Distributed Inference with Minimal Off-Chip Traffic for Transformers on Low-Power MCUs

Severin Bochem*, Victor J.B. Jung†, Arpan Suravi Prasad†, Francesco Conti‡, Luca Benini†‡
*D-ITET, ETH Zurich, Switzerland; †Integrated Systems Laboratory, ETH Zurich, Switzerland;
‡DEI, and Information Engineering, University of Bologna, Italy
sbochem@ethz.ch, {jungvi, prasadar, lbenini}@iis.ee.ethz.ch, f.conti@unibo.it

***Abstract*—Contextual Artificial Intelligence (AI) based on emerging Transformer models is predicted to drive the next technology revolution in interactive wearable devices such as new-generation smart glasses. By coupling numerous sensors with small, low-power Micro-Controller Units (MCUs), these devices will enable on-device intelligence and sensor control. A major bottleneck in this class of systems is the small amount of on-chip memory available in the MCUs. In this paper, we propose a methodology to deploy real-world Transformers on low-power wearable devices with minimal off-chip traffic exploiting a distributed system of MCUs, partitioning inference across multiple devices and enabling execution with stationary on-chip weights. We validate the scheme by deploying the TinyLlama-42M decoder-only model on a system of 8 parallel ultra-low-power MCUs. The distributed system achieves an energy consumption of 0.64 mJ, a latency of 0.54 ms per inference, a super-linear speedup of 26.1 ×, and an Energy Delay Product (EDP) improvement of 27.2 ×, compared to a single-chip system. On MobileBERT, the distributed system's runtime is 38.8 ms, with a super-linear 4.7 × speedup when using 4 MCUs compared to a single-chip system.**

***Index Terms*—TinyML, Transformer Models, Multi-chip Systems**

I. Introduction

Transformer models [1] have revolutionized the landscape of AI by achieving breakthroughs in areas such as Natural Language Processing (NLP) and Computer Vision (CV) [2]. The success of transformer-based language models such as BERT [3], GPT [4], or Llama [5] is largely due to their capability to capture contextual relationships within data, which makes them particularly appealing to use in contextual AI tasks commonly found in smart glasses, including personalized assistance and context-aware interactions.

Despite their success, deploying these Transformers on resource-constrained devices at the extreme edge presents formidable challenges, resulting from their high computational and memory requirements. Conventional Transformer models, which feature many millions to many billions of parameters [3], [6], are inherently too large to fit within the computation and memory budget of edge devices, necessitating reliance on off-chip memory or even cloud services. This dependency results in higher latency, increased power consumption, and privacy concerns, all critical in wearable devices. Smart glasses represent a promising wearable platform with the potential to enhance user experience through contextual AI [7]. By enabling seamless interaction with the environment, they could provide users with context-aware responses that enhance everyday life. However, deploying Large Language Model (LLM)s on such edge devices is infeasible due to their size and computational requirements. Tackling this challenge, Small Language Model (SLM)s with tens to a few hundred million, rather than several billion parameters have been proposed [8]–[10]. Still, even for SLMs, one main bottleneck during Transformer inference on smart glasses is the limitation in on-chip memory, which typically does not exceed 8 MiB [11]. Even for small Transformer models, weights and intermediate tensors might need to be stored and accessed from off-chip memory, which is both latency and energy-intensive.

Previous works have explored the distribution of Transformer models on multiple compute units, thus leveraging vast computational resources to execute intensive LLM workloads. These distributed methods [12], [13] allow Transformer models to meet their computational and memory requirements by partitioning workloads across multiple nodes, thereby overcoming the limitations of a single compute node. However, most of these works target high-performance computer architectures like Central Processing Units (CPUs), Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs). While data centers focus on increasing throughput and parallelism by batching tokens from multiple users to reduce the memory boundedness of the workload, the same methods cannot be applied to edge devices that require real-time and sequential processing. Moreover, edge devices such as smart glasses are subject to very strict constraints on latency, power, and form factor compared to cloud systems. This discrepancy calls for a novel approach to deploy Transformers at the edge, requiring careful optimization of latency, power consumption, and form-factor constraints for devices like smart glasses.

To tackle the challenges above, we propose a distributed inference scheme to facilitate the efficient deployment of SLMs on resource-constrained Systems-on-Chip (SoCs).

We deploy our scheme on a network of Siracusa [14] chips designed for smart glasses, featuring a cluster of 8 parallel RISC-V cores with instruction extensions for Machine Learning (ML) and Digital Signal Processing (DSP) workloads.

Our approach enables running TinyLlama with 42 million parameters [15] and MobileBERT [10] models solely from on-chip memory [16], with minimal overhead associated with inter-chip communication. The main contributions of our paper include:

- A strategy to partition the Transformers' Decoder and

978-3-9826741-0-0/DATE25/© 2025 EDAA

Encoder onto a distributed system of MCUs. This scheme minimizes chip-to-chip communication and needs only two synchronizations per Transformer block. The weights are scattered and never duplicated to reduce the on-chip memory footprint. This strategy enables individual Transformer blocks to be run only from on-chip memory, leading to lower energy per inference and super-linear latency reduction.

