Evaluating Model Compression Techniques for On-Device Browser Sign Language Recognition Inference

Mauricio Salazar
Digital Humanities Laboratory
Pontificia Universidad Católica del Perú
Lima, Perú
mauricio.salazare@pucp.edu.pe

Danushka Bandara
Dept. of Comp. Sci. and Engineering
Fairfield University
Fairfield, Connecticut 06824
dbandara@fairfield.edu

Gissella Bejarano
Department of Computer Science
Marist College
Poughkeepsie, NY-USA
gissella.bejarano@marist.edu

***Abstract*—Due to the improvement of sign language (SL) recognition models, more work has started to build upon them to create end-user platforms such as video-based dictionaries. Although most of them base their design on cloud services or APIs, this approach can become expensive, especially when working with such substantial dimensional data as videos. Other realities, such as regions with low internet connection and few economic resources, can benefit from the offline alternative for deploying these systems. However, models deployed in websites' front-end or edge devices have some constraints on their size. Therefore, our work evaluates two model compression techniques to enhance the performance of on-device inference for a SL recognition model. We test knowledge quantization and distillation on three datasets varied in size, ultimately deploying and evaluating them on a locally hosted web page. To the best of our knowledge, this is the first evaluation of a quantized and distilled transformer-based SL recognition model for on-device browser inference. Furthermore, the combination of both compression techniques achieves a size reduction of up to 17.83 times and a speedup of 5.9 times the baseline.**

***Index Terms*—quantization, vision transformer, edge computing, sign language recognition**

I. Introduction

Sign Language (SL) recognition models are crucial to enhancing communication and education access for the deaf and hard-of-hearing communities. Significant strides have been made in democratizing access to these models through the Internet, narrowing the gap between model development and their practical use in production environments. For example, recent works provide publicly available online platforms for French Sign Language [1] and Peruvian Sign Language [2]. However, cloud-based AI services such as these platforms, which aim to be available to many users, face several challenges such as stable connection, latency and computational cost.

The stable connection required for AI-powered cloud services such as sign language dictionaries is a particularly relevant challenge for developing countries where a large portion of their population does not have reliable access to the Internet. For example, in Peru, the 32.1% of the deaf population is in Lima while the rest reside outside of it, in regions that coincide with the locations where it is more challenging to get a reliable internet connection [3]. To further exacerbate this issue, the need for access to education and information where sign language dictionaries can help is felt the most in rural areas with limited access to professional sign language interpreters since most interpreters reside in Lima [4]. On top of these difficulties, computational costs of tools such as an online sign language dictionary could increment easily. For instance, inference calls of YOLOv8, an object recognition model, on up to 118,000 images range from $12.87 on AWS to $46.92 on Azure daily [5] based on different number of instances used. For sign language recognition, computational costs should consider sequences of images instead of individual ones.

On-device deployed models represent an alternative to offload the cloud computational cost by reducing the amount of data transferred to an external server [6]. Consequently, the number of simultaneous users does not impact the service's availability or cost, making it possible to deploy models in real-world applications to any number of users without scaling costs. Moreover, eliminating this connection with an external server removes a data exchange that could be challenging for users without a stable internet connection. Besides, an on-device approach protects the privacy of individuals signing in front of a camera because the inference occurs on the user's device rather than an external server. Nevertheless, available resources on local devices can constrain the users' experience of an AI-based sign language dictionary. The model parameters and the required runtime for running the inference require allocated memory from the browser. By reducing the model size and computational load, these techniques can facilitate efficient and effective real-time SL recognition on standard web browsers. In this work, we explore both quantization and knowledge distillation to improve on-device inference of SLR models through users' browsers. We evaluate our approaches on three datasets—LSA64, AUTSL, and DGI305 using one of the most popular SLR models, SPOTER [7]. We deploy

979-8-3503-7834-4/24/$31.00 © 2024 IEEE

the models using the ONNX Runtime Javascript API on a locally hosted web page to validate their functionality and performance in a near-real-world environment.

