TSAcc: An Efficient Tempo-Spatial Similarity Aware Accelerator for Attention Acceleration

Zhuoran Song[1]*, Chunyu Qi[1], Yuanzheng Yao[1], Peng Zhou[2], Yanyi Zi[2], Nan Wang[2], Xiaoyao Liang[1]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[2]Alibaba Cloud

ABSTRACT

Attention-based models provide significant accuracy improvement to Natural Language Processing (NLP) and computer vision (CV) fields at the cost of heavy computational and memory demands. Previous works seek to alleviate the performance bottleneck by removing useless relations for each position. However, their attempts only focus on intra-sentence optimization and overlook the opportunity in the temporal domain. In this paper, we accelerate attention by leveraging the tempo-spatial similarity across successive sentences, given the observation that successive sentences tend to bear high similarity. This is rational owing to many semantic similar words (namely tokens) in the attention-based models. We first propose an online-offline prediction algorithm to identify similar tokens/heads. We then design a recovery algorithm so that we can skip the computation on similar tokens/heads in succeeding sentences and recover their results by copying other tokens/heads features in preceding sentences to reserve accuracy. From the hardware aspect, we propose a specialized architecture TSAcc that includes a prediction engine and recovery engine to translate the computational saving in the algorithm to real speedup. Experiments show that TSAcc can achieve 8.5×, 2.7×, 14.1×, and 64.9× speedup compared to SpAtten, Sanger, 1080TI GPU, and Xeon CPU, with negligible accuracy loss.

ACM Reference Format:
Zhuoran Song[1]*, Chunyu Qi[1], Yuanzheng Yao[1], Peng Zhou[2], Yanyi Zi[2], Nan Wang[2], Xiaoyao Liang[1]. 2024. TSAcc: An Efficient Tempo-Spatial Similarity Aware Accelerator for Attention Acceleration. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655982

1 INTRODUCTION

Attention-based models including Transformer [14], BERT [3], and GPT-2 [10] have opened up a series of breakthroughs in natural language processing (NLP) and computer vision (CV) domains. And they are widely adopted in various applications, including machine translation, computer vision, and recommendation systems.

The remarkable achievements of attention come with a significant computational burden. To enhance the efficiency, numerous studies have delved into sparsification methods [4, 6, 7, 9, 11, 13, 17]. However, as shown in Fig. 1, such pruning methodology primarily concentrates on optimizing attention within individual sentences, overlooking the potential for optimization across sentences. This limitation confines these methods from fully harnessing their potential for attention acceleration.

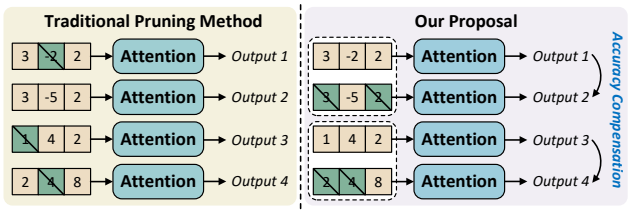


Figure 1: Comparision of our proposal and existing works.

Unlike previous research, our approach focuses on uncovering acceleration opportunities by analyzing inter-sentence similarity. Our key insight revolves around consecutive sentences that share many semantically similar words and exhibit a form of **temporal similarity**, which can lead to computational savings. For example, in Fig. 1, we have four sentences, each containing three tokens; we can see a high similarity between the first and second sentences, as well as the third and fourth sentences. Correspondingly, the tokens "3", "-2", and "2" in the first sentence and "-5" in the second sentence are identified as *pivotal tokens*, requiring computations for their features. Tokens "3" and "2" in the second sentence can be recognized as *similar tokens* referencing the pivotal token, and their computations can be safely skipped. Since the features of pivotal and similar tokens are independently captured using the same attention-based model, we can precisely recover the features of similar tokens by directly retrieving the features of their referenced pivotal tokens, compensating for errors resulting from the removal of associated computations of similar tokens. Additionally, we observe that multiple heads within the attention-based model exhibit **spatial similarity**, as some of them serve consistent roles, such as positional and syntactic functions [15]. We identify *pivotal heads* that act as reference heads for the remaining *similar heads*, allowing us to further reduce the size of the attention-based model. It is also worth noting that our proposal is orthogonal to the existing sparsification techniques and can be seamlessly integrated to deliver greater speedup.

