# ProSE: The Architecture and Design of a Protein Discovery Engine

Eyes Robson*
University of California, Berkeley
Berkeley, California, USA
eyes.robson@berkeley.edu

Ceyu Xu*
Duke University
Durham, North Carolina, USA
ceyu.xu@duke.edu

Lisa Wu Wills
Duke University
Durham, North Carolina, USA
lisa@cs.duke.edu

## ABSTRACT

Protein language models have enabled breakthrough approaches to protein structure prediction, function annotation, and drug discovery. A primary limitation to the widespread adoption of these powerful models is the high computational cost associated with the training and inference of these models, especially at longer sequence lengths. We present the architecture, microarchitecture, and hardware implementation of a protein design and discovery accelerator, ProSE (Protein Systolic Engine). ProSE has a collection of custom heterogeneous systolic arrays and special functions that process transfer learning model inferences efficiently. The architecture marries SIMD-style computations with systolic array architectures, optimizing coarse-grained operation sequences across model layers to achieve efficiency without sacrificing generality. ProSE performs Protein BERT inference at up to 6.9× speedup and 48× power efficiency (performance/Watt) compared to one NVIDIA A100 GPU. ProSE achieves up to 5.5 × (12.7×) speedup and 173× (249×) power efficiency compared to TPUv3 (TPUv2).

## CCS CONCEPTS

• **Computer systems organization** → **Special purpose systems**; *Neural networks*; *Heterogeneous (hybrid) systems*; **Systolic arrays**; • **Computing methodologies** → **Natural language processing**; • **Applied computing** → *Computational biology*.

## KEYWORDS

Accelerators, Neural Networks, Transformers, Domain-Specific Architecture, NLP, Protein Design

**ACM Reference Format:**
Eyes Robson, Ceyu Xu, and Lisa Wu Wills. 2022. ProSE: The Architecture and Design of a Protein Discovery Engine. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), February 28 – March 4, 2022, Lausanne, Switzerland*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3503222.3507722

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ASPLOS '22, February 28 – March 4, 2022, Lausanne, Switzerland*
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9205-1/22/02…$15.00
https://doi.org/10.1145/3503222.3507722

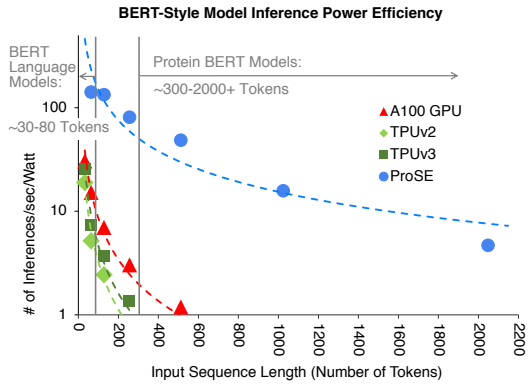


**Figure 1: BERT-style model inference efficiency (number of inferences per second per Watt) decreases dramatically as input sequence length increases. For short input sequences, representative of human language BERT, ProSE is able to obtain one order of magnitude better efficiency than GPU and TPU systems. As input sequence length passes 300 tokens, representing Protein BERT models, commodity platforms fail to perform >1 inference/sec/Watt[2], necessitating the need for architectural innovation.**

## 1 INTRODUCTION

Recent innovations in Natural Language Processing (NLP) enable powerful deep learning algorithms to accurately predict both experimental protein structure [8, 45] and drug-target interactions [43, 53], both of which can cut down the extraordinary costs (~$80 billion per year), failure rate (~90%), and lab-to-market average timeline of 12 years for drug discovery and validation [13]. These bio-repurposed NLP models, especially those employing BERT-style model structures [11], produce inferences much more speedily than wetlab experiments, and much more accurately than traditional computational biology algorithms[1], increasing the quality of drug candidates in expensive human trials. Unfortunately, commodity accelerated platforms such as the latest GPU and TPU systems lack power efficiency when executing BERT-style models. Further, the performance of these systems is optimized primarily for either non-NLP models or NLP models that target human language with short input lengths, while protein engineering and discovery requires longer lengths.

[1]For the specific case study of CYP450 mentioned in "Hunting for New Drugs with AI" [13], the prediction accuracy of CYP450-related toxicity went from ~66% to 95% when it is AI-assisted.

A plethora of specialized accelerators deliver exceptional performance and efficiency when performing inferences on CNNs [6, 7, 40] and DNNs [14, 20, 21, 26, 37]. However, few works have explored accelerating the newly developed BERT-style models [9, 11, 27, 30, 48, 51]. These models exhibit fundamentally different computational patterns to CNNs, and CNN accelerators dedicate significant resources to accelerate convolutional filters, something not present in most Transformers. As such, these architectures cannot efficiently predict protein binding affinity for drug discovery.

Of architectures handling Transformers [25], few compete with modern tensor-core-based GPUs or systolic-array-based TPUs and no accelerators attempt long length inputs to date. Figure 1 depicts the impact of input sequence length *in tokens* of BERT-style Transformer inference efficiency. For language models, a token is a word or subword and a sentence is typically 30–80 tokens. For protein models, an amino acid is a token and a protein is an input sequence of typically 300+ tokens. At these longer input lengths, the proportions of the operation mixes change, rendering GPU or TPU platforms far less efficient than performing the same tasks with short input lengths. Further, Transformers occasionally require matrix multiplications with smaller matrices than the TPU uses (128x128 systolic arrays) but larger matrices than what the GPU tensor core optimizes (4x4x8 tensors), leaving the TPUs underutilized and the GPUs less performant.

Besides GPUs and TPUs, few accelerators target BERT-style models [18, 19] and these accelerators only accelerate a portion of the model such as the attention mechanism, leaving large portions to be executed on commodity platforms. Sparse GEMM accelerators [42] require significant sparsity and do not accelerate large portions of Transformer models, while other systolic-array-based accelerators [15] do not accelerate Transformer models at all.

We design and evaluate a systolic engine targeting BERT-style model inference optimized for protein-size inputs, called ProSE. ProSE is a multi-threaded heterogeneous software-hardware co-designed system that contains a collection of heterogeneous systolic arrays and special functions, with each type of systolic array capable of executing a prominent operation sequence for a BERT-style model in dataflow fashion, such as matrix multiplication followed by matrix addition or GELU function. ProSE operates on *streaming* input data using an *output*-stationary systolic array with no provision for specialized intermediate data storage such as a local scratchpad on the accelerator. Instead ProSE uses the accumulator register within each multiply-accumulate unit as intermediate storage to reduce the memory requirements of large AI models [28].

This paper makes the following contributions:

- A performance-, power-, and area-efficient protein discovery systolic engine called ProSE. Using a collection of heterogeneous streaming systolic arrays with varying sizes and functionalities, ProSE demonstrates an architecture that is capable of executing matrix multiplications, SIMD ALU operations, and special activation functions efficiently, providing two to three orders of magnitude better efficiency compared to the latest GPU and TPU systems.
- The ProSE design process including computational pattern analysis of a bio-repurposed BERT-style model, a detailed design space exploration, and microarchitecture optimizations to eliminate expensive intermediate storage, reduce superfluous data movements, approximate special functions, and increase efficiency via task parallelism across threads and pipeline parallelism within threads.
- A detailed microarchitecture comparison of ProSE and TPU to highlight three novelties: 1) significantly better efficiency with increased input sequence lengths using heterogeneous systolic arrays, 2) exploitation of a streaming systolic array without dedicated local scratchpad via *local dataflow* as opposed to TPU's *global dataflow* that employs a large, power-hungry *Unified Buffer*, and 3) a novel left-rotation-capable systolic array that tightly integrates with a SIMD unit to perform element-wise SIMD ALU operation or special function without having to communicate to a local buffer.