- Benchmarking of our partitioning strategy for the autoregressive and prompt mode of the decoder-only TinyLlama model as well as MobileBERT's encoder. We extended our results with a scalability study on up to 64 Siracusa MCUs to test the limits of our approach.

We perform experiments using the open-source event-driven simulator GVSoC [17]. From the simulator, we extract latencies and the number of accesses to different memory levels, which are fed into an analytical model to estimate the system energy. Our partitioning improves the performance of autoregressive Tinyllama inference by $26.1\times$, while incurring a similar energy per inference when using 8 chips compared to a single chip. This demonstrates a super-linear scaling for the autoregressive TinyLlama mode, as it relies solely on on-chip memory to run a single Transformer layer. By eliminating long-latency off-chip memory accesses during inference, we achieve the aforementioned super-linear speedup. A scaled-up model achieves $60.1\times$ performance improvement and $1.3\times$ energy reduction for 64 chips, showing our approach's scalability to larger networks. In the prompt mode of TinyLlama, using 8 chips improves performances more than linearly by $9.9\times$. Finally, for the MobileBERT model, using 4 chips improves performance by $4.7\times$ per chip without costing any additional energy per inference.

II. Background

A. Transformer Networks

Transformers have revolutionized the field of NLP and achieved State of the Art (SotA) performance in many other domains, such as CV. In NLP, encoder-decoder and decoder-only models dominate, while for CV, mostly encoder-only models are employed. Despite their large memory and compute demands, Transformers have found use in resource-constrained environments [18]. The two main building blocks of a Transformer are the Multi-Head Self-Attention (MHSA) and the Full-Connected Layer. Due to its computing intensity and high memory footprint, the MHSA is the most challenging to deploy, especially on resource-constraint devices.

The dimensions specifying the Transformer operations are the *sequence length S*, the *embedding dimension E*, the *projection dimension P*, and the *head dimension H*. The first step projects the input $X \in \mathbb{R}^{S\times E}$ onto the queries, keys, and values, $Q, K, V \in \mathbb{R}^{S\times P}$ as shown in equation 1.

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_{\text{query}}\,, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_{\text{key}}\,, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_{\text{value}} \tag{1}$$

In the Attention step, Q, K and V are combined by

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) := \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d}}\right)\mathbf{V}, \tag{2}$$

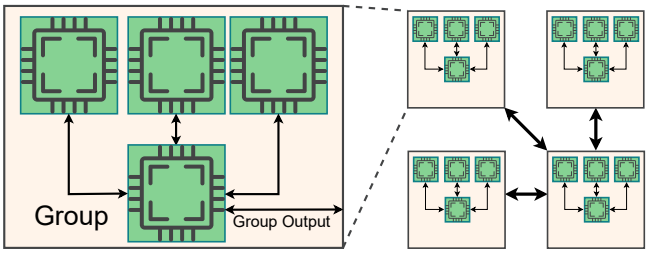

Fig. 1: Hierarchical interconnection of the Siracusa chips in the proposed system. Chips are placed in groups of four for improved scalability of the system. We use MIPI for the chip-to-chip link.

where d is the dimension of K used to scale the Attention. The softmax function is applied to each row of the matrix and defined as

$$\text{softmax}(\boldsymbol{x})_i = \frac{e^{x_i - \max(\boldsymbol{x})}}{\sum_{j=1}^{n} e^{x_j - \max(\boldsymbol{x})}} \tag{3}$$

for the *i-th* element of a row of size n. This operation is performed independently for each of the H heads.

The output of the MHSA is fed into the Fully-Connected Layer consisting of two Linear layers, a row-wise normalization, and a Gaussian Error Linear Unit (GELU) [19]. The shapes of the weight matrices in the linear layers are $E \times F$ and $F \times E$ respectively, where F is the *intermediate* dimension of the Transformer model.

This paper focuses on two different modes of Transformer inference, namely autoregressive and prompt mode. In autoregressive mode, each output token is predicted sequentially, based on all the previously predicted tokens using a data structure called Key-Value (KV)-Cache to store results of previous computations. In prompt mode, multiple outputs get predicted from multiple inputs simultaneously in one inference. Therefore, the main kernel of prompt mode inference is a General Matrix Multiply (GEMM), whereas in autoregressive mode General Matrix-Vector Multiply (GEMV) operations are dominant, which implies that prompt mode is more computationally intensive than autoregressive mode.