The contribution of this paper is summarized as follows:

1) We propose the tempo-spatial similarity aware attention acceleration mechanism, which for the first time identifies the temporal and spatial similarity between and within successive sentences to reduce the workload of attention.

2) We propose a pivotal/similar token/head prediction algorithm that can predict the significance of tokens and heads and skip the computations of similar tokens and heads for speedup. More

This work is partly supported by the National Natural Science Foundation of China (Grant No. 62202288). *Zhuoran Song is the corresponding author.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655982

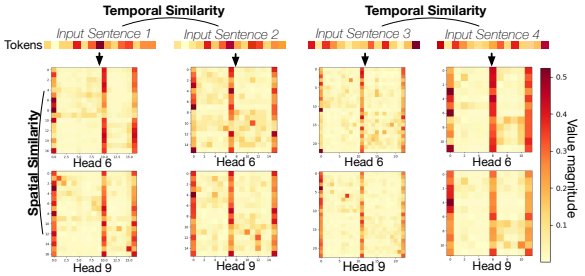


Figure 2: Visualization of the data similarity.

importantly, to avoid severe accuracy loss, the algorithm introduces a recovery operator to recover similar tokens/heads' features by copying their reference pivotal tokens/heads.

3) We propose a customized tempo-spatial similarity aware accelerator (abbreviated as TSAcc), consisting of a dedicated prediction engine and recovery engine. The prediction engine can efficiently predict the pivotal tokens on the fly. The recovery engine fine-grained manages the buffer to recover similar tokens/heads without burdening the memory.

2 BACKGROUND AND MOTIVATION

2.1 Attention-based Models

The multi-head attention mechanism is a key operation in attention-based models, consisting of three stages: input-weight, query-key, and softmax-value stages. The first *input-weight stage* performs three matrix multiplications by multiplying the input matrix (I) with the multi-head weight matrices q, k, v. The step generates three matrices, respectively denoted as query matrix (Q), key matrix (K), and value matrix (V), each of which is constituted by multiple heads.

Next, attention obtains a head of the score matrix (Sc_h) by multiplying a head of query matrix (Q_h) and the transpose of a head of key matrix (K_h^T) in the *query-key stage*. Afterwards, Sc_h goes through the softmax function for normalization; the normalization result is denoted as the head of the softmax matrix (S_h). S_h is then fed into the *softmax-value stage* and multiplied with a head of value matrix (V_h) to get the head of output matrix. The above stages are repeated until all heads of the output matrix are generated. And we can finally get the attention output (O) by concatenating all heads.

2.2 Accelerators for Attention

The large computation overhead of attention-based models attracts many researchers to enhance their efficiency [4, 6, 7, 9, 11, 13, 17] by hardware-software co-design. Among them, GOBO [18] proposed to quantize attention-based models like BERT to 3 bits, thus significantly reducing DRAM access. EdgeBERT [12] employs the early exit mechanism to perform dynamic voltage-frequency scaling (DVFS) at a sentence granularity, which can gain minimal energy consumption. SpAtten [17] and Sanger [7] explore the structured and unstructured pruning to prune away the redundant values in the query and key matrices based on the magnitude of the softmax matrix. TSAcc is different from the existing hardware-software co-design solutions for efficient attention because TSAcc tries to predict and skip the computation on all the following similar tokens/heads in the tempo-spatial domain.

2.3 Motivation

To verify the speculation of successive input sentences having strong temporal similarity, we first visualize the tokens of four randomly selected input sentences in the GLUE dataset [16]. As in Fig. 2, we can observe that the tokens in input sentence 1 and 2 are quite similar, and so for the input sentence 3 and 4. The temporal similarity in turn implies that their attention outputs can be easily obtained without intensive attention computations via our proposed scheme.

Additionally, considering that several heads in the attention mechanism play consistent and linguistically-interpretable roles [15], we envision that the heads with the same and consistent functions correspond to the similar heads in the softmax matrix that are irrelevant to the input sentence. To study the spatial similarity of heads, we then visualize the heads 6 and 9 of the softmax matrix and see that they are similar in all four sentences. This observation tells us that the spatial similarity of the heads is inherited, so we can spend a huge effort searching the spatial similarity in the training dataset and then directly take this indication to skip the computations of the similar heads in the test dataset.

