ConSmax: Hardware-Friendly Alternative Softmax with Learnable Parameters

Shiwei Liu[1,2†], Guanchen Tao[2], Yifei Zou[2], Derek Chow[1], Zichen Fan[2], Kauna Lei[2], Bangfei Pan[1], Dennis Sylvester[2], Gregory Kielian[1‡], and Mehdi Saligane[2‡]

[1]Google Research
[2]Department of Electrical Engineering and Computer Sciences, University of Michigan

Abstract—**The self-attention mechanism distinguishes transformer-based large language models (LLMs) apart from convolutional and recurrent neural networks. Despite the performance improvement, achieving real-time LLM inference on silicon remains challenging due to the extensive use of Softmax in self-attention. In addition to the non-linearity, the low arithmetic intensity significantly limits processing parallelism, especially when working with longer contexts. To address this challenge, we propose Constant Softmax (ConSmax), a software-hardware co-design that serves as an efficient alternative to Softmax. ConSmax utilizes differentiable normalization parameters to eliminate the need for maximum searching and denominator summation in Softmax. This approach enables extensive parallelization while still executing the essential functions of Softmax. Moreover, a scalable ConSmax hardware design with a bitwidth-split look-up table (LUT) can achieve lossless non-linear operations and support mixed-precision computing. Experimental results show that ConSmax achieves a minuscule power consumption of 0.2mW and an area of 0.0008mm^2 at 1250MHz working frequency in 16nm FinFET technology. For open-source contribution, we further implement our design with the OpenROAD toolchain under SkyWater's 130nm CMOS technology. The corresponding power is 2.69mW and the area is 0.007mm^2. ConSmax achieves 3.35× power savings and 2.75× area savings in 16nm technology, and 3.15× power savings and 4.14× area savings with the open-source EDA toolchain. In the meantime, it also maintains comparable accuracy on the GPT-2 model and the WikiText103 dataset. The project is available at https://github.com/ReaLLMASIC/ConSmax.**

Index Terms—**LLM, Transformer, Hardware-Software Co-Design, Softmax, ConSmax**

I. Introduction

Transformer-based LLM models have become foundational across a wide range of machine learning domains, including natural language processing [1, 2] and computer vision [3]. The notable improvement can be attributed to the unique self-attention mechanism. Different from previous convolutional or recurrent algorithms (CNN and RNN), Self-attention mechanisms enhance LLMs' ability to capture information across input contexts (i.e., tokens) regardless of their distance. Accelerating LLM inference on silicon is challenging due to its low arithmetic intensity, which results in poor energy efficiency. It prevents the further applications of LLMs, particularly on edge devices.

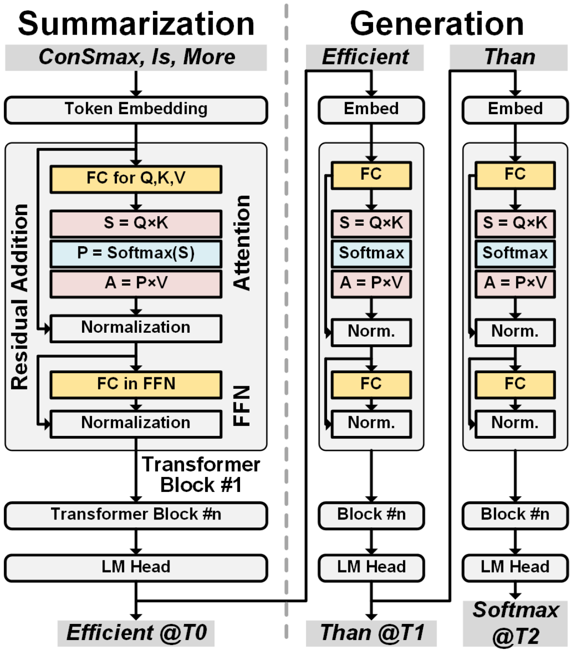


Fig. 1: LLM structure and processing flow with summarization and generation stages.

Softmax is the primary factor exacerbating the inefficiency of LLMs when processing long contexts. Fig. 1 provides a detailed illustration of the LLM structure, including the self-attention mechanism within the model. The self-attention first transforms the input token embeddings into separate representations known as Query (**Q**), Key (**K**), and Value (**V**). Self-attention then consists of two matrix multiplications, with a Softmax normalization in between them. The Q×K multiplication generates attention scores (**S**), which indicates the relevance between different tokens. The self-attention always extracts contextual information from highly-related tokens. Subsequently, the Softmax function normalizes the score matrix to derive attention probabilities (**P**). Finally, the P×V multiplication calculates a weighted average of all value vectors, producing the self-attention output. Previous works [4, 5, 6] primarily focus on optimizing the matrix multiplications in self-attention, but often overlook the Softmax operation. However, Softmax can be the actual bottleneck in LLM inference. Computing Softmax requires allocating and

†Work done during an internship at Google Research. ‡The corresponding authors are Gregory Kielian (gkielian@google.com) and Mehdi Saligane (mehdi@umich.edu).

iterating over the entire score vector to determine the maximum score and the score summation. Consequently, the after-Softmax P×V should be blocked until the before-Softmax Q×K and Softmax are completed. Due to the low parallelism opportunity, Softmax reduces the hardware utilization of mainstream GPUs and TPUs, as well as other potential hardware. The situation exacerbates when self-attention processes longer contexts, as the Softmax complexity increases linearly with the token numbers. Softermax [7] reveals that the Softmax can lead to over 30% latency overhead during LLM inference, particularly with a token sequence exceeding 4K, a typical size in state-of-the-art LLMs [1, 8, 9].