[2]For input sequence lengths >512 on the A100 and input sequence lengths >128 and >256 on the TPUs, the inference efficiencies are lower than 1 and therefore not shown on this log-scale graph.

## 2 BACKGROUND AND MOTIVATION

Substantial groundbreaking work has documented the success of NLP models in protein modeling [1, 3, 8, 35, 43, 45], especially those employing BERT-style structures [11]. These models uniquely benefit from *unsupervised pre-training*, something CNNs struggle to do for sequence data. This ability does not merely enable highly accurate downstream protein structure prediction [43, 45], but can permit strong zero-shot performance [35] and the potential to augment or outpace the impressive AlphaFold2 [8, 22]. Despite the success of these bio-repurposed NLP models, commodity platforms overlook the computational needs of these NLP models, especially runtimes encountered executing BERT-style models on proteins.

To begin, we provide some background on 1) the fundamental differences between a human language BERT model and a Protein BERT model, 2) a feasibility study aimed at the software Protein BERT model we developed to validate a well-known antibody target for breast cancer, the HER2 protein, and 3) the profiling of inference of a Protein BERT model to motivate and guide our design decisions.

### 2.1 Human Language BERT Model vs. Protein BERT Model

The human language BERT model developed at Google AI has expanded into various BERT-style models including ALBERT [27] – a lite BERT for self-supervised learning representations, RoBERTa [30] – an augmented training procedure for the BERT with input manipulations for model robustification, and MobileBERT [48] – a compact task-agnostic BERT for resource-limited devices. As depicted in Figure 2(a), regardless of model parameters, these models take a sentence or sentences as inputs, *tokenize* the inputs by words (each token is represented as a rectangular bounding box such as "cat"), perform inference on the BERT model, and use downstream models for desired language tasks.

To perform NLP tasks, these BERT-style models use pre-trained parameters specific to learning human language. These tasks might be as simple as language translation, e.g. translating "Where is Berlin?" from English to German (a frequently short-length task). Another prominent task is multiple-choice question answering, e.g. answering the question "Where is Berlin?" by selecting between

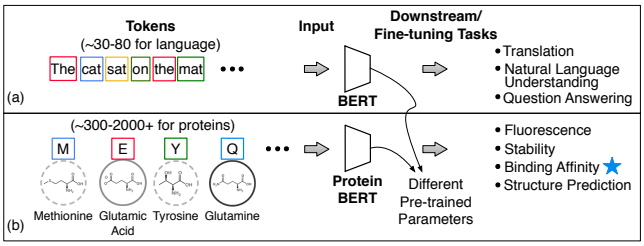


**Figure 2: Summary of inputs and downstream tasks for (a) natural language processing applications of BERT and (b) protein design applications of Protein BERT models. The primary difference between the two is the pre-trained parameters. However, a typical language task has an input between 30-80 tokens in length while the majority of protein sequences are 300-2000+ tokens. This gives the two vastly different execution profiles in practice. The star annotates the *Binding Affinity* task we used in Protein BERT.**

A) Nigeria, B) United States, or C) Germany. An alternative to this is free-response question-answering by spelling out the answer, e.g. G-E-R-M-A-N-Y. Other tasks might include natural language understanding, such as taking in simple inputs like "Berlin is in Germany and Austria is not a part of Germany", then determining whether "Berlin is the capital of Austria". To train such BERT-style models, users often make use of supervised learning based on large-scale labeled datasets in a specific language).

The Protein BERT models we use are identical in structure to human language BERT models. However, rather than taking in a human language sentence, the model takes in a protein sequence, represented as an amino acid alphabet, tokenizes sequence into individual characters per token, performs inference on the Protein BERT model, then uses downstream models to perform desired protein design tasks as shown in Figure 2(b). To perform antibody drug development tasks, Protein BERT models can use pre-trained parameters specific to learning protein representations (e.g., parameters from TAPE [43] or ESM [35, 45]). These models are then trained on downstream fine-tuning tasks such as fluorescence (certain proteins fluoresce under the right biological conditions), stability (i.e., predict if a protein will remain in its native folded conformation or unfold and change conformations), binding affinity (i.e., the strength of the interaction between two proteins), or structure prediction (i.e., predict a protein's three-dimensional structure from its amino acid sequence, as in AlphaFold [10, 22]). In our experiment below, we train a downstream model on a binding affinity prediction task.

Even though there are no structural differences between BERT-style models for human language and our Protein BERT model (model parameters aside), due to the difference in input domain (language vs. protein sequence), the typical token lengths for these two types of models differ significantly. For a human language model, a typical input sequence length (in tokens) is 30 to 80 tokens. For a Protein BERT model, however, protein inputs require a minimum input sequence length between 300 to 2000 tokens depending on the task, as multiple functional structural units (protein domains) can exhibit long-range effects that requires joint modeling [22]. While some tasks in human language (e.g. translating an entire book from scratch) can achieve higher performance with additional sentence context (longer length) [50], this is optional based on user discretion, unlike protein sequences, which are *obligately* long in length. In the future, more long-length tasks may emerge for human language models, and architectures like ProSE will benefit these tasks as well.

The input sequence length significantly alters the runtime behavior of BERT-style models (see Section 2.3), presenting scalability challenges such as both compute time and memory footprint increase quadratically as a function of input sequence length for some operations. This issue of length further motivates a specialized design more efficient for longer sequence lengths, as commodity platforms such as TPUs or GPUs optimize for non-BERT models or short input BERT-style models only.

## 2.2 Software Protein Binding Evaluation

Pre-trained protein language models (e.g. the aforementioned Protein BERT using TAPE parameters) have had their transfer learning predictive capacity validated by multiple wet lab experimentation papers [3, 45]. Here we perform an *in silico* validation of a general task for drug development, binding affinity prediction, via training a downstream model on the antibody Trastuzumab (brand name Herceptin) and similar antibody BH1. In general, the higher the binding affinity between antibody and pathogen the more likely the antibody will bind to the protein and neutralize it, making for a promising drug candidate. In our experiment, both Herceptin and BH1 antibodies bind the HER2 protein, a critical target in breast cancer [4]. We construct a Protein BERT and downstream task in PyTorch to assess the feasibility and applicability of our software model for drug development. The inputs to our Protein BERT model are the *Fab* subsequences of Herceptin and BH1 variants. The *Fab* subsequence is the part most responsible for protein binding (~450 amino acids in length).

Our downstream model performs feature extraction via the Protein BERT model from TAPE and fits a regularized linear regression model [3] on 39 variant Herceptin *Fab* sequences with 35 BH1 *Fab* sequences as an independent test set [46]. We measure the test set accuracy using *rank correlation*, a statistic that measures the degree of similarity between different rankings of the same variables. The rank correlation informs which antibody is predicted to be most likely to bind to the HER2 protein. This software experiment achieves a rank correlation of 0.5161, which while perhaps not quite as accurate as the geometric binding affinity model OSPREY [17], is more than sufficient for experimental validity (i.e., near or above 0.5). OSPREY is a high-accuracy approach that simulates *every* atom in detail via costly protein structure information at the cost of slow software inference (~4 orders of magnitude). Unlike many traditional approaches, BERT-style models are able to mimic the accuracy of OSPREY while exceling at hardware acceleration, and unlike OSPREY, which only performs binding affinity predictions, BERT-style models are applicable to arbitrary downstream tasks, as detailed in Figure 2. Furthermore, the modularity of BERT-style protein design software gives our workflow the ability to automatically improve (without manual engineering) as larger and more powerful Protein BERT-style models are developed [8, 35, 45], which will likely close this accuracy gap.