B. Deployment platform

We partition the model on a multi-chip architecture consisting of multiple generic Siracusa chips as shown in Fig. 1. As chip-to-chip link, we use the Mobile Industry Processor Interface (MIPI) serial interface with $100\ \mathrm{pJ\,B^{-1}}$ energy consumption and $0.5\ \mathrm{GB\,s^{-1}}$ bandwidth. All-reduce operations are performed hierarchically in groups of four to reduce the contention on the interconnect, as shown in Fig. 1.

Each chip of the multi-chip architecture is a Siracusa [14] low-power, heterogeneous RISC-V MCU which features an accelerator cluster of eight RISC-V cores, enabling Single Program Multiple Data (SPMD) processing [14], [23]. An overview of the Siracusa architecture is depicted in Fig. 2. To keep assumptions about the deployment platform minimal and the setup general, we do not use Siracusa's N-EUREKA

TABLE I: Comparison of SotA works on model partitioning of machine learning inference

Work	Model	Scale	Platform	Pipelining	Weight Duplication
Deepthings [20]	CNN	Low-Power	Raspberry Pi	No	Yes
Efficiently Scaling Transformer Inference [13]	Transformer	Datacenter	TPU	No	No
DeepSpeed Inference [12]	Transformer	Datacenter	GPU	Yes	No
When the Edge Meets Transformers [21]	Transformer	Low-Power	CPU	No	Yes
Hermes [22]	Transformer	Low-Power	CPU	Yes	No
Ours	Transformer	Extreme Edge	Siracusa (MCU)	No	No

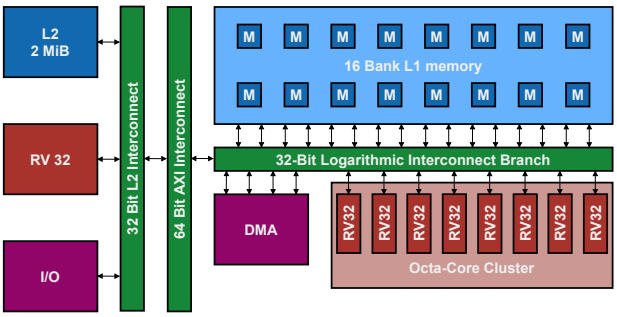

Fig. 2: Overview of the generic Siracusa SoC including an octa-core RISC-V cluster and host controller (red), memory hierarchy with two levels of scratchpad memory, two arbitrated interconnects towards the L1 memory and an Advanced eXtensible Interface (AXI) interconnect (green), and peripherals such as the cluster Direct Memory Access (DMA) and chip-level I/O (purple). Note that the image does not depict the N-EUREKA accelerator, as it was not used in this work.

accelerator. To enable single-latency access from cluster cores to the L1 Tightly Coupled Data Memory (TCDM), the cores are connected to the 16 L1 memory banks through a logarithmic interconnect using one 32-bit port each, granting a total memory bandwidth of $256\,\mathrm{bit/cycle}$ to the compute cluster. While each chip is equipped with significant computing capabilities, its on-chip memory is not sufficient to run inference of SLMs such as MobileBERT [10] or TinyLlama [9].

III. Related Work

A. Small Language Models

Foundation Models (FMs), such as decoder-only LLMs, like Llama [24] and Mixtral [25] come with large compute and memory demands, often requiring TBs of storage which makes them challenging to deploy on edge devices. SLMs address this gap by condensing Large Language Models LLMs into tens to hundreds of MBs. Some notable examples of SLMs include TinyLlama [9], the Phi series [26], [27] and MobileLLM [28]. Methods like incorporating high-quality data [27] and structured pruning techniques [29] aim to improve the efficacy of SLMs.

Embedding FMs into edge devices may enable a new wave of intelligent, responsive, and autonomous devices such as smart glasses. This work contributes to the goal of efficiently deploying SLMs on edge devices by proposing and benchmarking a partitioning scheme that can be applied to a wide range of FMs, from autoregressive decoder-only to encoder-only ones.

B. Distributed Model Inference

One main bottleneck of Transformer inference on edge devices is the available on-chip memory that can be used to store model weights and intermediate tensors. Each Siracusa chip used in this work features only $256\,\mathrm{KiB}$ in L1 and $2\,\mathrm{MiB}$ in L2 memory (See II-B).

