Postprocessing
P^{-1}
Step 4: Proposal Engine
IOU Measurements
Proposal Voting and Merging

Figure 3.4: General design of SoftFusion system: In step 1 we synthesize a diverse set of inputs for object detection. In step 2, we run a batched inference on the augmented list of inputs. During step 3, the impact of individual augmentations on the proposed bounding boxes is reversed. In step 4, object proposals from individual images are fused for the final set of bounding boxes.

3.3.2 Step 1: Hierarchical Augmentation

The goal of augmentation is to generate a diverse set of inputs for intra-fusion. This would provide the opportunity for the detector to analyze the same input from different synthetic viewpoints. To achieve this goal, we are deploying a hierarchical scheme of augmentation. Each level of hierarchy would double the number of images in the augmentation set. As a result, if we have an N-level hierarchy, we would end up with 2^N samples of augmented images.

We start by initializing the augmentation set with the original input. Next, at each level of the hierarchy, a different image processing technique is applied to individual samples of the current set, and the result is appended back to the set.

The hierarchical scheme provides us the opportunity to generate the desired number of augmented samples with the least number of image processing techniques. This would significantly reduce the search space for selecting the image processors. For example, we would need to search for 7 different image processors to generate an augmented set of size 8 in the exhaustive approach. However with hierarchical scheme, we would only need to search for 3.

Figure 3.5 compares the mean average precision(mAP) gains for hierarchical and traditional augmentation methods, both using total number of 4 images in the augmentation

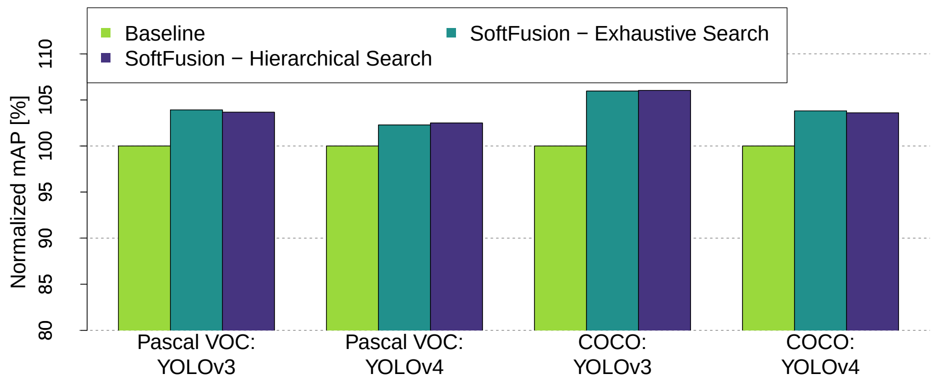


Figure 3.5: Comparison of hierarchical and exhaustive augmentation over YOLOv3 and v4 benchmarks on Pascal VOC and COCO datasets

list, on YOLO family of object detectors across two datasets. The y-axis presents the mAP values normalized with respect to the baseline values. We observe that the differences in the mAP gains are similar on both augmentation methods.

Since one of the goals in SoftFusion is to limit the overhead of latency, the selection of image processing techniques is limited to linear transformations that do not alter the original distribution of the input images. This ensures achieving similar accuracy levels on the inference of augmented images, without requiring any retraining or fine-tuning of the detector. Image processors such as flipping the image, scaling, and translation, all meet the mentioned criteria. On the other hand, non-linear techniques, such as histogram equalization, alter different features of the image like brightness intensity, and are prone to significant accuracy drops with the original detector.

3.3.3 Step 2: Batched Inference

Due to the careful selection of augmentation techniques, SoftFusion can use the same detector to do inference on all samples. This gives us the opportunity to deploy batch-processing of the inputs. As a result, the overhead of the multi-inference approach is significantly reduced compared to running inference on individual samples sequentially.

To further analyze the effect of batched inference on latency, we run an experiment on the YOLOv4 [11] using the validation set of the COCO [73], in which the number of inputs

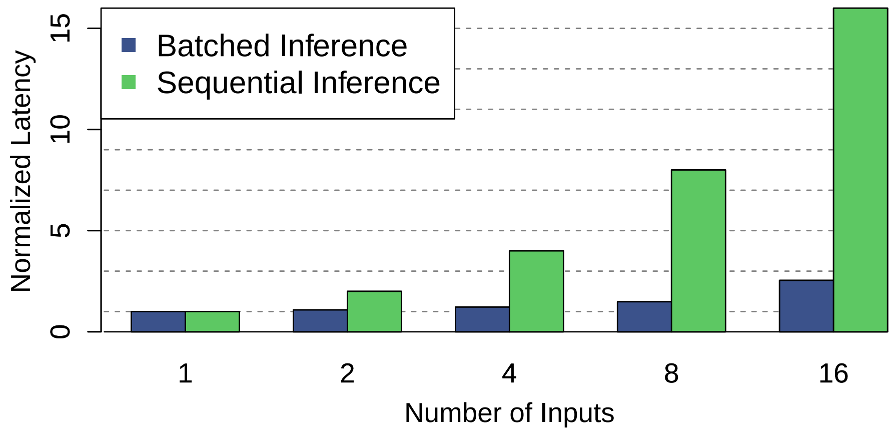


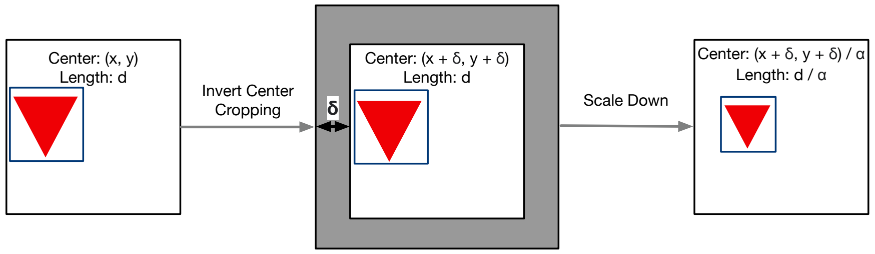
Figure 3.6: Normalized latency comparison of batched inference and sequential inference on YOLOv4 and COCO dataset.

for inference is changed from 1 to 16. We use the Ultralytics [116] framework and NVIDIA Tesla V100 [1] to run the inference. Figure 3.6 presents the average latency of sequential and batched inference, which is normalized with respect to single input inference. Based on Figure 3.6, batched inference causes very low overhead over single image inference with varying size of inputs. To be more specific, the overhead of batched inference for input size of 4 is 22%, which is also the same number that majority of the benchmarks use in Section 3.4.

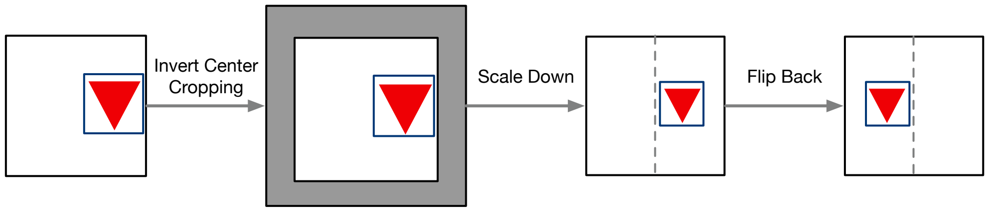
3.3.4 Step 3: Proposal Post-processing

Since the augmentations used in Section 3.3.2 would affect the location of objects in the original image, we need to post-process the proposed bounding boxes to revert the impact of augmentation before passing them to the proposal engine.

Figure 3.7a presents the required post-processing for scaled up bounding boxes. First, we need to reverse the center cropping by shifting the center of the bounding box. Next, each bounding box is scaled down with respect to the scaling factor. Figure 3.7b also shows the required post-processing for the samples that are both scaled-up and flipped. First, the bounding boxes are scaled down, and then, they are mirrored in order to map them to the



(a) Post-processing scaled up bounding boxes



(b) Post-processing flipped and scaled up bounding boxes

Figure 3.7: Post-processing the proposed bounding boxes by inference engine

original image.

3.3.5 Step 4: Proposal Engine

At the final step, proposal engine gathers all bounding boxes (BBs) from individual inferences to generate the final set of predictions. Each input BB includes three different features: The first one is the location with respect to the image. The second feature is the confidence of the detector about the existence of an object in the generated BB. Finally, the last feature is the label of the object.

Algorithm 1 describes the proposal function that is called for each set of bounding boxes that have the same label. First, the BBs are sorted based on their confidence levels (line 3). Next, starting from the most confident prediction, each prediction is compared with the remaining ones with respect to their Intersection over Union (IoU) (line 13). If the IoU is bigger than the provided threshold (IOU_thr), the two BBs are overlapping with each other and can be merged (line 17). The merging function receives the location of BBs and their confidence levels as input (line 18), and merges them together by taking a weighted average according to their confidence values. The confidence value for the merged BB is

Algorithm 1: Proposal engine called for each class

```
Input: BBs, BBs_len, IoU_thr, Vote_thr
Output: BBs_out
Sort(BBs)
visited = $[false] \times BBs\_len$
for $i = 0$ to $BBs\_len$ do
   if $visited_i$ then
      continue
   end if
   $visited_i = True$
   $cur\_votes = 1$
   $cur\_BB = BBs_i.BB$
   $cur\_conf = BBs_i.conf$
   for $j = i$ to $BBs\_len$ do
      if $visited_j$ then
         continue
      else
         if $IoU(cur\_BB, BBs_j.BB) > IOU\_thr$ then
            $cur\_BB = Merge(cur\_BB, BBs_j.BB,$
                  $cur\_conf \times cur\_votes, BBs_j.conf)$
            $cur\_votes{+}{+}$
            $visited_j = True$
         end if
      end if
      if $cur\_votes > Vote\_thr$ then
         $BBs\_out.append([cur\_BB, cur\_conf])$
      end if
   end for
end for
```

further emphasized with respect to the number of votes that it has received so far. Finally, the merged BB is appended to the output list if it has received sufficient votes (line 24).

The proposal engine results in more accurate and reliable predictions in two different ways:

- It makes the originally detected BBs more accurate, thereby improving the overall precision of predictions.
- It improves the reliability of predictions by detecting the originally missed objects, thereby increasing the overall recall of objects by the detector.

3.3.6 SoftFusion System Design

To configure SoftFusion for a provided detector, there are two sets of decisions to be made. The first decision is the number of levels and the choice of image processors to be used in hierarchical augmentation. The second design point is the threshold values to be selected for the proposal engine. Configuring both of these settings is done offline by evaluating the accuracy of different setups over the validation set.

We tried hierarchical augmentation setups with different number of levels and choice of linear image processing solutions. Based on our preliminary experiments, we saw that a hierarchy of up to 2 levels gives us the sweet spot in the accuracy gains and speed. As for the augmentation methods, flipping horizontally was giving us the maximum accuracy improvements. Therefore, the first level of hierarchy is fixed with horizontal flipping. The second level is also using either scale up plus center cropping, or translation in both x and y dimensions. The selection of the second image processor and the corresponding transformation factor, e.g., how much the input image should be scaled up or translated, is decided based on the profiling experiment. We further discuss the decision choices for individual detectors in Section 3.4.

After the augmentation hierarchy is fixed, we profile the threshold values of IoU_thr and $Vote\_thr$ offline for the maximum accuracy gains. Based on our experiments over different detectors and datasets, an IOU_thr in the range of 50% to 70%, and a $Vote\_thr$ of $\frac{N}{2}$ (N being the total number of images of the augmentation set), was giving us the desired accuracy improvements. However, these threshold values could be configured more/less aggressively depending on the tolerance of false positives in the corresponding application.

3.4 Evaluation

We evaluate SoftFusion in multiple aspects. First, we evaluate its impact on overall accuracy of detections on two different datasets. Next, we analyze the performance overhead

Dataset	Detector	Level 1 Preprocessor	Level 2 Preprocessor
Pascal VOC	YOLOv3	Horizantal Flip	Scale Up
	YOLOv4	Horizantal Flip	Scale Up
COCO	EfficientDet-D0	Horizantal Flip	Translation
	YOLOv3	Horizantal Flip	Scale Up
	RetinaNet	Horizantal Flip	Translation
	Faster-RCNN	Horizantal Flip	Translation
	YOLOv4	Horizantal Flip	Scale Up

Table 3.1: Benchmark set used to evaluate SoftFusion.