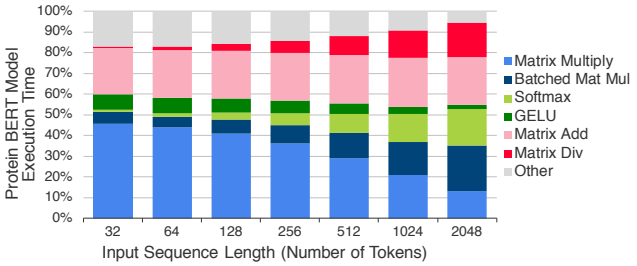


**Figure 3: Runtime breakdown of operations for Protein BERT as a function of input sequence length. While matrix multiply operations take up the bulk of execution time with shorter sequence lengths, the ratio decreases as input lengths increase and other operations start to dominate.**

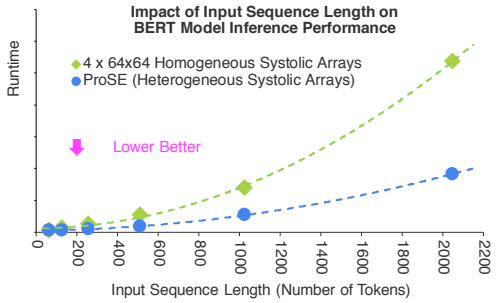


**Figure 4: BERT-style model inference runtime increases quadratically as input sequence length increases. For short input sequences, there is little difference in performance when using large homogeneous systolic arrays vs. heterogeneous systolic arrays. However, as input sequence length passes 300 tokens, representing Protein BERT models, the heterogeneous architecture eliminates unnecessary overhead and provides much better performance.**

## 2.3 Profiling of the Protein BERT Model

To design an efficient inference accelerator for protein discovery, we profiled a Protein BERT model implemented in PyTorch performing inferences on input sequence lengths ranging from 32 to 2048. The inputs to this Protein BERT model are synthetic protein strings. Human language BERT models and Protein BERT models do not differ in structure, only in weights and the typical input sequence lengths. The execution breakdown on an A100 GPU with machine configuration depicted in Table 1 is presented in Figure 3. We use a batch sizes of 24576, 12288, 6144, 2048, 512, 128, and 64 for input lengths 32, 64, 128, 256, 512, 1024, and 2048, respectively, as this yields the best inference time using the GPU's limited 40 GiB memory. We observe the percentage of matrix multiplications (Matrix Multiply) decreases while that of other operations, especially element-wise operations (Matrix Add/Div), as well as special functions (e.g., Softmax), increases. However, matrix multiplications (batched and unbatched) comprise of 35%–52% of the total runtime for all the input lengths profiled, signaling the importance of an accelerator designed to perform matrix operations efficiently.

**Table 1: Configuration of the A100 GPU Platform used for BERT Profiling.**

| A100 GPU Platform Machine Configuration | |
|---|---|
| Host Processor | Intel Xeon<br>96 C, 3 GHz |
| Memory | 1152 GiB DDR4 |
| GPU | A100-SXM4<br>6912 CUDA Cores, 432 Tensor Cores |
| GPU Memory | 40 GiB HBM2 |
| External Interface | NVLink 3.0 |

From the inspection of the Protein BERT software model, we observe: 1) matrix multiplications (MatMuls) and batched matrix multiplications (BMMs) use a handful of input matrix sizes with the majority of the time spent executing much smaller matrices than what a TPU provisions (i.e., 128×128 systolic arrays for both TPUv2 and v3) but much bigger than what an A100 tensor core (i.e., 4×4×8 tensor computations) provisions and 2) element-wise SIMD operations including special functions are usually dependent on the result of a MatMul or a BMM. These observations drive our design decisions to use *smaller and varying sized systolic arrays* to execute smaller input matrix sizes to increase utilization, and to architect ProSE to use novel *left-rotation-capable systolic arrays* to perform MatMuls followed by dependent element-wise SIMD operations or special functions in a single dataflow without the need to store and refetch intermediate data.

## 2.4 Impact of Input Sequence Lengths and Heterogeneity on BERT Performance

Systolic-array-based designs have shown exceptional efficiency when executing CNN and DNN models [6, 7, 20, 21]. Noting the mismatch between hardware provisioned by a homogeneous and large systolic-array-based architecture such as the TPU and the smaller input matrix sizes utilized by the Protein BERT model, we conduct an experiment investigating relationships between input sequence lengths and the heterogeneity of systolic arrays shown in Figure 4. We vary the hardware systolic array sizes to better match software by constructing a heterogeneous systolic-array-based architecture we call ProSE and compare the inference runtime with a homogeneous systolic-array-based architecture that is comprised of four 64×64 systolic arrays, resource-equivalent to the number of processing elements (PEs) on one TPU 128×128 systolic array. The hardware resource provisioned for both the homogeneous and heterogeneous (ProSE) systolic engines are roughly the same to ensure fairness (16K PEs). The runtime for both systolic-array-based architectures increase steadily as the input sequence length increases. However, the homogeneous architecture has a much steeper slope as input lengths increase to beyond 128 tokens due to the unnecessary overheads of executing small matrices as inputs on several large homogeneous systolic arrays and the lack of sufficient SIMD units to support element-wise and special function operations. This experiment shows that a *heterogeneous systolic-array-based architecture* that matches the software behavior can eliminate unnecessary overheads, significantly increase execution efficiency, and provide much better scalability.

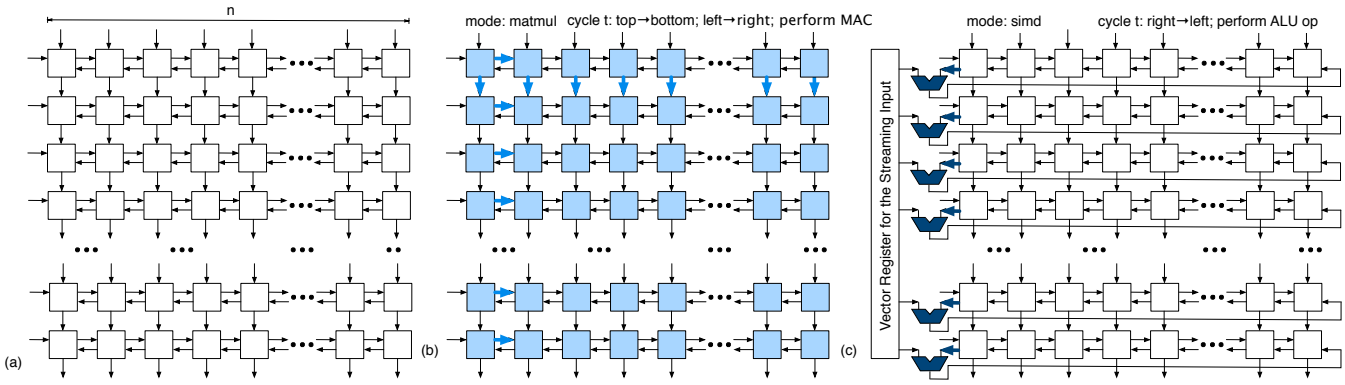


**Figure 5: This figure shows the traditional systolic array design with dimensions $n \times n$ in (a), the systolic array performing a matrix multiplication in matmul mode depicted in (b), and the systolic array performing a SIMD ALU operation in simd mode illustrated in (c). In order to perform SIMD ALU operations using the systolic array, a column of $n$ ALUs need to be implemented. To eliminate unnecessary stall, pipelining interleaved matrix multiplication and matrix SIMD ALU operations informed our design to have the systolic array act as a large column left-rotator, producing $n$ SIMD operation results per cycle.**

