Joint inference for speech recognition models using edge devices and the cloud

Akshat Jain Harsh Rawat Niharika Tomar

December 15, 2020

1 Abstract

Despite the advancements and soaring use of deep neural networks (DNNs) for speech recognition on smart devices, uniformly sustaining high-performance inference on edge devices has been elusive due to the excessive computational demands of modern DNNs and heterogeneity in the processing capabilities of these devices. The traditional (and more popular) alternative for speech recognition has been to offload DNN processing to powerful cloud-based data center. However, this approach is heavily hindered by the significant wide-area network (WAN) latency, leading to poor real-time performance as well as low quality of user experience. In this paper, we investigate a distributed inference system that employs device-cloud computation together to deliver fast and robust inference for speech data. Our experiments show that when such a joint inference method is used along with state-of-the-art compression techniques, data transmission over the WAN can be reduced by 8 times as compared to a cloud only inference model. This has the potential to speed up speech-inferences by 1.5 to 2 times.

2 Introduction

Automatic speech recognition, or ASR, is used for speech to text conversion using machine learning models. Traditional speech recognition models rely on manually engineered stages such as those of Hidden Markov Models which are difficult to tune and upgrade. With the advancements made in Deep Neural Networks in the last decade for speech recognition [1][2], ASR's performance is on par with the humans. Applications like Siri, Amazon Alexa, and Google Assistant are very popular and they have brought ASR technologies into people's daily life with the help of edge devices such as Amazon Echo and Google Home among others. The most widely used and accurate ASR systems on edge devices perform inference on the cloud [3], due to its computational capabilities. However, some of the recent researches [4] have highlighted the benefits of performing inference on the edge devices with respect to costs and security.

These two extreme practices for speech recognition on edge devices have their own benefits and pitfalls. While device only inference for DNNs can be costly in terms of computation power requirement, cloud only inference is expensive in terms of latency, thus making it unsuitable for interactive applications, which can suffer from the dynamic connectivity conditions and the uncertain availability of the cloud.

A happy medium is possible between these two extreme endpoints wherein the model can be split into two parts and joint inference can be performed using the device as well as the cloud. Some researchers have tried to use a similar approach for video analytics [5] and geo-analytics [6]. The main idea of joint inference is that one part (head) of the model performs inference on the device and the remaining part (called tail) of the model performs inference on the cloud. One of the prominent pitfalls of such an approach would be the presence of a large number of activations that we have to send across to the cloud server. Large models such as BERT have more than 340M parameters and transmitting a large number of activations can increase network latency as well as introduce the possibility of error leading to reduced accuracy. However, applying suitable compression techniques to the intermediate activations can help reduce network latency.

In this paper, we aim to split speech recognition models and analyse the results by-

- Comparing the joint inference model with the more popular approach of cloud-only inference.
- Comparing lossy (Huffman Coding) and lossless methods (Auto-encoders) of compression in terms of latency and accuracy in joint inference.

Our initial experiments suggest that-

- Joint inference on speech recognition models can speedup inference as compared to the cloud only approach.
- Lossless compression is beneficial for activation compression in speech models whereas lossy com-

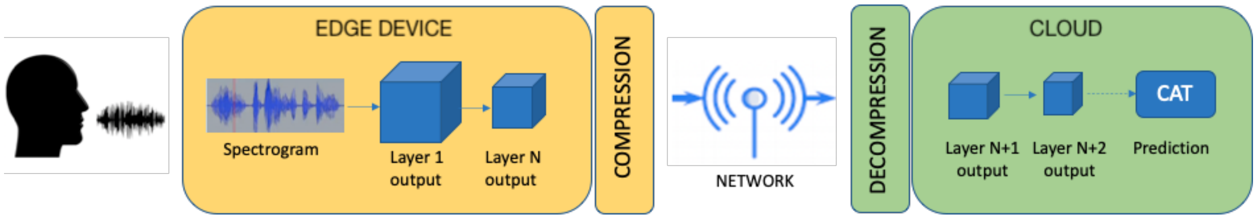


Figure 1: Joint inference using mobile device and the cloud

pression techniques can have serious impact on inference accuracy.

The rest of this paper is divided as follows. In section 3, we give the background of various approaches used for speech inference. In section 4, we provide a detailed description of our design for performing joint-inference and in section 5, we describe our implementation. In section 6 and 7, we evaluate and conclude the results and in section 8, we comment on the scope of future research in this area.

