High-Level Frameworks: Effect on Transformer Inference Time and Power on Embedded GPU Devices

Marika E. Schubert*, David Langerman†, Alan D. George*
* *Department Electrical and Computer Engineering*
University of Pittsburgh
Pittsburgh, PA USA
{marika.schubert,alan.george}@pitt.edu
† *NSF Center for Space, High-Performance, and Resilient Computing*
Pittsburgh, PA USA
david.langerman@nsf-shrec.org

***Abstract*—Developing software for machine- and deep-learning (ML and DL) workloads is often a daunting task to individuals with minimal programming experience or to organizations with limited engineering capacity. ML frameworks address these issues by providing a high-level API able to perform otherwise complex tasks with less engineering time. This high level of abstraction can reduce and hide many of the challenges that are induced by unclean datasets, complicated pre/postprocessing pipelines, and low-level dependencies like CUDA. This high-level approach encourages model portability and can dramatically increase design iteration speed, as well as providing model speedup in some cases. This research demonstrates that these high-level ML frameworks are also more performant out-of-the-box on embedded systems than their pure PyTorch reference implementations likely due to their myriad of optimizations related to data movement and memory management. In this research, we benchmark a state-of-the-art transcription model, wav2vec2, and compare performance across different frameworks: the reference implementation from the Fairseq framework and the two higher-level frameworks HuggingFace and Lightning Flash. Overall, we observe that both Lightning Flash and HuggingFace are substantially faster than the original unoptimized PyTorch model. In general, these models ran between 1.8× and 2.0× faster than the base PyTorch implementation on the embedded NVIDIA Jetson platforms targeted. As a secondary result, we also observe the high-level frameworks to be more power efficient for the same computation.**

***Index Terms*—benchmarks, GPUs, software abstraction, machine learning, speech-to-text**

I. Introduction

As deep-learning (DL) models become larger and more computationally demanding, the task of running these models on constrained embedded hardware becomes more challenging. For applications where inference latency is of special concern, model processing is often done entirely on the edge device to maximize service availability and remove the communication overhead of network requests. This scheme of edge-system processing is common with many speech-based applications, including those using newer transformer-based models.

Transformer-based models are favorable for text- and audio-based tasks as their self-attention mechanisms can process an arbitrarily-sized input sequence. However, the memory and computational requirements for self-attention scale quadratically relative to the input sequence length. Often, this scaling behavior necessitates an embedded GPU for a practical balance of computing and power characteristics. However, many of these modern models are trained on server-class GPUs, which do not suffer the same constraints as their embedded counterparts. Therefore, after training, there is limited understanding of how these models will work on embedded systems.

In order to detail the characteristics of transformer-based speech-to-text models on embedded hardware, this research evaluates model inference performance with the wav2vec2 model [1], which has achieved state-of-the-art (SOTA) accuracy on many common speech-to-text datasets. In this paper, wav2vec2's inference latency and power is characterized with three ML frameworks: a PyTorch-only reference implementation from the fairseq repository [2] and two higher-level ML frameworks, HuggingFace [3] and Lightning Flash [4]. HuggingFace is a widely used high-level API for ML with transformers. Lightning Flash is an object-oriented library based on PyTorch that imposes a strict model format and provides higher-level APIs to access the underlying model structure. These model APIs allows the library to provide better bridges to integrate novel software optimizations or hardware support into existing code, like the Optimum optimization extension [5]. Both of these APIs access the HuggingFace repository's copy of the wav2vec2 model, meaning that the primary difference being tested between these models is pre- and post-processing and data-loading. To make sure that the performance of these models scales as expected with increased hardware resources, these tests are executed on three embedded GPUs: the NVIDIA Jetson TX2, the Xavier NX,

This work was supported by SHREC industry and agency members and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783.

and the AGX Xavier.

This study overall aims to aid in the design of a speech-to-text algorithm amenable to operation in a remote or offline systems with constrained hardware. This is beneficial for applications in remote areas, environments with noisy or unreliable wireless connections, and in situations where data must be managed offline due to legal, privacy, or latency reasons. Practically, these offline cases could include autonomous cars, accessibility devices, voice assistants in factories, and even human-spaceflight.