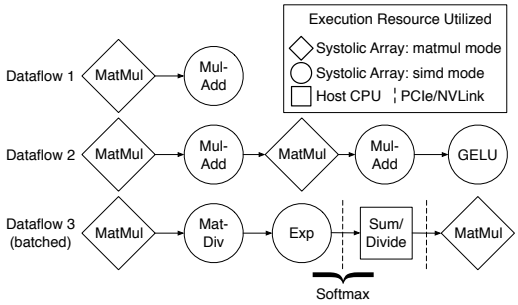


**Figure 6: Across model layers, we found three major operation sequences are executed for around 90% of the total inference time. If each operation sequence can be executed in a dataflow-esque fashion on the accelerator without communicating with the host, it would maximize the efficiency of the design. Note that Dataflow 3 still has a softmax operation that requires host-accelerator communications, trading performance for hardware simplicity.**

# 3 PROSE SYSTEM

## 3.1 Software-Hardware Co-Design

***ProSE Computational Patterns.*** To exploit common computational patterns in ProSE, we analyze Protein BERT model execution and group around 90% of the operations into three major operation sequences, Dataflows 1, 2, and 3, as shown in Figure 6. Each pattern can be performed wholly on the accelerator via pipelined dataflow-esque chaining to eliminate unnecessary data movement and host-accelerator communication.

***ProSE Systolic Arrays.*** Figure 5(a) depicts a systolic array containing $n \times n$ processing elements (PEs). To support the three major dataflows, all of which contain matrix multiplication followed by SIMD ALU operations, we design our systolic arrays to operate in two modes: *matmul* and *simd*. In matmul mode, systolic arrays perform MatMuls by moving data across PEs top-to-bottom, left-to-right, and execute multiply-accumulate per PE as depicted in Figure 5(b). In simd mode, systolic arrays act as large left column rotators that perform element-wise SIMD-style ALU operations by moving data across PEs right-to-left, executing SIMD ALU operations using inputs from the left-most column of the systolic array and the vector register that stores the streaming input as depicted in Figure 5(c). To enable stall-free dataflow pipelining between operations, the simd mode moves data right to left so the SIMD ALUs can start executing as soon as the left-most column of the systolic array has completed its matrix multiplication without having to wait for the rest of the columns to complete operations. The two modes work in concert to allow stall-free interleaved MatMul and SIMD ALU operations by keeping the intermediate data in the PE accumulators (PE microarchitecture described in Section 3.2).

***Systolic Array Types.*** Prior work has demonstrated rectangular (non-square) systolic arrays can improve performance on irregular matrices during GEMM operations [42]. However, this gain is from improved tiling, which is not a primary limitation at longer input lengths as our typical square-shape systolic array demonstrates high utilization in our evaluations.

To support the dataflows depicted in Figure 6, each with a different special function, three obvious design choices suffice: 1) a homogenous collection of systolic arrays each equipped with the capability to execute GELU and Exp, 2) a homogenous collection of systolic arrays with a subset equipped with the capability to execute GELU and a subset equipped to execute Exp, or 3) a heterogenous collection of systolic arrays with a subset equipped with the capability to execute GELU and a subset equipped to execute Exp. Since the three major operation sequences are mutually exclusive and execute different portions of the model, we choose to divide up the systolic arrays into three types, each capable of executing one operation sequence, to maximize system efficiency. There are three types of systolic arrays incorporated in ProSE: the

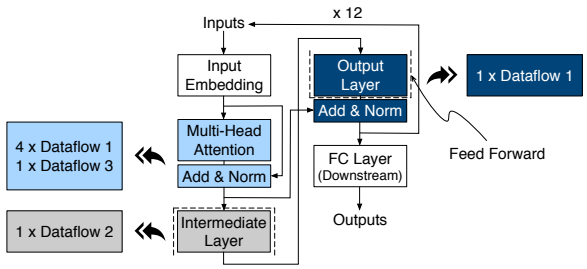


**Figure 7: This figure depicts one layer of the Protein BERT model: attention, intermediate, and output sublayers, as well as their dataflow mappings. A full Protein BERT model has 12 consecutive layers ("x 12") before outputting the results (e.g. to a downstream FC layer for a downstream task).**

*M-Type* supports MatMuls and SIMD ALU operations, the *G-Type* supports MatMuls, SIMD ALU operations, and GELU special functions, and the *E-Type* supports MatMuls, SIMD ALU operations, and Exponential functions (Exps).

***Systolic Array Mapping and Sizes.*** We characterize the Protein BERT model sublayer by sublayer and analyze the matrix multiplication sizes produced by PyTorch. The model is separable into the *Attention*, the *Intermediate*, and the *Output* sublayers as shown in Figure 7 along with the dataflow mappings. For example, the *Attention* calculations as well as the *Output* sublayer use the largest matrices for computations, usually matrix dimensions $m = 65536$, $k = 768, 3072$, and $n = 768$. These corresponds to executing the operation sequence of Dataflow 1 and the M-Type systolic arrays. In order to maximize performance for such large MatMul computations, we chose to implement the M-Type systolic array as the largest systolic array in ProSE at the dimension of 64×64.

When computing MatMul of varying sizes, smaller systolic arrays suffer less startup and draining costs versus larger systolic arras if input matrices are small, but they take much longer to complete MatMuls. To compute SIMD ALU operations such as matrix additions or divisions (by multiplying reciprocal constants) using our design shown in Figure 5(c), the smaller the systolic array, the larger the ratio of available SIMD ALU computation units to PEs. For example, for a $n \times n$ systolic array, there are $n$ SIMD ALU units and there are $n^2$ PEs, making it a 1:$n$ ratio. Therefore, smaller systolic array will have many more SIMD units per PE, allowing any SIMD intensive operation sequences to perform better.

The batched matrix multiplications in Figure 3 consist of dot products in the attention sublayer and use the smallest matrices, usually dimensions $m = 1024$, $k = 64$, and $n = 512$. This is covered by Dataflow 3 and E-Type systolic arrays, which interleave matrix division and Exp with MatMul, producing the most SIMD-intensive operation sequences. To maximize utilization, we chose to implement E-Type systolic array as smaller systolic arrays, either 32×32 or 16×16. Dataflow 2 and G-Type systolic array that support interleaving GELU and MatMul are also implemented with medium or small systolic arrays for similar reasons. We verify our intuition for choosing a heterogeneous collection of systolic arrays by performing a design space exploration in Section 4.2.