3 Background

Traditional DNN computation is either performed solely on the mobile devices or are offloaded to the cloud. Both these approaches can lead to poor performance as discussed in section 2. Recent research work has considered using a middle path between these two. We will expatiate upon these in the following sub-sections.

3.1 Background on DNNs

Deep Neural Networks (DNNs) are feed forward neural networks composed of many hidden layers. Each layer consists of neurons which takes the input, perform computations on it based on the associated weight and then produces a non-linear output after applying a non-linear activation function such as tanh or relu. The output of the hidden layers is known as ”activations”. Each layer in the DNN transforms its input data into more abstract and composite representation.
DNNs are widely used in a variety of domains and have become a critical component of computing. All major commercial speech recognition systems such as Microsoft’s Cortana, Apple’s Siri, Amazon Alexa, Google Now, etc are based on deep learning.

3.2 Device only inference

With the improvements in the hardware technology, hosting complete inference on the mobile device is a lucrative idea. By hosting the model closer to the user, we can achieve flexible execution as well as addresses the issues of privacy associated with sending the data to the cloud [7]. However, it can be challenging to execute computation intensive operations on the device due to its limited computation power. Large models such as BERT would produce inference at a slow rate which would be unacceptable for the end user. One possible solution to mitigate such challenges is to include specialized hardware on the device or optimize the run time for efficient execution of computations [8]. However, this can increase the cost of the devices. Another avenue of research in this area is to optimize the model architecture so as to achieve comparable performance using moderate amount of computations [9]. Again, this may not be feasible for all the models or the accuracy trade-off might not be acceptable for some application.

3.3 Cloud only inference

Another classical approach for inference is to offload the entire computations to the cloud. Here, we send the speech signal to the cloud data center via bandwidth limited channel such as 3G, 4G LTE or Wifi. Then the inference is performed on the cloud machine. This approach is scalable with respect to memory and computation requirements of the task. However, this requires us to provide a sustained high capacity network link for low latency and equivalent accuracy. This can be a bottleneck in areas with poor network connectivity. One possible solution for decreasing the network usage would be to compress the input data before transmitting it over the network. However, compression can lead to loss of finer features of the input leading to decreased accuracy. Additionally, there would be compute overhead associated with the compression and decompression of the transmitted signal.

3.4 Joint inference using device and the cloud

This is one of the run-time optimization method which is a middle path between the device only and the cloud only approach [Figure 1]. [5] investigates a research agenda that involves opening up the black box of neural networks for object detection in edge devices (specifically cameras) and describe new application scenarios that benefit from the joint inference between the edge devices and the cloud. In their proposed technique of split-brain inference, the DNN is being split into two parts, where the edge device is responsible for evaluating the first N layers of the DNN. This then produces result activations that are compressed using compression techniques and sent over to the cloud. Upon arrival, the activation decompression takes place and then resumes the DNN evaluation. They bring the compute communication trade-off into an interesting region where the data communicated after splitting the DNN is lower than the cloud-only baseline (with compressed images) and the computation needed on the edge device (including compression costs) is less than running the entire DNN.

4 Design

4.1 Speech Recognition Models

Speech Recognition is an inter-disciplinary sub-field of computer science and computational linguistics. The first attempt at end to end ASR was with Connectionist Temporal Classification (CTC) based systems introduced by Google DeepMind in 2014 [10]. The system was then expanded by Baidu on extremely large datasets [2]. In this project, we are using a slightly modified version of the DeepSpeech 2 architecture [2].
The performance of a speech recognition model is defined in terms of its accuracy. The two aspects of accuracy are Character Error Rate (CER) and Word Error Rate (WER). CER measures the error of the characters between the ground truth and the model's output. Similarly, WER takes the transcription produced by the model and measures the error with respect to true transcription.

$$WER^{1} = \frac{S + D + I}{N}$$

where -
S is the number of substitutions
D is the number of deletions
I is the number of insertions, and
N is the number of words in the reference

4.2 Split mechanism

The split point in the DNN model is of particular interest as the activations to be sent across the network are obtained from the output at the given split layer. Additionally, it needs to be ensured that sufficient computation is performed on the device to capture the finer features of the data. We call the first part of the model which needs to be executed on the device as "head" of the model. The remaining layers needed to be executed on the cloud for inference are called "tail" of the model.
Selecting this split point dynamically is an active research area. [11] introduces a framework called Edgent which can generate an execution plan in both static and dynamic environments. Edgent tunes two design knobs, specifically - (1) The partitioning of the DNN model taking into account the device resources and the network latency (2) DNN right sizing to decrease the compute latency by early exiting inference at an appropriate DNN intermediate layer. By tuning these knobs, Edgent can provide on-demand low latency inference.
Other approach is to manually select the split point based on the resulting activation size. We have used the manual approach of finding the split point (described in section 5.3) and dynamic selection remains in the future scope of the project.

