

WebInf: Accelerating WebGPU-based In-browser DNN Inference via Adaptive Model Partitioning

Bing Dong[†], Tianen Liu[†], Borui Li[†], Xiaolei Zhou[‡], Shuai Wang^{†*}, Zhao-Dong Xu[§]

[†]School of Computer Science and Engineering, Southeast University

[‡]The Sixty-Third Research Institute, National University of Defense Technology

[§]School of Civil Engineering, Southeast University

Email: {dongbing, toliutianen, libr, shuaiwang, xuzhdgqy}@seu.edu.cn, zhouxiaolei@nudt.edu.cn

Abstract—Artificial intelligence (AI) model inference performance in browsers is constrained, and transmitting data to the server consumes substantial transfer time by cloud computing. In this paper, we investigate the status quo of cloud and browser processing and explore model computation partitioning methods. Our study is rooted in WebGPU and employs the TensorFlow.js framework, encompassing seven AI models spanning computer vision, natural language processing, and automatic speech recognition domains. Leveraging the characteristics of neural network layers, we find a significant performance boost through a method that partitions AI models at layer granularity. We design a system called WebInf to partition AI models at layer granularity between the browser and server for faster inferencing-based adaptive model partitioning. WebInf supports diverse hardware, wireless networks, neural network structures, servers, and adaptive partitioning models for optimal inference performance. We evaluate WebInf on two laptops and servers, demonstrating that WebInf yields inference time improvements of 30% and 52%, respectively, when compared to separate inference execution in servers and browsers. The improvements can even peak at 33% and 69% respectively.

Keywords—Model partitioning, WebGPU, Neural network, Edge device

1. INTRODUCTION

Currently, the AI models that have been deployed on the web are increasingly becoming more convenient, including intelligent search [1], recommendation system [2], automatic translation [3], sentiment analysis [4], and virtual assistant [5]. It allows the same AI model to perform inference seamlessly across various browser platforms, regardless of the hardware configurations (personal computers, smartphones, and wearables) and operating systems (Windows, Mac, iOS, and Android) [16]. Modern browsers incorporate hardware acceleration features, like WebGL [6] and WebGPU [7], to optimize the performance and efficiency of AI models. However, deploying AI models in browsers is still a challenge due to the limited computing resources and memory capacity of browsers [17]. Additionally, the high latency of requests involved in uploading and downloading data from browsers to servers further complicates the process [34]. This limitation is particularly noticeable with latency-sensitive models such as object detection [8], which are unable to be effectively deployed in a browser environment.

*Corresponding author.

To solve the above challenge, researchers introduce various methods to enhance inference performance and reduce communication costs on servers. Serverless computing improves the inference efficiency of AI models in the server [9], [10]. Elastic algorithms [10] automatically adjust model training and inference based on different resource configurations and workload requirements. AI models deployed exclusively in the server have fast inference speeds, but they suffer from significant network communication latency during user requests, and they also raise potential privacy concerns. On the other hand, some researchers choose to deploy AI models on the browser to reduce communication costs [11], [12]. For instance, WebDNN [11] aims to swiftly execute deep neural networks within the browser and WebGL technology for edge machine learning inference [12]. While this approach reduces network communication latency during requests, the resource-constrained browser environment may diminish model inference efficiency. We are exploring approaches that strike a balance between network communication latency and model inference latency.

In response to the shortcomings of the aforementioned work, we consider dividing AI models into two parts, with one part executing on the browser and the other on the server to balance inference efficiency and communication latency. However, achieving the trade-off of efficiency and latency of different models is challenging. Firstly, how do we study the characteristics of AI models? We analyze the output data size and inference time of each layer of the AI models. Secondly, how can dynamic model partitioning be implemented? We take into account the model's structure, data size, variations in local workload levels, and fluctuations in wireless network bandwidth to decide the optimal partitioning point. Based on the considerations outlined above, we propose an adaptive model partitioning system called **WebInf**, comprising browser and server components. It is capable of autonomously identifying the optimal partition point within neural networks. The specific contributions of this paper are as follows:

- We propose a method for partitioning computations within neural networks based on WebGPU. The approach is easier to extend to multiple models, heterogeneous platforms and devices.
- We design **WebInf**, a system for adaptive offloading

AI model computations between browser and server in layer-level granularity by leveraging the computing capabilities of both, simultaneously trade-off communication latency.

- Comparing independent cloud-only and browser-only processing, WebInf improves model inference latency and achieves better performance and accuracy than the state-of-the-art methods.