While many previous designs exploit quantization [10, 11, 12], weight sparsity [13, 14], token relevance sparsity [4, 5, 6, 15, 16, 17] and workload partition [18, 19, 20] to optimize the matrix multiplications in LLM, these endeavours are still hindered by the Softmax bottleneck. A few pioneers [7, 21, 22] focus on Softmax optimization. They can be divided into two categories, namely computing approximation and workflow scheduling. For computing approximation, Look-up tables (LUTs) [23, 24, 25] and Taylor expansions [26, 27] are two methods to approximate the non-linear Softmax. However, approximate computing invariably compromises LLM accuracy and fails to enhance Softmax parallelism. Conversely, workload scheduling prioritizes enhancing Softmax parallelism. For example, SpAtten [6] computes Q×K, Softmax and P×V for a sequence of tokens in pipeline. While SpAtten demonstrates high efficiency on encoder-only BERT models [28], it still suffers from a low utilization on decoder-only GPT models, which are the representative of the prevailing LLMs. On the other hand, partial Softmax, such as FlashAttention [20, 29], increases Softmax parallelism for both encoder and decoder LLMs. The central concept involves partitioning the score vector into multiple partial vectors and subsequently applying standard Softmax on each of them in parallel. Nevertheless, partial Softmax requires synchronization across partial vectors to determine the ultimate maximum score and score summation, which accounts for about 20% runtime latency for self-attention computing [21].

In summary, previous Softmax-oriented works suffer from inefficiency stemming from a lack of high parallelism and accuracy. There is a noteworthy observation that the maximum score can be replaced by arbitrary values to scale the numerator and denominator in Softmax. Furthermore, the probability vector resulting from Softmax normalization does not necessarily have to be a unit vector to maintain LLM accuracy. Inspired by the above insights, this paper proposes Constant Softmax (ConSmax), a software-hardware co-design that serves as an efficient alternative to Softmax. Diverging from previous strategies, ConSmax enhances computing parallelism and ensures lossless non-linear computing. The key contributions are listed as follows:

- We utilize two differentiable normalization parameters to substitute for the maximum score and denominator in the original Softmax, thereby preventing the data synchronization required for maximum searching and score summation. These parameters are learnable during training and remain fixed during inference, thus achieving inference efficiency.
- We propose a bitwidth-split ConSmax hardware design to generate lossless non-linear functions and mitigate the lookup table (LUT) overhead. Furthermore, the ConSmax hardware is scalable to accommodate mixed-precision computing, a prevalent feature in state-of-the-art LLMs [8, 30].
- We extensively evaluate ConSmax on the GPT-2 model and the WikiText103 [31] dataset. Experimental results show that ConSmax achieves 3.35× power savings and 2.75× area savings in 16nm FinFET technology, and 3.15× power savings and 4.14× area savings with the open-source OpenROAD toolchain under SkyWater's 130nm CMOS technology. In the meantime, ConSmax also maintains comparable accuracy on the GPT-2 model and the WikiText103 dataset.

II. Background

A. Large Language Model Preliminaries

*1) **Structure:*** Fig 1 illustrates the LLM structure. It typically contains three main architectural components: the embedding layer, the transformer block and the language model (LM) head. The embedding layer consists of token embedding and positional encoding, encoding the discrete input tokens to high-dimensional representations. Token embedding captures the semantic meaning, while positional encoding records the relative positioning order. In contrast, at the very end is the LM head, which serves the converse function to the embedding layer. It takes in the transformer outputs and transforms them back into linguistic tokens by predicting the probabilities of the next token. Connecting the embedding layer and LM head is a stack of transformer blocks, constituting the bulk of the LLMs. Each block can be further divided into a multi-head self-attention layer and a feed-forward layer. The self-attention layer allows each token to attend to every other token, thus enabling the model to capture global information across input tokens regardless of their distance. The self-attention is applied multiple times to form multi-head self-attention. All attention heads operate in parallel to capture linguistic dependencies from various representation subspaces. The feed-forward layer further processes the attention output through two linear transformations, providing additional representational capacity to LLM models.

*2) **Workflow:*** As shown in Fig. 1, for text generation tasks, LLMs operate between a summarization stage and a generation stage. The summarization stage provides the initial prompt context to condition the LLM model, while the generation stage uses the context to produce a continuation. Both stages utilize the same model structure but with distinct workflows. In the summarization stage, the self-attention layer simultaneously processes a set of input tokens and extracts their key and value representations in parallel. The resultant representation matrix is then reused in the generation stage to generate the first new token. In contrast, the generation stage generates only a single token at each inference iteration. Subsequently, the output tokens from the previous iteration are iteratively fed back as input to generate subsequent output tokens.

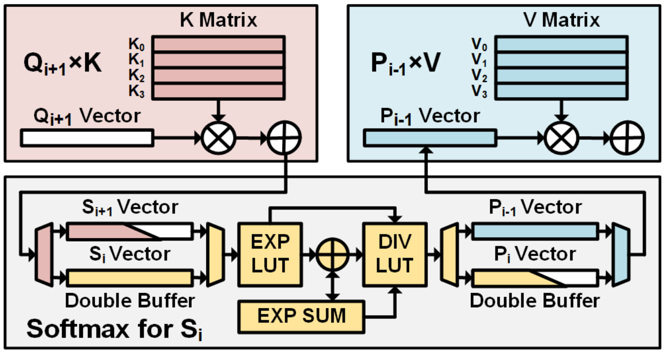


Fig. 2: Self-attention inference on hardware with token-pipeline workflow.

B. Softmax Bottleneck in LLM Inference

As the summarization stage processes a set of input tokens, it involves matrix-matrix multiplications, rendering it compute-bound. Consequently, the summarization stage can effectively saturate GPUs/TPUs utilization. In contrast, the generation stage performs vector-matrix multiplication to generate a single new token at each inference, which is a memory-bound operation. This can result in a reduced utilization of GPUs/TPUs or even previous transformer-oriented accelerators [5, 6, 15, 16, 17]. Softmax leads to the underutilization in the generation stage. Unfortunately, this aspect has not been fully explored in these prior works.