The main contributions of this paper are as follows. First, we characterize these frameworks on embedded GPUs by measuring latency and power consumption under various loads and hardware configuration. Second, we show derived metrics describing the realtime latency and performance (throughput-per-watt) values of these frameworks in the tested conditions. Finally, we provide recommendations for best use of these frameworks for inference on embedded systems.

II. Background

This section is broken into two parts. First, the high-level practical methods for accelerating transformers will be discussed. Second, the embedded GPUs that are used in this research will be summarized.

A. Accelerating Transformers

Short of developing a custom accelerator or attempting to integrate research software into their code, most DL developers have three options for increasing throughput or decreasing latency of transformer inference. The first of these is to increase the batch size performed with each inference. This method takes advantage of efficient tensor operations on GPUs, but may not be practical in situations where data is sporadic or latency is a strong constraint. Additionally, batched inputs all need to be padded to the same length, meaning that systems with a high variability in the input length will process data less efficiently. In all cases, system memory may also quickly become a factor limiting maximum batch sizes.

Another common method for combating slow inference times is to export the model to a format supported by a runtime such as ONNX runtime [6] or TensorRT [7]. Depending on the particular model being exported, the support for conversion to the ONNX model format may not be available due to incompatibilities with specific operations. New versions of the PyTorch to ONNX converter support transformer operations, but this feature was not available at the time of data collection for the models in this research. It is also worth noting that many runtimes, including TensorRT, use the ONNX as a common intermediate data format, meaning that without support for ONNX it is difficult to use the model with any runtimes other than the one it was developed with (i.e. Tensorflow or PyTorch). There are some additional options that do not rely on the ONNX model format, such as TVM [8], but these frameworks have their own limitations as well. TensorRT, if available, is a strong option for acceleration on embedded GPUs, as some researchers have reported substantial speedup with these frameworks. However, researchers have also observed non-deterministic execution time and memory behaviors with TensorRT [9].

The third option, which is explored in this research, is using a high-level framework like HuggingFace or Lightning Flash with built-in optimizations for data movement and loading. Unlike the aforementioned runtime frameworks, the high-level frameworks tested run the exact same underlying PyTorch model without modification. For this reason, one might assumed that the execution time across these frameworks should be roughly the same. However, the frameworks take different approaches to memory management and data movement, which can lead to overall faster or slower inference speed. These frameworks are also more likely to provide support or extensions for newer optimizations through community contributors rather than a single, isolated model designer would be able to keep up with on their own. HuggingFace in particular has been very quick to include emerging models, which allows engineers to migrate and test newer models with few code changes on their end system. Huggingface has also maintained extensions for optimizations like Optimum [5] and hardware-specific support.

B. NVIDIA Embedded GPUs

NVIDIA, known for its high-performance consumer- and server-grade GPUs, has an additional line of system-on-module (SoM) GPU platforms. These heterogeneous architectures combine an ARM CPU with an NVIDIA GPU on the same die and packaged in an embedded form factor. The entire system can be constrained to meet the requirements of severe and remote environments. Compared to their server-grade counterparts, these boards attain only a fraction of the memory bandwidth ($\sim$50GB/s vs $\sim$900GB/s) but also operate at significantly lower power ($\sim$10-30W vs $\sim$300W) [10], [11]. For these reasons, embedded GPUs are able to provide acceleration for sufficiently small or optimized models in remote or limited-power scenarios. This research leveraged the NVIDIA development kits, but these chipsets can be found in a wide variety of third-party single board computers.

While embedded GPUs exhibit desirable size and power properties, they are also less capable than their consumer- or server-grade counterparts. As most transcription models are trained on high-performance hardware, there is minimal benchmarking performance data for these edge devices. Embedded GPUs have a shared memory path between the CPU and GPU, which limits their performance for memory-bound applications, but also improves access speed for the GPU. The most important characteristics of embedded GPUs for the purpose of this research is the feasibility of deployment in scenarios ranging from remote-sensing to robotics.

The three devices examined in this research are the Jetson TX2, the Xavier NX, and the AGX Xavier. Each of these boards has roughly twice as many CUDA cores than its predecessor. The GPU on the Jetson TX2 is comprised of the older Pascal architecture while the NX and AGX are running feature the newer Volta architecture. These devices are all widely used in remote ML applications as they are all portable, self-contained, and have strong performance characteristics. The full descriptions of these devices and their power modes are located in Appendix A.