II. Related Work

Previous works on compression techniques have successfully applied quantization and distillation to transformer models, achieving good performance. The following section provides examples of these implementations and explains their underlying principles. Additionally, it includes details about prior deployments of deep learning models for on-device browser inference.

A. Quantization of Transformer Models

Quantization is a technique that reduces the precision of the model parameters, compressing the original 32-bit floating point weights into smaller 8-bit, 4-bit, or even 2-bit weights. This reduction significantly decreases the model size and computation requirements, making it ideal for on-device inference. Nevertheless, it comes at the cost of reduced accuracy compared to the original model. This technique arose from the notion that models tend to be over-parameterized, with considerable redundancy among their weights, which results in unnecessary size [8]. A solution for this became using techniques such as binarization, product quantization, or scalar quantization with k-means to map the original 32-bit floating point weights to a smaller range of possible values [9].

This compression technique has been successful in reducing the size of transformer models while maintaining their performance scores by transforming their full precision 32-bit weights into a reduced 8-bit integer form for tasks such as machine translation and image classification [10]–[12]. A notable implementation similar to the proposed approach is ESPnet-ONNX [12], which tests both quantization and node fusion to convert speech processing models from Pytorch into an optimized ONNX format. These experiments resulted in a speedup between 1.3 to 2 times faster than the original Pytorch models. However, the inference of the model was tested through an isolated AWS EC2 instance, which differs from the browser runtime environment proposed for this research.

B. Knowledge distillation on Transformer Models

Knowledge distillation is a compression technique applied during the training stage. In this method, a smaller student model is trained with the guidance of a larger teacher model that has been tested to perform better. This process involves leveraging the output probability distribution of the teacher model to enhance the training of the student model. By learning from the teacher model, the student model can achieve high performance while maintaining a more efficient and compact architecture.

This has been used previously on vision transformers which focus on image classification. For instance, distillation has been previously used to improve an action recognition transformer model performance while identifying actions in the dark [13]. A teacher model was trained with light enhancement preprocessing over the video data, and its outputs were used to train an identical model but trained with videos without proper lighting. The output increased performance in its accuracy for predicting actions without increasing its computational complexity by preprocessing the videos with adjusted lighting.

Meanwhile, another approach for leveraging knowledge distillation to improve vision transformer performance has been to use a convolutional neural network as a teacher model [14]. The probability distribution of a convolutional network was introduced into the distillation loss function of a transformer student. As a result, the accuracy of the transformer model in image classification over ImageNet improved from 83.1% without a teacher to 85.2% through knowledge distillation.

C. On-device Inference through Browsers

The integration of deep learning model inference on the edge, directly through users' browsers, has been a topic extensively explored by various libraries such as TensorFlow.js [15], ONNX Runtime [16], WebDNN [17], among others. These libraries provide both training and inferencing capabilities through browsers by integrating them in Javascript [18].

The closest example to this approach for sign language recognition has been performed by the american sign language dictionary [19]. As confirmed through consultations with the authors, the CoreML library from Apple [20] was used for this dictionary. This tool enabled deploying the original model to iOS devices to run on-device inference. However, it also meant a restriction of availability only for iOS devices, since CoreML is not compatible with other devices. Moreover, the original model without any compression applied was deployed, meaning that it could be further optimized still. Other noteworthy examples of prior implementations include using a KNN image classifier in TensorFlow.js. This was employed to recognize custom gestures or signs for controlling an Amazon Alexa device via a computer webcam [21]. Another notable instance is the use of ONNX Runtime to deploy Segment Anything from Meta [16], [22]. This implementation, executed through WebAssembly or WebGPU, employs an encoder-decoder architecture for image segmentation, showcasing the feasibility of deploying models with complex architecture on the web with on-device browser inference.