3 TEMPO-SPATIAL SIMILARITY AWARE ALGORITHM

3.1 Online-Offline Prediction

3.1.1 Pivotal/Similar Token Online Prediction. In this section, we propose an online pivotal/similar token prediction that can predict the most representative tokens across sentences during run-time. The prediction procedure is presented as follows:

1. **Token similarity calculation.** To analyze the tokens' similarities, we first calculate the L1 distance between tokens. Given n sentences, which correspond to n input matrices, each with m tokens, there are a total of $n \times m$ tokens. We construct $(n \times m) \times (n \times m)$ token pairs, where each token is drawn from n input matrices. Next, we generate a distance matrix D by $D_{i,j} = \sum_{p=1}^{P} \left| T_i^p - T_j^p \right|$, where $i, j <= n \times m$ and P is the dimension of each token. $D_{i,j}$ represents the L1 distance of a token pair (T_i, T_j). After that, we apply a predefined token threshold Thr_T to binarize D and generate a $(n \times m) \times (n \times m)$ binary map B by $B_{i,j} = binarize(D_{i,j}, Thr_T)$, where $B_{i,j}$ is the token similarity between T_i and T_j. $B_{i,j} = 1$ indicates that the token pair is similar, while $B_{i,j} = 0$ indicates dissimilarity. As the example in Fig. 3(a), given 2 input matrices ($n = 2$) and each has 3 tokens ($m = 3$), we generate a 6×6 binary map by calculating the L1 distances of the token pairs and comparing the distances with the predefined threshold $Thr_T = 0.4$.

2. **Pivotal/similar token identification.** Once we obtain the binary map that indicates the similarities between tokens, the next thing is to identify the pivotal tokens. As stated above, pivotal tokens are defined as those that have strong relationships with other tokens, and their key features need to be computed and preserved. By utilizing the binary map that portrays the similarity between tokens, we can formulate the problem of identifying pivotal tokens as a well-defined Minimum Dominating Set problem [1]. Our goal is to minimize the number of pivotal tokens connecting all tokens. As this operation is executed online, we leverage the efficient greedy algorithm to solve our problem.

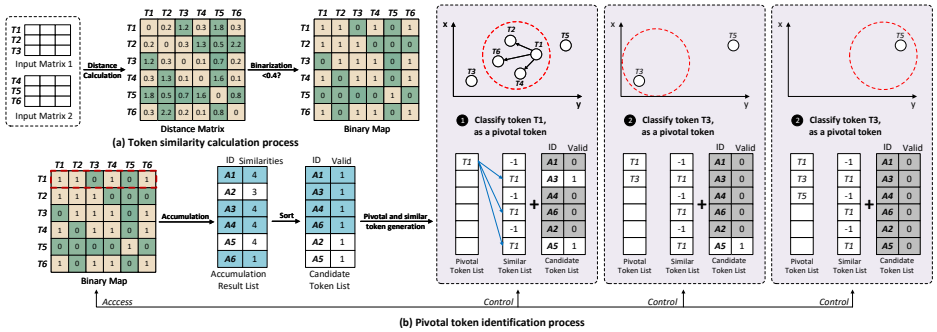


Figure 3: Pivotal/similar token online prediction with (a) token similarity calculation, (b) pivotal/similar token identification.

Initially, we accumulate the number of "1", also known as token similarities, in the row of the binary map B. The accumulation results are stored in the accumulation result list A, where the magnitude of A_i indicates the relation of token T_i with all other tokens. As in Fig. 3(b), we can get A_1-A_6, where $A_1 = 4$ means the token T_1 is similar to four tokens, including tokens T_2, T_4, T_6, and itself. Given that the larger accumulation result a token has, the more important the token will be. So we sort the accumulation results in descending order for the later pivotal/similar token classification. In the example, we sort the accumulation results to get a candidate token list whose token order is A_1, A_3, A_4, A_6, A_2, and A_5.