The remainder of this paper is as follows. In Sec. 2, we introduce the current state of model inference on browsers. Sec. 3 analyzes the characteristics of AI models partitioning at layer granularity and gives key observations. Sec. 4 shows the system of WebInf. Experimental evaluation is presented in Sec. 5. Sec. 6 summarizes the related work of this paper and Sec. 7 concludes this paper.

2. STATE OF MODEL INFERENCE ON BROWSER

Cloud providers currently rely on performing all neural network model processing in the server to enable intelligent applications. However, this approach incurs a significant overhead due to the high cost associated with transmitting data over wireless networks. An alternative approach is to perform neural network model inference on the browser. However, this method is not optimal due to the limited computational resources and memory available in the browser.

We evaluate the performance of model inferencing in server and browser. Specifically, we use MobileNetV1 [13] as our application model and deploy a convolutional neural network for image classification. We use a laptop with an integrated graphics card as the edge device (GPU: Inter(R) UHD Graphics 730) and a discrete graphics card-equipped machine as the server (GPU: NVIDIA RTX 3060).

In Fig. 1, we conducted a test using a 192kb image to measure the end-to-end latency (communication latency and computation latency) of MobileNetV1 under different inference modes. The end-to-end latency of 4G is estimated to be approximately 131ms from browser to server, while it is about 39ms for WiFi. Although the computation latency (approximately 5ms) is lower in the server, the transmission latency is much larger than the computation latency. The abscissae “WebGPU” and “WebGL” express the computation latency in the browser, about 38ms and 47ms, respectively. Compared to computing on the server, browser-only computing saves transmission costs, but sacrifices computing costs.

In a word, cloud computing offers significant computational power, but data transmission time can impact performance, particularly with WiFi. The model inferencing on the browser eliminates the communication cost but increases the computation latency. Therefore, we consider combining the advantages of browser and server to balance communication latency and computing latency.

3. COMPUTATION PARTITIONING OF AI MODELS

The findings of Sec. 2 provide a rational basis for the division of AI models between the browser and the server. We

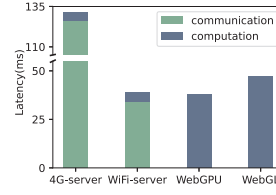


Fig. 1: The comparison of model inferencing based on different configurations.

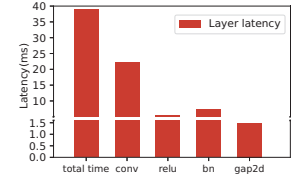


Fig. 2: The analysis of pre-layer latency based on categories.

analyze the size of output data and the latency of execution at layer-level granularity.

3.1 The characteristic of pre-layer in MobilenetV1

We begin our investigation by analyzing the data and computational characteristics of each layer in MobileNetV1, aiming to gain insights into the potential benefits of partitioning neural network model computations between the browser and server domains.

Per-layer latency. The red bars in Fig. 3 display the execution time of each layer on the WebGPU, arranged from left to right in their sequential execution order. Convolution layers *conv* (including pointwise convolution *pw* and depthwise convolution *dw*) emerge as the most time-intensive, accounting for over 80% (Fig. 2) of the overall execution time. Notably, despite lower computational complexity, pointwise convolution *pw* layers often exhibit higher latency than regular convolution layers due to the intricate memory access patterns involved. Batchnormalization *bn* layers and global average pooling *gap2d* layers demonstrate faster execution times.

Data size variation. The markers on the lines in Fig. 3 indicate the size of the output data for each layer, with the output of the preceding layer serving as the input for the subsequent layer. Early-stage convolution *conv* (including pointwise convolutions *pw*) yields a substantial volume of output data, represented by the tallest bars in Fig. 3. Because early-stage convolution uses hundreds of convolution kernels on input feature maps to extract informative features. Output data size remains constant after batch normalization and activation layers *relu6*. However, the application of depthwise convolution *dw* significantly reduces data size, resulting in reductions of up to 6×. Finally, the output layer *dense* reduces the data to a single classification label.

Key observations. We conclude the following observations with the above experimentation based on per-layer latency and data size variation.

- Each layer possesses distinct computational and data profiles based on its type and position within the model.
- The data size in later layers is significantly smaller than that of the input layer. Notably, higher computation times are generally observed in the later layers of the model.
- Front-end data volume tends to decrease, while back-end latency per layer tends to be higher, indicating promising opportunities for computational partitioning between the browser and the server in neural network models.