III. Related Research

Much of the value of ML in speech-processing domains is in the ability to train a single model to serve in place of previously hand-tuned or large, empirically derived statistical models. Following early recurrent neural network models [12], [13], the creation and application of the transformer with its unique self-attention mechanism [14] led to the development of a large number of transcription models using this backbone [1], [15]–[18].

Wav2vec2 is an improvement on both wav2vec [19] and wav2vec-vq [20]. As a whole, these models represent iterations on an open-source acoustic model for speech-to-text tasks. Wav2vec2 has a SOTA accuracy with a word error rate (WER) of 0.08. It uses Mel-Frequency cepstrums (frequency bands weighted by the sensitivity of the human ear) as inputs to a convolutional neural network (CNN) which generates feature vectors. These feature vectors are then provided to the transformer, which performs the bulk of the acoustic processing. Typically, these models are trained and used with a connectionist temporal classifier (CTC) [21], but have also been demonstrated outside of literature to be compatible with KenLM language models [22]. At the time of this writing there are now newer augmentations of wav2vec2 with slightly improved accuracy. These augmentations include integrating lessons and code from BERT-models for semisupervised learning [17], [23], the inclusion of SpecAugment to aid in pre-training [24], and generating a language model in an unsupervised fashion using text-based datasets [16].

Previous work in wav2vec2 benchmarking on embedded systems has been explored [25]. This previous research looked at a single device (Xavier NX) and compared the performance of the transformer-based model with and without clustering optimizations, as well as with respect to the device power mode and the input sequence length. It was concluded that these optimizations, which were originally intended for accelerating transformer training, had a measurable effect on performance and this effect was generally detrimental if input sequence lengths were short and beneficial if sequences were long. It was also shown that the clustering augmentation "improved-clustering" which increased the accuracy of the model, always resulted in a slow-down of the model inference. The supporting repository can be shared upon request.

IV. Approach

Benchmarking is accomplished primarily through a containerized application (targeting JetPack v4) which maintains the software environment for the test across devices and runs. This application must execute for latency measurements and a separate time for power measurements. This separation is required because the system calls that are used to poll the system power observably slow down the execution time. The frameworks tested are HuggingFace v4.18.0 and Lightning Flash v0.5.1. A pure, un-optimized PyTorch implementation is used as a baseline of comparison. The PyTorch benchmarking module [26] is used to control the number of iterations adaptively power measurements are averaged over 10-samples taken at 1 second intervals while the model is run repeatedly. The power rails sampled for each device can be referenced in

In addition to varying the inference framework, two parameter sweeps are performed in order to better understand the library operation on the device. The first sweep is input sequence length from 2 to 28 seconds and a batch size of 1. Wav2vec2 is trained on 32-bit floating-point data sampled at 16kHz, so data is generated at the same bit depth and and is described in terms of real-time representation (number of seconds that would be represented by this audio). 28-second inputs were the largest supportable by the Jetson TX2, and were therefore used as the maximum across the test. The other parameter that is varied in this study is the power mode of the device, which can be referenced in Appendix A. These modes vary the clock speeds of memory, CPU, and processors as well as the number of active cores.

At the time of development of this benchmark, it was observed that Lightning Flash was recreating the underlying PyTorch data loader with each prediction, substantially slowing down inference because of the significant setup and teardown time of the data-loader object. Since this behavior is more implementation or system-specific, a small patch was added by overriding the default predict call using a single re-used data loader. HuggingFace was observed to have instability with repeated executions over long periods of time. Since this was not the focus of the study, manual garbage collection and cache clearing was necessary to limit the effect of these leaks.

Despite performing the complete parameter sweep, many data points are excluded for the sake of brevity. For instance, the value from studying every power mode on every device is minimal, as many power modes differ primarily in number of active CPU cores which has a minimal effect on relative performance. In cases where trends are similar across all devices, the AGX will be the only device presented. The correlated data for all tests in this parameter sweep can be produced upon request to support other studies.

