VEXP: A Low-Cost RISC-V ISA Extension for Accelerated Softmax Computation in Transformers

Run Wang*, Gamze Islamoglu*, Andrea Belano†, Viviane Potocnik*
Francesco Conti†, Angelo Garofalo†, Luca Benini*†
*IIS, ETH Zurich, Switzerland
runwang@ethz.ch, {gislamoglu, vivianep, lbenini}@iis.ee.ethz.ch
†DEI, University of Bologna, Italy
{andrea.belano2, f.conti, angelo.garofalo}@unibo.it

***Abstract*—While Transformers are dominated by Floating-Point (FP) Matrix-Multiplications, their aggressive acceleration through dedicated hardware or many-core programmable systems has shifted the performance bottleneck to non-linear functions like Softmax. Accelerating Softmax is challenging due to its non-pointwise, non-linear nature, with exponentiation as the most demanding step. To address this, we design a custom arithmetic block for Bfloat16 exponentiation leveraging a novel approximation algorithm based on Schraudolph's method, and we integrate it into the Floating-Point Unit (FPU) of the RISC-V cores [1] of a compute cluster, through custom Instruction Set Architecture (ISA) extensions, with a negligible area overhead of 1 %. By optimizing the software kernels to leverage the extension, we execute Softmax with 162.7× less latency and 74.3× less energy compared to the baseline cluster, achieving an 8.2× performance improvement and 4.1× higher energy efficiency for the FlashAttention-2 kernel in GPT-2 configuration. Moreover, the proposed approach enables a multi-cluster system to efficiently execute end-to-end inference of pre-trained Transformer models, such as GPT-2, GPT-3 and ViT, achieving up to 5.8× and 3.6× reduction in latency and energy consumption, respectively, without requiring re-training and with negligible accuracy loss.**

***Index Terms*—LLM, transformer, flashattention, softmax, neural network acceleration, exponential function, RISC-V**

I. Introduction

Transformer-based models such as the GPT family [2] and the LLaMa family [3], have emerged as a cornerstone of machine learning, demonstrating state-of-the-art performance in diverse domains, including natural language processing (NLP), computer vision, and audio processing. These models leverage pre-trained representations on large-scale unlabeled datasets, enabling remarkable accuracy improvements in fine-tuned downstream tasks such as sentence classification and question answering. At the core of their success is the Transformer architecture [4], which utilizes the self-attention mechanism to model complex relationships within input sequences.

Despite the interest in deploying Transformer-based models on mobile and edge devices, their substantial computational and memory requirements present challenges in meeting the resource and energy constraints of these devices. In encoders and the prefill stage of decoders, the computational complexity of attention layers scales quadratically with the input sequence length, leading to memory and computational overheads that necessitate mitigation by means of dedicated acceleration. Although many architectures utilize General Matrix-Matrix Multiplication (GEMM) acceleration to alleviate the computational burden, performance bottlenecks are increasingly shifting toward non-linear operations, especially the Softmax function within the attention layers.

This work was supported by the NeuroSoC project, funded under the European Union's Horizon Europe research and innovation programme (Grant Agreement No. 101070634).

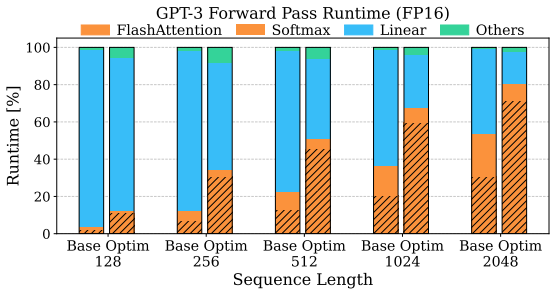

Fig. 1. Runtime breakdown for GPT-3 on a RISC-V multi-cluster platform [5]. For each sequence length, the left bar shows unoptimized GEMM results, while the right bar reflects optimized GEMM results.

Accelerating Softmax poses challenges due to its non-linear, non-pointwise nature and its reliance on a transcendental function, i.e. the exponentiation. The low arithmetic intensity of Softmax constrains parallelism and processing efficiency, a limitation that becomes more pronounced as GEMM latency decreases with acceleration. For example, the runtime breakdown for BERT on Volta GPU from Steven et al. [6] shows that Softmax contributes more than 30% for long sequences. Moreover, as shown in Figure 1, deployment of GPT3-XL on the RISC-V multi-cluster platform [5] reveals that Softmax contributes to 30% of the runtime prior to GEMM operator acceleration, and 70% afterwards for sequence length of 2048.

The use of large accelerators is justified for GEMMs, which constitute the majority of a Transformer's workload in terms of number of operations. However, allocating considerable silicon area for Softmax acceleration is sub-optimal, as it represents only a small portion of the overall computational workload. Hence, addressing these challenges necessitates innovative solutions that optimize the Softmax function with minimal area costs while preserving accuracy.

Existing software-level optimizations [7] often fall short of delivering the accuracy, performance and efficiency improvements necessary for large-scale or low-power deployments. Although hardware accelerators improve performance and energy efficiency, they typically lack flexibility due to their dependence on fixed-function datapath, and often lead

2576-2265/25/$31.00 ©2025 IEEE
DOI 10.1109/ARITH64983.2025.00016

to considerable integration area overhead in existing systems. Moreover, achieving accuracy parity often necessitates re-training [6] [8]—a technique that is undesirable, and often impractical for Large Language Models (LLMs).