From previous research, pre-loading the models is one important factor to consider when deploying these on-device inference services. Since the model loading time significantly exceeds that of the inference itself [18], it is recommended to warm up the model before a user requests a prediction. This approach minimizes delays and ensures a more responsive user experience when executing real-time inference tasks in the browser.

III. Methodology

To evaluate the effectiveness of compression techniques for on-device browser inference, several factors first needed to be considered. These included the selection of datasets to test, the architecture of the model, the integration with the compression techniques, and the deployment method on browsers. The

following sections provide a detailed explanation of each step in the process.

A. Preprocessing Datasets

Three primary datasets were considered: the Argentinian Sign Language dataset (LSA64) [23], the Ankara University Turkish Sign Language Dataset (AUTSL) [24], and the PUCP University Peruvian Sign Language Dataset (DGI305) [25]. For each dataset, only labels with more than 10 instances were selected. This filtering process primarily impacted the DGI305 dataset, reducing its size from 1645 videos down to 584 videos. Nonetheless, filtering was necessary to ensure the model received sufficient data for each gloss (written representation of a sign). At the end of this stage, the total of classes for each dataset was 64 for LSA64, 226 for AUTSL, and 19 for DGI305. Subsequently, each dataset was split into training and test sets using an 80/20 ratio, preserving class balance for both sets by using *train_test_split* from the scikit-learn library

Keypoints from all three datasets were extracted using the pose estimator model provided by the Mediapipe Holistic framework. This model could extract 543 keypoints for each frame from the videos, covering the face, hands, and body. These keypoints provide a detailed representation of the signer's movements, which could later be used as an input for the SL recognition model.

From the initial set of 543 extracted keypoints, a subset of 54 keypoints was selected for each frame. The selection process was based on the relevance of these keypoints signs used in [2]. Once selected, these keypoints underwent a normalization process. Normalization involved standardizing and scaling the keypoints to a consistent range, making them suitable for the following model training stage. This step ensures that variations in keypoint positions due to different signers or camera setups are minimized, improving the robustness of the final model.

After the preprocessing stage, each input entry was shaped as [frames, keypoints_ID, coordinates]. More specifically, the shape of the preprocessed vectors becomes [dynamic axis, 54, 2]. Since the first dimension is dynamic, the model can handle videos of varying lengths. The second dimension corresponds to the 54 selected keypoints, selected from the 543 available as previously described. Thirdly, the last dimension represents each keypoint's x and y coordinates, indicating their position for each frame.

B. Model Architecture

For this research, the SPOTER architecture proposed by Matyáš Boháček and Marek Hrúz was selected [7]. This architecture is a variation of the standard transformer framework, specifically designed for SL recognition tasks. The difference resides in the removal of the self-attention from the decoder. Since the queried target is a single element representing the class of the input sign, the decoder layers do not use self-attention for their calculations. The final decoder output is passed to a linear layer that classifies the sign in the video accordingly through a final softmax function. This model was chosen because its architecture prioritizes low computational cost, with fewer parameters and much fewer floating point operations (FLOPs) for inference than other SL recognition models [7]. After compressing it further, its already low computational cost should allow the model to minimize its impact on the browser's available resources.