Afterward, we can classify the pivotal tokens according to the candidate token list. To do this, we pick out the token T_i that exhibits significant similarities with other tokens by searching from top to bottom of the candidate token list. Moreover, we should ensure that the potential token has not been previously classified as a pivotal/similar token by checking whether the corresponding valid bit is "1". In the figure, we first classify token T_1 as a pivotal token (covered by a red circle) because it is sufficient to represent T_2, T_4, and T_6. We record the token T_1 in the pivotal token list and set its valid bit in the candidate token list as "0" to avoid repeated classification. Later on, given the classified pivotal token, we can get the similar tokens as well as the reference relationship between the pivotal and similar tokens by accessing the binary map B. For the pivotal token T_i, we retrieve ith row in B to find any column j that satisfies $B_{i,j} = 1$. We classify T_j as the similar token that refers to the pivotal token and record the ID i at index j in the similar token list, as long as index j still holds the value "-1". In addition, we set the valid bit of T_j in the candidate token list to "0" to imply its unavailability. To illustrate this process, consider the example in Fig. 3(b). Based on the classified pivotal token T_1, we observe that tokens T_2, T_4, and T_6 satisfying $B_{1,2}$, $B_{1,4}$, and $B_{1,6} = 1$ (covered by a red block). Consequently, we classify them as similar tokens that refer to T_1. We store the ID 1 in the indices 2, 4, and 6 of the similar token list, denoted by the blue arrows. And we set the valid bit of T_2, T_4, and T_6 in the candidate token list to "0".

In essence, this step entails finding the area in the 2D space where the red circle covers the maximum number of points using a greedy algorithm. One of the points is classified as a pivotal token, while the rest, which fall within the same red circle can be replaced by the pivotal token, are identified as similar tokens. Subsequently, these points are removed from the space. The step is repeated until the space becomes empty.

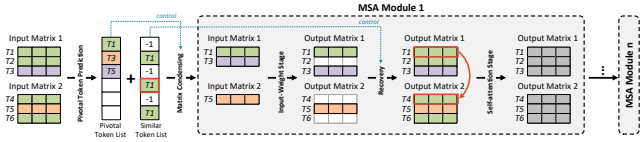


Figure 4: Overview of the proposed algorithm.

3.1.2 Pivotal/Similar Head Offline Identification. Based on the characteristic of the inherited spatial similarity of heads, we propose to identify the pivotal/similar heads offline, which differs from the online prediction in the following aspects:

- We use sentences from the training dataset to implement the pivotal/similar head offline identification, rather than identifying them during attention computing;
- We apply a head threshold Thr_H to conduct the binarization, which will generate several binary maps. Since the binary maps belonging to different sentences are slightly different, we find their intersection as the most representative binary map by performing the AND Boolean operation on them;
- We apply the brute force rather than the greedy algorithm to the most representative binary map. This is because the offline condition allows us to spend more effort to get more precise pivotal/similar heads.

The IDs of pivotal heads are recorded in the pivotal head list and the reference relationships between pivotal and similar heads are in the similar head list.

3.2 Condensed Matrix Multiplication

With the pivotal/similar token/head list, the original attention becomes multiple condensed matrix multiplications by eliminating the computations on similar tokens/heads. We take reducing the similar tokens' computations as an example, as depicted in Fig.4, given the pivotal token list containing T_1, T_3, and T_5, the algorithm compresses the input matrices belonging to two sentences by excluding all similar tokens. This step constructs two smaller input matrices, where input matrix 1 holds two tokens and input matrix 2 has one token. The condensed matrices are then fed into the input-weight stage for performance enhancement. Subsequently, the resultant condensed output matrices are expanded, waiting for later recovery. Note that we abbreviate the query, key, and value matrices (Q, K, V) as the output matrices for simplicity.

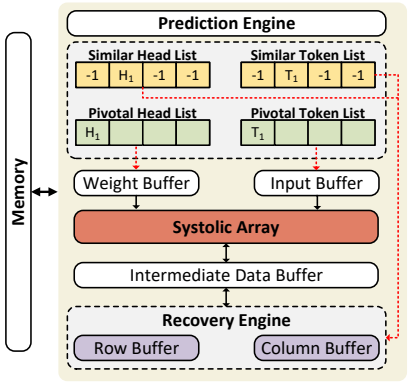


Figure 5: TSAcc overview.