In contrast, the growing adoption of the open and extensible RISC-V ISA to design domain-specialized programmable compute units offers a promising approach for addressing these limitations. In this work, we identify the exponential function as the primary computational bottleneck in Softmax computation, and we accelerate it on RISC-V-based systems, with low hardware overhead, through specialized ISA extensions. We demonstrate significant performance and energy efficiency gains in accelerating Transformer inference at native precision (Bfloat16), without compromising model accuracy, and without jeopardizing area and power consumption of the compute system. The contributions of this paper are:

- We design a custom arithmetic block for accelerating the exponential function on Brain Floating-Point (Bfloat16 or BF16) data, integrate it with ultra-low overhead in the FPU of programmable RISC-V processors of an octa-core compute cluster, and extend their ISA with a custom EXP instruction;
- We implement the cluster using GlobalFoundries 12 nm technology down to silicon-ready design and demonstrate that our solution incurs only a 1.0% area overhead at the cluster level and a negligible power overhead of 1.8% on workload with peak FPU utilization that do not exploit EXP, while the energy required to execute exponential operation is reduced by two orders of magnitude;
- By exploiting the proposed ISA extensions, we optimize the execution of Softmax in software, demonstrating 162.7× latency reduction and 74.3× less energy consumption compared to a non-optimized kernel on the baseline cluster and achieving 1.4× better area efficiency and 7.4× lower power consumption compared to state-of-the-art Softmax accelerators, as detailed in Section VI. Moreover, we integrate our fully optimized Softmax kernel into FlashAttention-2, showing 8.2× performance and 4.1× energy efficiency improvements;
- We scale up the proposed cluster to a 16-cluster system to evaluate the acceleration capabilities of our solution on end-to-end execution of pre-trained, un-tuned Transformers. We benchmark models such as GPT-2, GPT-3, and Vision Transformers (ViT), achieving up to 5.8× latency reduction and up to 3.6× less energy consumption compared to a baseline system without the proposed optimizations. Notably, these gains are achieved without re-training and with an accuracy loss of less than 0.1%.

II. Related Work

Optimization techniques for Softmax can be broadly divided into two categories: workflow scheduling and computational approximations. Workflow scheduling techniques such as FlashAttention [9] and FlashAttention-2 [10] enhance data reuse through a tiling technique, thereby improving both memory efficiency and parallelism. While these techniques do not directly target optimizing the Softmax computation kernel, they are complementary to the kind of optimizations explored in this work, which focus specifically on improving the efficiency of the computation kernel itself.

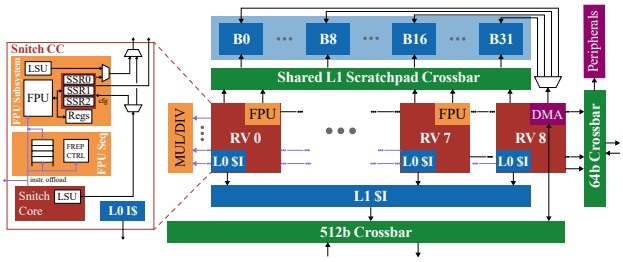

Fig. 2. Architecture of the RISC-V compute cluster with ISA extension FREP and SSR [1].

Computational approximations target the core Softmax operations: exponentiation and division. Full-precision exponentiation units based on iterative methods like Taylor series [11] and Coordinate Rotation Digital Computer (CORDIC) [12] offer accuracy but suffer from slow convergence, hence, long latency and high implementation costs. Lookup Table (LUT)-based methods [13] pre-compute values for faster computation but face scalability challenges due to high memory usage. Piecewise linear approximations [14] balance accuracy and efficiency but require input preprocessing, which can introduce additional overhead. Schraudolph's method [15] achieves fast exponential performance but is limited by accuracy. Division, a key component of Softmax normalization, further adds complexity. Methods like log-sum-exp [16] eliminate the need for division at the cost of logarithm computations. Alternatively, a single division can compute the reciprocal of the denominator, which is then multiplied by all values for normalization [6].

Recently, hardware accelerators for Softmax have emerged, particularly for Transformers. Most accelerators [6], [16]–[19] rely on fixed-point approximations for exponentiation and division, enabling efficient circuitry but complicating precision handling due to conversions from floating-point or integer formats. Methods like [20], [21] overcome this by operating directly on integer or floating-point formats without fine-tuning, while [6], [22] are tailored specifically for 8-bit quantized networks but rely on fine-tuning.

To achieve a better balance between flexibility and efficiency, we design a custom arithmetic block for the fast execution of the exponential function on BF16 data, a widely adopted precision for Transformers [23], based on an enhanced version of Schraudolph's method. We integrate it into RISC-V processors within a parallel compute cluster through lightweight custom ISA extensions, providing a detailed description and evaluation of the proposed approach, including accuracy and design tradeoffs, while demonstrating significant acceleration speed-ups, minimal implementation costs, and negligible accuracy loss. As highlighted in Section VI, our approach offers superior efficiency and flexibility compared to state-of-the-art Softmax accelerators, enabling end-to-end execution of Transformers without the need for fine-tuning.

III. Background

A. Snitch Cluster

Figure 2 illustrates the architecture of the Snitch cluster [1] that we use as a baseline. The Snitch cluster is an energy-

efficient compute architecture designed for high-performance workloads. It integrates eight RISC-V RV32IMAFD cores, each paired with a tightly coupled 64-bit Single Instruction Multiple Data (SIMD)-capable FPU supporting a wide range of data formats (FP64 to FP8, including BF16) and a private L0 instruction cache.

A 128 KiB, 32-banked scratchpad memory (SPM) is shared across the cluster, connected via a single-cycle logarithmic interconnect that delivers high-bandwidth, low-latency data access. A dedicated direct memory access (DMA) control core facilitates asynchronous data transfers between the SPM and external memory systems (e.g., HBM2E or other clusters), achieving bandwidths of up to 512 bit/cycle. To ensure efficient data movement, the hierarchical interconnect incorporates a 512-bit wide crossbar for L1 instruction cache and data access and a 64-bit crossbar for peripheral communication.

The architecture also supports advanced ISA extensions, including `FREP` (Floating-Point Repetition) [1] for hardware loops and `SSR` (Stream Semantic Register) [24] for managing data access with minimal software overhead. The `FREP` instruction configures the FPU sequencer to automatically repeat and autonomously issue the next n floating-point instructions to the FPU. The `SSR` extension allows the configuration of up to three memory streams with affine address patterns, effectively eliminating explicit memory operations.