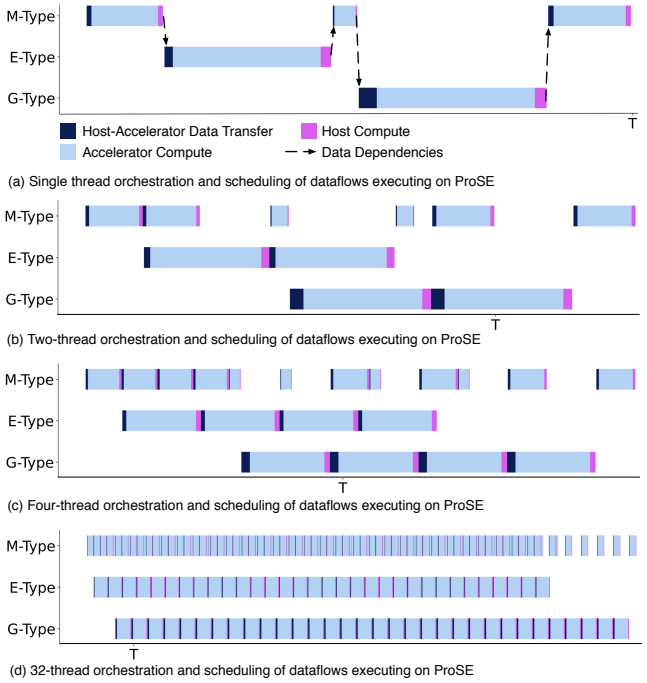


**Figure 8: ProSE uses an orchestration and scheduling mechanism that maps software threads to systolic arrays. Multithreading enables parallel execution of systolic arrays, significantly improve system throughput and efficiency.**

***Multithreaded Execution of Heterogeneous Systolic Arrays.*** Figure 8 shows a typical sequence of data-dependent dataflows (Dataflow 1 → 3 → 1 → 2 → 1) from the Protein BERT model and how the software is mapped onto the three types of systolic arrays. Exploiting task parallelism (i.e., a data-dependent dataflow is a task), we enable the parallel execution of multiple systolic arrays, significantly increasing system throughput. We show the orchestration and scheduling of a single-thread, a 2-thread, a 4-thread, and a 32-thread ProSE . Note that every time a type of systolic array is mapped to an active software thread, a host-accelerator data transfer happens. We implement three I/O buffers, one for each type of systolic array, and handle thread contentions with mutex locks. With more threads, fewer data-dependency bubbles occur, but the overhead of thread contention goes up. Through experimentation, we chose 32 threads for ProSE.

## 3.2 ProSE Architecture and Microarchitecture

In this section we describe the ProSE architecture and the microarchitecture in detail. Since ProSE is a systolic-array-based architecture, we carefully compare the differences between a TPUv2 microarchitecture and ProSE to highlight the novelty of our proposal. Note that because TPU's exact microarchitectural steps are not publicized, we extrapolate the steps based on the block diagrams and publications that describe the functionalities of these microarchitectural units.

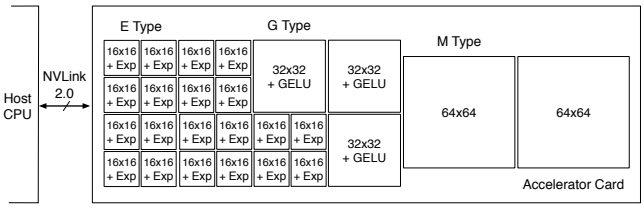


**Figure 9: ProSE is a collection of heterogeneous systolic array with varying sizes and functionalities. This configuration of ProSE consists of 2 64×64 M-Type, 3 32×32 G-Type, and 20 16×16 E-Type systolic arrays.**

***System Overview.*** ProSE is a heterogeneous collection of *output-stationary*, *streaming* systolic arrays packaged on one accelerator card communicating with a host CPU via a high-bandwidth external interface such as NVLink 2.0 [12, 39]. Figure 9 shows an example configuration of ProSE. For our ProSE system, we envision a host CPU that is capable of supporting four NVLinks similar to what the latest NVIDIA Grace CPU [38] is capable of, with each NVLink connecting to one ProSE instance, totaling four ProSE instances per system.

***ProSE Operations.*** The systolic arrays designed for ProSE support the datatype *bfloat16* [5] and a combination of the following five primitive operations derived from careful characterizations of the Protein BERT model:

*MatMul* performs matrix multiplication $C = A \times B$ where $A$ and $B$ are the input matrices and $C$ is the output matrix.

*MulAdd* performs $C = \alpha A + \beta B$ where $\alpha$ and $\beta$ are scalar constants.

*MatDiv* performs an element-wise reciprocal multiplication $C = A \times \frac{1}{\alpha}$ where $\alpha$ is a scalar constant.

*Exp* performs an element-wise exponential function $exp(x)$ using a lookup table where $x$ is one element of the input matrix $A$ and $C = exp(A)$. This operation is used to support the *softmax* activation function in conjunction with the host CPU. The summation and the division of the softmax activation $softmax(A) = \frac{exp(A_{ij})}{\sum exp(A_{ij})}$ are performed on the CPU after the exponential functions are completed on the systolic array.

*GELU* performs an element-wise Gaussian Error Linear activation function $GELU(x) = 0.5x(1 + tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$ where $x$ is one element of the input matrix $A$ and $C = GELU(A)$.

***ProSE Processing Element.*** Instead of having large local scratchpads or accelerator-attached memory as other deep learning accelerators have employed such as the TPU and the A100 GPU, ProSE implements output-stationary systolic arrays (unlike the TPU, which is weight-stationary) and uses the local multiply-accumulate (MAC) accumulators as intermediate storage. ProSE *streams* inputs continuously from the host, preventing unscalable memory usage on large models. However, the streaming nature of our design would be bottlenecked if external interface bandwidths are insufficient, making it necessary to deploy ProSE with fast and high-bandwidth host-accelerator interfaces.

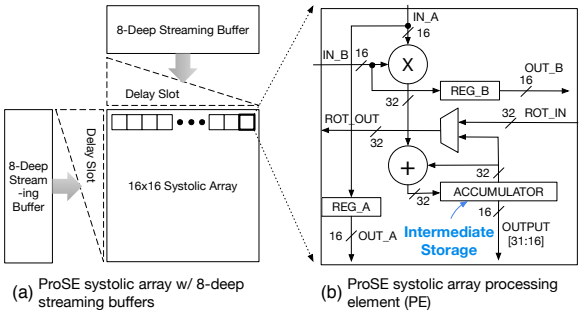


**Figure 10: (a) Each ProSE systolic array is equipped with two 8-deep streaming buffers and delay slots to allow streaming inputs. (b) Each PE is implemented with a 16-bit multiplier, adder, and a 32-bit accumulator.**

Figure 10(a) shows a diagram of a 16×16 systolic array that accommodates uninterrupted streaming from the host using an 8-deep streaming buffer for both input matrices. We validate that 8-deep streaming buffers are sufficient to cover the latency given the NVLink bandwidth provisioned for all types and all sizes of the ProSE systolic arrays (i.e., M-, G-, and E-Types as well as 64×64, 32×32, and 16×16 sizes) using Little's Law and our performance model. These stream buffers are synthesized as registers. A close-up microarchitecture block diagram for our systolic array processing element (PE) is shown in Figure 10(b). MACs are executed using bfloat16 datatype and accumulated using a 32-bit accumulator similar to TPUs to prevent precision loss. The 32-bit accumulators in each PE are used as intermediate storage.

***MatMul Executed on TPUv2 vs. ProSE.*** Figure 11 shows how a matrix multiply is performed using a systolic array in a TPUv2 and in ProSE. To illustrate, we show a MatMul on two 4×4 input matrices using a 2×2 systolic array. The operations needed to run on a TPUv2 depicted in Figure 11(a) while the operations needed to run on ProSE depicted in Figure 11(b) and show the first two steps to perform MatMul in Figure 11(c). The $A_{4\times4} \times B_{4\times4}$ MatMul is decomposed into four $A_{2\times4} \times B_{4\times2}$ MatMuls to execute on a 2×2 systolic array.