Detector	Backbone	Input Size	mAP
YOLOv3	Darknet53	416 x 416	78.83
SoftFusion YOLOv3 - L1	Darknet53	416 x 416	**81.34**
SoftFusion YOLOv3 - L2	Darknet53	416 x 416	**81.72**
YOLOv4	CSPNet	416 x 416	80.17
SoftFusion YOLOv4 - L1	CSPNet	416 x 416	**81.6**
SoftFusion YOLOv4 - L2	CSPNet	416 x 416	**82.17**

Table 3.2: Average precision results on Pascal VOC dataset.

of the proposed solution. Then, we analyze the impact of individual components of the SoftFusion in terms of accuracy gain. And finally, we evaluate the overall impact of SoftFusion on the robustness of the predictions.

3.4.1 Methodology

Benchmarks: We use two different datasets on object detection. The first one is the Pascal VOC [27] which includes 20 different classes of objects. We use Pascal VOC 2007 and 2012 for training purposes, and the test set of Pascal VOC 2007 to evaluate the accuracy. The 2007 validation set is used to configure the SoftFusion system. The second dataset is COCO [73] which is composed of 80 different object classes. The 2014 training set is used to train the models, and the 2017 validation set is used for evaluation purposes. We also use 20% of the validation set to configure the SoftFusion system on COCO benchmarks. Table 3.1 summarizes the 7 different benchmarks and the proposed SoftFusion configuration.

Frameworks: Darknet [5] framework is used to train and evaluate the YOLO bench-

Detector	**Backbone**	**Input Size**	**FPS (V100)**	**AP**	**$AP_{0.5}$**	**$AP_{0.75}$**	**AR_1**	**AR_{10}**	**AR_{100}**
EfficientDet-D0	EfficientNet-B0 + BiFPN	512 x 512	**62.5**	31.4	46.1	34.3	26.8	38.2	40.3
SoftFusion EfficientDet-B0 - L1	EfficientNet-B0 + BiFPN	512 x 512	**54.4**	**32.1**	47.5	34.6	**27.6**	39.3	41.5
SoftFusion EfficientDet-B0 - L2	EfficientNet-B0 + BiFPN	512 x 512	**51.8**	32.1	**47.6**	**34.7**	**27.7**	39.5	41.8
YOLOv3	Darknet53	416 x 416	**54**	38.1	67.8	39.2	30.5	47.9	51
SoftFusion YOLOv3 - L1	Darknet53	416 x 416	**50.1**	**40.1**	69.4	41.6	**31.6**	49.8	53.3
SoftFusion YOLOv3 - L2	Darknet53	416 x 416	**42.5**	**40.4**	69.9	42	**31.7**	50.1	54
RetinaNet	ResNet-101 + FPN	800 x 800	19.6	40.4	60.2	43.2	33.6	53.2	56.3
SoftFusion RetinaNet - L1	ResNet-101 + FPN	800 x 800	12.55	**41**	60.4	44	**34.2**	54.5	58.3
SoftFusion RetinaNet - L2	ResNet-101 + FPN	800 x 800	5.9	41	60.2	**44.2**	34.1	**55.1**	**59.7**
Faster-RCNN	ResNeXt-101 + FPN	800 x 800	9	43	63.7	46.9	34.1	53	53.3
SoftFusion Faster-RCNN - L1	ResNeXt-101 + FPN	800 x 800	6.2	**43.9**	64.3	48.2	**34.6**	54.5	57.1
SoftFusion Faster-RCNN - L2	ResNeXt-101 + FPN	800 x 800	2.9	**44.1**	64.3	48.3	**35**	55.3	58.5
YOLOv4	CSPNet	416 x 416	**96**	47.2	71.3	51.1	35.7	56.6	59.8
SoftFusion YOLOv4 - L1	CSPNet	416 x 416	**82.1**	**48.5**	72	52.6	**36.5**	0.5	61.6
SoftFusion YOLOv4 - L2	CSPNet	416 x 416	**78.6**	**48.9**	72.3	53.7	36.5	**58.3**	62.7

Table 3.3: Average precision, recall, and speed comparison results on COCO val-2017 dataset.

marks. Detectron2 [127] and Pytorch frameworks are used for the pre-trained models of EfficientDet-D0, RetinaNet and Faster-RCNN.

Performance Analysis: We use NVIDIA Tesla V100 GPU for speed evaluation. Ultralytics [116] framework is used for YOLO benchmarks, while Detectron2 and PyTorch are used for speed evaluation of RetinaNet and Faster-RCNN. EfficientDet-D0 is also evaluated on TensorRT [117].

3.4.2 Accuracy Results

First, we analyze the effect of SoftFusion on the accuracy of the benchmarks. Table 3.1 includes the detail for the image processing techniques used in each level of the hierarchy. We evaluate two versions of the SoftFusion designs. The first version only includes a single level of hierarchy for the augmentation (SoftFusion-L1). All benchmarks used horizontal flipping as the augmentation method in L1. The second version of SoftFusion deploys two levels of augmentation (SoftFusion-L2). Benchmarks use scaling up or translation in L2 depending on the offline profiling results.

Table 3.2 presents the mean average precision(mAP) gains on Pascal VOC benchmarks.

On average, we can achieve 2.48% normalized mAP improvement over baseline with single level of augmentation. In addition, by deploying a 2-level hierarchy, the mAP gains can further be improved to 3.08%.

Next, we analyze the accuracy gains on COCO benchmarks. Two groups of metrics are used to evaluate SoftFusion. The first group is average precision which includes three different configurations with respect to the IoU threshold used: $AP_{0.5}$ uses an IoU threshold of 50%, whereas $AP_{0.75}$ uses a threshold value of 75%. On the other hand, *AP* is the mean of average precision evaluated with different IoU threshold values ranging from 50% to 95% with a step of 5%. The second group of metrics analyzes the average recall of the detector, i.e., how many objects it successfully detects in each image. *AR* is measured according to the top-k detections for each image based on the confidence values of the proposals.

Table 3.3 presents the results for both average precision and recall. By applying SoftFusion-L1, we achieve an average of 3.16% normalized improvement on *AP* compared to the baseline. By moving to a higher level of hierarchy we can get an additional 0.53% of gains in *AP*. The highest gain is achieved in the YOLOv3 benchmark which achieves 6.3% improvement with SoftFusion-L2. Whereas RetinaNet receives the lowest amount of gain. By inspecting the *AR* values we can also see a higher recall factor with increasing hierarchy level, which leads to lower number of undetected objects.

3.4.3 Performance Analysis

Table 3.3 also presents the performance evaluation results for COCO benchmarks. For each benchmark, we compare the speed of baseline model on a single image inference with the speed of SoftFusion designs. Due to low overhead of the batch-processing of the inputs, we can still maintain the real-time performance of the baseline models with the SoftFusion applied. On average, SoftFusion-L1 reduces the FPS of the real-time applications by less than 12.9%. By increasing the hierarchy level to 2, we observe 22.6% reduction in the number of frames processed per second.

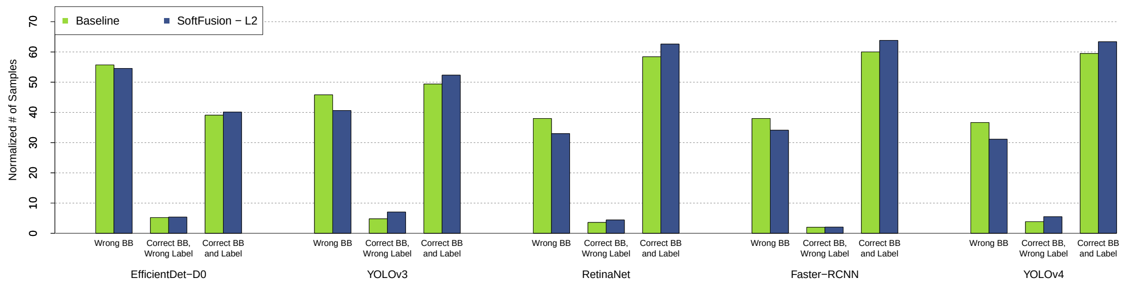


Figure 3.8: Comparing the reliability of predictions made by SoftFusion-L2 with the baseline detector across 5 different benchmarks on 2017 COCO validation set

3.4.4 Reliability Analysis of Object Detections

Finally, we compare the robustness of the predictions made by SoftFusion to the baseline by running a similar experiment as described in Section 3.2.2. Each object in individual images of the COCO validation set is cross checked with respect to baseline and SoftFusion-L2 proposed bounding boxes.

Figure 3.8 compares the distribution of all objects across the three possible categories of outputs for baseline and SoftFusion-L2. In every benchmark, we can observe the reduction in undetected objects, and increase in the other categories. SoftFusion-L2 results in 9.70% reduction in the number of undetected objects, and achieves 5.96% increase in the correct bounding box and label detections.

Overall, SoftFusion results in more robust predictions by reducing the number of undetected objects and either correctly predicting the labels and bounding boxes for them, or at least realizing their existence.

3.5 Related Works

A set of related works includes the robust fusion of available perception sensors which plays a vital role [57, 104, 18] in the functional safety of AVs. An important aspect of fusion is to deal with noisy inputs from individual sensors and generate a probabilistically reasonable estimate of the environment [100, 97]. [16, 10] discuss how combining vision camera systems with radars can improve the detection rates with decreased false detection in different climate conditions. Radars benefit from lateral resolution and cameras from feature richness, and the absence of sensing range of cameras is compensated by radars [48, 13, 123]. [18] evaluated a two-layer sensor fusion system consisting of 14 on-board sensors to achieve 93.7% detection rate. [69, 119] fuses LiDAR and camera data to improve the accuracy of object detection on KITTI dataset. While the hard fusion results in significant gains in precision of detectors, they are also susceptible to higher runtime overheads due to the need to process both image and LiDAR data.

Another set of prior works attempts to improve the accuracy of object detectors by fusing the intermediate semantics representations that are generated throughout the object detection models. Feature Pyramid Networks (FPNs) is an example for these line of techniques, which are applied to different baseline object detectors for accuracy gains [72, 76, 134]. Unlike multi-sensor counterparts, the FPN fusion techniques cause minimal runtime overhead, since the semantics representations used in fusion are already computed throughout the baseline object detection procedure. While FPNs are the most related to SoftFusion in terms of deploying intra-sensor fusion, we can further improve their average precision by deploying SoftFusion on top of them. This orthogonality effects can be observed in three of the evaluated benchmarks (EfficientDet-D0, RetinaNet, and Faster-RCNN) which are deploying FPNs and also benefit in accuracy by addition of SoftFusion.

Sensor fusion also provides the system redundancy, both in spatial domain (different available sensors) and also time domain(different instances from the same sensor in continuous frames). Space redundancy and time redundancy are recognized standard solutions to increase the reliability of computer systems. In time redundancy the computation or data transmission is repeated and the result is compared to the previous result [68, 43, 109]. A well-known technique to increase the reliability based on replication is n-modular redundancy [106, 8]. Dual modular redundancy (DMR) requires two replications of each element. It can only detect a mismatch by voting and is not able to help with recovery. To help the recovery issue, triple modular redundancy (TMR) uses three replications of each element to output the correct result when one of the replications fails [77, 107].

Another group of related works focuses on the deployment of image processing techniques with the goal of increased accuracy and reliability [53, 126, 61]. Ensemble methods for object detection is analyzed by recent work [15], which also proposes the test time augmentation. But, the solution requires retraining of the networks which eliminates the possibility of batched inference.

3.6 Conclusion

DNNs are getting deployed in mission critical applications such as object detection in self-driving cars, or computational health. However, an important question is how reliable available object detection models are. A widely deployed solution in the AV domain, is sensor fusion, which introduces redundancy into the system by incorporating different physical sensors, and results in resiliency against individual sensor failures. In this paper, we propose SoftFusion which is inspired by the sensor fusion. SoftFusion introduces the concept of intra-sensor fusion, in which the input for individual sensors could be processed to generate a diverse set of augmented samples with the goal of adding redundancy per sensor. In SoftFusion, a hierarchical augmentation unit is proposed to produce the desired diversity levels for the detector without any implication of retraining required. As a result, the augmented set of input could be processed in a batched-inference fashion to minimize the overhead of the multiple inferences. Next, SoftFusion deploys the proposal engine to effectively combine the bounding boxes proposed for individual samples by taking into account the confidence level and the number of votes that each prediction receives. With the combination of augmentation module and proposal engine, SoftFusion can lead to a more robust object detection in a cost-effective manner. The proposed solution is evaluated on 7 benchmarks and 2 datasets, which shows that SoftFusion can increase the average precision of the predictions by 3.45% with less than 23% overhead on the baseline latency.