Fig. 2 illustrates the architecture of the typical transformer accelerators [5, 6]. For brevity, the peripheral modules such as the sparsity detector in these works are not depicted. These types of accelerators process Q×K, Softmax and P×V in a pipeline manner. Note that the original Softmax requires data synchronization to calculate the maximum factor and denominator. Therefore, for each token, the Softmax module should allocate all Q×K results and halt the P×V computation until Softmax is completed. To enhance computing parallelism, these cascaded modules in Fig. 2 process different tokens in a pipeline, known as the token-pipeline workflow. For example, Q×K module operates on Token_{i+1}, while Softmax and P×V modules handle Token_{i} and Token_{i-1}, respectively. Within each operation, the partial result is stored in double buffering and interleaved to the next module once the final results are ready. Token-pipeline flow facilitates the summarization stage, during which LLM models process plenty of tokens from the input prompt. However, as mentioned in Section II-A, the generation stage cannot fully utilize the pipeline because the LLMs operate with only a single input token. As a result, only one of the three modules works at a time, causing a decrease in hardware utilization.

In addition to its low parallelism, Softmax also faces challenges due to hardware-unfriendly non-linear operations such as exponents and reciprocals. To implement Softmax on silicon, previous works utilize LUTs as well as Taylor expansions to approximate Softmax as piece-wise linear functions. These methods are effective for accelerating CNNs and RNNs, where Softmax accounts for a minimal portion as the final classification layer. However, this prerequisite is no longer valid for transformer-based LLM models, which employ Softmax as the key component in the self-attention mechanism. On the one hand, using approximated Softmax can compromise LLM accuracy. On the other hand, it does not enhance computing parallelism.

C. ConSmax Motivation

In summary, the Softmax operation hinders the existing LLM accelerators due to its low computing parallelism and hardware-unfriendly non-linear operations. To address these challenges, we propose ConSmax, a software-hardware co-design that serves as an efficient alternative to Softmax. Our work is orthogonal to previous works, contributing:

1) ***High Computing Parallelism****:* Rather than focusing on the already optimized matrix multiplication, we thoroughly investigate the Softmax bottleneck in LLM acceleration. The maximum searching and denominator summation contribute to approximately 20% of the latency in the attention operation during token generation [21]. To mitigate this synchronization overhead, ConSmax introduces differentiable normalization parameters that replace the maximum factor and denominator in the original Softmax, enhancing computing parallelism by eliminating the need for maximum score searching and score summation.

2) ***Lossless Computing and Scalability****:* A bitwidth-split ConSmax hardware can perform lossless non-linear operations while minimizing LUT overhead. Additionally, the ConSmax hardware is hierarchical and scalable, supporting mixed-precision computing, which is commonly used in state-of-the-art LLM models [8].

III. ConSmax Algorithm

Although the Softmax operation represents a relatively small portion of the overall self-attention layer, it poses a much greater challenge to be designed into efficient hardware compared to well-optimized matrix multiplication. Therefore, the Softmax can result in significant overhead if not handled appropriately. In this section, we introduce the ConSmax algorithm, specifically designed to alleviate data synchronization requirements for computing the exponential maximum and summation within the original Softmax. As a consequence, ConSmax significantly enhances computing parallelism during LLM inference, notably benefiting the generation stage.

A. Convert Softmax to ConSmax

The Softmax operation depicted in Fig. 3(a) requires the entire score vector to determine the maximum score and score summation. However, generating all scores simultaneously in a single cycle is impractical, particularly considering the current LLM models that accommodate thousands to hundreds of thousands of input tokens. [1, 8, 9] (from 8K to 128K tokens). Therefore, the potential hardware should buffer the exponent results for all score elements before Softmax can proceed. Note that the exponent operation would explode the numerical range. It results in more memory consumption to maintain the precision of these intermediate results. To increase Softmax parallelism, ***the primary challenge lies in computing each***

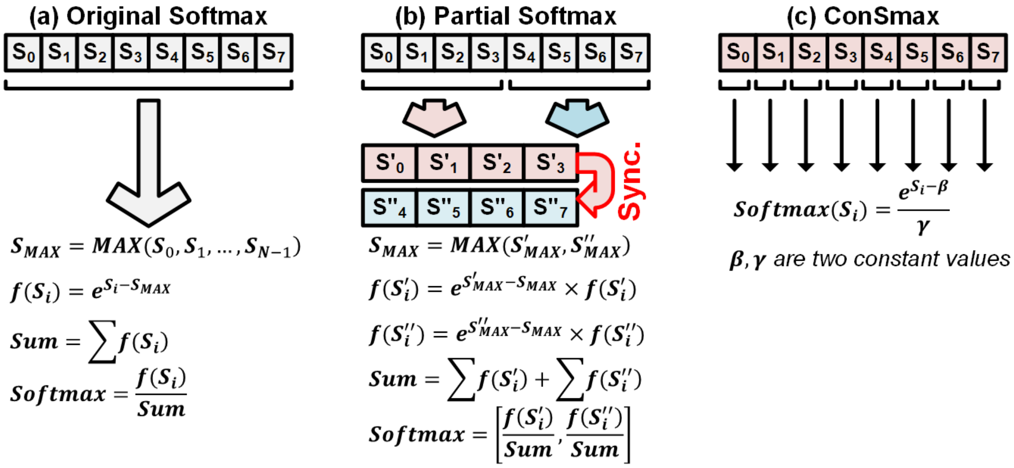

(a) Original Softmax
S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7
$S_{MAX} = MAX(S_0, S_1, \ldots, S_{N-1})$
$f(S_i) = e^{S_i - S_{MAX}}$
$Sum = \sum f(S_i)$
$Softmax = \frac{f(S_i)}{Sum}$
(b) Partial Softmax
S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7
S'_0 S'_1 S'_2 S'_3
S''_4 S''_5 S''_6 S''_7
Sync.
$S_{MAX} = MAX(S'_{MAX}, S''_{MAX})$
$f(S'_i) = e^{S'_{MAX} - S_{MAX}} \times f(S'_i)$
$f(S''_i) = e^{S''_{MAX} - S_{MAX}} \times f(S''_i)$
$Sum = \sum f(S'_i) + \sum f(S''_i)$
$Softmax = \left[\frac{f(S'_i)}{Sum}, \frac{f(S''_i)}{Sum}\right]$
(c) ConSmax
S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7
$Softmax(S_i) = \frac{e^{S_i - \beta}}{\gamma}$
β, γ are two constant values

Fig. 3: Comparing ConSmax with (a) original Softmax and (b) partial Softmax.

partial Softmax result independently, without relying on the maximum score and summation results from other partial Softmax computations.