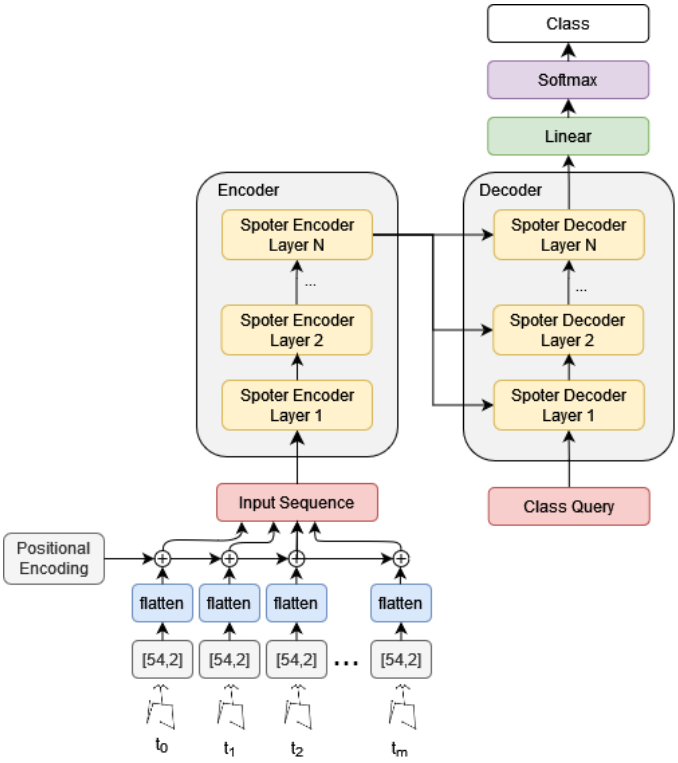

Fig. 1. Architecture of the SPOTER models used for the experiments (Based on the diagram from [7])

Two versions of the SPOTER model trained with the standard cross-entropy loss were used as benchmarks to evaluate the effectiveness of the chosen compression techniques. The first version is the full SPOTER model, with six encoder and six decoder layers (N=6). The second version is a smaller variant, featuring only one encoder layer and one decoder layer (N=1). These baseline models provide a reference point for assessing the impact of compression techniques on model performance and computational efficiency. All models, including these benchmarks, were trained for 15 epochs with a learning rate of 0.001, using the SGD optimizer. Additionally, due to the different number of frames in each input keypoint tensor, a batch size of 1 was employed during training.

C. Distillation Loss and Quantization

The training process for knowledge distillation involves using a loss function that combines two components: the conventional cross-entropy loss and the Kullback-Leibler divergence between the student's and the teacher's outputs [26]. The cross-entropy loss L_{CE} in (2) ensures the student's predictions align with the true labels, while the KL divergence L_{KL} in

(3) helps the student replicate the teacher model's patterns. By balancing these two loss components in the overall knowledge distillation loss L_{KD} in (1) with a weighting factor α, the student model learns to reproduce the teacher, reducing its size while maintaining its performance.

$$L_{\text{KD}} = \alpha L_{\text{CE}}(y, \sigma(z_s)) + (1-\alpha) L_{\text{KL}}(\sigma(z_t/T), \sigma(z_s/T)) \quad (1)$$

$$L_{\text{CE}}(y, \sigma(z_s)) = -\sum_i y_i \log(\sigma(z_{s_i})) \quad (2)$$

$$L_{\text{KL}}(\sigma(z_t/T), \sigma(z_s/T)) = \sum_i \sigma(z_{t_i}/T) \log\left(\frac{\sigma(z_{t_i}/T)}{\sigma(z_{s_i}/T)}\right) \quad (3)$$

For this loss function, there are two important parameters: the temperature (T) and the weighting factor (α). The temperature determines the softening of the probability distributions from the teacher model. When it is high, the probabilities are more spread out, which helps the student to learn the difference between all classes, while it will focus more on the most likely class when T is low. Meanwhile, α defines the balance between the KL divergence and the cross-entropy loss. If α is too high, the model might overfit to the teacher's outputs and not learn enough from the true labels. However, if α is too low, the student might not capture enough of the teacher's distilled knowledge.