For Deep Neural Networks (DNNs), a commonly used method to overcome the bottleneck of available on-chip memory is to partition the inference workload across multiple devices to reduce the memory and compute demands for each chip. Deepthings partition Convolutional Neural Network (CNN) inference across multiple Internet of Things (IoT) devices by splitting its input feature maps [20]. Follow-up works like EdgeFlow [30] introduced support for network and device heterogeneity. However, these methods are all tailored towards CNN inference and are not directly applicable to Transformer models. [21]. Recent work from Google partitions Transformer inference across multiple TPUs [13] tailored towards data center applications and inference of models with more than 500 Billion parameters. This makes memory considerations vastly different from the inference of small models at the edge. Groq proposes a software-defined datacenter-scale system that aims to minimize off-chip memory access during inference [16]. PipeEdge [31] partitions Transformer models on edge devices leveraging pipeline parallelism. Hermes [22] chooses a similar pipeline parallel approach. However, pipeline parallelism is infeasible for real-time single-user applications like smart glasses as it requires a sufficient batch size to keep the pipeline utilized and is unable to optimize the latency of an individual request. Another work [21] that aims for low-power Transformer inference targets CPU applications and needs to replicate model weights across devices. While this approach can reduce computational demands, the reliance on off-chip memory persists. An overview of previous works on distributed model inference can be found in table I.

In this work, we propose a tensor parallelism-based distributed inference scheme across Siracusa chips to facilitate the efficient deployment of SLMs on resource-constraint low-power MCUs. By not having to replicate any model weights, this partitioning enables running TinyLlama [9] and MobileBERT [10] solely from on-chip memory. This is especially beneficial in models that are bound by memory rather than compute latency, such as the autoregressive mode of TinyLlama. With

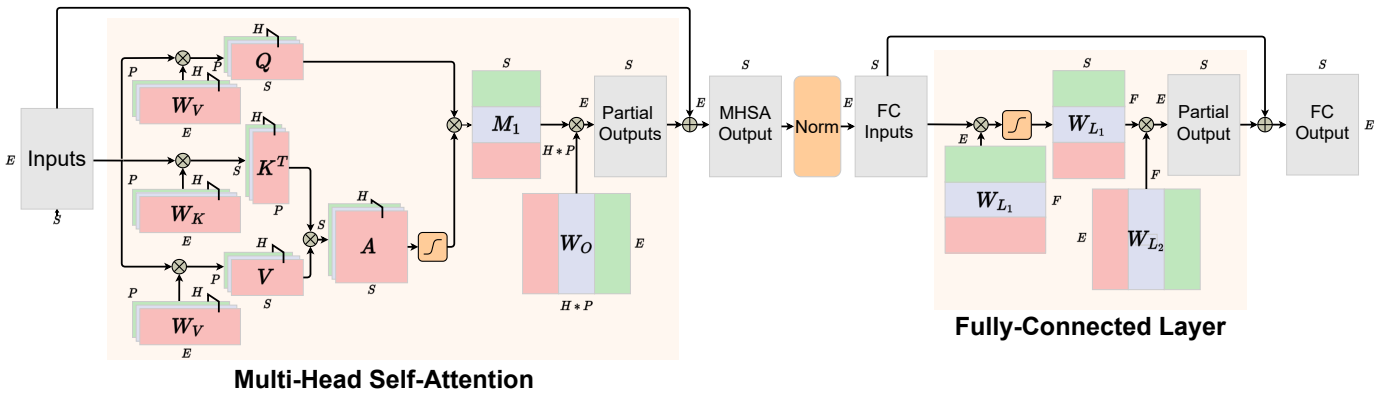

Fig. 3: Partitioning of Transformer Inference for three chips. Tensor colorings indicate on which chip a tensor is present. Tensors with grey coloring are present in all chips. Softmax and Norm are shown in orange.

previous approaches, these models would not fit into on-chip memory [21] or would lead to an insufficient chip utilization [31] for real-time inference. This partitioning scheme does not face the high communication cost common for tensor parallelism, as communication between chips is minimized.

IV. Partitioning Scheme

A visualization of our partitioning of the MHSA can be found in 3. In this example, we assume an MHSA with 3 attention heads distributed across 3 chips for visualization purposes. The input to the MHSA is broadcast to all chips. The weight matrices W_Q, W_K, and W_V are evenly split across chips, which results in each chip holding one slice of the weight tensors of shape $E \times P \times \frac{H}{Num\_Chips}$, divided across the attention head dimension. Note that in Fig. 3, we assume $H = Num\_Chips = 3$ for ease of visualization. Each chip will hold a partition of dimension $S \times P \times \frac{H}{Num\_Chips}$ of the tensors Q, K and V. Partitioning the MHSA across the head dimension is favorable, as the computations along H are fully independent of one another, requiring the chips to communicate only once after the MHSA.

Each chip holds a slice of the W_O matrix of shape $\frac{H*P}{Num\_Chips} \times E$, which is applied to a slice of the intermediate tensor of shape $S \times \frac{H*P}{Num\_Chips}$. After the partial MHSA, each chip holds a partial output of shape $S \times E$, which means that an all-reduce operation is needed before the normalization can be applied. As an all-to-one reduce operation lacks the required scalability, we perform a hierarchical reduction in groups of chips. First, a reduction is applied in a group of four chips by sending all partial outputs to one specific chip of the group, on which the partial outputs are accumulated. The outputs of this reduction are then again reduced until the final output of the MHSA is computed on one of the chips as visualized in Fig. 1. The skip connection from the MHSA input to the output shown in Fig. 3 can be merged into the all-reduce operation as all chips hold the full input tensor. After this output is normalized on a single chip, it is then broadcast back to all chips in the same manner as it is reduced.