According to the mathematical formula, Softmax operation works as a normalization function, wherein the maximum value serves as the scaling factor to prevent data overflow. There is a noteworthy observation that the scaling factor can be an arbitrary value rather than strictly using the maximum value. Therefore, the Softmax operation is converted as:

$$Softmax(S_i) = \frac{e^{S_i - S_{max}}}{\sum_i e^{S_i - S_{max}}} = \frac{e^{S_i - \beta}}{\sum_i e^{S_i - \beta}} \qquad (1)$$

where $\mathrm{S_i}$ represents the i-th score element and β could be arbitrary value. Further, the denominator could also be arbitrary as long as it satisfies the normalization function. Therefore, we replace the denominator with another constant value, denoted as γ, to propose the final ConSmax formulation as:

$$ConSmax(S_i) = \frac{e^{S_i - \beta}}{\gamma} \qquad (2)$$

However, it's essential to note that the scaling factor β cannot be arbitrary due to the risk of exponential computation overflow. For example, the overflow occurs in the case where $\mathrm{S_i} >> \beta$. In contrast, the exponential result infinitely tends to be zero if $\mathrm{S_i} << \beta$, leading to numerical precision loss. Similarly, the denominator γ cannot be chosen arbitrarily, since it normalizes the exponential score to probability distribution. Take the most extreme case as an example, where $\gamma \to 0$ or $\gamma \to +\infty$. In such cases, the probability values tend towards to infinity or zero, respectively, rendering them unable to effectively distinguish the token relevance.

To determine the optimal scaling and denominator, we designate the β and γ as learnable parameters. During the training phase, these parameters evolve in response to the characteristics of the practical dataset. In addition, the combination of β and γ varies across different self-attention heads, allowing for a more flexible and customized approach to normalization. The next problem is how to initialize the β and γ before training commences. Optimizing β and γ is pivotal for enhancing the effectiveness of the ConSmax function, presenting a promising avenue for improving LLM efficiency. This exploration can be conducted through a hyperparameter tuning process, where various combinations of initial β and γ are tried during the warming-up iterations. The combination that yields the best performance, such as the lowest validation loss, is selected.

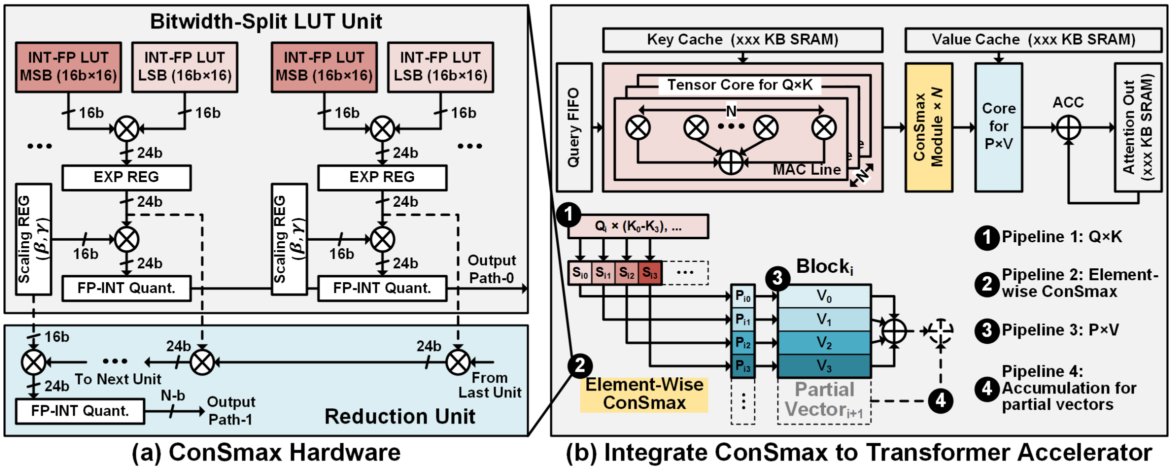
Finally, while a pretrained denominator cannot guarantee that the probability vector is a unit vector, the experimental results presented in Section V demonstrate that this constraint does not degrade LLM accuracy. As long as the probability distribution can magnify the small differences in input scores, the LLM performance remains robust. In addition, the ConSmax in Equation 2 can be rewritten as:

$$ConSmax(S_i) = \frac{e^{S_i - \beta}}{\gamma} = C \times e^{S_i}, where\ C = -\frac{e^{\beta}}{\gamma} \qquad (3)$$

During inference, we merge β and γ into a single constant value. However, during training, we maintain them as independent parameters to mitigate against exponential overflow.

B. Comparison with Partial Softmax

Softermax [7] and FlashDecoding++ [21] are two alternative approaches to Softmax, aimed at reducing memory consumption and enhancing computational parallelism. Similarly, they employ the partial Softmax technique, where the global maximum factor is replaced by the local one. More specifically, the main idea is to divide the score vector into several partial vectors and subsequently apply the standard Softmax to each partial vector separately. Although these approaches improve Softmax parallelism, they necessitate additional synchronization among different partial vectors. As shown in Fig. 3(b), the partial Softmax recalculates the exponential maximum and summation after all partial vectors are processed by the standard Softmax. Subsequently, their normalized results are adjusted based on the synchronized maximum and summation values. Such synchronization leads to 18.8% overheads in the attention computation with 1024 input tokens, which is anticipated to worsen with longer input sequences. In contrast,



(a) ConSmax Hardware

(b) Integrate ConSmax to Transformer Accelerator

Fig. 4: (a) Bitwidth-split ConSmax hardware unit and (b) Integrate ConSmax hardware to transformer accelerator.

ConSmax avoids any synchronization, distinguishing itself from these previous works.

IV. ConSmax Hardware

Based on the algorithm in Section III, we proceed to explore the customized ConSmax hardware. The bitwidth-split LUT architecture can produce lossless non-linear functions and enable the scalability for the mix-precision computing. In addition, we also elaborate on how to integrate the ConSmax hardware into state-of-the-art transformer accelerators. Leveraging its synchronization-free property, ConSmax enhances parallelism in both LLM summarization and generation stages.