3.3 Similar Token/Head Recovery

In this section, we propose a recovery operation to recover similar tokens/heads' features.

3.3.1 Similar Token Recovery. The similar token recovery recovers the query, key, and value matrices Q, K, and V in a copy-and-paste manner without heavy computation. To recover them, we first fill the holes in the expanded query, key, and value matrices by copying known results of the pivotal tokens row-by-row. Specifically, given similar token T_j referring pivotal token T_i as recorded in the similar token list, the token recovery copies the row i to row j in the expanded matrix. The token recovery is highlighted by a red block in in Fig. 4. Given the similar token T_4 whose pivotal token is T_1 (T_1 is recorded in $4th$ index of the similar token list), we copy the row 1 to row 4 to recover T_4's output features.

3.3.2 Similar Head Recovery. The similar head recovery is executed by filling the empty similar heads column-by-column. The head recovery copies the pivotal heads of the matrices to their similar heads. As it is trivial, we skip its details.

4 TEMPO-SPATIAL SIMILARITY AWARE ACCELERATOR

4.1 Architecture Overview

In this section, we propose the Tempo-Spatial Similarity Aware accelerator (TSAcc), which is shown in Fig. 5. TSAcc equips a novel prediction engine to efficiently predict the pivotal/similar tokens during runtime, and a recovery engine with fine-grained buffer management to handle the token recovery with minimal memory accesses. To perform the condensed matrix multiplication, TSAcc leverages the systolic array [5] without dedicated sparsity control.

4.2 Prediction Engine

In this section, we design a prediction engine to support the pivotal/similar token prediction efficiently.

4.2.1 Token Similarity Calculation Module. In the prediction engine, we first calculate the token similarity between tokens by designing an adder systolic array. The array can shift the reusable token within the systolic array. Specifically, each PE inside the array equips two adders. One calculates the absolute difference between two tokens, while another one directly sums the partial distance

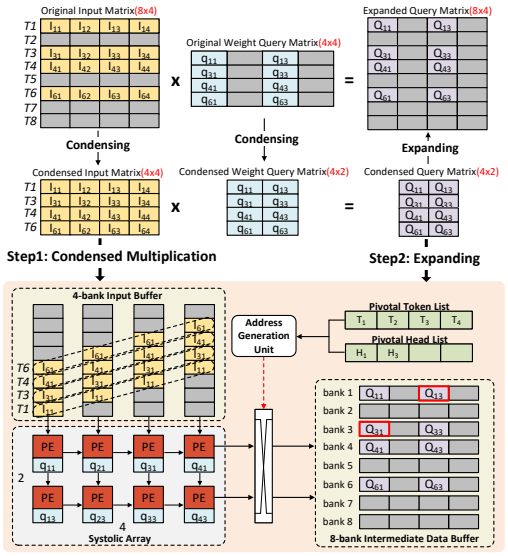


Figure 6: Fine-grained buffer management example.

addition result from the above PE. As a result, each column of the adder systolic array calculates a distance by receiving token values from the left and shifting the partial addition results to the below PE in the meantime and finally exports the distance to the distance buffer. We then feed the distances into comparators to perform paralleled binarization and the results are kept in the 1-bit binary map buffer. Entry i of bank j in the binary map buffer holds the binarization result $B_{i,j}$, indicating the similarity between tokens T_i and T_j.

4.2.2 Pivotal/Similar Token Identification Module. To identify the pivotal/similar tokens, the pivotal/similar token identification module first leverages the adders and a Top-k unit to complete the token similarity accumulation and accumulation result ranking. The token IDs of the Top-k accumulation results are recorded in the index buffer. Next, we get the pivotal tokens by traversing the tokens in the index buffer. The IDs of the pivotal tokens are kept in the pivotal token list. Subsequently, given the pivotal token T_i, we identify its similar token T_j by providing comparators in the pivotal/similar token identification module to identify whether $B_{i,j}$ is equal to "1". Afterward, the token ID i is updated into the entry j of the similar token list, meaning that pivotal token T_i is the reference token of the similar token T_j. Note that the entries in the similar token list are updated once because one similar token can only refer to one pivotal token.