B. FlashAttention

A Transformer consists of multiple blocks, each containing a multi-head attention (MHA) module and a feed-forward module. In MHA, token vectors are projected through query (Q), key (K), and value (V) matrices, with attention computed as $\text{Softmax}((QK^\top)/\sqrt{d_k})V$. FlashAttention [9] optimizes this computation by dividing Q, K, and V into blocks that can be efficiently processed in fast SRAM memory, thereby reducing costly HBM accesses. FlashAttention-2 [10] further enhances performance through optimized memory layouts and aggressive operator fusion.

For numerical stability, we adopt the Softmax function with maximum subtraction:

$$\text{Softmax}(x_i) = \frac{\exp(x_i - \max(\mathbf{x}))}{\sum_j \exp(x_j - \max(\mathbf{x}))}$$

, which requires storing the complete attention matrix and performing row-wise operations. To address this limitation, FlashAttention introduces *partial* Softmax, which processes blocks incrementally while maintaining running statistics (maximum values and exponential sums). For each new block, these statistics are updated and used to compute partial results, enabling numerically equivalence to standard Softmax while significantly reducing memory overhead. This online computation approach not only ensures numerical stability but also eliminates the need to materialize the full attention matrix in memory.

C. Execution Model on Snitch Cluster

Baseline Softmax: The baseline kernel is written in C without leveraging Snitch's extended ISA (FREP, SSR and SIMD). Data is transferred via DMA from HBM to the local SPM with double buffering to mask data marshalling latency while the eight Snitch cores process sequences in parallel. The division in Softmax is performed by the FPU's division block, and the exponential function is based on `math.h` library and uses a piecewise polynomial approximation method with software LUTs.

Baseline FlashAttention-2: Following the approach in [5], we adapt FlashAttention-2 to the Snitch cluster architecture with an optimized tiling strategy. The implementation first loads a Q tile to SPM via DMA, then iteratively transfers and processes corresponding K and V tiles. To maximize throughput, we employ double buffering for efficient overlap between memory transfers and computation. The tile size is optimized based on SPM capacity under double buffering constraints. Within each tile, both GEMM and partial Softmax computations are parallelized across the cluster cores. The partial Softmax computation is parallelized by having the eight cluster cores simultaneously compute multiple row statistics. The GEMM implementation leverages Snitch's specialized instruction-level optimizations as detailed in [5], which serves as the foundation for all GEMM operations in this work.

D. Exponential Approximation Algorithm

For efficient exponential computation, we adopt Schraudolph's method [15], which exploits the memory arrangement of floating-point numbers to approximate e^x with few basic operations. The input x is scaled to the base-2 domain as $x' = x/\ln(2)$, then decomposed into integer and fractional parts: $\text{int}(x') = \lfloor x' \rfloor$ and $\text{frac}(x') = x' - \lfloor x' \rfloor$. The approximation is reconstructed as $\exp(x) \approx 2^{\text{int}(x')} \cdot (1 + \text{frac}(x'))$. Based on the method proposed by Belano et al. [25], to enhance accuracy, the fractional term $(1 + \text{frac}(x'))$ is replaced with a polynomial $P(\text{frac}(x'))$, yielding:

$$\exp(x) \approx 2^{\text{int}(x')} \cdot (1 + P(\text{frac}(x'))). \tag{1}$$

To better approximate $2^{\text{frac}(x)}$, the interval $[0, 1)$ is split into two equal-length partitions, determined by the most significant bit of the mantissa. For each partition, a polynomial in the form $ax(x + b)$ is applied:

$$P(x) = \begin{cases} \alpha x\,(x + \gamma_1), & x \in [0, 0.5), \\ \text{not}\,(\beta\,\text{not}(x) \cdot (x + \gamma_2)), & x \in [0.5, 1). \end{cases} \tag{2}$$

Here, α, β, γ_1, and γ_2 are optimized for minimal error, with $1 - x$ approximated by $\text{not}(x)$ for hardware efficiency. Adjustments to γ_1 and γ_2 account for fixed-point arithmetic constraints. Parameters $\alpha = 0.21875$, $\beta = 0.4375$, $\gamma_1 = 3.296875$ and $\gamma_2 = 2.171875$ are derived via a heuristic Monte Carlo optimization with 10^6 trials by Belano et al. [25] to minimize the error between the true exponential function and its approximation.

IV. Methods

A. EXP Custom Arithmetic Block

Figure 3c shows the proposed arithmetic block to efficiently compute the approximation of exponential function on Bfloat16 data. This block is structured around the algorithm introduced in Section III-D and consists of two cascaded stages as described above: $exps(x)$, which implements the Schraudolph's method in hardware, and the subsequent $P(x)$, which performs the mantissa correction for improved precision.

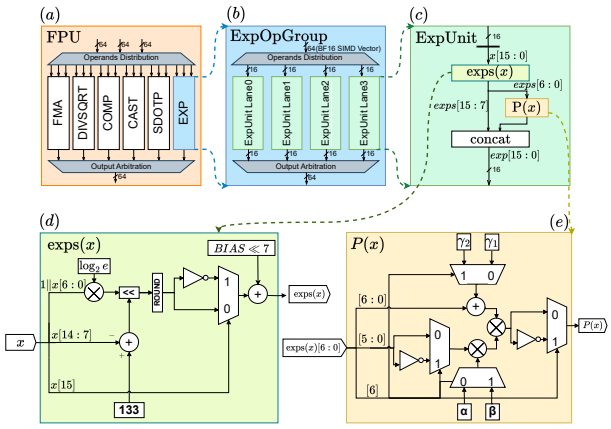

Fig. 3. Block diagram of (a) the extended FPU, (b) the *ExpOpGroup*, (c) the *ExpUnit*, (d) the $exps(x)$ stage, and (e) the $P(x)$ stage.

At the input of the $exps(x)$ stage (shown in Figure 3d), the data in Bfloat16 format is decomposed into its sign, exponent, and mantissa bits, with the implicit leading 1 appended to the latter. Next, the mantissa is multiplied by the precomputed constant $(\log_2 e)$, and the result of this multiplication is then shifted by an amount equal to the difference between the exponent of the argument and the maximum exponent after which the exponential function is guaranteed to overflow (133 in the case of BFloat16 numbers). Then, the first 15 bits of the shifted mantissa are selected and appropriately rounded to maintain precision, and finally, if no overflow occurs, the result is obtained by appending a leading zero (the sign bit of the result) to the rounded mantissa and adding the bias to the new exponent. If an overflow or infinity is detected, the output is assigned to either ∞ or 0, depending on whether the argument is positive or negative. For subnormal values, the data is flushed to zero following BFloat16 simplifications relative to IEEE-754 behaviour [23].