For a TPUv2 to compute step 1, it performs eight operations illustrated in Figure 11(a): ① Load weight matrix B into Weight FIFO from DDR, ② Pre-load weights into systolic array as (weight-stationary unlike ProSE), ③ Stream (the entire) matrix A from the host into the Unified Buffer (a large local scratchpad), ④ Setup input matrix A, ⑤ Shift input matrix A into systolic array, ⑥ Perform MatMul, ⑦ Perform accumulation, and ⑧ Write partial results to the Unified Buffer. Because the Unified Buffer has already streamed in matrix A in its entirety, the second step would repeat operations ④ through ⑧, what we term *global dataflow*.

For ProSE to compute step 1, it performs four operations illustrated in Figure 11(b): ① Stream matrix B-left-half from the host and shift into systolic array, ② Stream matrix A-top-half from the host and shift into systolic array, ③ Perform MatMul, and ④ Write partial results back to the host. For the second step, ProSE will

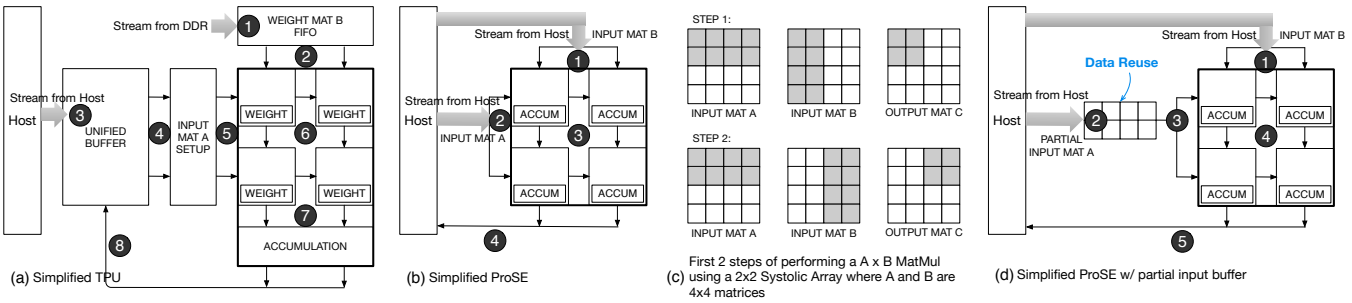


**Figure 11: A MatMul performed on (a) a TPUv2 microarchitecture and (b) a ProSE microarchitecture. The main difference is that TPUv2 uses a weight-stationary systolic array while ProSE uses an output-stationary systolic array. Another distinction is that TPUv2 employs a large local scratchpad to store intermediate results (the Unified Buffer) whereas ProSE uses the 32-bit accumulators as intermediate storage. (c) A larger MatMul decomposed into smaller MatMuls to execute on a smaller systolic array. (d) An input partial buffer is implemented to allow data reuse and boost performance in a limited bandwidth scenario.**

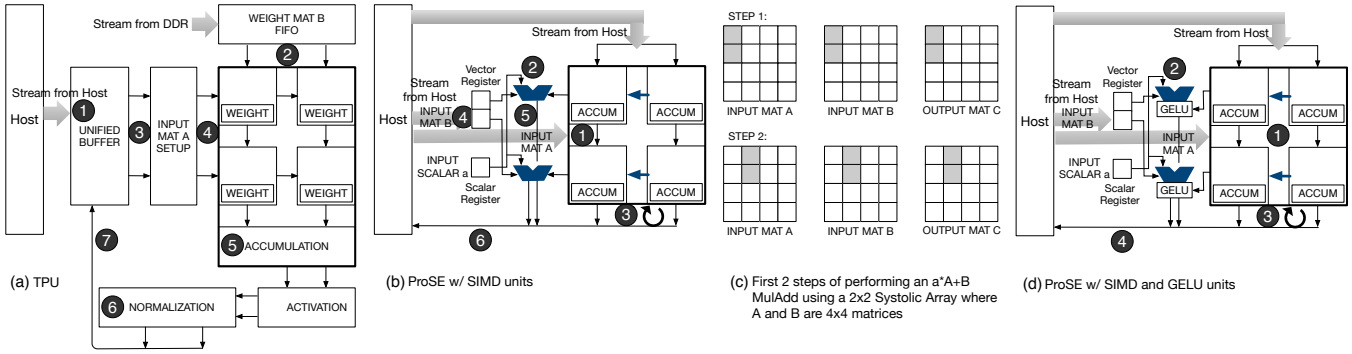


**Figure 12: A MulAdd performed on (a) a TPUv2 microarchitecture and (b) a ProSE microarchitecture. The main difference is that TPUv2 uses Normalization to perform scaling and Accumulation to perform addition while ProSE uses SIMD ALUs and left-rotating columns to exploit local dataflow and pipeline parallelism. (c) A larger MulAdd decomposed into smaller MulAdds to execute on a smaller systolic array. (d) A GELU lookup table is implemented per SIMD ALU unit to provide fast approximations of the activation function.**

repeat operations ① through ④. To exploit data reuse and alleviate the host to accelerator streaming bandwidth requirements, we implement a configuration of ProSE that provides an *input partial buffer* as shown in Figure 11(d). The input partial buffer has enough room to hold one step of the input matrix A (in this example, a 2×4 matrix) for step 2 to reuse matrix A-top-half without having to stream from the host. What we termed *local dataflow* is then performed via operations ① ③ ④ ⑤ in Figure 11(d) and repeated until the input data stored in the partial buffer have to be replaced. Note the input partial buffer only stores partial inputs streamed from the host and not intermediate data since that data is stored in the accumulators.

***MulAdd Executed on TPUv2 vs. ProSE.*** Figure 12 shows how a MulAdd $a^*A_{4\times4} + B_{4\times4}$ is performed on a systolic array in a TPUv2 (Figure 12(a)) and ProSE (Figure 12(b)), where $a$ is a scalar constant. ProSE employs a vector register of the same width as one column of the systolic array (in this toy example, two elements wide) and a SIMD unit that contains the same number of ALUs. It also employs a scalar register and can broadcast the scalar value to the SIMD ALUs. In order to perform element-wise SIMD operations without local scratchpads, the systolic array uses the accumulator register per MAC as intermediate storage and the systolic array is equipped with a left-rotation capability, shifting matrix stored in the systolic array to the left by one column after performing a SIMD operation. To perform MulAdd, ProSE performs the following operations: ① Stream matrix A-upper-left-quadrant (top half of column 1 and column 2) from the host and shift into the 2x2 systolic array, ② Broadcast scalar $a$ from the scalar register to the SIMD ALU units, ③ Rotate systolic array to the left by one column and shift the left-most column into the SIMD ALUs and perform $a$*A for two elements (i.e., set ALU op to MUL), ④ Stream matrix B-upper-left-column (top half of column 1) from the host and shift into the vector register, ⑤ Perform $a$*A + B by using the input from the vector register (i.e., set ALU op to ADD), and ⑥ Write partial results back to the host.