CHAPTER IV

Efficient Sparsely Activated Transformers [*] [†]

4.1 Introduction

Attention-based deep neural networks (DNNs) such as Transformer [118] and BERT [24] have been shown to exhibit state-of-the-art performance across a variety of machine learning domains, including natural language processing [124] and computer vision [26]. Due to their size and complexity, they are expensive to train and deploy, especially on resource-constrained hardware. In particular, attention layers, which form the building blocks of such networks, account for the majority of network runtime. Figure 4.1 illustrates this using the Transformer-XL network [22]; here, we show the proportion of inference latency that each layer type is responsible for on two different GPUs: the NVIDIA V100 and NVIDIA A100. We notice that on both GPUs, attention layers (shown in red) account for over 80% of total inference latency, with the rest coming from feed-forward (blue) and embedding layers (yellow). Due to their outsize influence on total inference latency, recent work has explored various approaches for runtime performance optimization that specifically target attention layers; this includes work such as PAR Transformer [78], where attention layers are re-distributed within the network to optimize performance, and various papers on pruning either attention heads and/or entire attention layers [121].

[*] Published in the 1st Workshop on Dynamic Neural Networks, International Conference on Machine Learning (ICML'22) [59]

[†] Submitted to 6th Conference on Machine Learning and Systems (MLSys'23) [60]

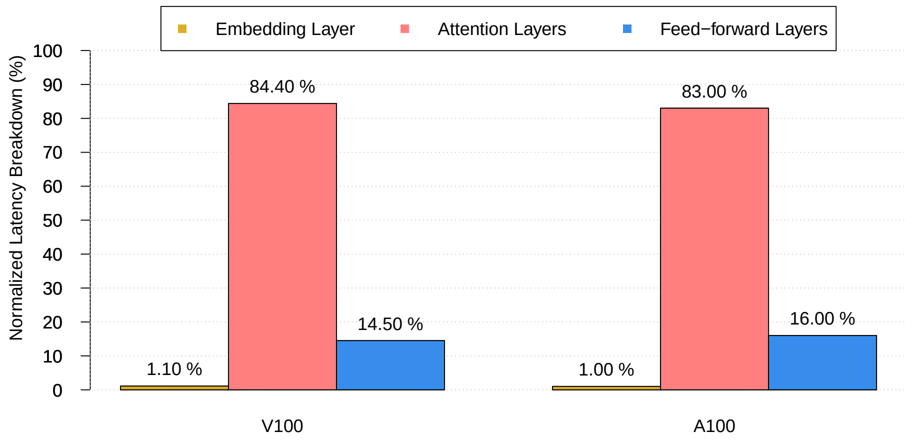


Figure 4.1: Profiling results for different Transformer-XL layers on NVIDIA V100 and A100 GPUs

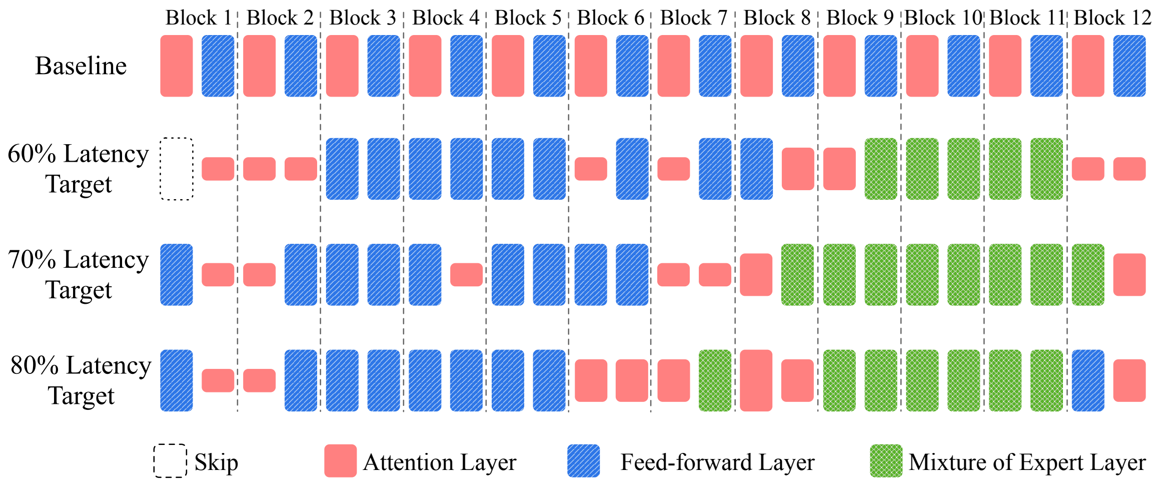


Figure 4.2: Exploration results for Transformer-XL Base model on the *enwik8* dataset for different latency targets. The width of the attention layer blocks represent the relative number of heads.

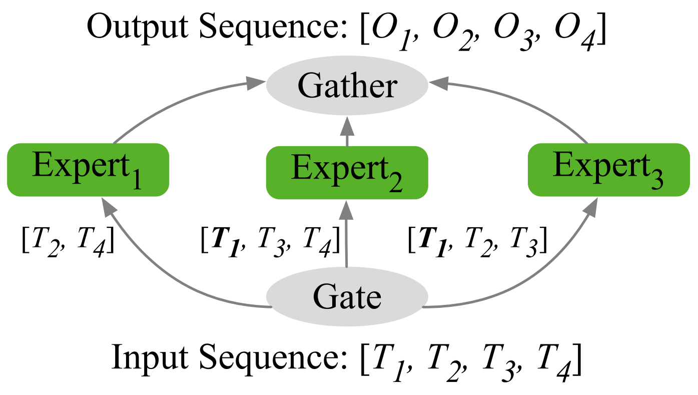
A separate body of work has explored the addition of sparsely activated layers to Transformer models to improve task performance [102]. In particular, mixture-of-expert (MoE) Transformer variants such as Switch Transformer [29] have demonstrated state-of-the-art task performance while simultaneously improving training and inference costs. While most work in this direction has focused on improving task accuracy, in this paper we attempt to answer the following question: *can the addition of sparsely activated layers help* ***preserve*** *accuracy in the face of latency-optimizing network transformations such as skipping/pruning attention layers? And if so, to what extent?*

To help answer this question, we present PLANER, a novel system for designing latency-aware sparsely activated Transformer networks. Given a Transformer-based model as input, along with an inference latency target expressed as a percentage of the baseline model's latency, PLANER produces a sparsely-activated Transformer model that fulfills the latency objective while preserving baseline accuracy. PLANER employs an efficient two-phase gradient descent-based neural architecture search (NAS) strategy with a dynamic loss formulation to achieve this. During the search process, PLANER efficiently explores the large number of alternative architectures arising from different combinations of feed-forward, attention (with varying number of heads), and mixture-of-expert layers; as a concrete example, PLANER considers over 68 billion unique architectures for the Transformer-XL model in our evaluation. The optimized architecture obtained from NAS is then fine-tuned using a load-balancing loss term to produce the final network. Figure 4.2 demonstrates how PLANER infers different architectures depending on the user-provided inference latency targets. Here, each of the inferred architectures matches baseline accuracy, but has different inference latencies. Depending on the latency target, we notice that PLANER progressively reduces the number of attention layers and their widths, while using additional MoE and/or feed forward layers to compensate for potential accuracy drops.

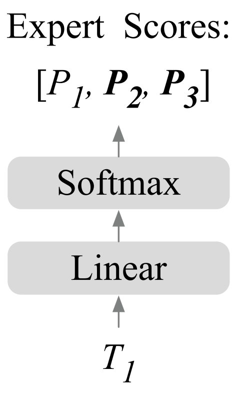
We evaluate PLANER on three different Transformer-based networks drawn from language modeling, and demonstrate an inference latency reduction of at least 2$\times$ for each network while maintaining baseline accuracy. We also compare PLANER with prior work such as PAR Transformer [78] and Sandwich Transformer [91], and with parameter-matched non-MoE implementations of the final optimized networks.

4.2 Background and Motivation

Mixture-of-expert (MoE) networks [79] dynamically partition the input domain so that each sub-network or "expert" specializes in one or more input partitions, yielding a sparsely activated network. Recent work has explored the application of MoE layers to efficiently increase the model capacity of Transformer-based architectures [102, 67, 29,


Output Sequence: $[O_1, O_2, O_3, O_4]$
Gather
Expert$_1$
Expert$_2$
Expert$_3$
$[T_2, T_4]$
$[\boldsymbol{T_1}, T_3, T_4]$
$[\boldsymbol{T_1}, T_2, T_3]$
Gate
Input Sequence: $[T_1, T_2, T_3, T_4]$

(a) MoE Layer



(b) Gate

Figure 4.3: General overview of MoE layers and gate function.

33]. These sparsely-activated architectures are shown to achieve similar accuracy gains without the proportional increase in computation compared to traditional scaling of network parameters [93]. In this work, we focus on applying MoE layers to improve inference latency while maintaining baseline accuracy.

Figure 4.3a depicts a general implementation of an MoE layer with three experts. The sequence of input tokens are distributed among the experts for processing, where each token is processed by one or more experts. The number of experts per token is denoted as Top_K in this work. In Figure 4.3a, Top_K is two. A single-layer linear classifier called a *Gate* (Figure 4.3b) decides which expert(s) to use to process a specific token. The Gate generates a probability distribution across the experts per token, which will then be used to select the Top_K experts.

Layer-wise Performance Analysis: To better understand the performance behavior of Transformer-based networks, we present layer-wise profiled latencies for the Transformer-XL Base network in Figure 4.4. Here, each bar represents the latency of a network block normalized to the latency of default multi-head attention with 8 heads. Note that the default FFL inner dimension in the Transformer-XL Base network is 2048. Profiling is performed with a model dimension of 512, sequence length of 192, and batch size of 64 on an NVIDIA A100 GPU. We observe three key points from the figure: (1) the significant cost of the

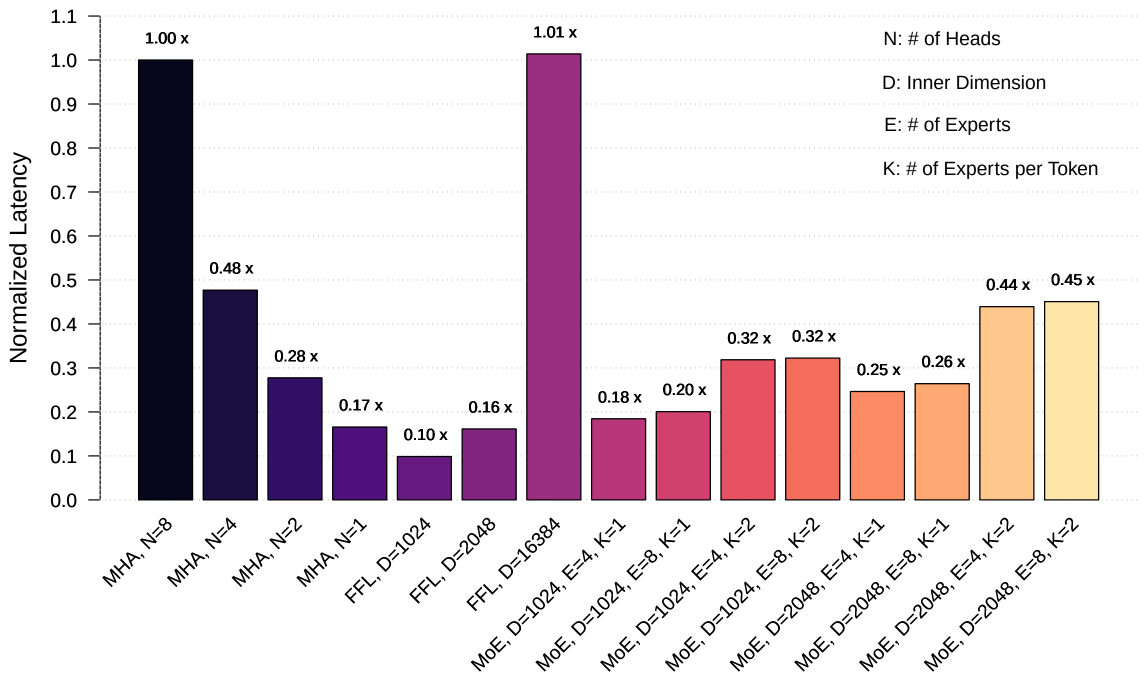


Figure 4.4: Latency comparison of attention, FFL, and MoE layers normalized w.r.t. attention with 8 heads, profiled on NVIDIA A100 GPU with batch size of 64, sequence length of 192, and half-precision.

default attention configuration, amounting to a 6.2× higher runtime compared to the default feed-forward layer (FFL) with an inner dimension of 2048, (2) the approximately linear scaling of the attention cost with respect to the number of heads (pruning attention heads and/or blocks could thus play a significant role in improving network performance), and (3) the compute efficiency of the MoE blocks compared to both attention and iso-parametric FFL blocks (iso-parametric FFL blocks are obtained by scaling up an FFL block according to the number of experts to match the number of parameters in a corresponding MoE block), signifying the promise of using MoE blocks as a cost-effective solution to scale the learning capacity and compensate for the potential accuracy loss caused by aggressive attention pruning.