For the experiments, the previously explained parameters were set constant to $T = 3$ and $\alpha = 0.5$. This configuration was used to train an alternative small SPOTER model with a 1-layer encoder and a 1-layer decoder. This architecture was chosen to evaluate the effect of distillation over the less complex architecture possible for the SPOTER model. The base SPOTER architecture, with six encoder layers and six decoder layers, is used as the teacher model. The outputs from this teacher model guide the training of the smaller, 1-layer student model. This distilled student model is then compared against the original 1-layer SPOTER benchmark model to evaluate performance gains due to knowledge distillation. At the end of this stage, we have a total of 3 versions:

- Full SPOTER
- 1-Layer SPOTER (1-layer encoder and 1-layer decoder)
- Distilled SPOTER (1-layer encoder and 1-layer decoder)

After training every variation of the models, 8-bit dynamic quantization is applied to each of them. Quantized models are then evaluated to ensure that the reduction in precision does not significantly impact their performance. Through the Pytorch framework, the accuracy and model size of the quantized models are compared against their non-quantized counterparts to assess the effectiveness of the compression techniques.

D. Model Conversion & Deployment

To enable the deployment of every model through on-browser inference, we convert all models to the ONNX (Open Neural Network Exchange) format. The conversion process involves exporting the trained models from their original frameworks (.pth in Pytorch) to .onnx ONNX format. This models are then deployed through the onnx web benchmark implementation provided by Microsoft [27]. This benchmark provides the inference speeds of the converted models that end users would perceive. The models were integrated by compiling their executions through the Web Assembly format available through ONNX Runtime. Consequently, this implementation accesses the CPU of the device when inferencing and not the GPU. For all the experiments, the device's CPU was an AMD Ryzen 7 5800HS.

IV. Results

The results of the experiments conducted for this research can be divided into the two following sections. The first set of experiments was conducted through the Pytorch framework, where the size and accuracy of every variation were measured and annotated. After that, the onnx web benchmark [27] was used to measure the time response of the converted .onnx models for on-browser inference in a locally hosted page accessed through Google Chrome. The code used for training, converting, and evaluating these models can be found on the GitHub repository linked to this research[1].

A. Size & Accuracy

Applying quantization on the model led to a size reduction of 3 times on average in PyTorch, as shown in Tables I, II, and III. The size reduction from the full fp32 precision version to the quantized version remained consistently proportional across all variants. Each version demonstrated a reduction in size to one-third of its original when quantized. For the AUTSL dataset, the Full SPOTER decreased from 22.72 Mb to 7.47 Mb, the 1-Layer SPOTER decreased from 3.9 Mb to 1.29 Mb, and the Distilled SPOTER also decreased from 3.9 Mb to 1.29 Mb, representing a reduction by a factor of 3 in each case. As it is also shown, the Distilled SPOTER exhibited the exact same size as the 1-layer version, which is expected since they share identical architectures with the same number of parameters. The only difference between both versions is their training method.

TABLE I
LSA64 Results

Model	Size (Mb)	Top-1 Acc.	Top-5 Acc.
Full SPOTER	22.65	98.28	100
Quantized Full SPOTER	7.45	98.28	100
1-Layer SPOTER	3.82	97.81	100
Quantized 1-Layer SPOTER	1.27	97.81	100
Distilled SPOTER	3.82	98.59	100
Quantized Distilled SPOTER	1.27	98.59	100

Additionally, the knowledge distillation technique proved useful in maintaining Top-1 accuracy even with 5 fewer layers in the encoder and decoder. When evaluated on the test set, each iteration of the distilled SPOTER proved to have higher top-1 and top-5 accuracy than its 1-layer counterpart. Even

[1]https://github.com/Mauricio1812/spoter_compression

without the need for on-device browser implementation, the benefits of quantization proved beneficial in enhancing the model from a backend perspective.