There are three metrics presented in the results. The first is the realtime factor, which describes the performance of the model relative to the speed at which data is generated with regards to temporal units. Note that some other literature presents realtime factor as a ratio of execution time to input length where lower values are preferred, but this value can drift into small decimal values. For the sake of clarity in regards to later performance metrics, realtime factor is shown here as the input sequence length in seconds divided by the execution time of the algorithm. For example, a realtime factor of 20 indicates that the model is processing data 20× faster than it can be generated. Higher is better for these values.

The second metric is throughput per watt. This metric divides the throughput of the model, in this case the realtime factor, by the average power used during that inference. While this unit is difficult to intuitively relate to real quantities, it is able to convey both performance and power considerations into a single value expressing the efficiency of a particular input size. Again, higher values are better.

The final metric is speedup. This metric is used as it is a succinct way to describe how much faster an algorithm is than its reference counterpart by dividing the execution time of the reference algorithm by that of the new algorithm. If the algorithm became faster, the metric is above 1 and if the algorithm experienced slow-down then the result will be below 1. Speedup is used in the results to highlight some specific behaviors of the frameworks presented.

V. Results

This section details results of benchmarking wav2vec2 in the HuggingFace and Lightning Flash frameworks as compared to a base-PyTorch implementation provided by fairseq. This includes primarily derived metrics, namely realtime factor, throughput per watt, and speedup. While this data was collected over all three devices in their default power modes, notable results are selected for brevity. The full dataset, with execution time and average power at each combination of parameters, is available upon request.

A. Execution Time

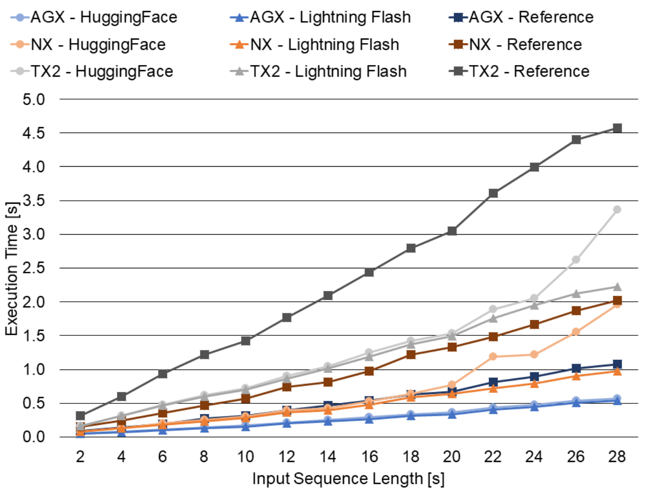


Fig. 1: Execution time of all devices in MAXN.

The overall measured latency of the tested devices is shown in Figure 1. The TX2 is the slowest device, with its worst execution time of 4.58 seconds. The AGX is the fastest device with 0.54 second execution times for the largest sizes. These execution times are approximately linear ($R > 0.9$), which echos similar results from previous research [25]. This approximate linearity helps predict execution times for these models at intermediate input sizes.

As expected from device specifications and previous literature, the AGX performs better than the NX, which performs better than the TX2. What is unexpected is how much worse the original PyTorch model performs relative to the inference frameworks. The difference between the Lightning Flash model and the reference model on the AGX is roughly twice the latency.

B. Realtime Factors Per Device

The realtime factors of each device for the tested frameworks can be seen in Figure 2. At 10-second input lengths, Lightning Flash has a realtime factor of 63.7, while HuggingFace has 59.0 and the reference model only achieves 32.1, which is similar to the measurements of the Xavier NX.

These frameworks are able to process data roughly twice as quickly as their base-PyTorch counterparts. While some discrepancy was expected, the degree of the improvement is substantial, especially for systems where execution times are particularly long. In this case when input lengths are around 30 seconds and inference times are on the order of seconds. From the perspective of latency and throughput, it is always preferable to use the higher-level frameworks over the bare-PyTorch reference implementation.

C. Performance by Device

The throughput per watt of each device in its MAXN power mode can be see in in Figure 3. Each device is displayed with a reduced set of power modes that represent the different power budgets associated with the device. Looking first at the AGX in Figure 3a, the higher power modes (2: 15W and 4: 30W) are shown to have the best performance for this metric. This result means that despite the higher average power, the hardware is utilized more efficiently. This trend does not hold for the other two devices, which are more efficient in lower power modes. Regardless of the power mode, the high-level frameworks were more performant than the base-PyTorch model.