A. Lossless and Scalable ConSmax Hardware

Fig. 4(a) illustrates the proposed ConSmax hardware featuring a two-level structure. In Level-1, multiple bitwidth-split ConSmax units operate in parallel to execute ConSmax operations. The reduction unit in Level-2 can allocate varying numbers of the basic ConSmax units to support mixed-precision computing. For brevity, Fig. 4(a) just displays two ConSmax units, capable of generating either two 8-bit ConSmax results or one 16-bit result simultaneously.

*1) **Bitwidth-Split ConSmax Unit**:* As depicted in Fig. 4(a), each ConSmax unit comprises four components as bitwidth-split LUTs, floating-point (FP) multipliers, FP-to-Integer (INT) converter and necessary buffers. Assume that matrix multiplication accelerators, such as GPUs/TPUs, produce the Q×K multiplication and generate 8b-INT scores sequentially. Upon receiving the 8b-INT results, the LUTs within the ConSmax unit first perform the exponential operation and concurrently dequantize the result to 16b-FP format. Specifically, the 8b-INT scores function as addresses to access the LUTs, which store the corresponding 16b-FP exponential/dequantized results. This approach allows for storing the exponential result within a limited bitwidth while maintaining higher precision, thus mitigating potential bitwidth explosion inherent in integer format representation. Moreover, the ConSmax unit circumvents the need for an additional INT-to-FP converter, thanks to the simultaneous conversion by the LUTs. To further minimize the LUT capacity, instead of employing a large LUT to enumerate all 256 combinations, we split the 8-bit input into two slices as MSB-INT4 and LSB-INT4. Each fragment is associated with a 16-entry LUT. The partial sums from these two LUTs are then merged in the downstream multiplier. This configuration allows the bitwidth-split LUTs to perform lossless exponential operations for all input combinations with minimal LUT overhead. In contrast to the straightforward shift and addition used for integral bitwidth alignment, the partial sum reduction between floating-point fragments is comparatively intricate:

$$\begin{aligned} e^{S_{INT8}} &= e^{(MSB_{INT4}<<4)+LSB_{INT4}} \\ &= e^{2^4\times MSB_{INT4}} \times e^{LSB_{INT4}} \end{aligned} \tag{4}$$

To avoid the implementation of non-linear $(e)^{2^4}$ in hardware, the MSB-LUT directly projects $e^{2^4\cdot x}$ for MSB-INT4, while LSB-LUT exclusively maps e^x. Following exponent calculation, the ConSmax normalization is further applied according to Equation 2. The pretrained β and γ are combined based on Equation 3and then multiplied with the exponential result in the second multiplier. Based on the above analysis, compared to previous works [23, 24], ConSmax can generate accurate nonlinear operations without using a LUT to approximate Softmax.

*2) **Reduction Unit**:* The mix-precision computing is a prevalent model compression technique for efficient LLM inference [8], wherein different operators of the model can be assigned with different precision. Therefore, this feature necessitates the need for supporting mixed-precision computing in the ConSmax hardware. To achieve this, the reduction unit allocates multiple bitwidth-split LUTs according to the precision requirements. As shown in Figure 4(a), two 8-bit bitwidth-split LUTs can execute one 16-bit ConSmax normalization. The 16-bit score element is initially divided into two 8-bit slices. Subsequently, each LUT receives one slice and independently generates an 8-bit ConSmax. The partial sums from different LUTs are then directed to the reduction unit, where they are merged using a floating-point multiplier chain, as depicted in Equation 4. The reduction unit modifies the length of the multiplier chain to accommodate other precision configurations.

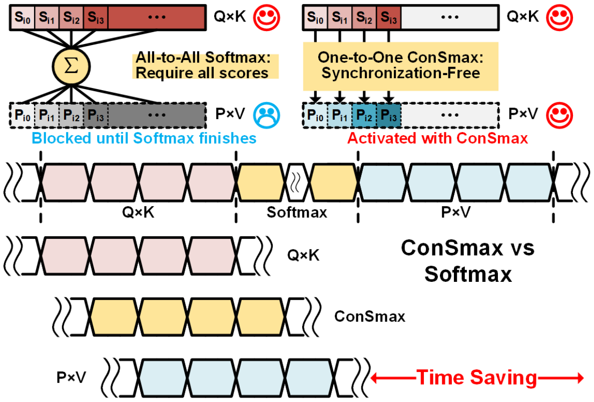


Fig. 5: Synchronization-free ConSmax leads to high parallelism and time savings.

B. Integrate ConSmax Hardware to Transformer Accelerator

Figure 4(b) elaborates on the integration of ConSmax hardware into dedicated transformer accelerators. Two TPU tensor cores (or alternative hardware, e.g. GPUs), alongside the inserted ConSmax hardware, produce Q×K, ConSmax normalization and P×V in pipeline. Note that the most advanced LLMs typically support input contexts with $\geq$8K tokens (e.g. 32K in GPT-4[1]). Therefore, it is improbable for the tensor core to execute such extensive matrix multiplication simultaneously. Nevertheless, thanks to ConSmax, the pipeline can continue to function effectively. ❶ First, the front-end tensor core conducts Q×K operation between the single given query vector and a portion of the key vectors. The size of the involved key vectors scales proportionally to the capacity of the tensor core. ❷ Secondly, the partial score vector is directly forwarded to the ConSmax unit for normalization. Meanwhile, the front-end tensor core simultaneously computes the Q×K multiplications for the next set of key vectors. ❸ Despite not having generated the complete attention probabilities, the normalized elements can be directly multiplied with the corresponding value vectors in the back-end tensor core. This is facilitated by ConSmax, which eliminates the need for the exponential maximum and summation calculations in the original Softmax function. ❹ Finally, the partial sum is accumulated and updated to generate the attention output.