For the Fully-Connected (FC) layer, we perform a similar approach. Both weight matrices of the fully connected stage W_{L_1} and W_{L_2} are sliced across the F dimension across chips, requiring no weight replication and resulting in each chip holding a slice of shape $E \times \frac{F}{Num\_Chips}$ of the W_{L_1} tensor and a slice of shape $\frac{F}{Num\_Chips} \times E$ of the W_{L_2} tensor. Similar to the MHSA, each chip produces a partial output of shape $S \times E$. From these partial outputs, the final output is produced in an all-reduce operation while also considering the skip connection. Note that this partitioning scheme replicates no weights across chips, which is crucial to save in on-chip memory of edge devices for Transformer applications. Furthermore, it requires only two synchronizations of chips at the end of the MHSA and fully connected layer.

V. Evaluation & Results

A. Experimental Setup

We conduct experiments using the open-source event-driven simulator GVSoC [17] to emulate the multi-chip architecture consisting of multiple Siracusa-like chips. From GVSoC, we obtain the latency and the number of accesses to each memory level, which is used by an analytical model to extract the energy consumption. For chip-to-chip interconnects, we use an analytical model of MIPI with $100\ \mathrm{pJ\,B^{-1}}$ energy consumption and $0.5\ \mathrm{GB\,s^{-1}}$ bandwidth [32]. The energy is computed analytically, assuming $100\ \mathrm{pJ\,B^{-1}}$ for accessing L3 memory and $2\ \mathrm{pJ\,B^{-1}}$ for accessing L2 memory. The average power consumption of one core is $13\,\mathrm{mW}$ [14] and the cluster of each SoC runs at $500\ \mathrm{MHz}$ [14]. The total system energy is computed as follows:

$$E_{\text{Total}} = N_{C2C} * E_{2C2} + \sum_{j=1}^{\#(\text{ Chips })} P * T_{Comp,j} + N_{L_3 \leftrightarrow L_2,j} * E_{L_3 \leftrightarrow L_2,j} + N_{L_2 \leftrightarrow L_1,j} * E_{L_2 \leftrightarrow L_1,j}.$$

Where P is the average power consumption, $T_{Comp,j}$ denotes the computation time of the chip j, N_{C2C} is the number of chip-to-chip transfer, $N_{L_2 \leftrightarrow L_1,j}$ and $E_{L_2 \leftrightarrow L_1,j}$ are the number of transfers and the transfer energy between L1 and L2 for the chip j, respectively.

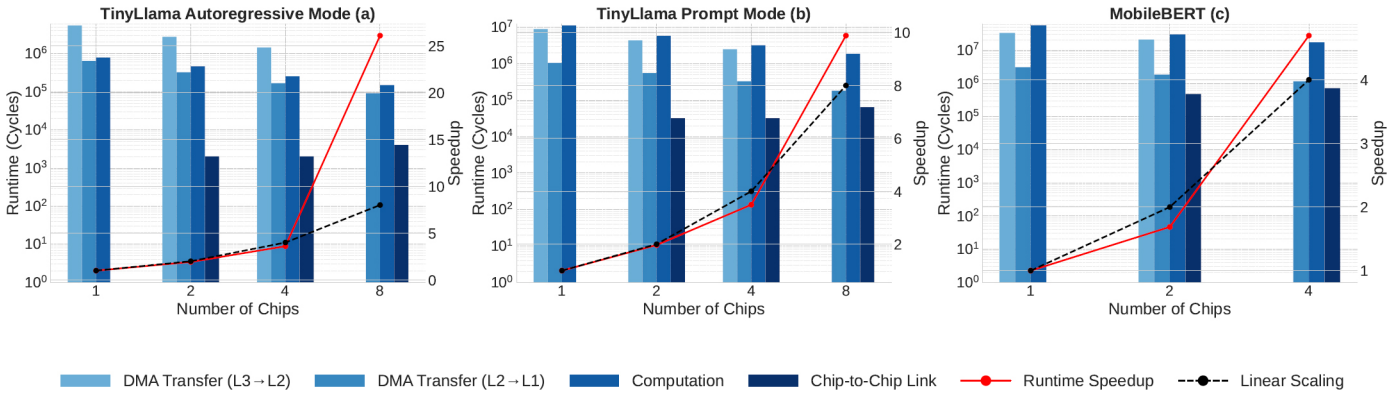

Fig. 4: Results of the MobileBERT model and TinyLlama model in prompt and autoregressive modes. The lines indicate the speedup when using 1-8 for TinyLlama or 1-4 for MobileBERT compared to a single-chip system. The bar plot shows the breakdown of runtime into computation, chip-to-chip communication, and access to L2 and L3 memory.