4.3 Searching for Efficient Transformers

In this section, we provide a thorough description of PLANER's two-phase NAS methodology for finding optimal latency-aware Transformers.

4.3.1 Phase 1: Search Space Exploration

Transformer-based models are composed of multiple blocks, where each block consists of multi-head attention (MHA) and feed-forward layers (FFLs) [118]. MoEs could thus be applied to either MHA or FFLs, or both. In this work, we only explore MoE FFLs in the design space; this is primarily due to the runtime overhead introduced by dynamic behavior, which we found to be prohibitively high for the already expensive attention layers. PLANER's first phase explores the large design space composed of different configurations of MHAs, FFLs, and MoE layers (see Section 4.4.1 for the full search space). The inputs to the first phase are the design space, the backbone of the baseline network architecture, and a target latency, expressed as a ratio with respect to the baseline latency.

For real-world networks, the design space of alternative architectures often gets prohibitively large; for instance, the Transformer-XL Base network on the *enwik8* dataset yields a search space size of over 68 billion architectures. To keep the search tractable, we deploy a differentiable NAS strategy, which has been shown to be significantly more efficient than reinforcement-learning-based approaches [135]. We follow a NAS algorithm similar to the one proposed by [125].

Phase 1 first composes a *search architecture* using the baseline network's backbone as depicted in Figure 4.5. The backbone includes details on the number of blocks (MHA or FFLs) and their configuration (number of heads or hidden dimension). Using the input backbone, each of the MHA or FFL blocks in the baseline network are replaced with *Super Blocks (SB)*, which includes all the search options in the design space. The goal is to find the best option for each block so that overall accuracy is maximized and the latency target is achieved. Figure 4.6 depicts the formulation of super blocks. Each of the search options

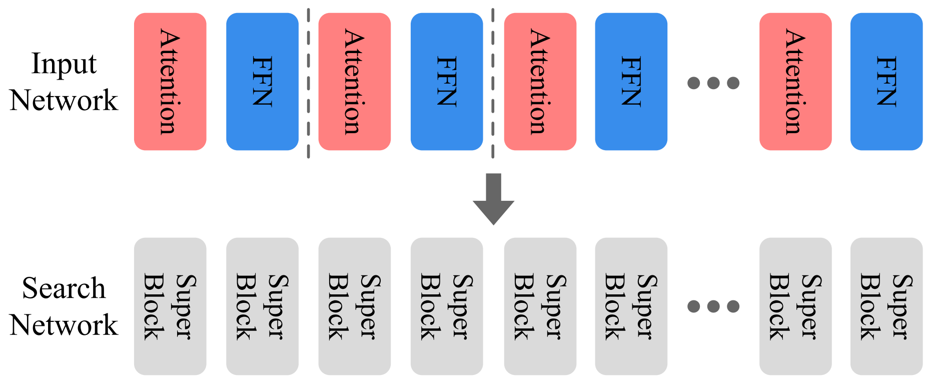


Figure 4.5: Composing the search network from the input network backbone

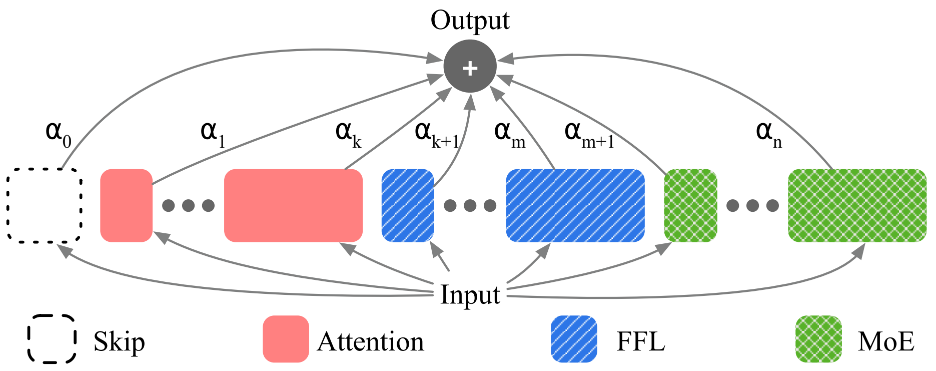


Figure 4.6: Formulating super blocks from the search space.

$Block_i$ is accompanied by corresponding architectural weights α_i, which are trained using gradient descent to represent the benefit factor of the search option [125]. To make the optimization graph differentiable with respect to the architecture weights, the output of the super block is formulated as:

$$Output = \sum_{i=0}^{n} P_i \times Block_i(Input)$$
$$s.t. \quad P_i = GumbelSoftmax(\alpha_i, [\alpha_0, ..., \alpha_n]) \tag{4.1}$$

Where the $GumbelSoftmax$ generates probability values by sampling the Gumbel distribution based on α weights.

This formulation yields two sets of parameters to be trained in Phase 1. The first group contains the actual network weights ($Block_i$), and the second group the architectural weights

(α_i). Training of each parameter group is done sequentially in each epoch, using separate optimizers. Thus each epoch of training in phase 1 consists of optimizing the network weights using 100% of the training samples, and then training the architecture weights using 20% of the randomly sampled training data. We use soft sampling for *GumbelSoftmax* during architecture optimization, and hard-sampling while training the network weights to reduce the overheads associated with the super blocks. To ensure that neither of the network weight sets are starved due to the hard-sampling of *GumbelSoftmax*, the architecture optimization is initially disabled for 10% of the epochs, and an annealing temperature scheduling is used for later epochs. These settings allow the blocks to be randomly sampled for the appropriate number of search epochs.

4.3.2 User-defined Latency Optimization

PLANER supports the specification of a latency target to provide more flexibility to users in exploring the performance-accuracy trade-off curve. To incorporate latency optimization in the search phase, we formulate an auxiliary loss based on the latencies of the search and baseline network, as well as the target latency. We use an estimation for the end-to-end latency of the search network as well as baseline in phase 1, using lookup tables filled with individual block latencies similar to prior work [125]. Equation (2) presents the formulation for the estimated latency which is composed of accumulating the latencies of each super block (*Lat_SB*).

$$\begin{aligned} Lat &= \sum_{b=0}^{B} Lat\_SB_b, \\ &s.t. \quad Lat\_SB_b = \sum_{i=0}^{n} P_{bi} \times Lat_i \end{aligned} \tag{4.2}$$

Here, Lat_i represents the profiled latency of $Block_i$ in isolation, and P_{bi} values correspond to the probability values for super block of b as sampled in Equation (1) with respect to the architecture weights.

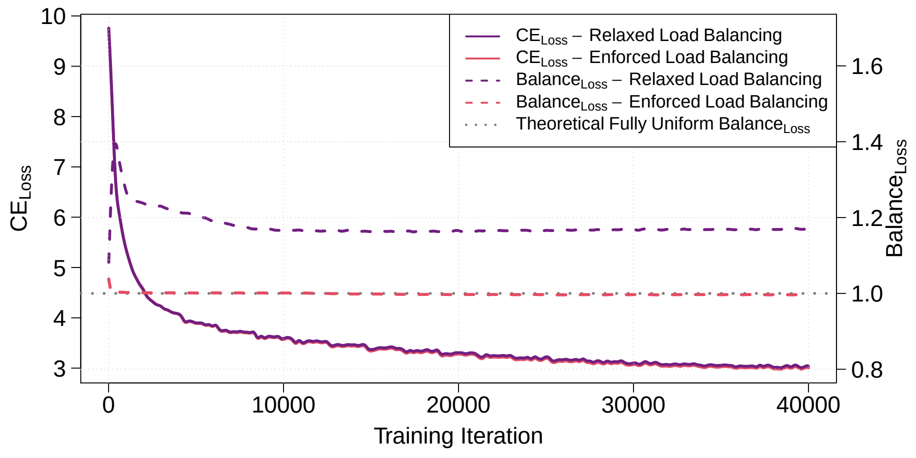
The latency loss Lat_{Loss} is implemented as the ratio of the estimated latency of the search network (Lat) over the normalized baseline latency with respect to the target latency.

$$\begin{aligned} & Loss = CE_{Loss} + \beta \times Lat_{Loss} \\ & \qquad s.t. \quad Lat_{Loss} = Lat \;/\; (Lat_{Baseline} \times Target_{Lat}) \\ & \qquad s.t. \quad \beta = 1 \quad if \; (Lat_{Loss} > 1) \quad else \; 0 \end{aligned} \tag{4.3}$$

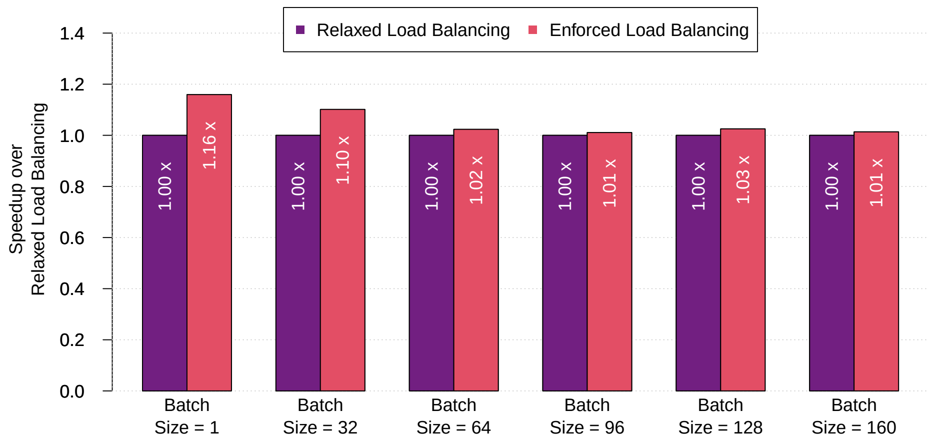
During the training of the architecture weights, the latency loss will be automatically activated depending on whether the estimated latency of the search network is meeting the target latency requirement. For example, if the target latency is set to 50% of the baseline, the latency loss will only get included if the estimated latency is higher than $0.5 \times Lat_{Baseline}$. Otherwise, the scalar factor of β would be 0 in Equation 3, leading the optimizer to adjust the architecture weights solely in the direction of minimizing the CE_{Loss}. This novel dynamic functionality helps the search progress towards the user latency target *without the need for additional hyper-parameter tuning*.

4.3.3 Phase 2: Architecture Sampling and Retraining

The optimized architecture obtained from Phase 1 is now instantiated for retraining. Since the weights of this final architecture were shared with other search points during Phase 1, a retraining step is necessary to avoid under-fitting and to obtain optimal accuracy. We construct the optimized architecture by selecting the blocks with the highest architecture weight values in each super block; from our empirical evaluation, this sampling strategy best balances additional training overheads with accuracy compared to other approaches such as the one described in [75]. We retrain the sampled architecture from scratch using the same settings as the baseline.



(a) Comparison of CE_{Loss} and $Balance_{Loss}$.



(b) Comparison of MoE Runtime across different batch sizes.