4.2.3 Tile-based Pivotal/Similar Token Prediction. The above online token prediction regards all tokens in successive input sentences as a whole, which can find more similar tokens in a larger search scope to achieve higher speedup. However, it demands prediction between all the tokens before the attention computation. Such serial execution will lead to long latency. In order to pipeline the token prediction and the following attention computation, we propose to divide the original input matrix into independent and smaller input tiles. The more tokens in each input tile, the lower the parallelism, and the higher the accuracy is. The token number can be configured so that we can find the best performance with satisfactory

Table 1: Area and Power Breakdown of TSAcc.

Modules	Parameter	Area (mm^2)	Power (mW)
Prediction Engine	16 × 16 adder systolic array 16 adders Comparators Buffers	0.5	258.7
Systolic array	64 × 32 MACs	6.5	1040.2
On-chip Buffer	32KB input buffer 192KB weight buffer 32KB intermediate data buffer	1.6	1792.1

accuracy. In the experiment, for the best balance between accuracy and hardware complexity, the token number in each input tile is set to 16.

4.3 Fine-grained Buffer Management

In this section, we will introduce fine-grained buffer management that collaborates with the systolic array to place the condensed matrices in the right locations of the intermediate data buffer. This is important because it simplifies the following recovery operation that makes TSAcc compatible with conventional accelerators.

After the systolic array completes the condensed matrix multiplications, the address generation unit scatters the result with the rule that, the cth PE row should place the result to (bank i, entry j), given the streamed token ID i and head ID j. Specifically, in cycle t, the cth PE row gets the pivotal token and head IDs by accessing the $(t-c+1)$th location of the pivotal token list and the cth location of the pivotal head list. As the example in Fig. 6, in cycle 2, the first PE row outputs the result $Q31$ to (bank 3, entry 1), because the first PE row receives the pivotal token and head IDs 3 and 1 by accessing the $(2-1+1=2)$th and 1th locations of the pivotal token and head lists, respectively. This rule guarantees no bank conflict throughout the result scattering.

4.4 Token and Head Recovery Engine

To recover the final query, key, and value matrices on a row or column basis, we provide the token and head recovery engine. The token recovery copies the results of pivotal tokens to similar tokens row-by-row. Such a manner reads the entire entry of a bank of the intermediate data buffer and writes them back to another entry of a bank, leading to a bank conflict problem. To avoid long latency, we propose to recover multiple similar tokens in the meantime by opening multiple banks at a time, then write them to the banks indicated by the similar tokens that refer to them. Moreover, the head recovery copies the results of pivotal heads to similar heads column-by-column, which can be easily pipelined by taking advantage of the dual-port intermediate data buffer.

5 EVALUATION

5.1 Evaluation Methodology

Software Implementation. We evaluate our method on BERT [3] and GPT-2 [10]. Our evaluation benchmarks include GLUE, PAWS, IMDB, PAWS-X, and BIOSSES datasets. All models are implemented and executed using PyTorch framework [8].

Hardware implementation. We implement the proposed TSAcc in Verilog and synthesize it by Synopsys Design Compiler to get the chip area and total power under 28nm technology with a working frequency of 500MHz. We also use CACTI [2] to get the energy and area given the width, size, and the number of read/write of each SRAM. Table 1 provides the detailed design parameter and area/power breakdown for TSAcc. To evaluate the performance of TSAcc, we developed a cycle-accurate performance model based on Sanger [7]'s open-source simulator.

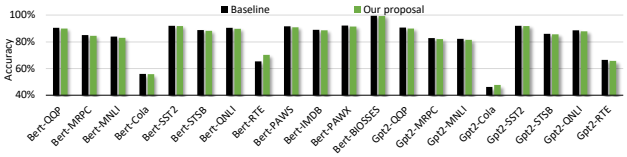


Figure 7: Accuracy of Baseline and the proposed algorithm.

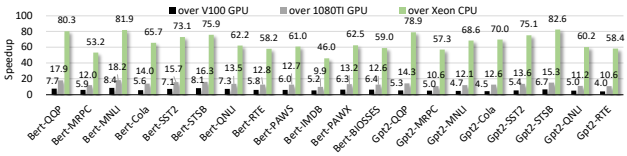


Figure 8: Speedup over GPU and CPU.