In contrast to the coarse-grained token pipeline outlined in Section II-B, ConSmax-integrated accelerators can employ a fine-grained element-wise pipeline to attain higher parallelism and realize significant time savings. Note that the LLM generation stage comprises only a single input token, which is insufficient to fully utilize the pipelined Q×K and P×V hardware modules in Figure 2. Therefore, as shown in Figure 5, the P×V operation in such accelerators is stalled until the original Softmax is completed, thereby leading to significant underutilization. The accelerator can allocate all computing logic to process LLMs in a layer-sequential manner [32], thereby achieving high utilization, However, this approach could result in numerous intermediate results which necessitate a large storage overhead. On the contrary, with its synchronization-free attribute, ConSmax allows for an element-wise pipeline instead of a token-based one. Consequently, the accelerator can fully utilize all processing modules even with a single token. This not only accelerates LLM summarization but, more significantly, enhances LLM generation with a uniform architecture.

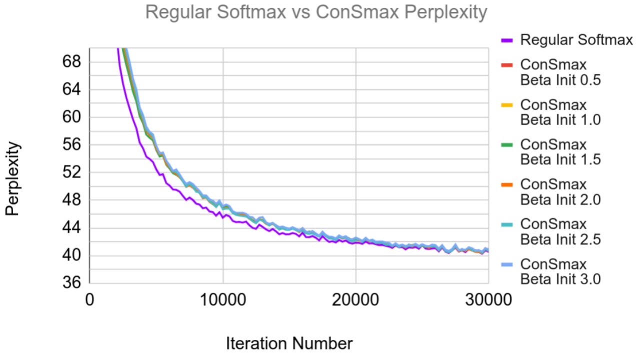


Fig. 6: Perplexity of GPT-2 models with Softmax and ConSmax, showing convergence of validation losses.

V. Experimental Results

A. Experiment Setup

We have developed a ConSmax prototype using Verilog RTL under 16nm FinFET CMOS technology. All digital modules are synthesized using Synopsys Design Compiler. For open-source contribution, we further synthesized our design with the OpenROAD toolchain [33] under SkyWater's 130nm CMOS technology. These resulting netlists are used to evaluate the power and area consumption. We compare the ConSmax hardware with two baselines: the DesignWare-based Softmax hardware and Softermax [7]. The former Softmax hardware faithfully implements the original Softmax function, whereas the Softermax is recognized as an efficient alternative to the conventional Softmax function. Since Softermax does not explicitly provide power and area results, we develop a corresponding counterpart according to the Softermax algorithm. These baselines are synthesized with the same configurations.

We evaluate the impact of ConSmax on the GPT model using the WikiText103 [31] dataset, where the Softmax in each self-attention block is replaced by the ConSmax. This benchmark model contains 6 transformer layers each equipped with 6 self-attention heads. Therefore, the embedding size is set to 384. Moreover, the default token length is set to 256. We report perplexity for the WikiText103 dataset. This metric measures the LLM's performance on text generation tasks. A lower perplexity value indicates better performance.

B. Software Performance

We first compare the perplexity of the standard Softmax and the proposed ConSmax. In this experiment, we initialize β within the range of [0.5, 2.5], while γ is set to a constant value of 100. As shown in Figure 6, with varying combinations of β and γ, ConSmax initially exhibits a marginally 2.3% higher perplexity than Softmax, and leads to less than 0.9% perplexity degeneration after 10K iterations. After about 20K iterations, both Softmax and ConSmax-based models converge,

TABLE I: ConSmax Hardware Performance Comparison with Softermax, Softmax

Proprietary EDA	ConSmax	Softermax	Softmax	Consmax	Softermax	Softmax
Process	16nm			130nm		
Max Frequency (MHz)	1250	1111	909	666.67	333.33	285.71
Area (mm^2)[a]	0.0008	0.0022	0.011	0.007	0.029	0.18
Power (mW)[b]	0.2	0.67	1.5	2.69	8.5	51
Optimum Energy per op(pJ)	0.2	0.7	1.5	4	25.5	178.5
Opensource EDA	ConSmax	Softermax	Softmax	Consmax	Softermax	Softmax
Process	16nm			130nm		
Max Frequency (MHz)	2000	1000	500	166.67	142.86	87.72
Area (mm^2)[a]	0.0009	0.0019	0.011	0.015	0.033	0.2
Power (mW)[b]	0.683	2.82	10.2	1.82	10.5	42.2
Optimum Energy per op(pJ)	0.3	1.4	2.7	16.7	73.5	255.3

[a] Power for 16nm is tested with 500MHz, for 130nm is tested with 80MHz.
[b] Area is measured at Max Frequency.

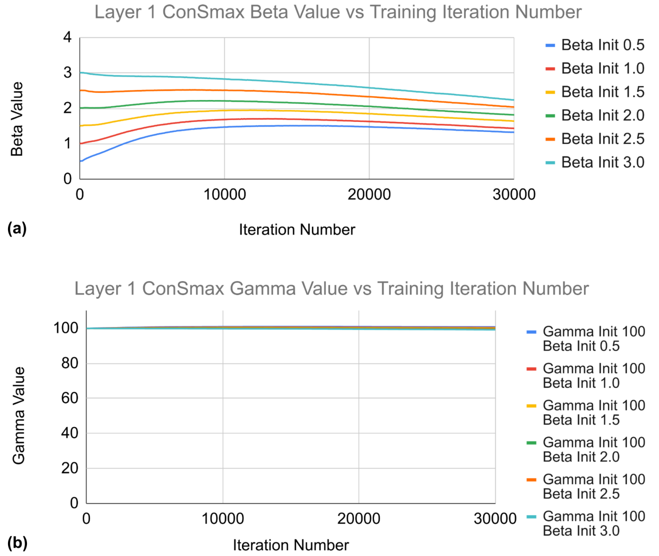


Fig. 7: Evolution of β and γ throughout training, each training run using a different starting value for β (as γ has been observed to have low % change). Spread of β values has been found to decrease with training.