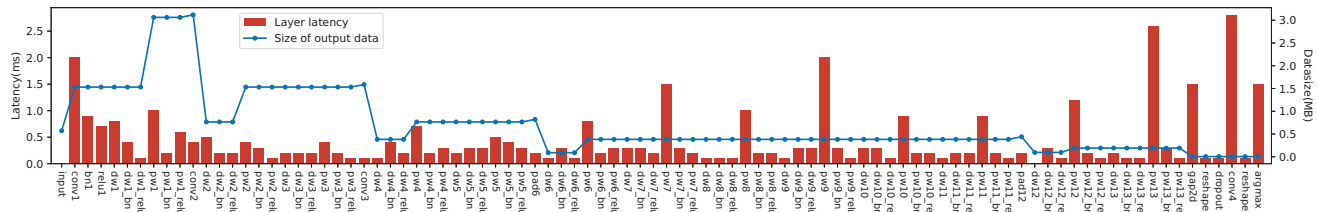


Fig. 3: The per-layer inference time (red bar) and size of output data (blue marker) after inferencing each layer in MobilenetV1.

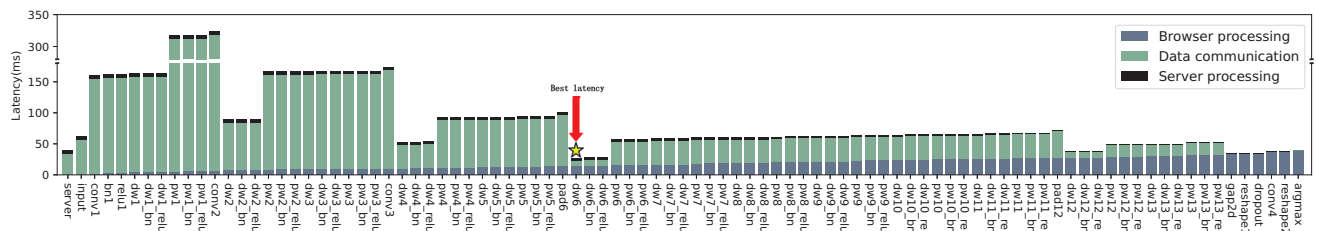


Fig. 4: The end-to-end latency of MobilenetV1 when choosing different partition points. The partition points for best latency are each marked by $\star$.

3.2 The selection of layer-granularity partitioning

We investigate the partitioning of each layer in MobilenetV1 between the browser and the server by employing WiFi as the wireless network configuration. More importantly, we analyze the end-to-end delay of MobileNetV1 through different partition points and give key observations.

Partition points. Each bar of Fig. 4 represents the end-to-end latency of MobileNetV1 when choosing different partition points. The layer before the partition point is processed on the browser, and the layer after the partition point is offloaded to the server for processing by WiFi transmission. The leftmost bar represents cloud-only processing on the server (i.e., partitioned from the beginning). As the partitioning point moves from left to right, more layers are processed on the browser. The rightmost bar represents execution browser-only (i.e., partitioned at the end).

Partition for latency. We observe that if partitioning occurs at the front end, data transmission becomes the dominant factor in the end-to-end latency. Although no data transmission is required in the back end, the poor inferencing ability of the browser is the dominant factor in the delay. The partitioning point somewhere in the middle provides better performance by minimizing data transmission overhead and utilizing the powerful server for computationally intensive layers. As is shown in Fig. 4, the partitioning point *dw6* achieves the lowest latency when combining the computing capacity of the browser and server and utilizing WiFi as a network transmission. We will process the layer before partitioning point *dw6* on the browser and send the inference result to the server by WiFi. The server continues to infer subsequent layers based on the uploaded results. Compared to independent cloud-only processing and browser-only processing, partitioning point-based processing improves 31% and 30% latency respectively.

Key observations. We conclude the following observations

TABLE 1: Benchmark specifications

APP	Network	Input	Layers
NLP	Spam detection	Word vectors	6
ASR	Speech commandsV1	Speech commands	23

by the above experimentation based on different partition points and inferencing latency.

- Partitioning at the layer granularity offers significant improvements in latency.
- For MobileNetV1 with WiFi transmission, the optimal partitioning point lies within the middle layers of the model.

3.3 Generalizing to more AI models

We extend our research to two intelligent applications Natural Language Processing (NLP) and Automatic Speech Recognition (ASR) to investigate their data and computational characteristics, as well as their impact on the computation partitioning points. Detailed information about the benchmarks is provided in Table 1. We partition each model from the first non-input layer to the final layer and employ a wireless network configuration.

NLP applications. Layer decomposition is depicted in Figures 5(a). In comparison to Computer Vision (CV) models, NLP models commonly incorporate inputs with temporal dependencies, which typically entail embedding layers and recurrent neural network (RNN) layers. These layers are computationally demanding, and due to the shorter input sequences, there are prospects for partitioning near the initial stages of the model. Fig. 6(b) shows the optimal partition point.