4.3 Compression schemes

Sending data over to the cloud from the edge devices incurs substantial network cost. If the data is huge (which in our case, it is), it becomes a network bottleneck. In order to avoid this issue, compression algorithms are used to reduce the data size by a significant margin. Compressed data leads to reduced network latency as well as bandwidth. In this paper, we use entropy coding algorithms like Huffman and Arithmetic encoding (lossless scheme) as well as DNN Autoencoders (lossy scheme) for compression of intermediate activations.

4.3.1 Huffman Encoding

Traditionally Huffman encoding has been used as a compression scheme for large data sets because of its lossless nature. The optimal prefix code helps to reduce the size of the data by exponential factors. Leveraging this property of this entropy compression-coding, we have decided to use this algorithm for our model. This algorithm pairs every character/symbol with a code in such a way that a code's length is dependent on the relative frequency (weight) of the corresponding character. It is generally visualized as a binary tree where the leaves store the encoded characters. This algorithm has been popularly used in GZIP, JPEG/PNG, MP3 for text, image and audio/video compression respectively. For the purpose of this project, the python library dahuffman [2] was used. Section 5.4 elaborates more on the usage of Huffman encoding specific to our use case.

[1]Source: Wikipedia

[2]https://pypi.org/project/dahuffman/

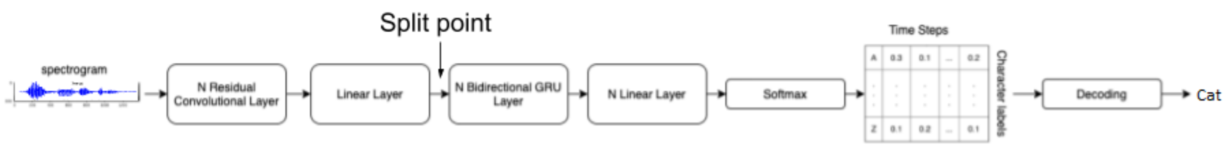


Figure 2: Speech recognition model and the selected split point

4.3.2 Arithmetic Encoding

Like Huffman encoding, Arithmetic encoding has also been used as a compression scheme for large data sets because of its lossless nature. Being a lossless entropy scheme, it represents a string using a fixed number of bits per character (like ASCII). The characters that are more frequently used are stored with fewer bits while the others are stored with more bits, which then results in fewer bits used in total. Unlike Huffman encoding, Arithmetic algorithm encodes the entire input into a single number as opposed to representing and replacing each component symbol of the input with a code. This entropy encoding is used in cryptography and data compression just like Huffman.
In section 5.4.1 we have analysed the results of Huffman and Arithmetic encoding upon DNN's intermediate activations.

4.3.3 Auto-encoders

An Autoencoder is a neural network based on unsupervised learning technique that helps with data compression. Unlike Huffman encoding scheme, Autoencoders tend to be lossy, however they pose the benefits of being adaptable and thus, custom fits a data model unlike lossless compression schemes that are standard. Since Autoencoders reduce the data dimensions by learning how to ignore the noise in the dataset, they are a good candidate for being used as a compression/decompression scheme. They are more commonly used in anomaly detection and image denoising- both of which work in the favor of this research- especially during decompression. However, sometimes this denoising might actually recreate a wrong output like mistaking the number 2 for 7 and this results in the loss of accuracy on the model's behalf. This is why they are termed lossy. Autoencoders take lower execution time than Huffman or Arithmetic compression and hence stand a good chance at competing with the lossless compression schemes at the expense of accuracy.
In our project, we compress the intermediate activations using autoencoders and compare the latency and accuracy trade-off with the cloud only approach. This is discussed in section 5.4.2 of this paper.