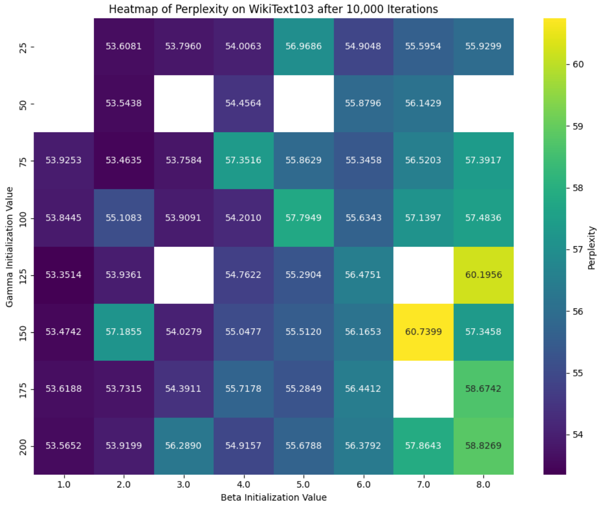
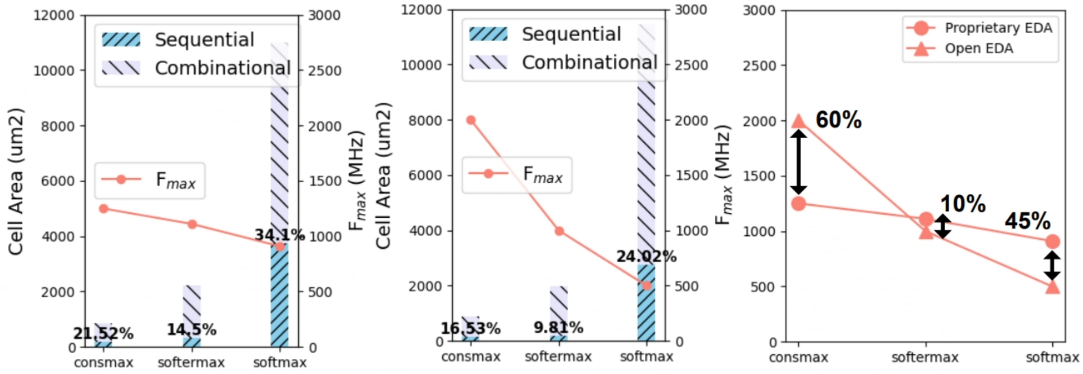


Fig. 8: Explore β and γ for ConSmax-based LLM training.

demonstrating a similar trend in their performance metrics. The additional learnable parameters in ConSmax can lead to instability during the early stages of model training. Moreover, the ConSmax could produce the non-unit normalization vector, which further exacerbates this instability. That's because the non-unit normalization vector cannot effectively amplify the differences in the Q×K results, causing the subsequent P×V computation to fail to extract information primarily from highly relevant tokens. Nevertheless, with sufficient training iterations, the learnable parameters in ConSmax, along with other weight matrices in LLMs, can effectively model text generation tasks. Figure 7 illustrates the evolution of β and γ throughout ConSmax-based model training. For brevity, we only present the results for one certain self-attention head in the GPT baseline model. Similarly, $beta$ is set within the range of [0.5, 2.5], while $gamma$ is set to 100. As the training proceeds, β demonstrates a converging trend toward a final value. At the same time, γ remains relatively constant across different beta configurations. The rest self-attention heads exhibit a similar trend. Finally, we examine the impact of β and γ initialization on GPT's performance in Fig. 8 With the same γ value, there is a tendency for lower perplexity with smaller β values after 10K warm-up training iterations. Conversely, when β values differ, the optimal γ selection varies on a case-by-case basis. Therefore, the combination of β and γ that results in the lowest perplexity is utilized to train the model until convergence.

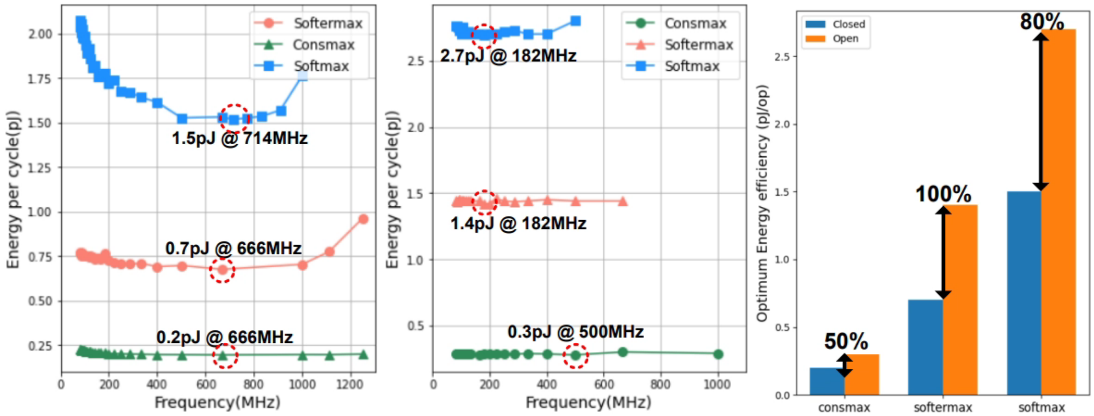
C. Hardware Performance

Table I summarizes ConSmax hardware performance and compares it with Softmax and Softermax hardware. For a fair comparison, the ConSmax and two baselines process a Softmax workload with a token sequence of 256. Under 16nm FinFET technology, 1250MHz working frequency and 0.8V power supply, the ConSmax hardware only consumes 0.2mW power and 0.0008mm^2 area. Therefore, the ConSmax only results in a minimal overhead when integrated into the LLM-oriented accelerators [15, 5, 6]. Compared to the Softermax counterpart, ConSmax achieves 3.35× power and 2.75× area savings. Compared to the Softmax counterpart, ConSmax further achieves 7.5× power and 13.75× area savings. For the open-source SkyWater's 130nm CMOS technology, ConSmax runs at a lower 667MHz working frequency and 0.8V power supply. ConSmax achieves 3.2× power saving and 4.1× area saving compared to Softermax, and 23.2× power saving and 25.7× area saving compared to Softmax. In summary, the



(a) Tested with proprietary tool (b) Tested with opensource tool (c) Comparison of Fmax

Fig. 9: Cell area comparison of ConSmax, Softermax and Softmax in 16nm process (a) tested in proprietary EDA tool, (b) tested in open-source EDA tool and (c) comparison of F_{max} using different EDA tool for 3 designs.