Figure 4.7: Impact of relaxing or enforcing the balance loss on training flow as well as MoE runtime.

4.3.4 Balancing Load Across Experts in MoE Layers

Since MoE blocks may be part of the final architecture, we incorporate an auxiliary loss during Phase 2 to enforce a balanced load across the experts. We follow the same implementation of the auxiliary loss for load balancing ($Balance_{Loss}$) as Switch Transformer [29].

Consider an MoE layer with E experts:

$$\begin{aligned} &Loss = CE_{Loss} + Balance_{Loss} \\ &\qquad s.t. \quad Balance_{Loss} = E \times \sum_{e=0}^{E} F_e \times G_e \end{aligned} \tag{4.4}$$

Here, F_e represents the fraction of the tokens processed by expert e, and G_e measures the average gate score received by expert e across the input tokens.

The $Balance_{Loss}$ provides an approximation for the load balancing score across experts. If the tokens are distributed uniformly across the experts by the gate function, we can expect each expert to process $\frac{1}{E}$ of the input tokens, while receiving an average score of $\frac{1}{E}$ from the gate. This would result in $Balance_{Loss}$ having an ideal value of 1 in a fully-uniform distribution of tokens across the experts. If there is more than one MoE layer in the architecture, the $Balance_{Loss}$ is the average of the individual loss values across the MoE layers.

Figure 4.7a compares the Phase 2 training progress of a Transformer-XL architecture with multiple MoE layers under two scenarios: (1) when the $Balance_{Loss}$ term is excluded from the loss function (*Relaxed Load Balancing*), and (2) when the loss function includes the $Balance_{Loss}$ term (*Enforced Load Balancing*). From the figure, we notice that trends for the CE_{Loss} term are similar in both scenarios, highlighting the fact that overall accuracy of the network is unaffected by load balancing constraints. From our experiments, we also notice that a balanced load improves the runtime of MoE layers by reducing tail latency - we illustrate this in Figure 4.7b. Here, we notice a runtime speedup of up to $1.16\times$ for MoE layers when load balancing is enforced.

4.4 Evaluation

We evaluate PLANER on three real-world language modeling tasks and compare the performance of the latency-optimized networks to other state-of-the-art efficient Transformer models. We also provide a detailed analysis of the impact of using our dynamic loss formulation.

4.4.1 Methodology

We use Transformer-XL (TXL) Base on the *WikiText-103 (WT103)*, *enwik8*, and *text8* datasets as our baseline networks. *WT103* is a large word-level language modeling dataset composed of more than 100M training tokens and a vocabulary size of 267735 words. *enwik8* and *text8* are both character-level language modeling benchmarks with more than 90M training characters. While *enwik8* treats the data as a sequences of bytes and has a vocabulary size of 204 Unicode symbols, *text8* is composed of English characters and spaces (vocabulary size of 27 characters).

The backbone architecture for all datasets uses a model dimension of 512 and an interleaved pattern of multi-head attention (MHA) with 8 heads and feed-forward layer (FFLs) with an inner dimension of 2048. The total number of blocks (MHA/FFL) is 24 for *enwik8* and *text8*, whereas *WT103* uses 32 blocks[‡]. The search space for phase 1 includes: (1) Skip connection, (2) MHA with 1, 2, 4, or 8 heads, (3) FFL with inner dimension of 2048, and (4) MoE FFL with inner dimension of 2048, 8 experts, where each token is processed by either 1 or 2 experts ($Top_K = 1\ or\ 2$).

To evaluate the performance of PLANER, we compare the latency and accuracy of the optimized models with the baseline TXL model and two prior papers: Sandwich Transformer [91] and PAR Transformer [78].

We explore the design space for *WT103* and *enwik8* using PLANER's 2-phase methodology (described in more detail in Section 4.3) with target latencies ranging from 50% to 95%. Due to its similarity to the *enwik8* dataset, we only perform phase 2 retraining for *text8* using the same network architecture found by PLANER for *enwik8* [22].

All training is performed on a node with 8 NVIDIA V100 GPUs. Inference evaluation

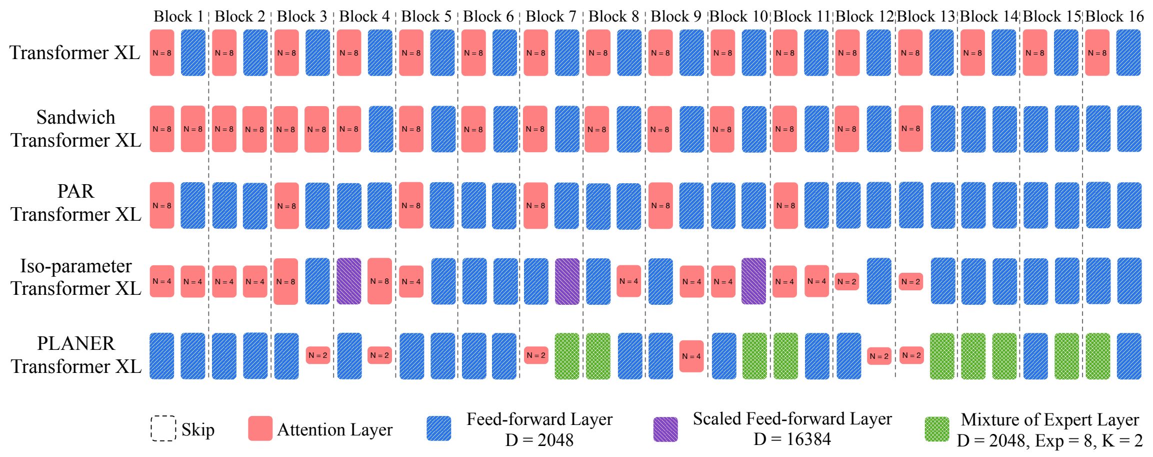
[‡]The number of MHA/FFL blocks is $2\times$ of the number of Transformer blocks.

Model	WT103 (PPL)		enwik8 (BPC)		text8 (BPC)	
	Dev	Test	Dev	Test	Dev	Test
Transformer-XL Base	22.7	23.4	1.114	1.088	1.128	1.194
Sandwich Transformer-XL	22.6*	-	1.107	1.083	1.122	1.188
PAR Transformer-XL	22.7*	-	1.121	1.119	1.129	1.197
PLANER Transformer-XL	22.5	23.5	1.109	1.083	1.127	1.191

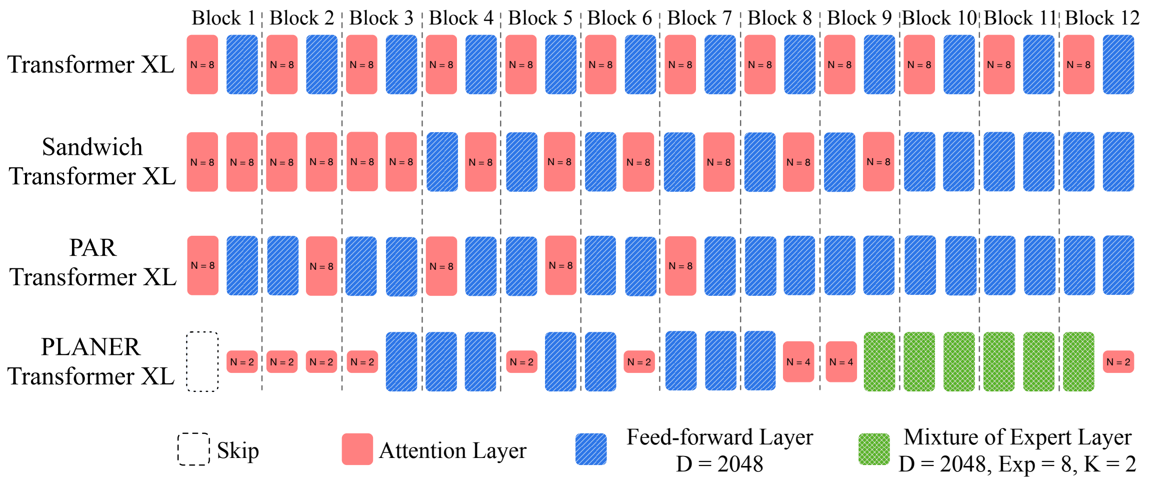
Table 4.1: Accuracy comparison of PLANER with prior work and baselines (scores marked with $*$ are referenced). Lower is better for both PPL and BPC metrics.

and look-up table generation done on A100. We use the settings published by NVIDIA for hyper-parameters [84]. The exact hyper-parameters used for each dataset are:

- **WikiText-103 - Network Weights (Phase 1 and 2):** JITLamb optimizer, learning rate of 0.01, batch size of 256, target and memory length of 192, dropout rate of 0.1 for non-MoE layers and 0.2 for MoE layers, and 40000 iterations.
- **WikiText-103 - Architecture Weights (Phase 1):** Adam optimizer, learning rate of 0.01, initial temperature of 5 for the Gumbel Softmax, and temperature annealing rate of 0.6.
- **enwik8 - Network Weights (Phase 1 and 2):** JITLamb optimizer, learning rate of 0.004, batch size of 64, target and memory length of 512, dropout rate of 0.1 for non-MoE layers and 0.3 for MoE layers, and 120000 iterations.
- **enwik8 - Architecture Weights (Phase 1):** Adam optimizer, learning rate of 0.01, initial temperature of 5 for the Gumbel Softmax, and temperature annealing rate of 0.7.
- **text8 - Network Weights (Phase 1 and 2):** Learning rate of 0.003, other settings same as *enwik8*.
- **text8 - Architecture Weights (Phase 1):** Same as *enwik8*.



(a) WT103

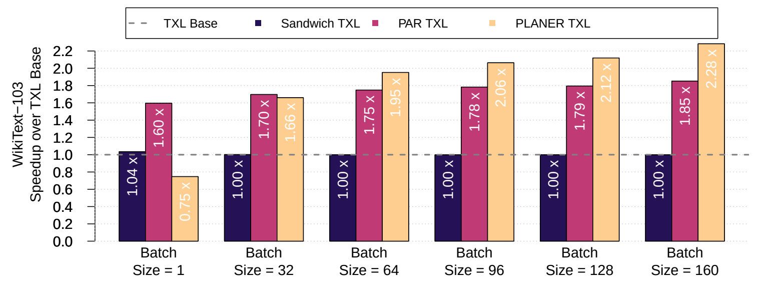


(b) enwik8 and text8

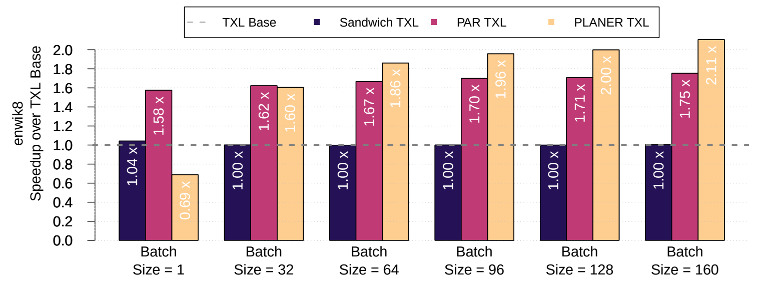
Figure 4.8: Layer-wise breakdown of each evaluated architecture for *WT103* (top) and *enwik8* and *text8* (bottom).

4.4.2 Accuracy and Performance Trade-offs

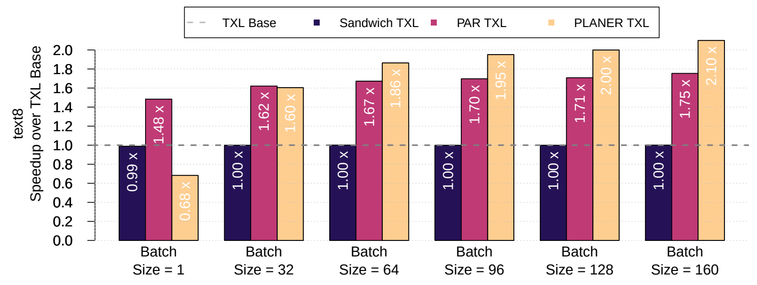
Table 4.1 lists the accuracy numbers obtained by PLANER and compares them with the baseline architectures across both validation and test sets (columns labeled 'Dev' and 'Test', respectively, in Table 4.1). Here, PPL and BPC denote model perplexity and bits-per-character, respectively. For the *WT103* dataset and Sandwich/PAR Transformer-XL, we reference the reported PPL numbers in the literature for the Dev set (entries labeled with asterisks). For *enwik8* and *text8*, all the architectures are re-trained and evaluated across both Dev and Test sets. We notice that all the TXL variants, including ones produced by PLANER,



(a) WT103



(b) enwik8



(c) text8

Figure 4.9: Speedups obtained by PLANER w.r.t. various baselines across different batch sizes and sequence length of 64, profiled on NVIDIA A100. For each task, the PLANER target latency was in the range of 50% to 65%.

maintain baseline accuracy levels. Figure 4.8 provides a deeper dive into the various network architectures we evaluate for the individual datasets. We notice that PLANER aggressively prunes/skips attention layers, while intelligently introducing sparsely activated layers for accuracy recovery.