5 Implementation

5.1 Setup

Our setup has two nodes hosted on the Wisconsin cluster in CloudLab machines. Both the machines have 8 CPU cores with 4 gigabytes of memory. All the results in section 6 have been calculated on these machines, one of which mocks an edge device while other is equivalent of a cloud node.

5.2 Speech recognition model

Our model architecture was based on the Deep Speech 2 [2]. The model architecture[3] has two major neural network modules - N layers of Residual Convolutional Neural Networks and N layers of Bidirectional Recurrent Neural Networks. The ResCNN layers are used to learn the relevant audio features and the Bidirectional RNN layers are used to utilize these learned features and correlate them to create an appropriate context. The fully connected layers are used to classify characters per time step.
For training purposes, CloudLab machines were used (as described above) and our model was trained on the LibriSpeech dataset.

5.3 Splitting

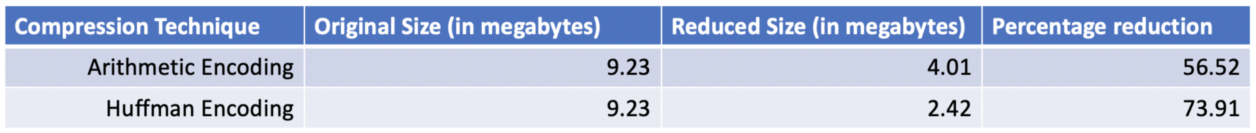
The model in this research was split manually into two parts. The optimal split point was decided to be set at the point after CNN + fully connected layers, because CNN layers are good at extracting abstract features and therefore our head model can capture the nuances of the data. The tail part of the model includes the bidirectional RNNs which can capture the context of both before and after frames.

5.4 Compression

5.4.1 Lossless compression

Huffman and Arithmetic encoding are the two most widely used lossless encoding techniques. For compress-

[3]https://www.assemblyai.com/blog/end-to-end-speech-recognition-pytorch



Compression Technique	Original Size (in megabytes)	Reduced Size (in megabytes)	Percentage reduction
Arithmetic Encoding	9.23	4.01	56.52
Huffman Encoding	9.23	2.42	73.91

Figure 3: Results of performing compression using lossy techniques on a 600*600*1 tensor.

ing our intermediate activations, we conducted an preliminary experiment to observe the effect of both of these techniques. First, we compressed intermediate activations of our DNN model using the 'dahuffman' module in python for huffman encoding. This reduced the size of our intermediate tensor by 73.91 percent (Figure 3). Next, we repeated the same experiment using Arithmetic encoding technique, specifically using arcode module in python. The compression using this technique was 56.52 percent.

5.4.2 Lossy compression

DNN Autoencoders were used for lossy encoding in the project. The autoencoder had four fully connected layers with the initial two layers used for encoding the activations and the remaining two for decoding the activations. The autoencoder was also split and deployed on two differnet nodes with the encoder placed alongside head model and decoder with the tail model.
The training procedure for the autoencoder included taking the activations at the split point as the input to the encoder. We used MSE loss between the input activations and the output of the decoder. The training was completed on the training set of the LibriSpeech dataset.

5.5 Communication

For sending our compressed data from the edge device to the cloud, we had to decide on a network transmission protocol. TCP, UDP and HTTP are the most popular ones which also have numerous libraries in python for implementation. We discarded the UDP approach in order to prevent the loss incurred due to lost packets, which narrowed our choice down to TCP or HTTP. TCP (Transmission Control Protocol) helps with data streaming by splitting the information into chunks along with a host of information about the chunk being sent/received. Similar to this, HTTP (HyperText Transfer Protocol) uses additional information to read and process the data at hand - once it arrives. Since our model doesn't require specific information about the data sent and received (and how to process it) and just an acknowledgment for both is enough, we decided to use the TCP protocol over the HTTP. For the purpose of this research, python's socket module[4] and Berkeley's socket interface API[5] were used. Also, we used python's pickle library to convert the activation tensors into bytes, before sending them over the network.[6]

5.6 Inference

In order to evaluate our model, we stored the labels for the entire data set on both the machines. After activations were decompressed, the tensors were reconstructed back into their original shape (using python's pickle library) and the inference was computed.

6 Results and Evaluation

6.1 Latency Comparison

In our first experiment, we compared the cloud-only inference method to our joint inference setting. In order to make a fair comparison, both the models were based on the same DNN, were trained on the same data set, and tested against the same data points (speech inputs). In the cloud-only setting, the voice signal which is received on the edge device, is sent directly over to the cloud machine for inference.