To deploy the model partitions, we extend the open-source Open Neural Network Exchange (ONNX) compiler Deeploy [33] that is tailored for Transformer models on edge devices. As workloads, we run a TinyLlama [15] and MobileBERT [10] model. We take the TinyLlama model from an open-source implementation with an embedding dimension E of 512, an intermediate size of 2048, and 8, matching the configuration of the model released initially. In autoregressive mode, this model leverages a KV-Cache to avoid unnecessary recomputation. We distribute the TinyLlama model across up to 8 chips. For our scalability study, we use a modified version of TinyLlama, containing 64 heads, and perform inference distributed on up to 64 chips. To do so, we leave all other model parameters unchanged. We use TinyLlama with a sequence length of 128 for autoregressive mode and 16 for prompt mode. The MobileBERT model has an embedding dimension and intermediate size of 512, 4 attention heads, and a sequence length of 268.

In our experiments, we depict the runtime and energy for a single Transformer block. The weights of the next Transformer block are loaded into L2 memory from L3 memory during the execution of the current block in a double-buffered fashion.

B. Runtime and Energy Consumption

In the following subsection, we showcase the results of our partitioning scheme with the setup and networks described in V-A. First, we partition the autoregressive and prompt modes of TinyLlama and MobileBERT models in their original configuration across Siracusa chips. Fig. 4 shows runtime speedup results and a runtime breakdown for all three models. Fig. 5 depicts the energy and latency for all three models in a 2D plot. Note that Fig. 5 also contains results of our scalability study that we address in Sec. V-C.

In autoregressive mode, we achieve a speedup of 26.1×, when using 8 chips, compared to using only a single chip, resulting in a super-linear scaling as seen in Fig. 4 (a). Super-linear speedup is not achieved for 1, 2, and 4 chips because the model weights of one TinyLlama block are too large to fit on the aggregated on-chip memory. Hence, for 1, 2, and 4 chips, many off-chip transfers are required in the execution of one transformer block, and they are the major contributor to the total runtime. Fig. 5 (a) shows that using 8 chips reduces the energy consumption per inference. This is a consequence of minimizing the chip-to-chip connection, not replicating model weights across chips, and storing intermediate tensors in L2 instead of L3.

In prompt mode, inference latency is reduced by 9.9× when using 8 chips over a single chip, which again leads to a super-linear runtime scaling as shown in Fig. 4 (b). Fig. 5 (b) shows that the energy consumption is reduced when using 8 chips, as, similar to autoregressive mode, we don't need off-chip transfer to process the current layer when 8 or more chips are used. Fig. 4 (a) and (b) show clearly that in autoregressive mode, accessing memory is the main contributor to overall runtime, whereas, in prompt mode, computation is the largest contributor. Therefore, in prompt mode, reducing the number of off-chip transfers to L3 leads to less speedup compared to autoregressive mode, as the workload is not bottlenecked by off-chip memory transfer in the first place.

Finally, Fig. 4 (c) and Fig. 5 (c) depict the latency and energy for the partitioning of MobileBERT. Partitioning on 4 chips results in a super-linear speedup of 4.7× due to the suppression of off-chip transfers to L3. However, using 4 chips results in a slight increase in inference energy. This is caused by the increased partitioning that scales down the kernel size of the Transformer. Therefore, it becomes more challenging to achieve high utilization of the RISC-V cores in each chip, which slightly hurts energy efficiency. In particular, for example, the runtime of a GEMM kernel does not scale down linearly as the overall kernel size is reduced, resulting in a runtime reduction that is less than linear at the network scale.

C. Scalability Study

Next, we study the scalability of our partitioning scheme to a larger number of chips. We increase the number of heads of the TinyLlama model from 8 to 64 while keeping the other parameters constant. Fig. 6 shows the speedup of the scaled-up