Figure 4.9 shows the speedups obtained by PLANER and the various baselines (described in Section 4.4.1) across all three datasets and varying batch sizes. For each task, the PLANER target latency was in the range of 50% to 65%. From the Figure, we notice that PLANER

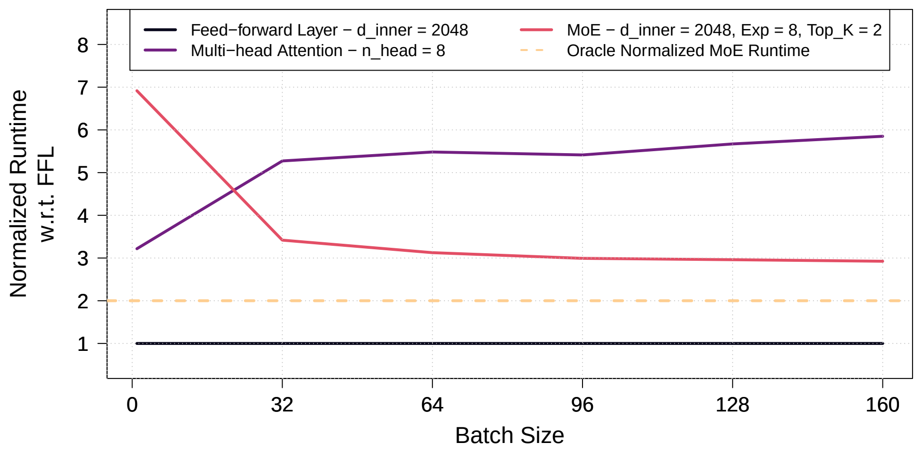


Figure 4.10: Runtime comparison of FFL, MHA, and MoE layers across different batch sizes normalized to FFL runtime.

provides speedups of over 2× at larger batch sizes. On smaller batch sizes, PAR Transformer outperforms PLANER; this is primarily due to the non-optimized MoE layers used in our current implementation. Specifically, our current implementation computes the outputs of each expert sequentially, where a batch of sequences with N tokens are sequentially processed in mini-batches of size $\frac{Top_K \times N}{Experts}$. This consequently leads to under-utilization of the compute units.

Figure 4.10 provides a more detailed overview of the current deficiencies in the sequential implementation of the MoE layers; here, we provide a runtime comparison of the FFL, MHA, and MoE layers across different batch sizes normalized with respect to FFL runtime. At lower batch sizes, MoE layers have an overhead of 7× over FFL, which is also higher than the MHA layers. However, as batch size increases, GPU resource utilization goes up, consequently decreasing the overhead of MoE layers to less than 3×. The oracle implementation (dashed orange line in Figure) shows the theoretically optimal runtime of the MoE layer. Since we use a Top_k value of 2 (viz., each input token is processed by 2 experts), we notice a corresponding 2× runtime overhead over the baseline FFL. Note that the oracle runtime does not take overheads related to gate function evaluation and the gathering/scattering of tokens across experts into account - the real-world runtime is thus

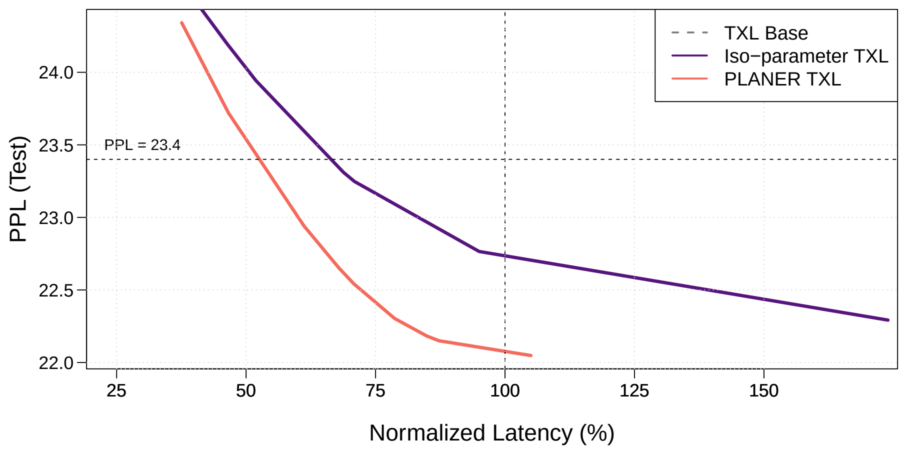


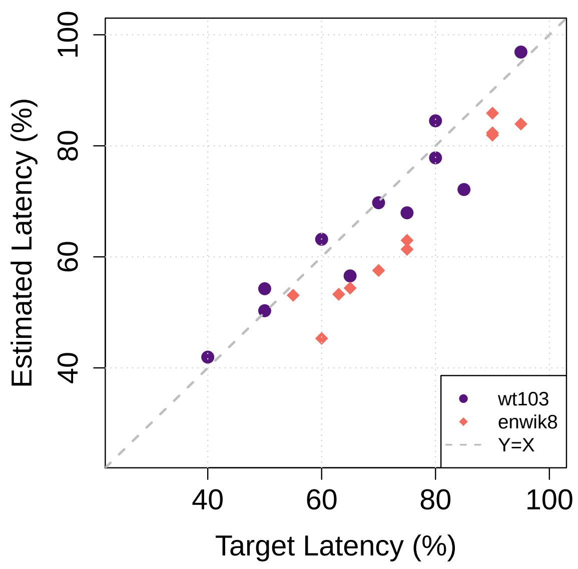
Figure 4.11: Comparison of the Pareto frontiers of the optimized architectures obtained by PLANER for MoE and Iso-parameter scaled FFL setups. Profiled on NVIDIA A100 with batch size of 64 and sequence length of 64.

likely to be higher. We are currently working on a more optimized parallel implementation of MoE layers, which will help plug this performance gap across various batch sizes.

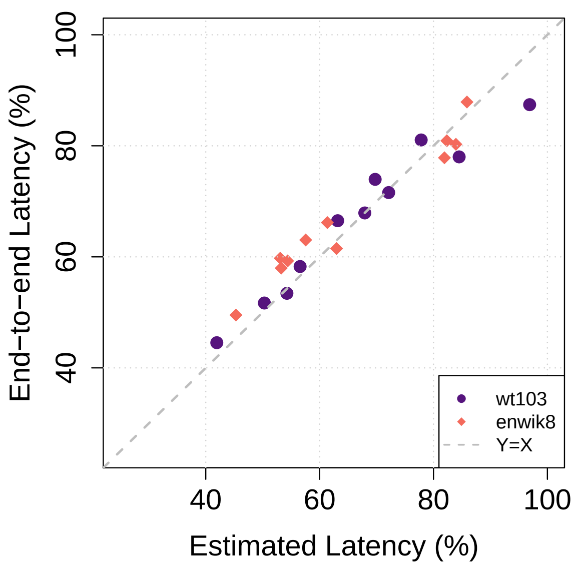
4.4.3 Comparison to Iso-parametric Setting

We also compare PLANER to an iso-parameter setup, which replaces the MoE with a scaled FFL in the search space. The scaled FFL has an inner dimension of 16384, which results in the same number of parameters as the MoE with 8 experts. The goal of the iso-parameter experiment is to analyze the effectiveness of different model scaling solutions in compensating for accuracy drops caused by aggressive attention pruning.

Figure 4.11 presents the comparison of the Pareto frontiers of the architectures obtained by PLANER with different latency targets on the *WT103* dataset. From the Figure, we clearly notice that the use of MoE layers results in higher performance architectures across the board at different accuracy levels. Further performance benchmarking reveals that scaled FFL layers are at least 2× slower than our (relatively unoptimized) MoE layers and actually approach the runtime of the much slower MHA layers with 8 heads. Naively scaling up the size of FFLs is thus not an ideal option for either improving accuracy or performance.



(a) Target vs estimated



(b) Estimated vs end-to-end

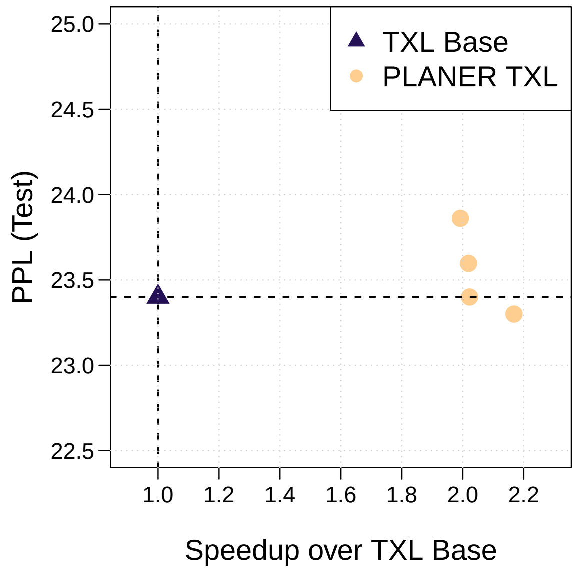
Figure 4.12: Correlation between target, estimated, and end-to-end latency.

4.4.4 Validating Estimated and End-to-end Runtime

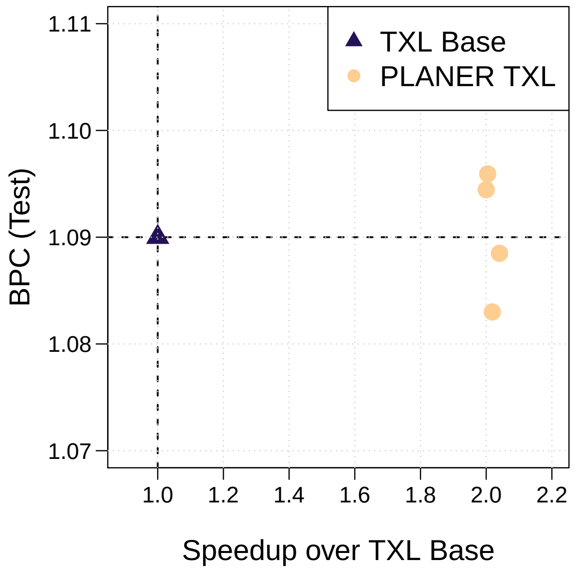
In this section, we analyze the performance of the dynamic latency loss used in Phase 1. Figure 4.12a shows the correlation between input target latency and the estimated latency of the architectures sampled at the end of Phase 1, while Figure 4.12b shows the correlation between estimated latency and profiled end-to-end latency. We make two important observations from the figures: (1) our dynamic latency loss formulation successfully steers the NAS towards architectures that match the input target latency, and (2) the latency estimated in Equation (2) is highly correlated with real-world latency, making it an appropriate option for PLANER's Phase 1 search.