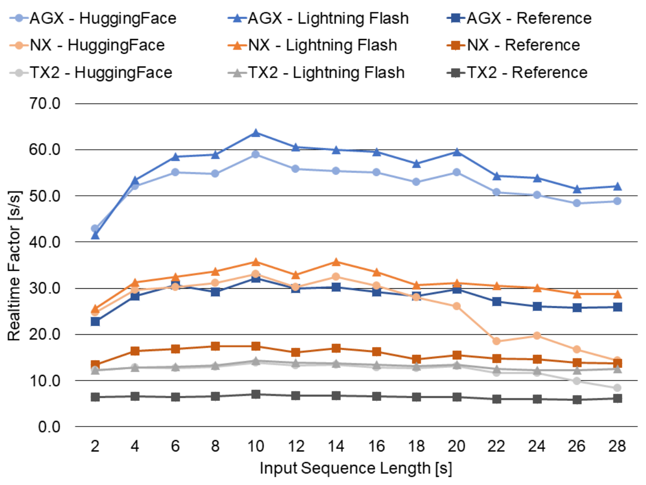
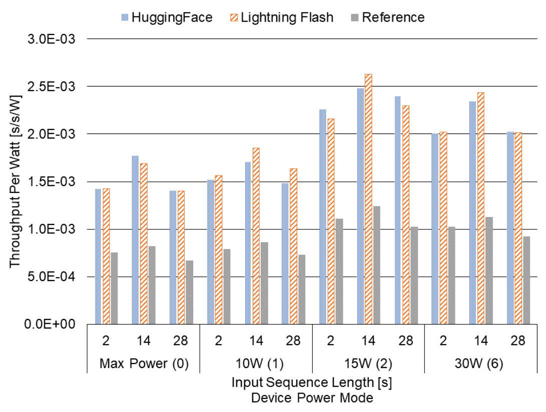
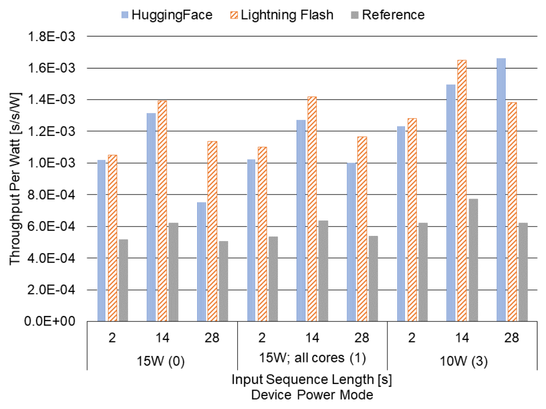


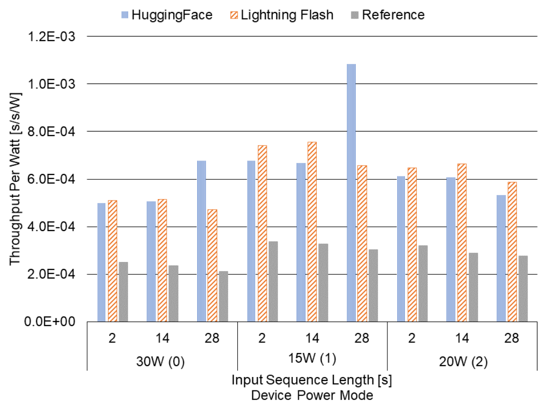
Fig. 2: Realtime factors of all devices across input space. Devices are shown in mode 0, which is the MAXN or equivalent mode for all devices. Errors are negligible. Higher values are better.



(a) AGX Xavier.



(b) Xavier NX.



(c) Jetson TX2.

Fig. 3: Throughput per watt of each device across representative power modes.

Between HuggingFace and Lightning Flash, the performance trends are less definitive. On the AGX, they perform similarly. However, for the NX, there are several instances where one framework dramatically outperforms the other. In power mode 0 on the NX, Lightning Flash performed better by this metric than HuggingFace, and slightly outperforms HuggingFace in most other points except for the largest size in the lowest power modes.