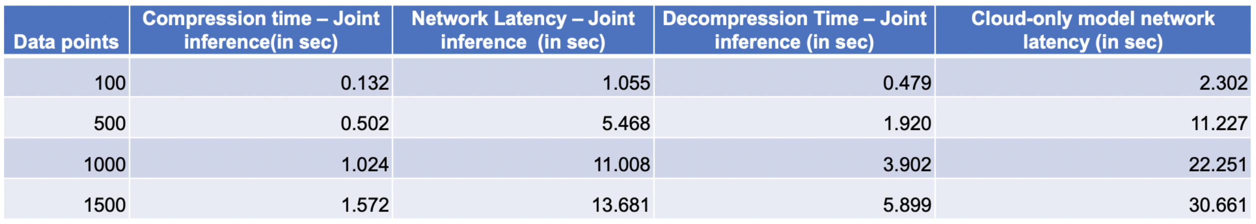
As evident from the table in figure 4, the inference time for 100 data points using 'joint-inference' method includes the time for compression, network latency and decompression. All this combined is still lower than the time taken to send the voice input across devices alone. This gap between the inference time only widens as the number of data points increase. This suggests that joint-inference can provide incredible results for speech recognition.

However, these results are not sufficient to conclude that joint-inference model always performs better. Applying modern voice compression techniques [12] [13] on the speech input is expected to produce low latency results with some accuracy trade-offs.

[4]https://docs.python.org/3/library/socket.html

[5]http://networkprogrammingnotes.blogspot.com/p/berkeley-sockets.html

[6]https://docs.python.org/3/library/pickle.html



Data points	Compression time – Joint inference(in sec)	Network Latency – Joint inference (in sec)	Decompression Time – Joint inference (in sec)	Cloud-only model network latency (in sec)
100	0.132	1.055	0.479	2.302
500	0.502	5.468	1.920	11.227
1000	1.024	11.008	3.902	22.251
1500	1.572	13.681	5.899	30.661

Figure 4: Comparing the performance of cloud-only inference with joint-inference. For joint-inference, the compression was done using auto-encoders and decoders, cloud-only models send the voice input as byte stream spectrograms

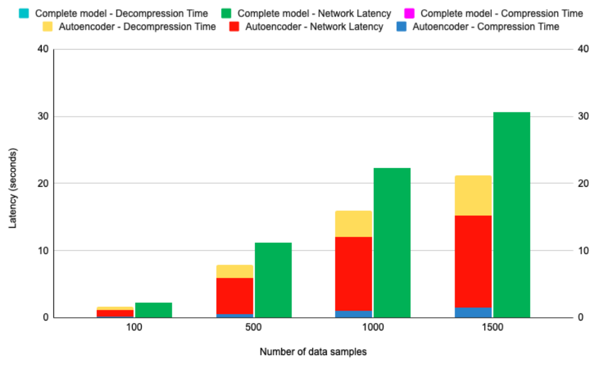


Figure 5: Latency comparison, from figure 4. The left bar for each comparison represents the joint inference model and the right bar represents the cloud-only model.

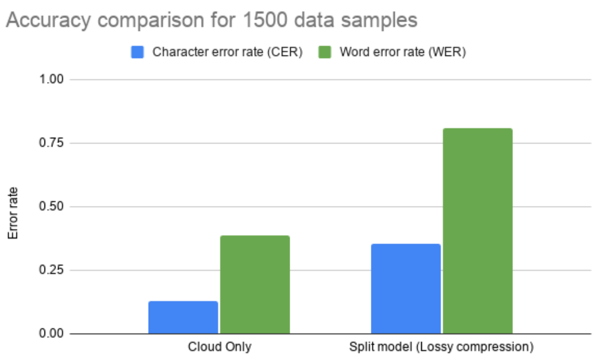


Figure 6: Accuracy comparison.