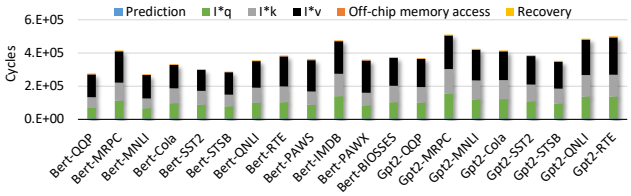


Figure 9: Execution cycle breakdown.

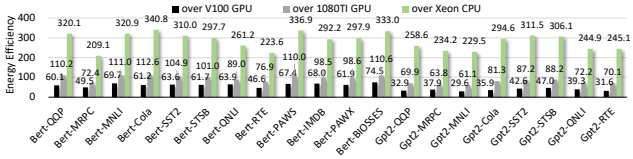


Figure 10: Energy efficiency over GPU and CPU.

Platforms for comparison. We compare TSAcc with modern hardware accelerators, including server GPUs (NVIDIA Tesla V100 PCIe 32GB, NVIDIA GTX 1080 Ti) and a server CPU (Intel(R) Xeon(R) CPU E5-2609 v4 @ 1.7GHz). We also compare with the state-of-the-art sparse attention accelerators, including SpAtten [17] and Sanger [7]. We get Sanger's performance results by taking their open-source code. We reproduce SpAtten's results by building a simulator according to their paper. For a fair comparison, we scale the number of multipliers of all accelerators to 64 × 32.

5.2 Evaluation Results

5.2.1 Accuracy Results. To verify the impact of our proposed algorithm on accuracy, we test it on BERT and GPT-2 models (denoted as "Baseline"). The results are presented in Fig. 7. Overall, our proposed algorithm demonstrates a remarkable performance, ensuring an accuracy loss of less than 0.8% when compared to the baseline. Similar results were observed with the GPT-2 model, where we achieved an accuracy loss ranging from 0.2% to 0.8%.

5.2.2 Comparison with CPUs and GPUs. Fig. 8 shows the speedup of TSAcc over V100 GPU, 1080TI GPU, and Xeon CPU. For BERT and GPT-2 models, TSAcc averagely achieves 6.7× and 5.1× speedup

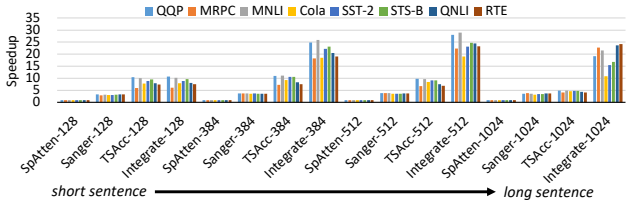


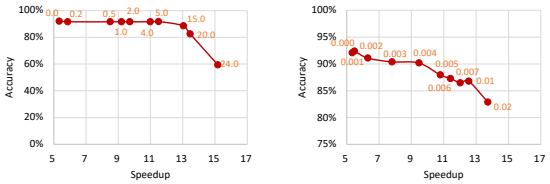
Figure 11: Comparison with other attention accelerators.

over V100 GPU, 14.1× and 12.5× speedup over 1080TI GPU, and 64.9× and 68.9× speedup over Xeon CPU. The speedup comes from the reduced computation and the optimized architecture of TSAcc that can highly parallelize the prediction and attention computation and support efficient recovery operation. Moreover, the limited input sentence length and batch size make GPU underutilized while TSAcc processes 64×32 operations in parallel so that our computing resources are easy to be fully-utilized.

We also give Fig 9 to show the detailed execution cycle breakdown of TSAcc. The figure shows that the prediction and recovery processes only account for 3% of the total execution cycles, thanks to the specialized hardware. The prediction engine alleviates the memory accesses for the token values, and the proposed recovery engine can fully pipeline the read and write operations and hold the intermediate matrices on-chip so as to eliminate the unnecessary memory access for recovering the matrices. Moreover, Fig 9 verifies that no matter in which NLP tasks, three matrix multiplications $I \times q$, $I \times k$, and $I \times v$ of the input-weight stage dominate the cycles.