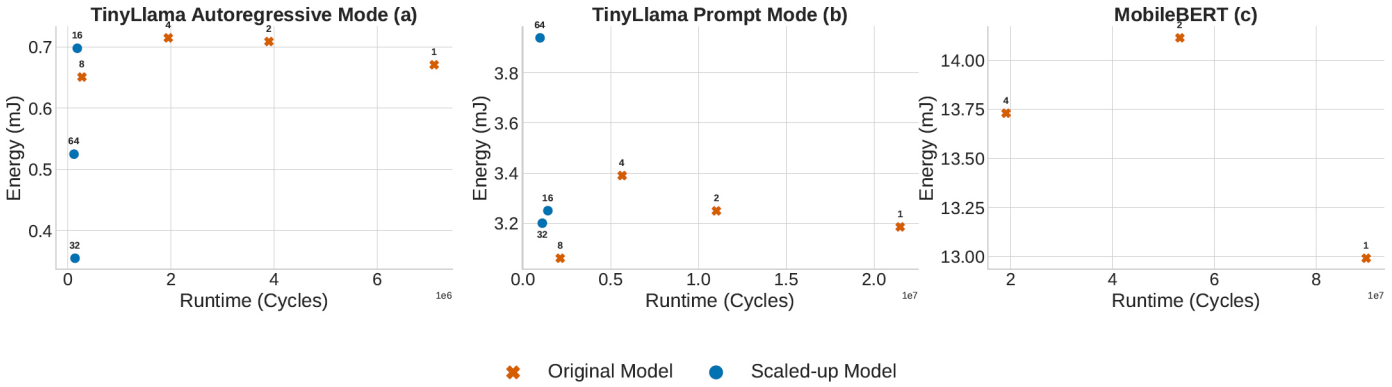

Fig. 5: This figure depicts runtimes and energies of TinyLlama in autoregressive mode (left), TinyLlama in prompt mode (middle), and MobileBERT (right). Red crosses are results obtained for models in their default configuration, whereas red circles show results for the scaled-up models.

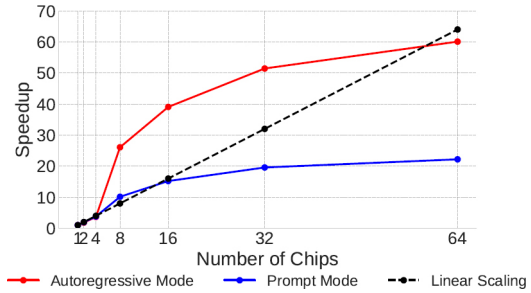

Fig. 6: Speedup on a scaled up TinyLlama model on 2-64 chips compared to a single-chip system.

model for both the autoregressive and prompt mode up to 64 chips.

In autoregressive mode, we achieve a speedup of $60.1\times$ using 64 chips instead of a single-chip system, demonstrating that our partitioning scheme achieves a quasi-linear speedup. Additionally, the energy consumption per inference is reduced by $1.3\times$ as shown in Fig. 4 (a). For 8 and 16 chips, we achieve a super-linear speedup for an individual Transformer block as one block can be run accessing only on-chip memory, whereas, for 1, 2, and 4 chips, off-chip memory is required to hold model weights and intermediate tensors of the current block. During the processing of one layer, the weights of the next layer can be loaded into on-chip memory, incurring an additional energy penalty. However, with 32 chips, all model weights fit on-chip, and double-buffering is no longer required, resulting in a further energy reduction that can be observed in Fig. 5 (a).

In the prompt mode of TinyLlama inference, we achieve a linear speedup up until a 16-chip system as seen in Fig. 6. Scaling the system further has diminished returns as the prompt mode is dominated by computation, and saving in off-chip memory accesses has a reduced benefit compared to autoregressive mode. Furthermore, the GEMM kernel's runtime scale is sub-linearly as the dimensions are reduced. Additionally, the number of chip-to-chip transfers and the accumulation of partial tensors introduce a larger overhead. Similar to the autoregressive mode, model weights need to be double buffered for 8 and 16 chips, whereas for 32 and 64 chips, on-chip memory is sufficient to hold all model weights, which results in reduced inference energy as can be seen from Fig. 5 (b).

Overall, the results demonstrate the scalability of our partitioning scheme for Transformer-based models, especially for models dominated by off-chip transfer to higher-level of the memory hierarchy, such as the autoregressive TinyLlama model for which we achieve super-linear speedup for 8-32 chips and a quasi speedup for 64 chips.

VI. Conclusion

In this paper, we presented a partitioning scheme tailored for deploying Transformer models on edge devices. With an approach inspired by tensor parallelism, this partition does not replicate any model weights across chips and only requires two synchronizations between chips, which allows the deployment of larger Transformer models at the extreme edge. We benchmark the partitioning scheme on the TinyLlama and MobileBERT models and demonstrated an above linear speedup of $26.1\times$ for autoregressive and $9.9\times$, for autoregressive and prompt TinyLlama mode, respectively, when using 8 chips instead of a single chip system. For MobileBERT, we achieve a speedup of $4.7\times$. To demonstrate the applicability to larger models, we showcase the scalability of our model. This work contributes to the active research field of deploying powerful Transformer-based models in highly resource-constraint devices.

VII. Acknowledgement

This work has received funding from the Swiss State Secretariat for Education, Research, and Innovation (SERI) under the SwissChips initiative. This work is funded in part by the dAIEDGE (#101120726) and CONVOLVE (#101070374) projects supported by the EU Horizon Europe research and innovation program.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[2] E. Xie, W. Wang, Z. Yu *et al.*, "Segformer: Simple and efficient design for semantic segmentation with transformers," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 12 077–12 090.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019.