TABLE II
DGI305 RESULTS

Model	Size (Mb)	Top-1 Acc.	Top-5 Acc.
Full SPOTER	22.63	53.85	88.88
Quantized Full SPOTER	7.44	53.85	88.88
1-Layer SPOTER	3.8	52.14	84.61
Quantized 1-Layer SPOTER	1.27	52.14	84.61
Distilled SPOTER	3.8	53.85	85.47
Quantized Distilled SPOTER	1.27	53.85	85.47

TABLE III
AUTSL RESULTS

Model	Size (Mb)	Top-1 Acc.	Top-5 Acc.
Full SPOTER	22.72	88.32	98.91
Quantized Full SPOTER	7.47	88.46	98.91
1-Layer SPOTER	3.9	87.18	98.49
Quantized 1-Layer SPOTER	1.29	87.22	98.49
Distilled SPOTER	3.9	87.82	98.72
Quantized Distilled SPOTER	1.29	88.05	98.72

B. On-device Browser Inference

The effectiveness of the final quantized distilled models was evaluated by deploying them in a web-based environment. The models were integrated by compiling their executions using the Web Assembly format available through ONNX Runtime. First, the model was loaded through an initial request for a warmup of 50 samples. Once loaded, another 1000 inferences were calculated while measuring the time required for each. Since the focus for this benchmark was to measure inference time, the input for the model was a set of randomly generated arrays of shape [70,54,2]. The duration for each inference was registered on a list, from which the average time and its standard deviation were calculated later. The models trained with the LSA64 dataset were used for these evaluations, the measured average inference time for each version is detailed on Table IV. Furthermore, single-thread processing was compared with inferencing through 4 parallel threads in Web Assembly to explore the benefit of activating this functionality. Overall, multi-threading was beneficial mainly for the larger full SPOTER model rather than for the smaller distilled versions.

V. DISCUSSION

The conducted experiments demonstrate significant improvements in model efficiency and performance by applying quantization and knowledge distillation techniques. To discuss in detail these results, the model size, accuracy, and inference speed need to be considered.

In terms of size, 8-bit dynamic quantization resulted in a constant reduction of a third of the original size across all model variations. This proportion was consistent across the Full SPOTER, 1-Layer SPOTER, and Distilled SPOTER models, indicating the uniform effectiveness of quantization. Despite this reduction, the quantized models maintained their accuracy levels for both the Top-1 and Top-5 accuracy.

TABLE IV
INFERENCE TIME ON BROWSER

Threads	Model	Time (ms)
1	Full SPOTER	1.95 ± 0.41
	Quantized Full SPOTER	1.99 ± 0.47
	Distilled SPOTER	0.21 ± 0.16
	Quantized Distilled SPOTER	0.33 ± 0.15
4	Full SPOTER	1.45 ± 0.31
	Quantized Full SPOTER	1.47 ± 0.44
	Distilled SPOTER	0.23 ± 0.12
	Quantized Distilled SPOTER	0.27 ± 0.12

On the other hand, knowledge distillation proved useful for maintaining the performance of smaller models. The Distilled SPOTER models, despite having the same reduced number of layers as the 1-Layer SPOTER, achieved higher Top-1 and Top-5 accuracies. Because of their identical architecture, both the 1-Layer SPOTER and the distilled SPOTER have an equivalent model size and an equivalent number of total operations. Thus, this improvement becomes highly useful as an accessible way to increase the accuracy of simplified model versions without increasing computational complexity.

By combining both quantization and distillation, it was possible to reduce the model size by 17.83 times for the LSA64 Dataset, 17.61 times for the DGI305 Dataset, and 17.61 times for the AUTSL Dataset. This significantly reduces overall size, decreasing the required browser memory to allocate for the model storage. As a result, these compression techniques could prevent the the slow loading times that on-browser inference tends to cause [28].

Regarding the inference speed measure for on-browser inference, the performance of the quantized models was evaluated in a web-based environment using the ONNX Runtime with WebAssembly. The inference times were measured on a locally hosted page accessed through Google Chrome, comparing single-threaded and multi-threaded processing. The results reveal that distillation can significantly enhance inference speed. Meanwhile, quantization resulted in similar inferring times as the full 32-bit precision version of the model. While using a single thread, the average time per inference was reduced from 1.95 ms down to 0.33 ms between the full SPOTER and the quantized distilled SPOTER, a speedup of 5.9 times. Similarly, for the experiments with four threads, a speedup of 5.05 was recorded.