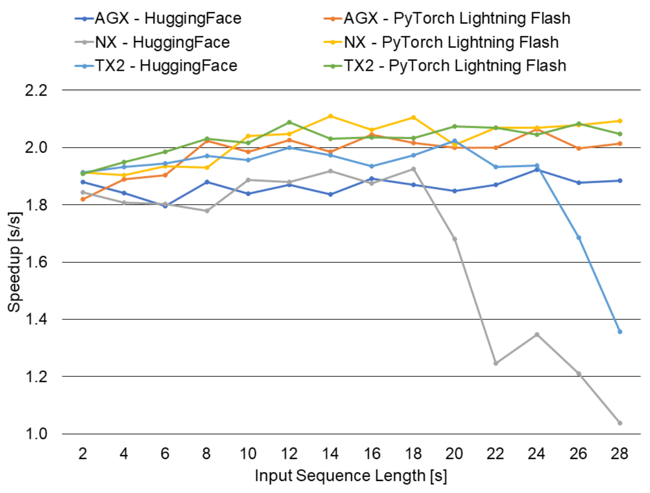


Fig. 4: Speedup of all devices in power mode 0.

D. Framework Speedup

The speedup derived from using these high-level frameworks can be seen in Figure 4. These speedup figures are calculated with respect to the reference model on the tested device with that input. Even in the worst case performance, these inference frameworks were all able to provide some speedup.

The worst-case behavior, however, does indicate an undesirable trend. HuggingFace experiences substantial speed losses for higher input sizes. This trend was also observed in Figures 3b and 3c. This loss is the least impactful for the AGX, and it most significant on the NX. The source of this issue is not immediately apparent and implementation-specific, and thus warrants further investigation. A possible source of high memory overhead could be the data loading mechanism used by the framework. HuggingFace's preferred method of performing inference is to construct an end-to-end data pipeline, which bundles together tokenization, model inference, and post-processing. This benchmark was constructed using HuggingFace's sample code for interacting with this model, which involves distinct calls to the tokenizers and model.

E. Discussion of Framework Drawbacks

While it seems to be a straightforward decision to use one of these high-level frameworks as they are able to outperform a manually-crafted model, it is also important to be mindful of the engineering difficulties associated with using these frameworks. Aside from additional dependencies which may not have the same long-term support as core frameworks like PyTorch will, there are two clear downsides. In this research, HuggingFace experienced slowdown at higher input sequence lengths, making HuggingFace less amenable for usecases where this is expected. Lightning Flash, on the other hand, is a newer framework that is still subject to growth and re-factoring that is less common in more stable frameworks like PyTorch.

VI. Conclusion

This research examined the utility of high-level inference frameworks to speed up the performance and increase the power efficiency of inference with the wav2vec2 model. It was found that these frameworks, specifically Huggingface and Lightning Flash, are roughly $2\times$ faster than reference PyTorch model on most tested inputs running on selected embedded GPU platforms. This speedup additionally enabled better throughput per watt for the system than the reference model, which equates to improved power efficiency of the entire system. These results indicate that optimizations built into the HuggingFace and Lightning Flash frameworks are observable and non-obvious from the base-PyTorch perspective. Moreover, despite calling the same underlying PyTorch model, HuggingFace and Lightning Flash have different performance characteristics due to the differences in their data loaders. HuggingFace was also shown to have undesirable scaling behavior at large input sizes and is not recommended for long audio sequences or repeated iterations.

Future work in this domain includes comparing these high-level training frameworks to runtime-specific frameworks, such as ONNX Runtime and TensorRT. At the time of data collection, this evaluation was not possible as many of the transformer operations in these models were not supported for PyTorch to ONNX converter, although could have been manually implemented through a stub. This barrier has been overcome, enabling benchmarking with ONNX and all of the runtimes that depend on ONNX or an intermediate data format. Another future direction could be examining the behavior of HuggingFace's slowdown to identify a solution to prevent this issue in out-of-the-box implementations of the model.

References

[1] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations," 2020. Publisher: arXiv Version Number: 3.

[2] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A Fast, Extensible Toolkit for Sequence Modeling," in *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[3] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: System demonstrations*, (Online), pp. 38–45, Association for Computational Linguistics, Oct. 2020.