The second stage of the exponential computation, $P(x)$, corrects the mantissa component of the approximation. First, the MSB of the mantissa determines whether the input falls within $[0, 0.5)$ or $[0.5, 1)$, selecting the appropriate polynomial branch. In the first branch, corresponding to $x \in [0, 0.5)$, the polynomial $\alpha x(x + \gamma_1)$ is evaluated directly using fixed-point arithmetic. For $x \in [0.5, 1)$, the computation proceeds with $\mathrm{not}(\beta\,\mathrm{not}(x) \cdot (x + \gamma_2))$, where the bitwise complement operation approximates the evaluation of $1 - x$.

Finally, the output of the EXP block is obtained by concatenating the corrected mantissa from the $P(x)$ stage with the sign and exponent fields from the $exps(x)$ stage.

B. Snitch ISA Extension and Microarchitecture

To exploit the fast exponentiation of Bfloat16 data enabled by the EXP block described in Section IV-A while preserving software programmability, we integrate the arithmetic block into an open-source, modular, energy-efficient multiformat FPU [26] for RISC-V processors. The target FPU already supports a wide range of floating-point operations, which are organized into specific multi-format modules that can be enabled at design-time through parameters. For our evaluation, it integrates `FMA` (fused multiply-add), `DIVSQRT` (division and square root), `COMP` (comparison), `CAST` (conversion), and `SDOTP` (dot product) modules. We extend it with a new dedicated single-format module, namely *ExpOpGroup*.

TABLE I
SNITCH RISC-V ENCODINGS FOR FEXP AND VFEXP

Format	Encoding (32-bit)
FEXP `rd, rs1`	`001111100000{rs1}000{rd}1010011`
VFEXP `rd, rs1`	`101111100000{rs1}000{rd}1010011`

The new operation group takes as input a single N-bit SIMD vector containing Bfloat16 elements and produces a single N-bit SIMD output vector, as illustrated in Figure 3b. Depending on the data-width of the FPU, which is configurable at design time, the *ExpOpGroup* integrates $k = N$-bit$/16$-bit *ExpUnit* lanes. This is preceded by additional logic that segments the input SIMD vector into k 16-bit elements and distributes them to the *ExpUnit*s. To meet the timing requirements of the processor that integrates the FPU enhanced with the proposed block, the *ExpUnit* includes a configurable number of pipeline stages, which can be utilized for retiming.

Furthermore, the extended FPU is integrated into the micro-architecture of the Snitch cores of the parallel compute cluster introduced in Section III-A. Since the Snitch core supports double-precision instructions, the FP register file contains 32 64-bit wide registers and its datapath is 64-bit wide. At the interface, the FPU accepts three 64-bit input operands and produces one 64-bit output per cycle. This configuration allows the packing of four Bfloat16 operands into a single SIMD 64-bit register. As illustrated in Figure 3c, the *ExpOpGroup* is parameterized with four *ExpUnit* lanes, each equipped with one level of pipeline registers to streamline processing. This design allows for the completion of a single exponentiation operation in two cycles, while still permitting back-to-back operations without stalling, thereby maintaining a peak throughput of four Bfloat16 exponentiation operations per cycle.

To enable this operation in software, we extend the Snitch's RISC-V ISA with two domain-specific instructions, namely `FEXP` and `VFEXP`. The first instruction is designed for scalar Bfloat16 operations, activating only one *ExpUnit* in the micro-architecture, while `VFEXP` performs a packed-SIMD exponential computation that fully utilizes the underlying micro-architecture's capabilities. Both instructions execute with a latency of two clock cycles in the Snitch core. As shown in Table I, the instruction formats use `rd` (destination register) and `rs1` (source register) as 5-bit fields to address the FPU's 32×64 register file, with the most significant bit of the entire instruction distinguishing between scalar and packed-SIMD operations. The Snitch core's decoder and FPU subsystem are updated to support these instructions and seamlessly activate the *ExpOpGroup* in the FPU.

C. Optimized Softmax Kernel

To speed up the execution of the Softmax function on the enhanced Snitch cluster, we develop optimized software routines that exploit the underlying ISA of the Snitch cores, including the designed `VFEXP` instruction. Since the maximum value is required for the exponentiation step and must be computed by looping over each row of the resulting QK^T

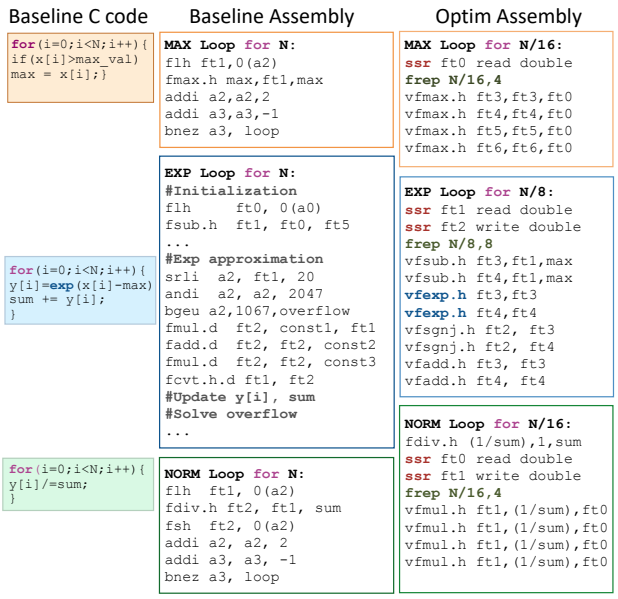

Fig. 4. Code comparison of Baseline and Optimized Softmax implementations. Baseline Softmax uses a piecewise polynomial approximation with software LUTs for the exponential (EXP) function, explicitly handling overflow to infinity and subnormals. The notation `frep n_frep, n_instr` represents a loop executing the following `n_instr` instructions for `n_frep` iterations. All `v` instructions in the code are packed-SIMD operations.

matrix, we construct the loop using the FREP instruction. As shown in Figure 4, targeting BF16 data and leveraging the core's 64-bit datapath, we utilize the VFMAX instruction in MAX step to balance computation and load costs - processing 4 SIMD operations per 64-bit data load. To streamline data loads and keep the datapath fully utilized, we exploit an SSR.