4.4.5 Repeatability Evaluation

To evaluate and validate the reproducibility of our experiments, and observe any potential variations in the final architectures, we also repeat the PLANER optimization of the architectures evaluated in Section 4.4.2. For this experiment, we keep all hyper-parameters fixed, but repeat PLANER's search process four times. Figure 4.13 presents the achieved



(a) WT103



(b) enwik8

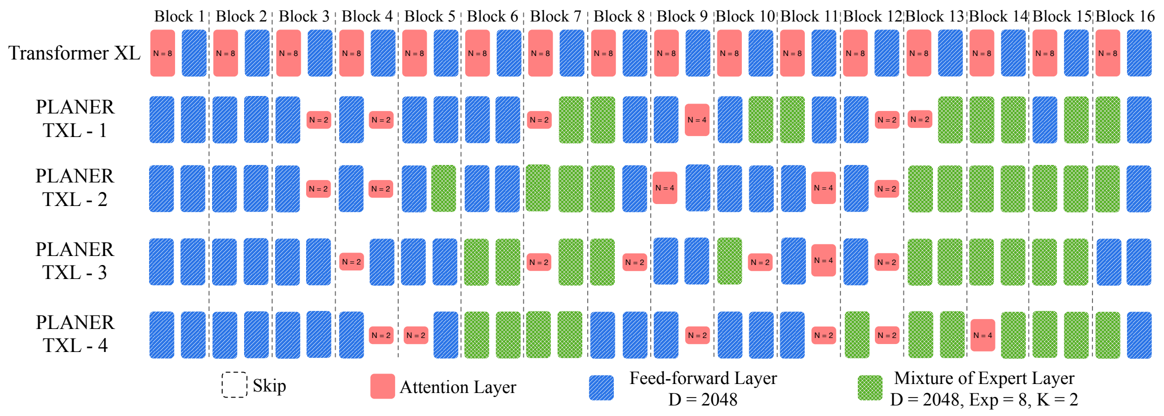
Figure 4.13: Speedup and accuracy results for the repeatability experiment.

accuracy and speedup numbers from our experiment. We notice that all the accuracy values are within 0.5% of the baseline, with speedups consistently over $2\times$.

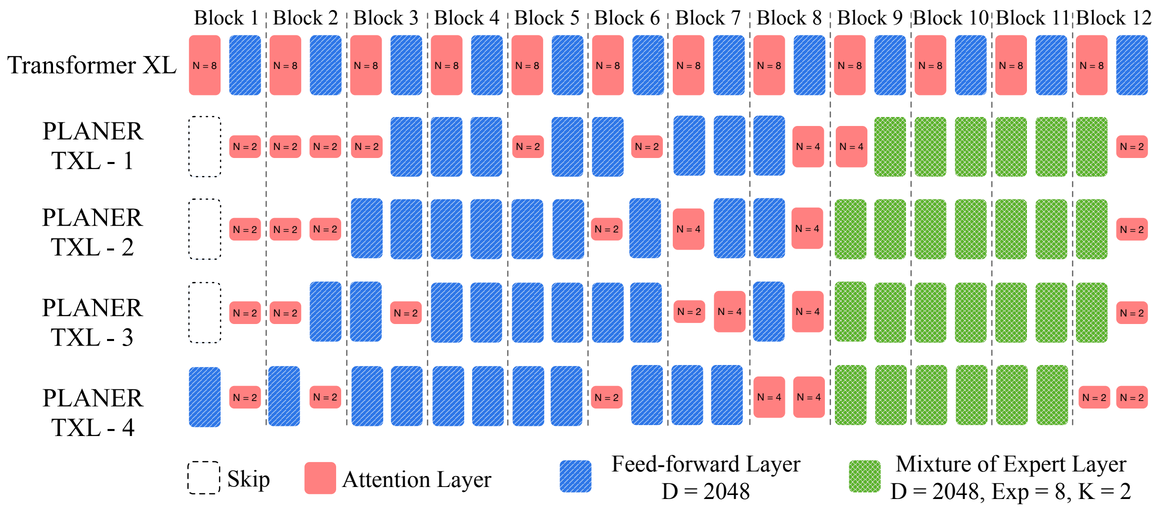
Figure 4.14 illustrates the variations in the final architectures explored at the end of phase 1 for both the *WT103* and *enwik8* datasets. Although the architectures do not match exactly, we notice a strong similarity in the total number of heads in the attention layers (14 ± 0 heads for *WT103* and 20 ± 2 heads for *enwik8*, across all four repeats). We also notice that MoE layers tend to be concentrated towards the end of the networks across both datasets.

4.5 Related Work

The introduction of the Transformer family of networks has overhauled the domain of NLP. These attention-based architectures have been shown to outperform their LSTM-based counterparts both in terms of effectively capturing time dependencies [118] as well as inference latency [103]. The general architecture of these models consists of multiple Transformer blocks, where each Transformer block consists of multi-head attention(s) and



(a) WT103



(b) enwik8

Figure 4.14: Explored architectures through repeatability experiment on *WT103* and *enwik8* dataset.

feed-forward layers.

Recent work has introduced Mixture-of-Expert (MoE) layers within networks to decompose tasks into sub-tasks, where experts could be trained on individual sub-tasks [79]. One motivation behind this idea is to dynamically partition the input space, with experts getting specialized on individual partitions. Recent work has also studied the application of MoE layers to efficiently increase the model capacity of Transformer-based architectures [102, 67, 29, 33]. These sparsely-activated architectures have been shown to achieve accuracy gains without the proportional increase in computation compared to traditional scaling of network parameters [93]. While MoE layers have been applied for accuracy improvement and training speed-ups, their use in designing latency-aware architectures have not been explored as thoroughly.

A separate body of work has also focused on optimizing the performance of Transformer models. In particular, [91] show that it is possible to achieve better accuracy by redistributing multi-head attention and FFL layers across the network while maintaining the original runtime. PAR [78] deploys NAS to explore the number and distribution of attention layers (while keeping the same head count) for improved latency. [121] prune attention heads and reduce the width of FFLs (while keeping the same backbone as the baseline) using an evolutionary NAS algorithm to design hardware-aware Transformers. While these works have explored the distribution or configuration of individual (non-MoE) layers in isolation, none of them consider both aspects simultaneously as part of a larger NAS search space. Additionally, as we demonstrate in this paper, the inclusion of MoE layers in the design space can help reduce inference latency further by removing/pruning attention layers more aggressively while maintaining baseline accuracy.

Finally, [42] explore the possibility of employing sparsely activated layers for designing more computationally efficient Transformer architectures. While this work shares some of the the same goals as PLANER, it primarily targets floating point operation (FLOP) reduction, which has been demonstrated to have little to no correlation with actual measured runtime, especially on parallel hardware [132]. Additionally, [42] limit the NAS space by constraining the location of attention and MoE layers to those in the baseline Transformer architecture. While this could potentially help reduce search complexity, it also significantly limits performance gains and speedups [91].

4.6 Conclusion

This paper has presented PLANER, an automated system for optimizing the inference latency of Transformer-based networks. PLANER employs a two-phase NAS methodology to systematically introduce sparsely activated layers into the given network, and uses a dynamic loss formulation to achieve user-provided latency targets while preserving accuracy. On three real-world NLP models, PLANER achieves inference latency reductions of over 2$\times$ at iso-accuracy.

CHAPTER V

Conclusion

5.1 Summary

The overarching goal of my research is to optimize deep learning applications both in the efficiency of executing them on various hardware platforms, and also trustworthiness of the outputs predicted by them. My work was aiming diverse domains in deep learning such as computer vision and natural language processing, including applications including autonomous vehicles such as self-driving cars, machine learning in healthcare such as precision medicine, and many other mainstream deep learning applications both in servers and also mobile platforms such as language modeling, machine translation, image processing, and etc.

In Chapters II and III, my focus was to improve the reliability aspects of deploying deep learning modules in the mission critical applications. The goal was to provide systematic solutions to efficiently identify the unreliable behaviors in the deep learning inference due to the inherent inaccuracy problems of these probabilistic models. In both of theses chapters, my optimizations were targeting both the reliability and efficiency of the deep learning systems. In Chapter IV, the goal was to optimize the compute efficiency of the Transformer architectures in the means of reducing the inference latency of these architectures. The reliability aspect of this project was on the lines of maintaining the baseline accuracy levels, in order to preserve the original quality of the Transformer architecture.

During the Chapter II, the goal was to develop a practical method to design and realize systems of CNNs that can increase robustness and reliability of classification results with imprecise and currently available CNNs as the building blocks. During the initial reliability assessment, I observed that regardless of the baseline accuracy of the CNN, nearly 10% of the predictions made by are wrong with a high confidence, which is a significant number. Next, inspired by the modular redundancy techniques in hardware reliability domain, I proposed the PolygraphMR system. PolygraphMR uses input preprocessing techniques to develop different variations of a CNN and combines them as a modular redundant system of heterogeneous CNNs. PolygraphMR enhances reliability by leveraging variations in behavior of each CNN that is trained in different circumstances. It takes advantage of behavior diversity to detect symptoms of unreliability from the predictions of individual CNN variants. Different performance optimizations were also applied to reduce the overhead of the system by: First, by reducing the energy consumption of the individual CNNs in the system by compressing the network weights using precision reduction. Second, by activating only a portion of the CNNs by default and activating other CNNs in case the reliability status could not be determined by the initially activated CNNs. Across six benchmarks evaluated, PolygraphMR detects an average of 33.5% of the baseline mispredictions with less than 2x overhead.

In Chapter III, I extended my learnings from image classification to the object detection domain. I studied intra-sensor fusion as complementary approach to traditional sensor fusion to increase both the precision as well as recall of the object detectors. With intra-sensor fusion, a diversity of inputs is synthesized at runtime, e.g., image/video filtering, to feed the original object detector. Intelligent combination of output data from individual inputs can lead to improvement in the precision of detections in the original objects as well as detection of the new objects that were missed by the original detector. My goal in this project was to achieve higher detection accuracy and reliability, with the least amount of overhead introduced in latency and cost. With careful design of synthesized inputs, it is possible to run inference in a batched fashion and stay away from the multiplicative cost of multiple inferences as in traditional modular redundancy. Furthermore, with intelligently processing the data from individual inputs, we can achieve more precise bounding box proposals with fewer objects left undetected. The evaluatation results show an improvement of 3.45% in the average precision of the predictions, with less than 23% latency overheads over the baseline.

Finally in Chapter IV, I targeted the efficiency of the Transformer architectures that are getting widely deployed both in natural language processing and computer vision domains. I discovered that main bottleneck in the inference of these models correspond to the multi-head attention layers. So a systematic optimization methodology is proposed to automatically alter the baseline Transformer architecture to satisfy user's latency targets. PLANER dynamically decides on the distribution of attention layers and their number of heads in order to comply with the latency requirements, and it employs mixture-of-expert layers when necessary throughout the network architecture to compensate for potential accuracy loss caused by attention pruning. PLANER achieves more than $2\times$ reduction in the inference latency of three language modeling tasks, while preserving the original accuracy levels.

5.2 Future Directions

Based on the solutions proposed in this dissertation, there are different new directions that could be explored as future directions.

PolygraphMR and SoftFusion adapt the idea of modular redundancy in computer vision domain for improved reliability. The proposed systems target image inputs in image classification and object detection tasks. However, there are two main directions that we could explore as future directions. First, we could extend the modular redundancy idea to other tasks in computer vision that are also being deployed in mission-critical tasks. An example for this is the Image Segmentation that has applications in the self-driving cars. Compared to image-classification, in segmentation we need to do the classification task for each individual pixel of the input image. This could provide new challenges and opportunities for us. For instance in the decision engine, we should consider the additional computational costs due to the scope of the problem. On the other hand, continuity of the objects in the neighboring pixels of an image, could provide us new opportunities to be explored in the decision policy for improved reliability.

Another direction that we could follow in the scope of reliability, is to adapt the idea of intra-sensor fusion proposed in SoftFusion to other input domains, i.e., LIDAR and Radar data. Considering the availability of LIDAR and Radar sensors in the self-driving cars, we could ask the question of what preprocessing techniques we could apply to these non-image input domains in the augmentation step.

Finally, the optimization methodology proposed in PLANER could be employed for any deep learning task using the Transformer architectures. This includes other tasks such as machine translation in the natural language processing domain, as well as the recently emerging applications in the computer-vision domain that are also based on these attention-based architectures [114, 52]. A future direction for PLANER, is to extend it to other domains in deep learning and assess the potential achievable speedup rates. On the other hand, the optimizations in the PLANER is hardware-aware, and I have only targeted GPU platforms. Therefore, analyzing the explored architectures by PLANER for other hardware platforms such as mobile accelerators, is another potential direction.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Nvidia tesla v100.

[2] Papers with code - coco test-dev benchmark (object detection).

[3] Papers with code - imagenet benchmark (image classification).

[4] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[5] AlexeyAB. Alexeyab/darknet.

[6] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174. IEEE, 2009.

[7] Wendy Bartlett and Lisa Spainhower. Commercial fault tolerance: A tale of two systems. *IEEE Transactions on dependable and secure computing*, 1(1):87–96, 2004.