ASR applications. Layer Decomposition is illustrated in Fig. 5(b). The model incorporates gated recurrent unit (GRU) [35] layers and long short-term memory (LSTM) [36] layers. These two layer types are computationally intensive.

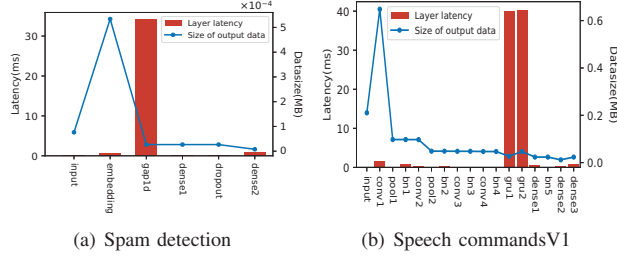


Fig. 5: The per layer latency on the WebGPU and size of data after each layer's execution.

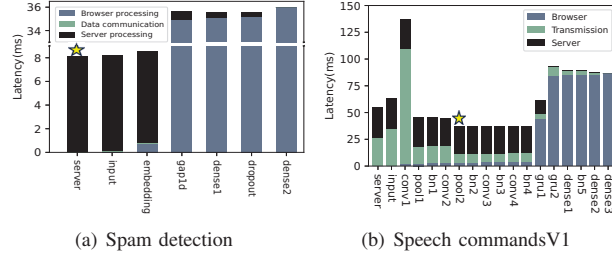


Fig. 6: The end-to-end latency when choosing different partition points.

Additionally, due to the shorter input sequences, there are opportunities for partitioning near the beginning of the model. Fig. 6(b) shows the optimal partition point.

Key observations. We conclude the above experimentation based on more AI models.

- In CV applications with convolutional layers, convolutions tend to increase the data size, while global average pooling layers significantly reduce it.
- Only models with similar-sized fully connected and activation layers exhibit minimal variation in layer latency and data size.
- The optimal partition point of a neural network model depends on its topology and the composition of its layers. CV models tend to benefit from partitioning around the middle layers, whereas for ASR and NLP models, partitioning closer to the input is more advantageous. The pronounced variability of the optimal partition point suggests the need for a system that dynamically allocates AI models between the browser and the server based on their neural network configuration.

4. WEBINF DESIGN

The optimal partitioning point of the model depends on the neural network's topology, which is reflected in the variations of computation and data size at each layer. Additionally, even for the same neural network structure, dynamic factors such as the state of wireless networks can influence the optimal partitioning point. Due to these dynamic factors, an automated system is required to intelligently select the optimal point for model partitioning to optimize end-to-end latency or energy

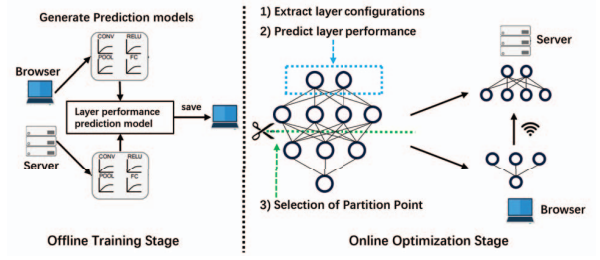


Fig. 7: WebInf overview.

consumption on mobile devices. To fulfill this requirement, we have designed an intelligent model partitioning engine, WebInf. WebInf consists of an offline phase and an online optimization system, aimed at managing the execution of partitioning in intelligent applications. Fig. 7 illustrates the design of WebInf.

Offline training stage. WebInf configures the browser and servers to generate layer performance prediction models. WebInf needs to be executed only once for a given set of browsers and server platforms, without the need to analyze each model individually. This set of prediction models is stored in the browser and subsequently utilized to predict the latency of each layer in the target model.

Online optimization stage. During runtime, while executing the model on the browser, WebInf dynamically determines the optimal partitioning point for the model. As shown in Fig. 7, the steps are as follows: 1) WebInf analyzes the layer types and configurations of the model's neural network. 2) WebInf uses stored layer performance prediction models to estimate the latency of executing each layer on both browser and server platforms. 3) WebInf combines the current wireless connection bandwidth to select the optimal partition point, optimizing the best end-to-end latency. 4) WebInf performs model execution, partitioning computation between browser and server.

4.1 Performance prediction model

The execution time varies across different layers. We observe that, for each layer type, there are significant latency variations based on different layer configuration parameters. We modify the parameters of the same layer and measure its execution time. We use regression functions to model both browsers and servers. In comparison to Neurosurgeon [33], we have increased the variety of layers and the amount of input data in our regression model, leading to improved prediction accuracy.