The results are then forwarded to the exponentiation step (EXP), where we maintain a similar kernel structure using FREP and SSR for efficient data loading. In this phase, we leverage the VFEXP instruction, which performs the exponentiation of a SIMD vector with 4 elements in 2 cycles. In contrast, the baseline kernel, described in Section III-C, computes the exponentiation in software with a latency of 319 cycles per BF16 item. For each computed exponential, we also accumulate the sum using VFADD within the same FREP−SSR loop.

Finally, we optimize the normalization step (NORM) by calculating $1/\text{sum}$ outside the loop and performing a point-wise scaling operation with a VFMUL instruction. Overall, these optimizations achieve 1.5 instructions/output, 2.125 cycles/output while also benefiting from loop unrolling advantages, significantly outperforming the baseline implementation, which requires 56 instructions/output, 360 cycles/output.

D. Optimized FlashAttention-2 Kernel

We optimize the partial Softmax part of the FlashAttention-2 kernel, which follows steps analogous to standard Softmax but performs them over multiple tiles. The optimization methods including FREP, SSR and SIMD instructions (VFEXP, VFMAX, VFSUB, VFMUL) are employed in the same manner for the partial Softmax in FlashAttention-2 for partial MAX, EXP, and NORM.

TABLE II
ACCURACY FOR GPT-2 AND VIT MODELS

Model	Dataset	Metric	FP32	BF16	BF16 EXP
GPT-2	WikiText	Perplexity (↓)	37.4	37.8	37.8
	ArcEasy	Accuracy (↑)	43.8	42.9	43.7
ViT-B	ImageNet	Accuracy (↑)	80.3	80.3	80.3
	CIFAR-10	Accuracy (↑)	98.5	98.5	98.5

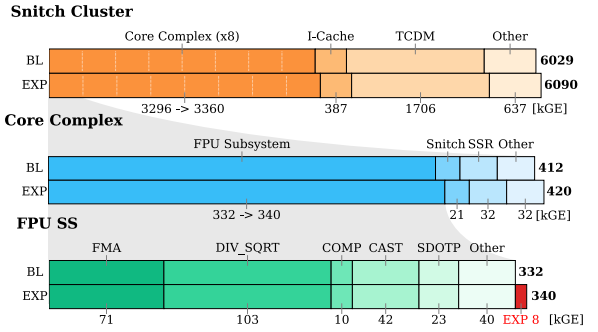

Fig. 5. Area breakdown of the Snitch cluster. BL: Baseline, EXP: Extended FPU with the EXP block.

V. EVALUATION AND RESULTS

A. Accuracy Analysis

Following Belano et al. [25], the proposed exponentiation algorithm achieves a mean relative error of 0.14% and a maximum relative error of 0.78% with respect to glibc's implementation. Building upon this, we evaluate the accuracy of our exponential implementation using pre-trained GPT-2 Small and ViT Base models. For GPT-2 Small, perplexity is measured on WikiText-2, and accuracy is measured on ARC Easy. For ViT Base, accuracy is evaluated on ImageNet-1K and CIFAR-10. Comparisons are made against FP32 precision, native BF16 casting, and BF16 casting with our optimized EXP implementation, which employs a software-simulated Schraudolph algorithm.

As shown in Table II, BF16 casting has minimal impact on the accuracy of the models. Moreover, our proposed EXP replacement demonstrates negligible differences compared to standard BF16 casting. These findings validate the proposed EXP approach as an efficient and accurate method for exponential computation, preserving model accuracy. Notably, this analysis demonstrates that Transformer models can be directly cast to BF16 without the need for re-training or fine-tuning, further highlighting the practicality of our approach.

B. Physical Implementation

We performed synthesis and place&route for one Snitch cluster with eight cores, 128 KiB of TCDM, and 8 KiB of instruction cache. Synthesis and implementation results are gathered with Synopsys' Fusion Compiler 2022.03 for GlobalFoundries 12nm FinFET technology.

For timing analysis, we constrained the design to 1 GHz. With the addition of the exponentiation block, the

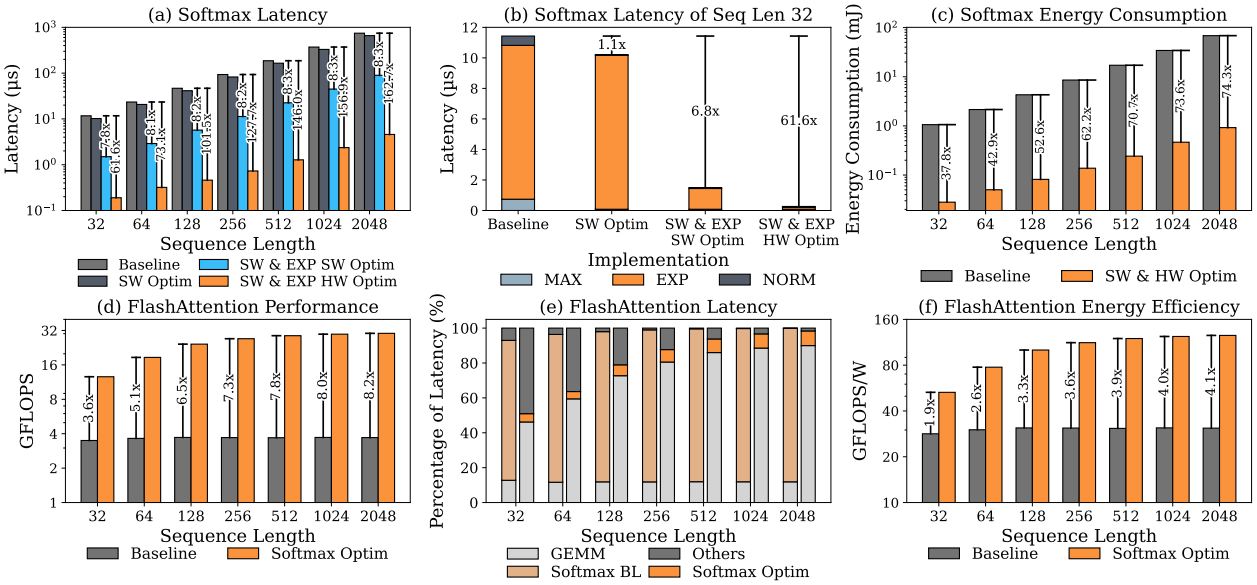

Fig. 6. Performance, latency, and energy analysis for Softmax and FlashAttention-2 kernels.