[4] "Lightning-Universe/lightning-flash," June 2023. original-date: 2021-01-28T18:47:16Z.

[5] "HuggingFace Optimum," June 2023.

[6] Microsoft, "Onnxruntime," June 2023.

[7] NVIDIA, "TensorRT," June 2023.

[8] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," Oct. 2018. arXiv:1802.04799 [cs].

[9] O. Shafi, C. Rai, R. Sen, and G. Ananthanarayanan, "Demystifying TensorRT: Characterizing Neural Network Inference Engine on Nvidia Edge Devices," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 226–237, Nov. 2021.

[10] NVIDIA, "Whitepaper: NVIDIA Telsa V100 GPU Architecture: The World's Most Advanced Data Center GPU," Tech. Rep. August, NVIDIA, 2017.

[11] NVIDIA, "NVIDIA Jetson AGX Xavier Series System-on-Module," Aug. 2022.

[12] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep Speech: Scaling up end-to-end speech recognition," Dec. 2014.

[13] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin," *International Conference on Machine Learning*, vol. 48, pp. 173–182, 2016.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5999–6009, 2017. _eprint: 1706.03762v5.

[15] A. Baevski, W.-N. Hsu, Q. Xu, A. Babu, J. Gu, and M. Auli, "data2vec: A General Framework for Self-supervised Learning in Speech, Vision and Language," 2022.

[16] A. Baevski, W.-N. Hsu, A. CONNEAU, and M. Auli, "Unsupervised speech recognition," in *Advances in neural information processing systems* (M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), vol. 34, pp. 27826–27839, Curran Associates, Inc., 2021.

[17] W.-N. Hsu, B. Bolte, Y.-H. H. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed, "HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units," tech. rep., June 2021. eprint: 2106.07447v1.

[18] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv*, 2019. Publisher: arXiv _eprint: 1907.11692.

[19] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "wav2vec: Unsupervised pre-training for speech recognition," in *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2019-September, pp. 3465–3469, International Speech Communication Association, 2019. ISSN: 19909772 _eprint: 1904.05862.

[20] A. Baevski, S. Schneider, and M. Auli, "Vq-wav2vec: Self-Supervised Learning of Discrete Speech Representations," tech. rep., 2019. _eprint: 1910.05453v3.

[21] A. Graves, S. Fernández, and F. Gomez, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *In Proceedings of the International Conference on Machine Learning, ICML 2006*, pp. 369–376, 2006.

[22] L. V. Burchell, A. Birch, N. Bogoychev, and K. Heafield, "An open model and dataset for language identification," in *Proceedings of the the 61st annual meeting of the association for computational linguistics: ACL 2023*, (Toronto, Canada), July 2023. tex.month_numeric: 7.

[23] Y.-A. Chung, Y. Zhang, W. Han, C.-C. Chiu, J. Qin, R. Pang, and Y. Wu, "W2v-BERT: Combining Contrastive Learning and Masked Language Modeling for Self-Supervised Speech Pre-Training," Sept. 2021. arXiv:2108.06209 [cs, eess].

[24] Y. Zhang, J. Qin, D. S. Park, W. Han, C.-C. Chiu, R. Pang, Q. V. Le, and Y. Wu, "Pushing the Limits of Semi-Supervised Learning for Automatic Speech Recognition," July 2022. arXiv:2010.10504 [cs, eess].

[25] M. E. Schubert and A. D. George, "Benchmarking Transformer-Based Transcription on Embedded GPUs for Space Applications," in *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, (Bangalore, India), pp. 01–06, IEEE, July 2021.

[26] H. Schueroff and B. Johnson, "PyTorch Benchmark," Oct. 2020.

[27] NVIDIA, "Power Management for Jetson TX2 Series Devices," 2022.

[28] NVIDIA, "Hardware Architectural Specification — NVDLA Documentation," 2021.

Appendix A
Devices

A. *Jetson TX2*

The Jetson TX2 is the oldest SoM in this research and utilizes the NVIDIA Tegra chip. This device is included because of its wide-spread integration in 3rd-party Jetson boards and legacy in other projects. It is a less capable device than others in this research, but may be more reflective of hardware already in existing systems.