For the TPUv2 to execute a MulAdd (see Figure 12(a)), it uses Normalization scale a*A, Accumulation to add a*A+B, and the Unified Buffer to store the intermediate results. Briefly, the operations might look like ① Stream matrix A from the host into the Unified

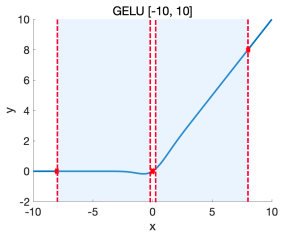


**Figure 13: GELU is only computed in the shaded areas. Inputs outside of these areas (either too large or too small) produce results that are unnecessary to store in the lookup table.**

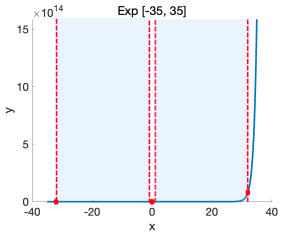


**Figure 14: Exp computation is also done in certain ranges to optimize hardware efficiency. We have validated that these truncation policies do not affect the accuracy of the models we study.**

Buffer, ② Load all 1's as weights into the systolic array, ③ Setup input matrix A, ④ Shift input matrix A into systolic array, pass through Accumulation and Activation, ⑥ Set the constant in Normalization to be the scalar $a$ and perform $a$*A, and ⑦ Write $a$*A back to the Unified Buffer. Repeat operations ① through ④ but this time stream in matrix B so that ⑤ B is ready in the Accumulation stage. Repeat again steps ② ③ by streaming in $a$*A from the Unified Buffer and perform an ADD in the ⑤ Accumulation stage, pass through Activation and Normalization, and finally ⑦ write the results of $a$*A + B into the Unified Buffer.

The comparison shows that while ProSE performed the MulAdd using one trip of the local dataflow, the TPUv2 is likely having to traverse two or three trips of the global dataflow, significantly reducing the efficiency of execution.

***ProSE Special Functions.*** ProSE supports two special functions GELU and Exp. These special functions are executed in simd mode as shown in Figure 12(d). While TPUv2 does not have a GELU activation unit and will be forced to use either a less-accurate version of a RELU activation or an approximation expansion of GELU (e.g. Taylor series expansion) that involves 10+ MulAdd operations, ProSE has these special function lookup tables implemented as part of the SIMD ALUs, greatly increase the performance and efficiency to perform these special functions. Both GELU and Exp are implemented using a two-level indexed lookup table. ProSE supports bfloat16 datatype with 1 sign bit, 8 exponent bits, and 7 mantissa bits similar to the TPU. The two-level lookup is done in one cycle, setting the critical path of the special function units. Because the protein language model accuracy is sensitive to the precision produced of GELU and softmax, special care is needed to preserve all 16 bits with lookup tables to match GPU precision.

For GELU, we designed the lookup table such that it only computes the output when the exponent is between -4 and 3 (i.e., [-4, 3]). As shown in Figure 13, when the input is with an exponent smaller than -3, it can be approximated as 0. When the input is with an exponent larger than 4, it can be approximated by a linear function. Only lightly shaded areas are computed for GELU using the lookup tables. For Exp, we followed the similar design choice and only computes the output when the exponent is in between -6 nd 5 (i.e., [-6, 5]) as shown in Figure 14.

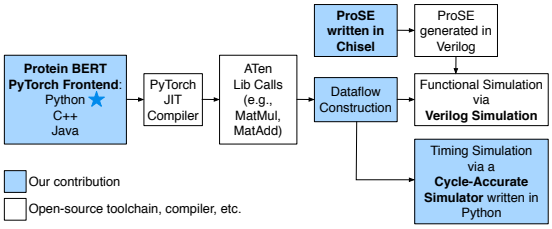


**Figure 15: Our simulation infrastructure is tightly integrated with the Protein BERT software model, providing flexible yet high fidelity evaluation. The star denotes that Python is our choice language for the PyTorch frontend.**

This design choice allows sufficient precision while keeping the lookup table at a reasonable size of 4 KB and 6 KB respectively. Each systolic array duplicates the lookup table $n$ times where $n$ is the number of SIMD ALU units. Each lookup table allows one lookup per cycle, trading area for performance.

***ProSE Efficiencies.*** Dataflow 1 and Dataflow 2 computational patterns (from Figure 6) can execute efficiently on existing systolic-array-based architectures as long as the model and intermediate data can utilize the local storage on the accelerator efficiently. However, as lengths increase, workloads that exhibit a computational pattern similar to Dataflow 3 suffer from the increasing portions of the matrix divisions, softmax operations, and interleaving MatMul and SIMD operations (see Figure 3), which are not handled well by existing architectures. In order to achieve high throughput at longer input lengths, ProSE aptly 1) batches CPU-essential operations like softmax efficiently via streaming, 2) accelerates the cost-intensive Exp with dedicated lookup tables, and 3) maximizes E-type utilization via interleaving operations on small systolic arrays. In addition to these long-length-specific design decisions, we gain significant length-agnostic efficiency from other choices such as our output-stationary design and left-rotating systolic arrays.

# 4 PROSE DESIGN AND EVALUATION

## 4.1 Methodology

***Implementation Methodology.*** Systolic arrays, GELU units, and Exp units are implemented in Chisel [44], compiled into Verilog using the Chisel toolchain. The Verilog is then synthesized using Synopsys [49], placed and routed using the FreePDK 15 nm technology node [33, 36] and scaled to 7 nm using the sub-10 nm technology scaling methodology [47] to obtain timing (frequency), power, and area of each ProSE hardware component. A summary of the component physical characteristics is provided in Table 2. The slowest MatMul-capable systolic arrays run at 1626 MHz and the slowest SIMD/GELU/Exp-capable systolic arrays run at 858 MHz setting the critical path. We choose to double-pump the systolic arrays when performing MatMul operations at 1.6 GHz and halve the frequency for SIMD or special functions to 800 MHz, simplifying design complexities. For the input buffer (`InBuf`), we assume they are implemented in SRAM and estimate their physical characteristics using the latest OpenRAM technology [16] at 45 nm PDK and scale the results to 7 nm.

**Table 2: This table presents the physical design characteristics of ProSE systolic arrays and special functions. The components are designed using the Predictive Technology Modeling (PTM) FreePDK 15 nm. For the input buffer (+InBuf), we use OpenRAM to synthesize, place, and route SRAMs at the latest technology available, 45 nm PDK. The obtained results are conservatively scaled to the same technology as the A100 at 7 nm and compared to the A100 power and area.**

| Heterogeneous Systolic Array Physical Characteristics | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Systolic Array Dimension | GELU LUT | Exp LUT | Frequency (MHz) | Power (mW) | +InBuf Power (mW) | %A100 Power | Area ($mm^2$) | +InBuf Area ($mm^2$) | %A100 Area |
| 16×16 | no | no | 1977.1 | 249.3 | 268.6 | 0.07% | 0.183 | 0.213 | 0.03% |
| | no | yes | 925.2 | 260.2 | 279.5 | 0.07% | 0.190 | 0.221 | 0.03% |
| | yes | no | 887.1 | 255.1 | 274.4 | 0.07% | 0.187 | 0.217 | 0.03% |
| 32×32 | no | no | 1707.1 | 802.6 | 841.2 | 0.21% | 0.706 | 0.766 | 0.09% |
| | no | yes | 886.8 | 830.0 | 868.5 | 0.22% | 0.725 | 0.786 | 0.10% |
| | yes | no | 870.3 | 808.4 | 847.0 | 0.21% | 0.719 | 0.779 | 0.09% |
| 64×64 | no | no | 1626.1 | 2552.1 | 2629.1 | 0.66% | 2.788 | 2.908 | 0.35% |
| | no | yes | 858.1 | 2578.2 | 2655.2 | 0.66% | 2.829 | 2.949 | 0.36% |
| | yes | no | 860.4 | 2514.8 | 2591.8 | 0.65% | 2.816 | 2.936 | 0.36% |
| | yes | yes | 858.1 | 2585.8 | 2662.9 | 0.67% | 2.863 | 2.983 | 0.36% |

***Evaluation Methodology.*** Figure 15 shows our simulation infrastructure. We developed a functional simulator and a cycle-accurate performance simulator including host-accelerator communications using conservative assumptions for achievable bandwidths. The Protein BERT model is implemented in PyTorch [41]. We instrumented our PyTorch model to produce raw sequences of its backend tensor and mathematical operation library calls (ATen calls) via the PyTorch JIT compiler. The produced ATen library calls are then constructed into dataflows and fed into the Verilog simulator for functional verification. The dataflows are fed into the cycle-accurate performance simulator to produce cycle counts. Combined with the physical characteristics in Table 2, we obtained the runtime of the ProSE system performing Protein BERT dataflows. In addition to operations executed on ProSE, we used a commodity Xeon CPU as the host CPU[3] to perform part of the softmax computations (i.e., Dataflow 3 in Figure 6) and incorporate the CPU-side runtime into our results.