[8] David Bernick, Bill Bruckert, Paul Del Vigna, David Garcia, Robert Jardine, Jim Klecka, and Jim Smullen. Nonstop/spl reg/advanced architecture. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 12–21. IEEE, 2005.

[9] Simone Bettola and Vincenzo Piuri. High performance fault-tolerant digital neural networks. *IEEE transactions on computers*, 47(3):357–363, 1998.

[10] Christophe Blanc, Laurent Trassoudaine, and Jean Gallice. Ekf and particle filter track-to-track fusion: A quantitative comparison from radar/lidar obstacle tracks. In *2005 7th International Conference on Information Fusion*, volume 2, pages 7–pp. IEEE, 2005.

[11] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[13] Luca Bombini, Pietro Cerri, Paolo Medici, and Giancarlo Alessandretti. Radar-vision fusion for vehicle detection. In *Proceedings of International Workshop on Intelligent Transportation*, pages 65–70, 2006.

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[15] Ángela Casado-García and Jónathan Heras. Ensemble methods for object detection. In *ECAI 2020*, pages 2688–2695. IOS Press, 2020.

[16] Josip Ćesić, Ivan Marković, Igor Cvišić, and Ivan Petrović. Radar and stereo vision fusion for multitarget tracking on the special euclidean group. *Robotics and Autonomous Systems*, 83:338–348, 2016.

[17] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018.

[18] Hyunggi Cho, Young-Woo Seo, BVK Vijaya Kumar, and Ragunathan Raj Rajkumar. A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1836–1843. IEEE, 2014.

[19] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[20] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, et al. Serving dnns in real time at datacenter scale with project brainwave. *iEEE Micro*, 38(2):8–20, 2018.

[21] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 3642–3649. IEEE, 2012.

[22] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive Language Models beyond a Fixed-length Context. *arXiv preprint arXiv:1901.02860*, 2019.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[25] Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

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

[27] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[28] Francesca M Favarò, Nazanin Nader, Sky O Eurich, Michelle Tripp, and Naresh Varadaraju. Examining accident reports involving autonomous vehicles in california. *PLoS one*, 12(9):e0184952, 2017.

[29] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[30] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[31] Ji Gao, Beilun Wang, Zeming Lin, Weilin Xu, and Yanjun Qi. Deepcloak: Masking deep neural network models for robustness against adversarial samples. 2017.

[32] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330, 2017.

[33] Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fast-MoE: A Fast Mixture-of-Expert Training System. *arXiv preprint arXiv:2103.13262*, 2021.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[35] Meijun He, Shuye Zhang, Huiyun Mao, and Lianwen Jin. Recognition confidence analysis of handwritten chinese character with cnn. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 61–65. IEEE, 2015.

[36] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[37] Dan Hendrycks, Mantas Mazeika, and Thomas G Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.

[38] Parker Hill, Animesh Jain, Mason Hill, Babak Zamirai, Chang-Hong Hsu, Michael A Laurenzano, Scott Mahlke, Lingjia Tang, and Jason Mars. Deftnn: Addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 786–799. ACM, 2017.

[39] Hossein Hosseini, Yize Chen, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. Blocking transferability of adversarial examples in black-box learning systems. *arXiv preprint arXiv:1703.04318*, 2017.

[40] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[41] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[42] Ganesh Jawahar, Subhabrata Mukherjee, Xiaodong Liu, Young Jin Kim, Muhammad Abdul-Mageed, Laks VS Lakshmanan, Ahmed Hassan Awadallah, Sebastien Bubeck, and Jianfeng Gao. Automoe: Neural architecture search for efficient sparsely activated transformers. *arXiv preprint arXiv:2210.07535*, 2022.

[43] Saurabh Jha, Timothy Tsai, Siva Hari, Michael Sullivan, Zbigniew Kalbarczyk, Stephen W Keckler, and Ravishankar K Iyer. Kayotee: A fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors.

[44] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[45] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[46] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Proteus: Exploiting numerical precision variability in deep neural networks. In *Proceedings of the 2016 International Conference on Supercomputing*, page 23. ACM, 2016.

[47] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks.

In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[48] Takeo Kato, Yoshiki Ninomiya, and Ichiro Masaki. An obstacle detection method by fusion of radar and motion stereo. *IEEE transactions on intelligent transportation systems*, 3(3):182–188, 2002.

[49] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017.

[50] Mahmoud Khairy, Jain Akshay, Tor Aamodt, and Timothy G Rogers. Exploring modern gpu memory system design challenges through accurate modeling. *arXiv preprint arXiv:1810.07269*, 2018.

[51] Salman Khan, Munawar Hayat, Syed Waqas Zamir, Jianbing Shen, and Ling Shao. Striking the right balance with uncertainty. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 103–112, 2019.

[52] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[53] Bo-Kyeong Kim, Hwaran Lee, Jihyeon Roh, and Soo-Young Lee. Hierarchical committee of deep cnns with exponentially-weighted decision fusion for static facial expression recognition. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 427–434. ACM, 2015.

[54] Alex Krizhevsky. cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks, 2012.

[55] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[57] Felix Kunz, Dominik Nuss, Jürgen Wiest, Hendrik Deusch, Stephan Reuter, Franz Gritschneder, Alexander Scheel, Manuel Stübler, Martin Bach, Patrick Hatzelmann, Cornelius Wild, and Klaus Dietmayer. Autonomous driving at ulm university: A modular, robust, and sensor-independent fusion approach. In *2015 IEEE intelligent vehicles symposium (IV)*, pages 666–673. IEEE, 2015.

[58] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6405–6416, 2017.

[59] Salar Latifi, Saurav Muralidharan, and Michael Garland. Efficient sparsely activated transformers. `https://dynn-icml2022.github.io/spapers/paper_10.pdf`.

[60] Salar Latifi, Saurav Muralidharan, and Michael Garland. Efficient sparsely activated transformers. *arXiv preprint arXiv:2208.14580*, 2022.

[61] Salar Latifi, Babak Zamirai, and Scott Mahlke. Polygraphmr: Enhancing the reliability and dependability of cnns. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 99–112. IEEE, 2020.

[62] Salar Latifi, Babak Zamirai, and Scott Mahlke. Softfusion: A low-cost approach to enhance reliability of object detection applications. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pages 344–351. IEEE, 2022.

[63] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[64] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[65] Kevin Lee, Vijay Rao, and William Christie Arnold. Accelerating facebook's infrastructure with application-specific hardware. *Facebook. Retrieved August*, 20:2020, 2019.

[66] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M Aamodt, and Vijay Janapa Reddi. Gpuwattch: enabling energy optimizations in gpgpus. *ACM SIGARCH Computer Architecture News*, 41(3):487–498, 2013.

[67] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[68] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 8. ACM, 2017.

[69] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *Proceedings of the European conference on computer vision (ECCV)*, pages 641–656, 2018.

[70] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[71] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 751–766. ACM, 2018.

[72] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[73] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[74] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

[75] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[76] Songtao Liu, Di Huang, and Yunhong Wang. Learning spatial fusion for single-shot object detection. *arXiv preprint arXiv:1911.09516*, 2019.

[77] Robert E Lyons and Wouter Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, 1962.

[78] Swetha Mandava, Szymon Migacz, and Alex Fit Florea. Pay attention when required. *arXiv preprint arXiv:2009.04534*, 2020.

[79] Saeed Masoudnia and Reza Ebrahimpour. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42(2):275–293, 2014.

[80] Seyed Mohsen Moosavi Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, number EPFL-CONF-218057, 2016.

[81] Michael Munz, Mirko Mahlisch, and Klaus Dietmayer. Generic centralized multi sensor data fusion based on probabilistic sensor and environment models for driver assistance systems. *IEEE Intelligent Transportation Systems Magazine*, 2(1):6–17, 2010.

[82] Mahdi Pakdaman Naeini, Gregory F Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *AAAI*, pages 2901–2907, 2015.

[83] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632. ACM, 2005.

[84] NVIDIA. Transformer-XL for PyTorch: NVIDIA NGC.

[85] NVIDIA. Nvidia drive agx self driving compute platform, 2011. https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/.

[86] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[87] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016.

[88] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[89] Vincenzo Piuri. Analysis of fault tolerance in artificial neural networks. *Journal of Parallel and Distributed Computing*, 61(1):18–48, 2001.

[90] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[91] Ofir Press, Noah A Smith, and Omer Levy. Improving Transformer Models by Reordering their Sublayers. *arXiv preprint arXiv:1911.03864*, 2019.

[92] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[93] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[94] Sivaramakrishnan Rajaraman, Sameer K Antani, Mahdieh Poostchi, Kamolrat Silamut, Md A Hossain, Richard J Maude, Stefan Jaeger, and George R Thoma. Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. *PeerJ*, 6:e4568, 2018.

[95] Shaan Ray. What is sensor fusion?, Sep 2020.

[96] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2017.

[97] G Rigatos and S Tzafestas. Extended kalman filtering for fuzzy modelling and multi-sensor fusion. *Mathematical and computer modelling of dynamical systems*, 13(3):251–266, 2007.

[98] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[99] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[100] JZ Sasiadek and P Hartana. Sensor data fusion using kalman filter. In *Proceedings of the Third International Conference on Information Fusion*, volume 2, pages WED5–19. IEEE, 2000.

[101] Brandon Schoettle. Sensor fusion: A comparison of sensing capabilities of human drivers and highly automated vehicles. *University of Michigan, Sustainable Worldwide Transportation, Tech. Rep. SWT-2017-12*, 2017.

[102] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[103] Yangyang Shi, Yongqiang Wang, Chunyang Wu, Ching-Feng Yeh, Julian Chan, Frank Zhang, Duc Le, and Mike Seltzer. Emformer: Efficient memory transformer based acoustic model for low latency streaming speech recognition. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6783–6787. IEEE, 2021.

[104] Patrick Y Shinzato, Denis F Wolf, and Christoph Stiller. Road terrain detection: Avoiding common obstacle detection assumptions using sensor fusion. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 687–692. IEEE, 2014.

[105] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[106] Daniel Siewiorek and Robert Swarz. *Reliable Computer Systems: Design and Evaluatuion*. Digital Press, 2017.

[107] Daniel P Siewiorek and Robert S Swarz. *The theory and practice of reliable system design*. Digital press, 1982.

[108] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[109] Jared C Smolens, Brian T Gold, Jangwoo Kim, Babak Falsafi, James C Hoe, and Andreas G Nowatzyk. Fingerprinting: bounding soft-error detection latency and bandwidth. In *ACM SIGPLAN Notices*, volume 39, pages 224–234. ACM, 2004.

[110] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[111] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[112] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[113] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, pages 303–314. ACM, 2018.

[114] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[115] Ryan Turner. A model explanation system. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*, pages 1–6. IEEE, 2016.

[116] Ultralytics. ultralytics/yolov5.

[117] Han Vanholder. Efficient inference with tensorrt, 2016.

[118] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[119] Sourabh Vora, Alex H Lang, Bassam Helou, and Oscar Beijbom. Pointpainting: Sequential fusion for 3d object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4604–4612, 2020.

[120] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. *arXiv preprint arXiv:2011.08036*, 2020.

[121] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020.

[122] Tao Wang, David J Wu, Adam Coates, and Andrew Y Ng. End-to-end text recognition with convolutional neural networks. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3304–3308. IEEE, 2012.

[123] Xiao Wang, Linhai Xu, Hongbin Sun, Jingmin Xin, and Nanning Zheng. On-road vehicle detection and tracking using mmw radar and monovision fusion. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):2075–2084, 2016.

[124] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[125] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[126] Weibin Wu, Hui Xu, Sanqiang Zhong, Michael R Lyu, and Irwin King. Deep validation: Toward detecting real-world corner cases for deep neural networks. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 125–137. IEEE, 2019.

[127] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[128] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[129] Jian-ru Xue, Di Wang, Shao-yi Du, Di-xiao Cui, Yong Huang, and Nan-ning Zheng. A vision-centered multi-sensor fusing approach to self-localization and obstacle perception for robotic cars. *Frontiers of Information Technology & Electronic Engineering*, 18(1):122–138, 2017.

[130] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *ICML*, volume 1, pages 609–616. Citeseer, 2001.

[131] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[132] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 692–693, 2020.

[133] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[134] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 9259–9266, 2019.

[135] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.