Convolution layer. GFLOPs (the computational power of models), the number of channels of the input feature map, and $(convolution\ kernel\ size/stepsize)^2 * (number\ of\ convolution\ kernels)$ are used as input variables.

Fully connected layer. This layer is usually used as the last layer of the neural network to map the feature maps output from the previous convolution or pooling layers to the classification or regression output. the size of input and

output feature maps, as well as the number of input and output neurons, are considered independent variables for the regression model. The handling of the **argmax** and **softmax** layer is similar.

Pooling layer. Pooling type and the size of the input and output feature maps are used as input variables.

Activation layer. There is a one-to-one mapping between the input and output, using the activation function type, the number of output neurons, and the size of feature maps as input variables. The handling of the **batchnormalization** layer is similar.

Embedding layer. The longer the length of the input data, the more computationally intensive the Embedding layer is. In this paper, the length of the input data, the Embedding dimension, and the vocabulary size of the Embedding layer are used as input variables.

RNN layer. The RNN Layer offers the flexibility to employ diverse types of recurrent cell architectures, including simple recurrent neural network (SimpleRNN) [37], LSTM, and GRU. These various cell types exhibit distinct parameter configurations, consequently leading to varying computational loads. Notably, the LSTM layer incurs a higher computational burden compared to the SimpleRNN layer. In this study, the input shape, dimensions of the hidden state, and the specific recurrent kernel type are adopted as input variables.

We use this approach to implement a performance prediction model for each layer that can generate a platform for analyzing the execution time of each layer of the network for a set of devices (browser WebGPU and server), then the execution time of each layer can be predicted without running the model. It is also shown that new models can be supported as well as different kinds of model inputs.

4.2 Dynamic AI model partitioning

Utilizing the layer performance prediction models, WebInf dynamically selects the best AI model partition points, as described in Algorithm 1.

Analysis of the model. In lines 10-11 of the algorithm, WebInf obtains the model's constituent layers L_i and utilizes the regressions model to estimate the execution time of each layer on browser WebGPU TB_i and on the server TS_i . Line 12 calculates the wireless data transfer latency TD_i based on the latest wireless network bandwidth.

Partition point selection. Line 13 calculates the total time to send the raw data to the server for inference. In Lines 14-15, WebInf predicts the total inference time for the model based on the partition after each layer, and also calculates the minimum time, and uses it as the best possible partition location. The two times are compared and the shorter time is chosen as the best cut position. Because of the simplicity of the regression models, the WebInf scheduling algorithm is lightweight and efficient.

4.3 WebGPU-based collaborative execution

We built a prototype of WebInf by creating tensorflow.js (TFJS) instances as the foundational infrastructure for both

Algorithm 1 WebInf partitioning algorithm

```

1: Input:
2: Data: Original Input Data
3:  $L_{i,i=0,1,\dots,L-1}$ : Number of layers in the model
4:  $D_{i,i=0,1,\dots,L-1}$ : Data size at each layer
5:  $f(L_i)$ : Performance prediction models predicting
   the latency of executing  $L_i$ 
6:  $W$ : Latest network transmission rate
7:  $K$ : Latest local load level
8: Procedure:
9:   for  $i$  in  $0 \dots L-1$  do
10:     $TB_i \leftarrow f_{\text{browser}}(L_i, K)$ 
11:     $TS_i \leftarrow f_{\text{server}}(L_i)$ 
12:     $TD_i \leftarrow D_i/W$ 
13:     $t_{\text{server}} = \sum_{i=0}^{L-1} TS_i + \text{Data}/W$ 
14:     $t_{\text{partition}} = \min_{t=0 \dots L-1} (\sum_{i=0}^j TB_i + \sum_{k=j+1}^{L-1} TS_k + TD_i)$ 
15:     $\text{index} = \arg \min_{t=0 \dots L-1} (\sum_{i=0}^j TB_i + \sum_{k=j+1}^{L-1} TS_k + TD_i)$ 
16:    if  $t_{\text{server}} \leq t_{\text{partition}}$  then
17:      Cloud processing: return -1
18:    else
19:      Collaborative processing: return index

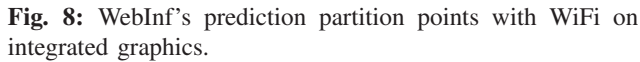
```

the browser and server-side components. We implemented a client-server interface using Websockets, a transmission control protocol (TCP) network protocol that enables bidirectional communication between the browser and server. To enable dynamic selection of partition points, the AI model has been loaded on both the browser and server. When the network transmission speed and local load change, lines 10 and 12 of the algorithm will adjust the location of the model partition point according to the latest network transmission speed and load level. Components in the browser and server select partition points at runtime. This approach allows for flexibility in adapting the optimal partitioning strategy based on specific requirements and constraints of the execution environment. The partition location is determined by the browser, which sends the output of the partition location from the browser to the server. Subsequently, the server executes the remaining layers. The final result is then sent back from the server to the browser. It's worth noting that the model precisely includes one partition point, which spans from the browser to the server.