TABLE III
ENERGY PER OPERATION FOR GEMM AND EXP

Energy/Op [pJ/Op]	Snitch Baseline	ISA Extended
GEMM	3.96	4.04
EXP	3433	6.39

Snitch cluster achieved 1.15 GHz under typical conditions (TT/0.8 V/25 °C) without introducing any new critical paths. Under worst-case conditions (SS/0.72 V/125 °C), the design reached up to 941 MHz.

For the area analysis, we evaluated the cluster, core complex (comprising the integer core and the FPU subsystem), and FPU subsystem (SS), as shown in Figure 5. At the cluster level, the total area increased by 1.0% compared to the baseline Snitch cluster due to the increase in the area of the eight core complexes. At the core complex level, the FPU SS exhibited a 1.9% area increase relative to the baseline. Within the FPU SS, the addition of the EXP block accounted for 8 kilo Gate Equivalents[1] (kGE), corresponding to a 2.3% increase in the area of SS.

To measure power, we performed parasitics-annotated gate-level netlist simulations using Synopsys' PrimeTime 2022.03 under typical conditions (TT/0.8 V/25 °C). In Table III, GEMM kernels (48×48, 85% FPU utilization) are compared between the baseline Snitch and ISA-extended Snitch. Adding the EXP block increased Snitch cluster's average power by 1.8%, with energy per operation rising from 3.96 to 4.04 pJ/Op.

For EXP implementation, the new EXP instruction was benchmarked against the baseline method (piecewise LUT with polynomial approximation), which requires 319 cycles per call and has low FPU utilization (6.5%). The ISA-extended Snitch core performs exponential calculations in hardware in two clock cycles. During the execution of the EXP kernel, the ISA-extended Snitch's average power increased by 2.4×. However, the execution time dropped from 319 cycles/output to 0.5 cycles/output (with 4-way SIMD instruction `VFEXP`), reducing the energy from 3433 to 6.39 pJ/Op.

[1]One Gate Equivalent (GE) represents the area of a minimum-sized two-input NAND gate, which is 0.121 µm^2 in GF 12nm technology.

C. Benchmarks

Softmax: We evaluated four Softmax implementation configurations: the baseline described in Section III-C, an optimized version using Snitch's existing ISA extensions (`SW Optim`), a further optimized version incorporating the software-implemented Schraudolph exponential function (`SW & EXP SW Optim`), and a final version combining Snitch's ISA extensions with hardware acceleration via the EXP instruction (`SW & EXP HW Optim`). Performance benchmarks were conducted using ModelSim-2022.3, with the system running at 1 GHz.

In Figure 6a, our final implementation (`SW & EXP HW Optim`) achieve up to 162.7× speedup over the baseline, while software-only optimizations show minimal gains due to the exponential operation bottleneck. The Schraudolph method in software offers some acceleration but is far outperformed by hardware by a factor of 19.6×. Figure 6b demonstrates the negligible impact of MAX and NORM on total latency, with software achieving only a 1.1× speedup, compared to 61.6× for combined hardware and software optimizations. Finally, Figure 6c shows energy reductions of up to 74.3×.

FlashAttention-2: We evaluated the FlashAttention-2 kernel on one Snitch cluster with a head dimension of 64 (GPT2 configuration). The results, shown in the second row of Figure 6, highlight several improvements. In Figure 6d, our implementation achieves up to 8.2× increase in throughput over the baseline. Figure 6e illustrates that Softmax dominates the latency in the baseline, while its contribution is reduced to

TABLE IV
COMPARISON OF STATE-OF-THE-ART SOFTMAX ACCELERATORS

Ref	Precision	Accuracy [MSE]	Evaluated Model	Tech [nm]	Frequency* [GHz]	Area* [µm²]	Power [mW]	Throughput [GOPS]	Strategy
Zhu et al. [16]	FX16	1.06e-10/2.28e-12[1]	Transformer-XL	28	2.78/1.64[1]	10081/18392[1]	-	22.24/13.12†,[1]	FX16 quant.
Koca et al. [17]	FX16	-	BERT	*FPGA*	-	-	-	-	No fine-tuning
Kim et al. [18]	FX8/FX16	71.2e-12/4.77e-12[2]	-	28	3.12/2.5[2]	7100/24900[2]	22.82/52.46[2]	24.96/20†,[2]	-
Xia et al. [19]	FP16/FP32-FX[3]	-	BERT	*FPGA*	-	-	-	-	Fine-tuning
Yu et al. [20]	INT32/FP16/FP32	-	RoBERTa, MobileBERT	7	1.5/0.74/0.62[4]	1009/498/1134[4]	0.06/0.02/0.04[4]	-	No fine-tuning
Wang et al. [21]	INT8/FP32	-	DeiT, Swin, BERT	28	1	-	-	-	No fine-tuning
Liu et al. [22]	INT8-FP[5]	-	GPT-2	16	1.25	800	0.2	-	Training
Our	BF16	1.62e-9	GPT-2, ViT	12	1	968[6]	7.1[6]	0.45[6]	No fine-tuning

* Results are reported only for standalone designs (all synthesis results except for [18]). For our design, we present the frequency of the full cluster and the post-layout area.
† Denotes peak throughput, which may differ from average throughput.
[1] The precision of the design is adjustable. The first value corresponds to the lowest precision setting, while the second value represents the highest precision setting ($P = 3$) evaluated in the referenced paper.
[2] The accelerator supports two input precisions: FX8 (first) and FX16 (second). For FX16, the reported results correspond to the version with a parallelization factor of 8.
[3] Internal computations are performed in fixed-point format, with input and output values converted from and to floating-point format.
[4] Values are reported for INT32, FP16, and FP32, respectively.
[5] Internal computations are performed in floating-point format, with input and output values converted from and to INT8.
[6] For our design, the reported area corresponds to the EXP unit per core, while the power and throughput are averaged over the entire Softmax operation per core.