[4] O. et al., "Gpt-4 technical report," 2024.

[5] H. Touvron, T. Lavril, G. Izacard *et al.*, "Llama: Open and efficient foundation language models," 2023.

[6] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," 2022.

[7] R. Konrad, N. Padmanaban, J. G. Buckmaster *et al.*, "Gazegpt: Augmenting human capabilities using gaze-contingent contextual ai for smart eyewear," 2024.

[8] R. Eldan and Y. Li, "Tinystories: How small can language models be and still speak coherent english?" 2023.

[9] P. Zhang, G. Zeng, T. Wang *et al.*, "Tinyllama: An open-source small language model," 2024.

[10] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "MobileBERT: a compact task-agnostic BERT for resource-limited devices," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds., Jul. 2020.

[11] H. Kwon, K. Nair, J. Seo, J. Yik, D. Mohapatra, D. Zhan, J. SONG, P. Capak, P. Zhang, P. Vajda, C. Banbury, M. Mazumder, L. Lai, A. Sirasao, T. Krishna, H. Khaitan, V. Chandra, and V. Janapa Reddi, "Xrbench: An extended reality (xr) machine learning benchmark suite for the metaverse," in *Proceedings of Machine Learning and Systems*, D. Song, M. Carbin, and T. Chen, Eds. Curan, 2023.

[12] R. Y. Aminabadi, S. Rajbhandari *et al.*, "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2022.

[13] R. Pope, S. Douglas, A. Chowdhery *et al.*, "Efficiently scaling transformer inference," *CoRR*, 2022.

[14] A. S. Prasad, M. Scherer, F. Conti *et al.*, "Siracusa: A 16 nm heterogenous risc-v soc for extended reality with at-mram neural engine," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 7, pp. 2055–2069, 2024.

[15] A. Karpathy, "llama2.c: Inference llama 2 in one file of pure c," https://github.com/karpathy/llama2.c, 2023, accessed: 2023-10-01.

[16] D. Abts, G. Kimmell, A. Ling *et al.*, "A software-defined tensor streaming multiprocessor for large-scale machine learning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 567–580.

[17] N. Bruschi, G. Haugou, G. Tagliavini *et al.*, "Gvsoc: A highly configurable, fast and accurate full-platform simulator for risc-v based iot processors," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, Oct. 2021.

[18] A. Burrello, M. Scherer, M. Zanghieri *et al.*, "A microcontroller is all you need: Enabling transformer execution on low-power iot endnodes," in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, 2021, pp. 1–6.

[19] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2023.

[20] Z. Zhao, K. M. Barijough *et al.*, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[21] C. Hu and B. Li, "When the edge meets transformers: Distributed inference with transformer models," in *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. Los Alamitos, CA, USA: IEEE Computer Society, 2024.

[22] X. Han, Z. Cai, Y. Zhang, C. Fan, J. Liu, R. Ma, and R. Buyya, " Hermes: Memory-Efficient Pipeline Inference for Large Models on Edge Devices ," in *2024 IEEE 42nd International Conference on Computer Design (ICCD)*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2024, pp. 454–461.

[23] E. Flamand, D. Rossi, F. Conti *et al.*, "Gap-8: A risc-v soc for ai at the edge of the iot," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–4.

[24] H. Touvron, L. Martin *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023.

[25] A. Q. Jiang, A. Sablayrolles, A. Roux *et al.*, "Mixtral of experts," 2024.

[26] Y. Li, S. Bubeck *et al.*, "Textbooks are all you need ii: phi-1.5 technical report," 2023.

[27] S. Gunasekar, Y. Zhang, J. Aneja *et al.*, "Textbooks are all you need," 2023.

[28] Z. Liu, C. Zhao, F. Iandola *et al.*, "MobileLLM: Optimizing sub-billion parameter language models for on-device use cases," ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 21–27 Jul 2024, pp. 32 431–32 454.

[29] M. Xia, T. Gao, Z. Zeng *et al.*, "Sheared LLaMA: Accelerating language model pre-training via structured pruning," in *The Twelfth International Conference on Learning Representations*, 2024.

[30] Y. Hao, Y. Liu, Z. Wu *et al.*, "Edgeflow: Achieving practical interactive segmentation with edge-guided flow," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021, pp. 1551–1560.

[31] Y. Hu, C. Imes, X. Zhao *et al.*, "Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 298–307.

[32] J. Gomez, S. Patel, S. Sarwar *et al.*, "Distributed on-sensor compute system for ar/vr devices: A semi-analytical simulation framework for power estimation," 03 2022.

[33] M. Scherer, L. Macan, V. J. B. Jung, P. Wiese, L. Bompani, A. Burrello, F. Conti, and L. Benini, "Deeploy: Enabling energy-efficient deployment of small language models on heterogeneous microcontrollers," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, Nov. 2024.