6.2 Accuracy comparison

One important observation to consider is that errors in the output produced by a speech recognition model are more pronounced as compared to vision models. Consider the case where we have a vision model which misconstrued a few pixels. The result would be tolerable as the overall meaning of the image would still be intact. On the other hand, consider that a speech recognition model misinterpreted ”c” of ”cat” as ”b”. The resulting word becomes ”bat” which has a completely different meaning as compared to the original word.
As evident from the figure 6, the error rate increases when we use Autoencoders for activation compression. This validates our hypothesis that changes in activation upon decompressing leads to significant changes in the final predictions. Moreover, increase in the character error rate is less than the corresponding increase in the word error rate.
This prompts an interesting conclusion that perhaps using lossy compression in split speech models is not a viable option. It remains to be investigated if this conclusion holds true in case of vision models which may be better suited for such a trade-off between accuracy and compression ratio.

7 Conclusion

In this paper, we observe that by splitting the speech recognition model over the edge device and the cloud, we can speedup inference as compared to the cloud only approach. We also observe that lossy compression mechanisms (Autoencoders) lead to loss of accuracy in the inference and therefore, should not be recommended for accuracy sensitive applications.

8 Future Work

We will now outline a few directions in which research can be taken forward for the joint inference models described in this paper.

- **Using model compression alongside split model approach**: In our experiments, we deployed the complete model on both the device node and the cloud node. However, inference using entire model may not be feasible when the model parameters are large. Therefore, model compression techniques may be explored alongside split model approach. There would be a trade-off between accuracy and the amount of data transmitted. Especially for large speech models if lossy compression is unsuitable, we can possibly use a compressed model and use lossless compression schemes in the split model.

- **Pipelining the tail model computations in a large cluster**: Another possible research avenue would be to run the inference on tail model in a distributed setting. Similar to the approach used in Pipedream[14], we can pipeline the computation for the tail model. This can prove to be effective considering that there may be a possibility of overlap between network transmission of compressed activations, decompression computation of a batch, and tail inference using these activations.

- **Integration of split model approach with online prediction serving systems such as Clipper[15]**: One of the major benefit of such an integration would be the batching of input data. If the activations of a number of inference requests can be batched then the amount of network latency can be reduced as compared to the computation speedup of running the tail model on a cloud cluster.

- **Dynamic selection of split point of the model**: As discussed in section 4.2, dynamic selection of split point can significantly improve the performance of the system. It can optimize the activation size to be sent across the network. Moreover, an optimal split point can help us learn the finer features efficiently at the head model.

References

[1] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition, 2014.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Vaino Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 173–182. JMLR.org, 2016.

[3] Xinyu Lei, Guan-Hua Tu, Alex X. Liu, Kamran Ali, Chi-Yu Li, and Tian Xie. The insecurity of home digital voice assistants – amazon alexa as a case study, 2019.

[4] R. Peinl, B. Rizk, and R. Szabad. Open source speech recognition on edge devices. In *2020 10th International Conference on Advanced Computer Information Technologies (ACIT)*, pages 441–445, 2020.

[5] John Emmons, Sadjad Fouladi, Ganesh Ananthanarayanan, Shivaram Venkataraman, Silvio Savarese, and Keith Winstein. Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary. New York, NY, USA, 2019. Association for Computing Machinery.

[6] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Konstantinos Karanasos, Jitendra Padhye, and George Varghese. Wanalytics: Geo-distributed analytics for a data intensive world. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, page 1087–1092, New York, NY, USA, 2015. Association for Computing Machinery.

[7] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

[8] Youngsok Kim, Joonsung Kim, Dongju Chae, Daehyun Kim, and Jangwoo Kim. layer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.

[9] Qingqing Cao, Niranjan Balasubramanian, and Aruna Balasubramanian. Mobirnn: Efficient recurrent neural network execution on mobile gpu, 2017.

[10] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In Eric P. Xing and Tony Jebara, editors,

Proceedings of the 31st International Conference on Machine Learning, volume 32 of *Proceedings of Machine Learning Research*, pages 1764–1772, Bejing, China, 22–24 Jun 2014. PMLR.

[11] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge ai: On-demand accelerating deep neural network inference via edge computing, 2019.

[12] Davis Yen Pan. Digital audio compression. *Digital Technical Journal*, 5(2):28–40, 1993.

[13] A. Gersho. Advances in speech and audio compression. *Proceedings of the IEEE*, 82(6):900–918, 1994.

[14] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, and Phillip B. Gibbons. Pipedream: Fast and efficient pipeline parallel DNN training. *CoRR*, abs/1806.03377, 2018.

[15] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, Boston, MA, March 2017. USENIX Association.

[16] J. Shao and J. Zhang. Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2020.