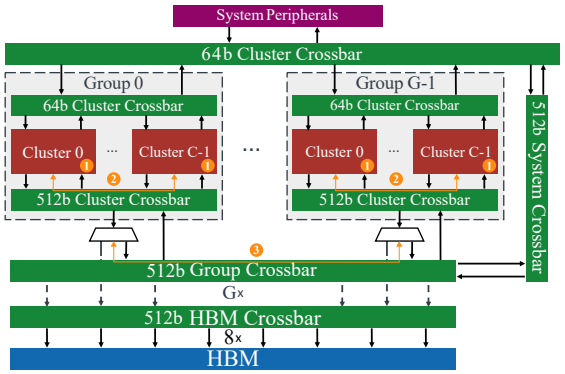

Fig. 7. Hierarchical multi-cluster architecture with heterogeneous memory interconnect: (1) Cluster-to-SPM interconnect, (2) Inter-Cluster communication, and (3) Inter-Group communication.

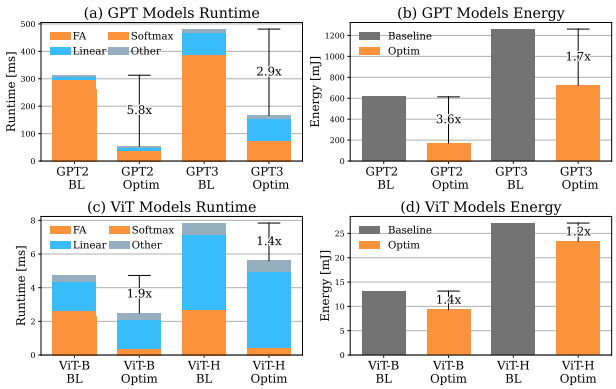

Fig. 8. Runtime and energy comparison of Softmax-optimized (Optim) system with the baseline (BL) for GPT and ViT models.

6% in the optimized version. Moreover, the energy efficiency of FlashAttention-2 improves up to 4.1× with the optimized Softmax as shown in Figure 6f.

D. Scalability Analysis

The Snitch cluster is designed to scale into a multi-cluster architecture, silicon-proven in Occamy [27]. As shown in Figure 7, a group of C compute clusters is connected by a 64-bit crossbar for fast synchronization and a 512-bit AXI crossbar for high-bandwidth inter-cluster access. Further scaling is achieved by linking G groups through a group-level AXI crossbar, enabling inter-group communication. Each group also interfaces with eight HBM channels through a wide crossbar, ensuring high-bandwidth access to main memory.

We benchmark runtime and energy metrics against [5] on GPT-2 Small, GPT-3 XL, ViT-Base, and ViT-Huge models. All models are evaluated non-autoregressively on a 16-cluster version of the Occamy system [27], with sequence lengths of 2048 for GPT models and 197 for ViT models. Following [5], we map each attention head to a single Snitch cluster, loading each Q tile from HBM to SPM via DMA and iteratively transferring and processing the corresponding K and V tiles.

As shown in Figure 8, the FlashAttention-2 kernel dominates runtime in the baseline implementation for both GPT and ViT models. With Softmax optimizations applied, overall runtime improves significantly, achieving speedups of 5.8×, 2.9×, 1.9×, and 1.4× for GPT-2, GPT-3, ViT-Base, and ViT-Huge, respectively. Similarly, energy consumption decreases substantially, with reductions of 3.6×, 1.7×, 1.4×, and 1.2× for these models, respectively.

VI. COMPARISON WITH THE STATE-OF-THE-ART

We compare our solution to state-of-the-art Softmax accelerators evaluated for Transformer models, as shown in Table IV. Unlike fully custom datapaths for all Softmax operations, our approach introduces an ISA extension to accelerate only the exponential function while optimizing the remaining operations in software. This hybrid method balances efficiency and flexibility, supporting a broader range of applications at a low cost.

Our approach employs BF16 precision and achieves a mean squared error (MSE) of 1.62e−9, which is comparable to fixed-point approximations by Zhu et al. [16] and Kim et al. [18]. In addition to this MSE, we demonstrate that our approximation preserves FP32/BF16 accuracy of GPT-2 and ViT-B, as detailed in Section V-A. Most other works do not evaluate their methods on LLMs but rather focus on smaller, encoder-only models. Although Liu et al. [22] achieves convergence to the same perplexity as the original GPT-2 during training, it remains unclear whether this approach can be applied without

fine-tuning. Moreover, their architecture is designed for INT8 inputs/outputs while internally utilizing FP16 precision.

Other works primarily report post-synthesis evaluations (except for [18]), omitting factors such as clock tree implementation and physical design. They also exclude timing, area, and power overheads arising from the integration of the custom datapaths into complex compute systems, making a thoroughly fair comparison impractical. Furthermore, while we report the average throughput per core over the entire Softmax computation, Zhu et al. [16] and Kim et al. [18] provide only peak throughput values, which neglect iterations required for sequence lengths exceeding the hardware-supported size of 8, as well as memory operations. Despite these limitations, our hybrid hardware-software approach, with a compact area footprint of $968\,\mu m^2$ per core, achieves $1.1\times$ better area efficiency (in terms of Op/cycle/mm^2) compared to the high-precision version of [16] and only $1.7\times$ lower area efficiency than the low-precision version, without compromising flexibility. Notably, our approach does not require fine-tuning or quantization. Furthermore, our method delivers $1.4\times$ greater area efficiency than the FX16 version of [18] while having $2.4\times$ lower area efficiency compared to the FX8 version. The lower power efficiency compared to [18] stems from the focus on optimizing the exponential function with higher precision, whereas [18] employs a softmax-specific hardware implementation with reduced fixed-point precision. Additionally, the reported power consumption accounts for the entire core over the full softmax computation, rather than only the exponential unit, with power consumption being $3.2\times$ (FX8) and $7.4\times$ (FX16) lower than that of [18].