5. EVALUATION

We evaluate WebInf using WiFi, and 4G wireless connections with two GPU backends (Integrated graphics: Inter(R) UHD Graphics 730, Discrete graphics: NVIDIA GTX 1650ti). We demonstrate that WebInf achieves significant end-to-end latency improvements compared to approaches

APP	Network	Input	Layers
CV	MobilenetV2	Image	156
	ResNet50	Image	177
NLP	Trivial expert	Word vectors	5
ASR	Speech commandsV2	Speech commands	17



5.1 WebInf's prediction partition points with WiFi on integrated graphics.

In Fig. 9, the transmission latency dominates the end-to-end delay of the CV model, and it is orders of magnitude larger than the layer latency. We observed that although the latency at the partition point is not the smallest compared to the average latencies before and after, the data output at the partition point is very small. This finding aligns with the WebInf prediction in Fig. 8, where the transmission time at the optimal partition point for the CV model is relatively small compared to other partition points. In Fig. 9, for the NLP



and ASR models, due to the very small model inputs and the relatively large execution time of RNN layers, inferring the models on the server is the optimal choice. This observation also corresponds to the results of the optimal partition point predicted by WebInf in Fig. 8. This phenomenon is consistent with the key observations from our analysis in Sec. 3.

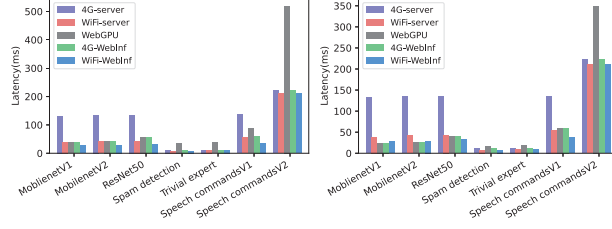
Partition point selection. Table 3 summarizes the partition point choices for latency optimization across 32 configurations (including seven models, two wireless network types, and two GPU backends) as facilitated by the Neurosurgeon [33]. Across all model and configuration permutations, WebInf consistently selects the optimal inference time.

5.3 Local load level and bandwidth variance

Authorized licensed use limited to: Southeast University. Downloaded on October 06, 2026 at 13:35:48 UTC from IEEE Xplore. Restrictions apply.

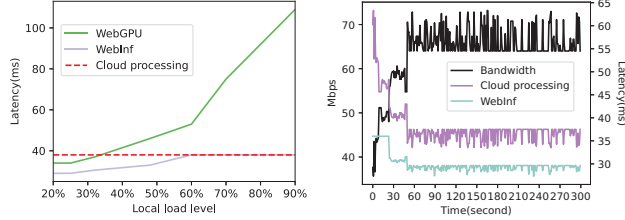
TABLE 3: WebInf’s prediction partition points for best end-to-end latency.

PC	Transmission	MobilenetV1	MobilenetV2	ResNet50	Spam detection	Trivial expert	Speech commandsV1	Speech commandsV2
Integrated graphics	4G	<i>argmax</i>	<i>argmax</i>	<i>argmax</i>	server	server	<i>conv1</i>	server
	WiFi	<i>dw6</i>	<i>pw6</i>	<i>conv3.1.1</i>	server	server	<i>bn4</i>	server
Discrete graphics	4G	<i>argmax</i>	<i>argmax</i>	<i>argmax</i>	server	server	<i>bn4</i>	server
	WiFi	<i>dw6</i>	<i>pw6</i>	<i>conv3.1.1</i>	server	server	<i>dense3</i>	server



(a) WebInf using integrated graphics. (b) WebInf using discrete graphics.

Fig. 10: Latency speedup achieved by WebInf normalized to status quo approach.



(a) WebInf adjusts partition based on local load levels. (b) The impact of wireless bandwidth variance.

Fig. 11: The impact of local workload variations and network fluctuations on WebInf.