Comparing the difference of inference speeds between 1 and 4 threads, the full SPOTER was the only model significantly affected. A speedup of 1.33 was achieved with the full SPOTER, while the distilled SPOTER and 1-Layer SPOTER were not accelerated by multi-threading. This suggests that multi-threading is particularly advantageous for larger models, where parallel processing can more effectively reduce latency.

VI. Conclusion

This research demonstrates the effectiveness of using knowledge distillation and quantization techniques to develop small but accurate SL recognition models suitable for on-browser real-time inference. By employing these techniques, we successfully reduced the model size and computational requirements without compromising accuracy, making these models viable for deployment in resource-constrained environments. Moreover, the deployment of these models was tested in a locally hosted web environment, evaluating their speed for on-device browser inference. This differs from the usual approach of integrating the optimized model on an external server where the inferences are calculated. Thus, the proposed implementation is the first for a sign language recognition transformer in which the previously mentioned compression techniques have been applied and compared to evaluate its performance for on-browser inference.

Future research will focus on comparing the effects of static versus dynamic quantization on inference times. Furthermore, the same ONNX Runtime framework can be used for deploying the model to mobile devices. Therefore, another important aspect to consider is the performance of these models on mobile native platforms. With further development, the solution could be adapted and tested in mobile environments to assess performance. Finally, integrating these models into a real-world web page will be a necessary next step. This integration would involve evaluating how these compressed models affect the loading times and responsiveness, conducting user experience studies to gather feedback on its usability. In this paper, we have limited our discussion to the technical performance aspects, however usability of this system is also quite important for successful deployment. The end goal for this implementation would be setting up a freely accessible web page that only needs an internet connection once to load the interface and model. Once the user has entered the web page, the integrated model could take their videos as input and classify the gestured signs offline.

This research has tested how to best compress these models to be integrated with the described web pages. Thus, it represents a first step in making real-time sign language recognition systems more accessible and efficient for on-browser deployment. We believe that this method could be easily generalized to other types of problems like speech recognition, gesture-based interfaces, or natural language processing for low-resource languages. By integrating both quantization and knowledge distillation, efficient model compression for resource-constrained devices can be achieved, potentially driving a broader impact across a diverse range of applications.

References

[1] J. Fink, P. Poitier, M. André, L. Meurice, B. Frénay, A. Cleve, B. Dumas, and L. Meurant, "Sign language-to-text dictionary with lightweight transformer models," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, E. Elkind, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2023, pp. 5968–5976, aI for Good. [Online]. Available: https://doi.org/10.24963/ijcai.2023/662

[2] G. Bejarano, J. Huamani-Malca, C. Vasquez, S. Oporto, F. Cerna, C. Ramos, and M. Rodrigues-Mondoñedo, "Lessons from deploying the first bilingual peruvian sign language- spanish online dictionary," in *Proceedings of the LREC2024 12th Workshop on the Representation and Processing of Sign Languages: Multilingual Sign Language Resources*. Torino, Italy: European Language Resources Association, 2024.

[3] OSIPTEL, "Erestel 2022 - encuesta residencial de servicios de telecomunicaciones," 2022. [Online]. Available: https://www.osiptel.gob.pe/media/hkkhfkf3/41661-erestel-2022.pdf

[4] E. Parks and J. Parks, "Encuesta sociolingüística de la comunidad sorda en perú," SIL International, 2015.

[5] P. Rajendran, S. Maloo, R. Mitra, A. Chanchal, and R. Aburukba, "Comparison of cloud-computing providers for deployment of object-detection deep learning models," *Applied Sciences*, vol. 13, no. 23, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/23/12577

[6] A. ICHINOSE, A. TAKEFUSA, H. NAKADA, and M. OGUCHI, "Performance evaluation of pipeline-based processing for the caffe deep learning framework," *IEICE Transactions on Information and Systems*, vol. E101.D, no. 4, pp. 1042–1052, 2018.