Fig. 10 shows the energy efficiency results. To be specific, TSAcc is 62.3× and 37.1× better than V100 GPU, 99.6× and 74.2× better than 1080TI GPU, and 295.3× and 265.6× better than Xeon CPU on BERT and GPT-2 models. Energy savings mainly come from the much reduced unnecessary data access of the off-chip memory and similar tokens/heads' related computations.

5.2.3 Comparison with Other Accelerators. Fig. 11 compares TSAcc with other state-of-the-art attention accelerators, including SpAtten [17] and Sanger [7]. The speedup is normalized to SpAtten. Since TSAcc concentrates on optimizing the input-weight stage, we distribute the query-key and softmax-value stages to the traditional neural network accelerator TPU [5]. To evaluate the effectiveness of TSAcc, we conducted tests with sentence lengths ranging from 128 to 1024. For instance, when the sentence length is 128, we label TSAcc as TSAcc-128. As shown in the figure, TSAcc performs better than SpAtten and Sanger, especially in scenarios involving shorter sentences such as 128 and 384. Specifically, with a sentence length of 128, TSAcc achieves an average speedup of 8.5× and 2.7× compared to SpAtten and Sanger, respectively. This substantial speedup can be attributed to two factors: 1) SpAtten and Sanger only accelerate the query-key and softmax-value stages, which limits their potential speedup, particularly in tasks with short sentences where $Q \times K^T$ and $S \times V$ constitute a small portion of computations. 2) Short-sentence tasks generally exhibit lower redundancy, making it challenging for SpAtten and Sanger to capture high sparsity. In contrast, TSAcc explores temporal-spatial similarity within the crucial input-weight stage, achieving significant performance improvements by constructing condensed matrices based on predicted pivotal tokens/heads.



(a) (b)

Figure 12: Trade-off curves between speedup and accuracy.

5.2.4 Detailed Analysis. To trade off the performance and accuracy of TSAcc, we can tune the token threshold Thr_T and head threshold Thr_H. Fig. 12 shows two trade-off curves between the speedup over 1080TI GPU and accuracy of BERT on the SST-2 task by tuning Thr_T (Fig. 12(a)) and Thr_H (Fig. 12(b)). To obtain the curve of the temporal-level similarity aware algorithm, the spatial-level algorithm is not applied, and vice versa. We finally chose $Thr_T = 5.0$ and $Thr_H = 0.004$ as the default settings for the SST-2 task to balance its accuracy and performance. Those settings help us to achieve 11.5× and 9.5× speedup over 1080TI GPU with negligible accuracy loss.

6 CONCLUSION

This paper proposes TSAcc, an algorithm-architecture co-design for an efficient attention inference by leveraging the tempo-spatial similarity across successive sentences. Experiments show that TSAcc can deliver satisfactory performance gain with trivial accuracy loss so TSAcc is promising to deploy the complex attention-based models on mobile devices.

REFERENCES

[1] Jochen Alber et al. 2004. Polynomial-time data reduction for dominating set. *Journal of the ACM (JACM)* 51, 3 (2004), 363–384.
[2] Rajeev Balasubramonian et al. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *TACO* 14 (2017), 1–25.
[3] Jacob Devlin et al. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[4] Tae Jun Ham et al. 2020. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *HPCA*. IEEE, 328–341.
[5] Norman P Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *ISCA*. 1–12.
[6] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451* (2020).
[7] Liqiang Lu et al. 2021. Sanger: A Co-Design Framework for Enabling Sparse Attention using Reconfigurable Architecture. In *MICRO*.
[8] Adam Paszke et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* (2019).
[9] Zheng Qu et al. 2022. DOTA: detect and omit weak attentions for scalable transformer acceleration. In *ASPLOS*. 14–26.
[10] Alec Radford et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[11] Aurko Roy et al. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics* 9 (2021), 53–68.
[12] Thierry Tambe et al. 2021. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO-54*. 830–844.
[13] Yi Tay et al. 2020. Sparse sinkhorn attention. In *ICML*.
[14] Ashish Vaswani et al. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[15] Elena Voita et al. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv* (2019).
[16] Alex Wang et al. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv* (2018).
[17] Hanrui Wang et al. [n. d.]. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *HPCA*. IEEE.
[18] Ali Hadi Zadeh et al. [n. d.]. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *MICRO*. IEEE.