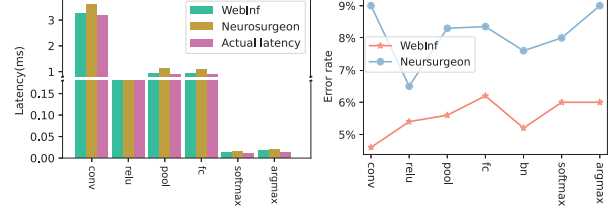
WebInf successfully mitigates the impact of network fluctuations by adapting its partitioning choices and adjusting data transfer volumes based on available bandwidth.

5.4 Comparing WebInf to Neurosurgeon

We employ Neurosurgeon [33] and WebInf for hierarchical performance prediction of VGG. We aggregate predicted latency by layer categories for comparison with actual latency. Fig. 12(a) illustrates that WebInf’s predicted times are closer to the actual times. Fig. 12(b) displays the average error rates for several layers on the test model. Neurosurgeon’s regression prediction model is relatively simple and has fewer input variables, leading to larger prediction errors. We have enhanced the regression prediction model with additional input variables and categories, and the results demonstrate that WebInf exhibits smaller prediction errors.

6. RELATED WORK

Deep learning in browser. In addition to the work on the utilization and efficiency of data center systems [18]–[20], [22]–[25], various algorithm optimization techniques have been proposed. For instance, neural network compression



(a) Latency comparison. (b) Error Rates comparison.

Fig. 12: Comparing WebInf to Neurosurgeon.

methods [26], [28] prune large-scale neural network models into smaller ones with lower computational demands, effectively enabling deployment on edge devices. Furthermore, to enhance cloud computing capabilities, a variety of accelerators such as GPUs, ASICs, and FPGAs have been introduced for data centers [30], [32]. However, these techniques are unlikely to address the direct needs of all existing browser platforms, particularly the challenge of accurate and efficient model inference.

Computation offloading. Transmission latency is one of the most significant factors affecting computation offloading and a core metric for evaluating network quality of service [38]. Various algorithms have been proposed to optimize the application performance via computation offloading [15], [21], [27], [31]. Kang *et al.* [33] has devised a system named Neurosurgeon, which automatically divides models between mobile devices and the server at the granularity of neural network layers. In this paper, we use the collaborative approach to design WebInf.

7. CONCLUSION

In this paper, we investigate the current methods involving cloud-only processing and browser-only processing. We find that transferring input data to server or executing neural network models in browser may not always be optimal. We investigate the computation and data characteristics of seven models including CV, NLP, and ASR. We provide a balance between partitioning computation at distinct points in neural networks. Building upon these findings, we design WebInf, a system capable of automatically partitioning models between browser and server at the granularity of neural network layers. WebInf adapts to various neural network architectures, hardware platforms, and wireless connections, selecting optimal partitioning points for minimal latency. Across seven benchmark tests, WebInf’s inference time improvements of 30% compared to individual server execution and 52% compared

to browser execution, reaching peak enhancements of 33% and 69% respectively. In future work, we will discuss how to deploy large language models in browsers via WebGPU for joint training and inference of large models.

8. ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant 62272098.