The properties of this mode can be seen in Table I. The SoC CPU is comprised of a quad-core ARM A57 and two NVIDIA Denver cores. The GPU runs on the older Pascal architecture with 256 CUDA cores. Additionally, this architecture does not contain Tensor Cores and was expected to perform much slower than its successors. The board used was equipped with 8GB of LPDDR4 memory and was running Ubuntu 18.04.

TABLE I: Summary of Jetson TX2 Power Modes [27]. Note that mode 4 is referenced in documentation, but was not present on the version of the TX2 used.

Mode ID	Estimated Power Budget (W)	Online A57 Count	Online D15 Count	A57 Max Frequency (MHz)	D15 Max Frequency (MHz)	GPU Max Frequency (MHz)
MAXN (0)	30W	4	2	2000	2000	1300
1	15W	4	0	1200	2000	850
2	20W	4	2	1400	1400	1120
3	20W	4	0	2000	1400	1120
4	N/A	0	2	2000	2000	1120

B. *Xavier NX*

This system has 6 Carmel ARM-based cores with an NVIDIA Volta GPU and 8GB of LPDDR4 memory. The GPU portion contains 384 CUDA cores, 48 Tensor Cores, and two Deep Learning Accelerators (DLAs). Tensor Cores are designed to accelerate tensor operations, specifically matrix multiplication [10]. DLAs are a structure that accelerate other deep-learning operations like convolution [28]. This board was running Ubuntu 18.04.

The GPU supports five standard power modes. The configurations of these power modes are summarized in Table II in Appendix A. There are two operational power budgets: 10W and 15W. Within each power budget, the main difference between power modes is number of available CPU cores and their operating frequency. This should not have an effect on the GPU operation as the function in question should be taking place exclusively on the GPU. For all modes, it is assumed that if fewer CPU cores are active, a smaller percentage of the power budget is used for the CPU, and the GPU may run at a higher power. All modes were considered as the CPU idle power was assumed to impact the maximum power available to the GPU.

TABLE II: Summary of Xavier NX Power Modes.

Mode ID	Power Budget (W)	Online CPU Count	CPU Max Frequency (MHz)	GPU Max Frequency (MHz)	Memory Max Frequency (MHz)
0	15	2	1900	1100	1600
1	15	4	1400	1100	1600
2	15	6	1400	1100	1600
3	10	2	1500	800	1600
4	10	4	1200	800	1600

C. *AGX Xavier*

The AGX Xavier is the largest of the embedded GPUs in the NVIDIA Jetson line. While it can be limited to run at 10W, it has a maximum power of 30W. The AGX device used was 32 GB. This board was also running Ubuntu 18.04. The summary of the power modes can be seen in Table III. It is important to note that to switch in or out of MAXN on this device, the board must be power-cycled. This board also runs the Volta architecture with 512 CUDA cores. Instead of the 6 Carmel cores of the NX, this board contains 8. Additionally, the AGX makes use of 64 Tensor cores and 2 DLAs.

TABLE III: Summary of AGX Xavier Power Modes.

Mode ID	Power Budget (W)	Online CPU Count	CPU Max Frequency (MHz)	GPU Max Frequency (MHz)	Memory Max Frequency (MHz)
0 (MAXN)	n/a	8	2265.6	1377	2133
1	10	2	1200	520	1066
2	15	4	1200	670	1333
3	30	8	1200	900	1600
4	30	6	1450	900	1600
5	30	4	1780	900	1600
6	30	2	2100	900	1600
7	30	4	2188	670	1333

APPENDIX B
POWER RAILS USED

TABLE IV: Power rails used for device power estimation.

Device	Rail Name	Description
Jetson TX2	VDD_SYS_GPU	GPU
	VDD_SYS_CPU	CPU
	VDD_SYS_DDR	DDR
	VDD_IN	Power regulator
	VDD_4V0_WIFI	Wifi
	VDD_SYS_SOC	SOC Fabric
Xavier NX	VDD_CPU_GPU_CV	CPU, GPU, Tensor Cores and DLAs
	VDD_SOC	SOC Fabric
	VDD_IN	Power Regulator
AGX Xavier	VDDRQ	DDR
	CV	Tensor Cores and DLAs
	GPU	GPU
	CPU	CPU
	SYS5V	System 5V
	SOC	SOC