[7] M. Boháček and M. Hrúz, "Sign pose-based transformer for word-level sign language recognition," in *2022 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, 2022, pp. 182–191.

[8] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," 2014.

[9] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," 2014.

[10] A. Bhandare, V. Sripathi, D. Karkada, V. Menon, S. Choi, K. Datta, and V. Saletore, "Efficient 8-bit quantization of transformer neural machine language translation model," 2019.

[11] G. Prato, E. Charlaix, and M. Rezagholizadeh, "Fully quantized transformer for machine translation," 2020.

[12] M. Someki, Y. Higuchi, T. Hayashi, and S. Watanabe, "Espnet-onnx: Bridging a gap between research and production," in *2022 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2022, pp. 420–427.

[13] C.-J. Chang, O. T.-Y. Chen, and V. S. Tseng, "Dl-kdd: Dual-light knowledge distillation for action recognition in the dark," 2024. [Online]. Available: https://arxiv.org/abs/2406.02468

[14] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers distillation through attention," 2021. [Online]. Available: https://arxiv.org/abs/2012.12877

[15] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel, S. Bileschi, M. Terry, C. Nicholson, S. N. Gupta, S. Sirajuddin, D. Sculley, R. Monga, G. Corrado, F. B. Viégas, and M. Wattenberg, "Tensorflow.js: Machine learning for the web and beyond," 2019.

[16] microsoft, "onnxruntime-inference-examples," https://github.com/microsoft/onnxruntime-inference-examples, 2024.

[17] M. I. L. Tokyo, "webdnn," https://github.com/mil-tokyo/webdnn, 2022.

[18] Y. Ma, D. Xiang, S. Zheng, D. Tian, and X. Liu, "Moving deep learning into web browser: How far can we go?" 2019. [Online]. Available: https://arxiv.org/abs/1901.09388

[19] M. Bohacek and S. Hassan, "Sign spotter: Design and initial evaluation of an automatic video-based american sign language dictionary system," in *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3597638.3614497

[20] Apple, "Coreml — apple developer documentation," https://developer.apple.com/documentation/coreml, 2024.

[21] A. Singh, "Alexa sign language translator," https://github.com/shekit/alexa-sign-language-translator, 2018.

[22] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," *arXiv:2304.02643*, 2023.

[23] F. Ronchetti, F. Quiroga, C. Estrebou, L. Lanzarini, and A. Rosete, "Lsa64: A dataset of argentinian sign language," *XX II Congreso Argentino de Ciencias de la Computación (CACIC)*, 2016.

[24] O. M. Sincan and H. Y. Keles, "Autsl: A large scale multi-modal turkish sign language dataset and baseline methods," *IEEE Access*, vol. 8, p. 181340–181355, 2020. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2020.3028072

[25] G. Bejarano, J. Huamani-Malca, F. Cerna-Herrera, F. Alva-Manchego, and P. Rivas, "PeruSIL: A framework to build a continuous Peruvian Sign Language interpretation dataset," in *Proceedings of the LREC2022 10th Workshop on the Representation and Processing of Sign Languages: Multilingual Sign Language Resources*, E. Efthimiou, S.-E. Fotinea, T. Hanke, J. A. Hochgesang, J. Kristoffersen, J. Mesch, and M. Schulder, Eds. Marseille, France: European Language Resources Association, Jun. 2022, pp. 1–8. [Online]. Available: https://aclanthology.org/2022.signlang-1.1

[26] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[27] microsoft, "onnxruntime-web-benchmark," https://github.com/microsoft/onnxruntime-web-benchmark, 2024.

[28] Q. Wang, S. Jiang, Z. Chen, X. Cao, Y. Li, A. Li, Y. Zhang, Y. Ma, T. Cao, and X. Liu, "Exploring the impact of in-browser deep learning inference on quality of user experience and performance," 2024. [Online]. Available: https://arxiv.org/abs/2402.05981