(a) Tested with proprietary tool (b) Tested with opensource tool (c) Comparison of Optimum Energy Efficiency

Fig. 10: Energy efficiency comparison of ConSmax, Softermax and Softmax in 16nm process (a) tested in proprietary EDA tool, (b) tested in open-source EDA tool and (c) comparison of Energy efficiency using different EDA tool for 3 designs.

ConSmax hardware presents stable performance improvement across different CMOS technologies and EDA toolchains.

Figure 9 presents the area breakdown and maximum operating frequency of the three different designs. The ConSmax demands minimal area consumption while achieving the highest operating frequency. This is because ConSmax completely eliminates the exponential maximum and summation calculations. As a result, the scratchpads for intermediate result storage and the costly floating-point accumulators can be minimized. Finally, we test the energy efficiency of ConSmax on the real Softmax workload. As depicted in Figure 10, under 16nm CMOS technology, both ConSmax and Softermax attain optimal energy consumption at 666 MHz, whereas the Softmax baseline achieves this at 714 MHz. ConSmax results in an energy efficiency of 0.2pJ, 3.5× and 7.5× better than Softermax and the Softmax baseline, respectively. The open-source EDA presents a similar result.

VI. Conclusion

This paper presents ConSmax, a software-hardware co-design for efficient Softmax acceleration. ConSmax improves Softermax computing parallelism by avoiding data synchronization, which produces maximum score and score summation. In addition, the bitwidth-split ConSmax hardware is lossless and scalable for calculating non-linear functions. The experiments show that ConSmax achieves a minuscule power consumption of 0.2mW and an area of 0.0008mm^2 at 1250MHz working frequency and 16-nm FinFEt CMOS technology. Compared to state-of-the-art Softmax hardware, ConSmax results in 3.35× power and 2.75× area savings with a comparable accuracy on GPT-2 model and the WikiText103 dataset.

References

[1] T. Brown et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020.

[2] Gong et al. Enhanced transformer model for data-to-text generation. In *Proceedings of the Workshop on Neural Generation and Translation*, pages 148–156, 2019.

[3] A. Arnab et al. Vivit: A video vision transformer. In *Proceedings of the international conference on computer vision (ICCV)*, pages 6836–6846, 2021.

[4] T. Ham et al. A3: Accelerating attention mechanisms in neural networks with approximation. In *International Symposium on High Performance Computer Architecture (HPCA)*, pages 328–341, 2020.

[5] T. Ham et al. Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *International Symposium on Computer Architecture (ISCA)*, pages 692–705, 2021.

[6] H. Wang et al. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110, 2021.

[7] J. Stevens et al. Softermax: Hardware/software co-design of an efficient softmax for transformers. In *Design Automation Conference (DAC)*, pages 469–474, 2021.

[8] A. Jiang et al. Mistral 7b. In *arXiv preprint arXiv:2310.06825*, 2023.

[9] H. Touvron et al. Llama: Open and efficient foundation language models, 2023.

[10] Z. Liu et al. Post-training quantization for vision transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 28092–28103, 2021.

[11] F. Frantar et al. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *arXiv preprint arXiv:2210.17323*, 2022.

[12] J. Chee et al. Quip: 2-bit quantization of large language models with guarantees. In *arXiv preprint arXiv:2307.13304*, 2023.

[13] F. Frantar et al. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning (ICML)*, pages 10323–10337, 2023.

[14] S. Liu et al. 16.2 a 28nm 53.8tops/w 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned nn and cim-based local-attention-reusable engine. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 250–252, 2023.

[15] A. Yazdanbakhsh et al. Sparse attention acceleration with synergistic in-memory pruning and on-chip recomputation. In *International Symposium on Microarchitecture (MICRO)*, pages 744–762, 2022.

[16] Y. Qin et al. Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14, 2023.

[17] Z. Qu et al. Dota: detect and omit weak attentions for scalable transformer acceleration. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 14–26, 2022.

[18] M. Zhou et al. Transpim: A memory-based acceleration via software-hardware co-design for transformer. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1071–1085, 2022.

[19] S. Hong et al. Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation. In *International Symposium on Microarchitecture (MICRO)*, pages 616–630, 2022.

[20] T. Dao et al. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 16344–16359, 2022.

[21] K. Hong et al. Flashdecoding++: Faster large language model inference on gpus. In *arXiv preprint arXiv:2307.08691*, 2023.

[22] Y. Zhang et al. Base-2 softmax function: Suitability for training and efficient hardware implementation. In *Transactions on Circuits and Systems I: Regular Papers*, volume 69, pages 3605–3618, 2022.

[23] K. Chen et al. Approximate softmax functions for energy-efficient deep neural networks. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 31, pages 4–16, 2023.

[24] Y. Joonsang et al. Nn-lut: neural approximation of non-linear operations for efficient transformer inference. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 577–582, 2022.

[25] G. Du et al. Efficient softmax hardware architecture for deep neural networks. In *Proceedings of the on Great Lakes Symposium on VLSI (GLSVLSI)*, pages 75–80, 2019.

[26] E. Banerjee et al. Exploring alternatives to softmax function. In *arXiv preprint arXiv:2011.11538*, 2020.

[27] A. Brébisson and P. Vincent. An exploration of softmax alternatives belonging to the spherical loss family. In *arXiv preprint arXiv:1511.05042*, 2015.

[28] J. Devlin et al. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*, 2018.

[29] T. Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *arXiv preprint arXiv:2307.08691*, 2023.

[30] J. Xu et al. Mixed precision quantization of transformer language models for speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7383–7387, 2021.

[31] S. Merity et al. Pointer sentinel mixture models. 2016.

[32] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W Fletcher. Mind mappings: enabling efficient algorithm-accelerator mapping space search. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 943–958, 2021.

[33] Tutu Ajayi, Vidya A Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, et al. Toward an open-source digital flow: First learnings from the openroad project. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–4, 2019.