VII. Conclusion

This work proposes a novel method to accelerate the Softmax function, a key bottleneck in Transformer models, by integrating a custom exponential instruction into the RISC-V Snitch architecture. Through hardware/software co-design, the approach achieves up to $162.7\times$ speedup, with $5.8\times$ and $3.6\times$ reductions in latency and energy for GPT-2 and ViT models. This research demonstrates the potential of RISC-V for energy-efficient AI in resource-constrained settings, balancing precision, power, and simplicity.

References

[1] F. Zaruba, F. Schuiki, T. Hoefler *et al.*, "Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1845–1860, Nov. 2021.

[2] Y. Guo, Y. Lang, and Q. Ren, "GPTQT: Quantize Large Language Models Twice to Push the Efficiency," Jul. 2024, arXiv:2407.02891.

[3] H. Touvron, T. Lavril, G. Izacard *et al.*, "LLaMA: Open and Efficient Foundation Language Models," Feb. 2023, arXiv:2302.13971.

[4] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[5] V. Potocnik, L. Colagrande, T. Fischer *et al.*, "Optimizing Foundation Model Inference on a Many-tiny-core Open-source RISC-V Platform," May 2024, arXiv:2405.19284.

[6] J. R. Stevens, R. Venkatesan, S. Dai *et al.*, "Softermax: Hardware/Software Co-Design of an Efficient Softmax for Transformers," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE Press, 2021, pp. 469–474.

[7] S. Kim, A. Gholami, Z. Yao *et al.*, "I-BERT: Integer-only BERT Quantization," Jun. 2021, arXiv:2101.01321.

[8] G. Islamoglu, M. Scherer, G. Paulin *et al.*, "ITA: An Energy-Efficient Attention and Softmax Accelerator for Quantized Transformers," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2023, pp. 1–6.

[9] T. Dao, D. Y. Fu, S. Ermon *et al.*, "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," Jun. 2022, arXiv:2205.14135.

[10] T. Dao, "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning," Jul. 2023, arXiv:2307.08691.

[11] J. van der Hoeven and F. Johansson, "Fast multiple precision exp(x) with precomputations," in *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*, Jun. 2024.

[12] H. Chen, L. Quan, and W. Liu, "HGH-CORDIC: A High-Radix Generalized Hyperbolic COordinate Rotation Digital Computer," in *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*, Jun. 2024.

[13] Q. Sun, Z. Di, Z. Lv *et al.*, "A High Speed SoftMax VLSI Architecture Based on Basic-Split," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, Oct. 2018.

[14] H. Dong, M. Wang, Y. Luo *et al.*, "PLAC: Piecewise Linear Approximation Computation for All Nonlinear Unary Functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, Sep. 2020.

[15] N. N. Schraudolph, "A Fast, Compact Approximation of the Exponential Function," *Neural Computation*, vol. 11, no. 4, May 1999.

[16] D. Zhu, S. Lu, M. Wang *et al.*, "Efficient Precision-Adjustable Architecture for Softmax Function in Deep Learning," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3382–3386, Dec. 2020.

[17] N. A. Koca, A. T. Do, and C.-H. Chang, "Hardware-efficient Softmax Approximation for Self-Attention Networks," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2023, pp. 1–5.

[18] J. Kim, S. Kim, K. Choi *et al.*, "Hardware-Efficient SoftMax Architecture With Bit-Wise Exponentiation and Reciprocal Calculation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 10, pp. 4574–4585, Oct. 2024.

[19] T. Xia and S. Q. Zhang, "Hyft: A Reconfigurable Softmax Accelerator with Hybrid Numeric Format for both Training and Inference," in *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '24, 2024, pp. 1–6.

[20] J. Yu, J. Park, S. Park *et al.*, "NN-LUT: neural approximation of non-linear operations for efficient transformer inference," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22, 2022, pp. 577–582.

[21] W. Wang, S. Zhou, W. Sun *et al.*, "SOLE: Hardware-Software Co-design of Softmax and LayerNorm for Efficient Transformer Inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, Oct. 2023, pp. 1–9.

[22] S. Liu, G. Tao, Y. Zou *et al.*, "ConSmax: Hardware-Friendly Alternative Softmax with Learnable Parameters," Nov. 2024, arXiv:2402.10930.

[23] N. Burgess, J. Milanovic, N. Stephens *et al.*, "Bfloat16 Processing for Neural Networks," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, Jun. 2019, pp. 88–91.

[24] F. Schuiki, F. Zaruba, T. Hoefler *et al.*, "Stream Semantic Registers: A Lightweight RISC-V ISA Extension Achieving Full Compute Utilization in Single-Issue Cores," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 212–227, 2021.

[25] A. Belano, Y. Tortorella, A. Garofalo *et al.*, "A Flexible Template for Edge Generative AI with High-Accuracy Accelerated Softmax & GELU," Dec. 2024, arXiv:2412.06321.

[26] L. Bertaccini, G. Paulin, T. Fischer *et al.*, "MiniFloat-NN and ExSdotp: An ISA Extension and a Modular Open Hardware Unit for Low-Precision Training on RISC-V Cores," in *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*, Sep. 2022, pp. 1–8.

[27] G. Paulin, P. Scheffler, T. Benz *et al.*, "Occamy: A 432-Core 28.1 DP-GFLOP/s/W 83% FPU Utilization Dual-Chiplet, Dual-HBM2E RISC-V-Based Accelerator for Stencil and Sparse Linear Algebra Computations with 8-to-64-bit Floating-Point Support in 12nm FinFET," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, Jun. 2024, pp. 1–2.