REFERENCES

- [1] Li H, Lu Z. "Deep learning for information retrieval." In: Proceedings of the 39th International ACM SIGIR 2016, pp. 1203-1206.
- [2] Koren Y, Bell R, Volinsky C. "Matrix factorization techniques for recommender systems." *Computer*, 2009, 42(8): 30-37.
- [3] Bahdanau D, Cho K, Bengio Y. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473, 2014.
- [4] Socher R, Perelygin A, Wu J, et al. "Recursive deep models for semantic compositionality over a sentiment treebank." In *Proc. of EMNLP*, 2013, pp. 1631-1642.
- [5] Tulshan A S, Dhage S N. "Survey on virtual assistant: Google assistant, siri, cortana, alexa." In *Advances in Signal Processing and Intelligent Recognition Systems: 4th International Symposium SIRS 2018, Bangalore, India, September 19-22, 2018*.
- [6] Khronos Group. "WebGL." [Online]. Available: <https://www.khronos.org/webgl/>. [Accessed: 2018].
- [7] W3C. "WebGPU." [Online]. Available: <https://www.w3.org/community/gpu/>. [Accessed: 2018].
- [8] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Proc. of ICCV*, IEEE, 1998, pp. 555-562.
- [9] R. Wang, B. K. Szymanski, and M. Zhi, "Serverless Computing: An Overview and Research Challenges," in *IEEE Transactions on Services Computing*, vol. 13, no. 3, pp. 438-446, 2020.
- [10] A. Kumar and J. Wang, "Elastic Machine Learning Algorithms in Amazon SageMaker," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 1499-1504, 2020.
- [11] S. Han, J. Wang, and X. Lu, "WebDNN: Fastest DNN Execution Framework on Web Browser," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1-6, 2018.
- [12] R. Li and L. Zhang, "Edge Machine Learning Inference with WebGL," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1254-1262, 2019.
- [13] A. G. Howard, M. Zhu, B. Chen, et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," in *Proc. of CVPR*, 2017.
- [14] Bilogrevic, E. von Zezschwitz, and H. Hussmann, "A Survey of Privacy Issues in Social Media," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Algarve, Portugal, Oct. 2012, pp. 1482-1487.
- [15] Li, B., Fan, H., Gao, Y. and Dong, W., 2022, June. Bringing webassembly to resource-constrained iot devices for seamless device-cloud integration. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services* (pp. 261-272).
- [16] Y. Ma, D. Xiang, S. Zheng, et al., "Moving Deep Learning into Web Browser: How Far Can We Go?" in *Proceedings of The World Wide Web Conference*, 2019, pp. 1234-1244.
- [17] Z. Chen, Y. Cao, Y. Liu, et al., "A Comprehensive Study on Challenges in Deploying Deep Learning Based Software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 750-762.
- [18] C.-H. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, L. Tang, J. Mars, and R. G. Dreslinski, "Adrenaline: Pinpointing and reigning in tail queries with quick voltage boosting," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2015.
- [19] J. Hauswald, M. A. Laurenzano, Y. Zhang, et al., "Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers," in *Proc. ACM ASPLOS*, 2015.
- [20] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: Qos awareness and increased utilization for nonpreemptive accelerators in warehouse scale computers," in *ACM SIGPLAN Notices*, vol. 51, pp. 681-696, 2016.
- [21] Li, B., Dong, W. and Gao, Y., 2021, May. Wipro: A webassembly-based approach to integrated iot programming. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications* (pp. 1-10). IEEE.
- [22] Y. Zhang, D. Meisner, J. Mars, and L. Tang, "Treadmill: Attributing the source of tail latency through precise load testing and statistical inference," in *Computer Architecture (ISCA)*, 2016 ACM/IEEE 43rd Annual International Symposium on, pp. 456-468, IEEE, 2016.
- [23] A. Jain, M. A. Laurenzano, L. Tang, and J. Mars, "Continuous shape shifting: Enabling loop cooptimization via near-free dynamic code rewriting," in *Microarchitecture (MICRO)*, 2016 49th Annual IEEE/ACM International Symposium on, pp. 1-12, IEEE, 2016.
- [24] M. A. Laurenzano, Y. Zhang, J. Chen, L. Tang, and J. Mars, "Powerchop: Identifying and managing non-critical units in hybrid processor architectures," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*, pp. 140-152, IEEE Press.
- [25] V. Petrucci, M. A. Laurenzano, Y. Zhang, J. Doherty, D. Mosse, J. Mars, and L. Tang, "Octopus-man: Qos-driven task management for heterogeneous multicore in warehouse scale computers," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2015.
- [26] X. Xie and K. H. Kim, "Source Compression with Bounded DNN Perception Loss for IoT Edge Computer Vision," in *Proceedings of The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1-16.
- [27] Li, B. and Dong, W., 2020, November. Edgeprog: Edge-centric programming for iot applications. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)* (pp. 212-222). IEEE.
- [28] T. Zhang, S. Ye, K. Zhang, et al., "A Systematic DNN Weight Pruning Framework Using Alternating Direction Method of Multipliers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 184-199.
- [29] K. Ovtcharov, O. Ruwase, J. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, no. 11, 2015.
- [30] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "Djinn and Tonic: DNN as a Service and Its Implications for Future Warehouse Scale Computers," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*, ISCA '15, New York, NY, USA, 2015.
- [31] Li, B. and Dong, W., 2021. Edge-centric programming for iot applications with automatic code partitioning. *IEEE Transactions on Computers*, 71(10), pp.2408-2422.
- [32] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, no. 11, 2015.
- [33] Kang Y, Hauswald J, Gao C, et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge[J]. *ACM SIGARCH Computer Architecture News*, 2017, 45(1): 615-629.
- [34] A. Finamore, M. Mellia, M. M. Munafo, et al., "Youtube everywhere: Impact of device and infrastructure synergies on user experience," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 345-360.
- [35] I. Sutskever, O. Vinyals, and Q. V. Le, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *Proc. of EMNLP*, 2014, pp. 1724-1734.
- [36] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.
- [37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533-536, Oct. 1986.
- [38] Fan, X., Deng, X., Xia, et al. (2023). Tensor-based Confident Information Coverage Reliability of Hybrid Internet of Things. *IEEE Transactions on Mobile Computing*.