Besides a detailed model of the ProSE microarchitecture, the cycle-accurate performance simulator includes a thread launching model, an orchestration/scheduling model, and a host-accelerator communication model. All of these overheads are included in our results. We measure A100 and TPU results and compare the accelerated portions (all operations except for "Other" in Figure 3).

For the A100 power consumption, we measure ProteinBERT via NVIDIA System Management Interface (nvidia-smi) to obtain power consumption at 395 W (close to the published TDP of 400 W). For the TPU power consumption, we use published TDP (e.g., v2 280 W per chip and 4 chips/device = 1120 W) as there are no power measurement tools publicly available. For ProSE power consumption, we synthesize ProSE's systolic arrays, with and without lookup tables, input buffers, and SIMD units to obtain results shown in Table 2 and Table 4. Additionally, we use Intel RAPL to measure CPU power under ProSE load (21.4% of the time consuming 50.21 W) and DRAM power (6.23 W) to obtain ProSE power executing the same ProteinBERT model. We conservatively do not include CPU power or NVLink/PCIe power for GPU/TPU platforms.

[3]Intel Xeon Gold 6140M (14 nm Skylake), dual socket with 36C/72T @ 2.3 GHz (Turbo 3.7 GHz), 24.75 MB L3 cache; 128 GiB DDR4 memory and 256 GiB SSD storage.

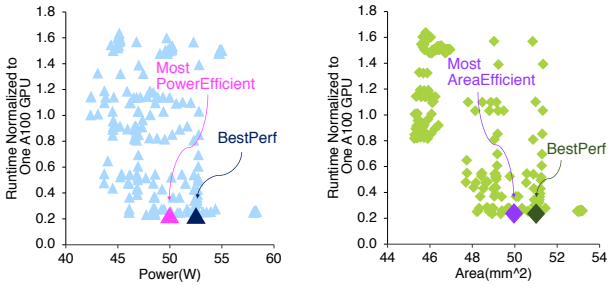


**Figure 16: We perform a detailed design space exploration (DSE) using 238 configurations and plot the normalized runtime vs. power on the left and normalized runtime vs. area on the right. We select the `BestPerf` and Pareto front `MostPowerEfficient` and `MostAreaEfficient` configurations.**

**Table 3: Heterogeneous systolic array types, sizes, and counts in our DSE.**

| Hardware Configurations for Design Space Exploration | | | |
|---|---|---|---|
| Systolic Array Type | Systolic Array Size | Maximum Count | Counts Explored |
| M-Type | 64×64 | 2 | 1 ... 3 |
| G-Type | 32×32 | 15 | 1 ... 15 |
| | 16×16 | 31 | 1 ... 31 |
| E-Type | 32×32 | 15 | 1 ... 15 |
| | 16×16 | 31 | 1 ... 31 |
| Homogeneous | 64×64 | 4 | 4 |

## 4.2 ProSE Systolic Array Mix Design Space Exploration

We perform a design space exploration (DSE) on ProSE hardware configurations. We explore mixes of types, sizes, and counts as shown in Table 3. We eliminate configurations that yield unreasonable performance or efficiency to reduce the number of ProSE configurations under consideration (e.g., M-Type systolic arrays have to be at least 64×64 for the performance to be competitive). We keep the total number of PE counts constant at 16384 PEs, resource-equivalent to a single TPU 128×128 systolic array. Since all systolic array types are needed for functionality, every configuration must have a count of one or more. Other factors considered include the

**Table 4: Six select ProSE designs with 16K and 20K PEs for further evaluation.**

| Select ProSE Instance Configurations for Further Evaluation | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Config | M size | M count | G size | G count | E size | E count | Power (mW) | Area ($mm^2$) |
| BestPerf | 64×64 | 2 | 16×16 | 10 | 16×16 | 22 | 12994 | 12.75 |
| MostEfficient | 64×64 | 2 | 32×32 | 3 | 16×16 | 20 | 12306 | 12.49 |
| Homogeneous | 64×64 | 2 | 64×64 | 1 | 64×64 | 1 | 10652 | 11.93 |
| BestPerf+ | 64×64 | 2 | 32×32 | 5 | 32×32 | 7 | 16918 | 48.50 |
| MostEfficient+ | 64×64 | 2 | 32×32 | 5 | 32×32 | 7 | 16918 | 48.50 |
| Homogeneous+ | 64×64 | 2 | 64×64 | 1 | 64×64 | 2 | 13315 | 14.92 |

use of an input buffer and how NVLink bandwidths are statically divided by lanes and connected to each systolic array type.

We explore how to provision bandwidth for each type of systolic arrays as the streaming nature of ProSE can become a bottleneck if not managed. We use NVLink 2.0 for our DSE and statically partition 6 45 GB/s lanes [12, 32] (6 * 45 = 270 GB/s, a conservative estimate of achievable NVLink 2.0 bandwidth at 90% of 300 GB/s), connected to the M-, G-, and E-Type systolic arrays. The number of lanes per systolic array type is swept as part of the design space exploration and a different option is chosen for each mix of systolic arrays.

We explored 238 hardware configurations including homogeneous configurations and the results are plotted in Figure 16. Performance (runtime normalized to one A100) vs. power and performance vs. area of the DSE are depicted on the left and right plots. Aside from the best performant configuration (`BestPerf`), choosing Pareto design points would give us the maximum power- and area-efficiency (`MostPowerEfficient` and `MostAreaEfficient`). The DSE results revealed that the `MostPowerEfficient` and the `MostAreaEfficient` happens to be the same configuration which we call `MostEfficient`. Table 4-top lists the select configurations from our DSE and a homogeneous configuration with a resource profile of 16K PEs for further evaluation.

## 4.3 ProSE Evaluation

To assess the impact of hardware resources on ProSE performance and efficiency, we perform DSEs sweeping number of PEs ranging from 8K to 24K at a fixed 90% achievable NVLink 2.0 bandwidth of 270 GB/s. Figure 17 plots resources vs. performance on the left and resources vs. power efficiency on the right for the `BestPerf` and the `MostEfficient` configurations from each resource profile. We observe that with hardware resources provisioned at 16K PEs (ProSE) or 20K PEs (ProSE+), the designs are the most balanced where each ProSE instance is comparably power-efficient and performant.

For the resource profile of 20K PEs, we explore the possibility of a larger ProSE instance being bandwidth-bound and perform another DSE with a larger communication link bandwidth of 540 GB/s, the 90% achievable NVLink 3.0 bandwidth employed by the A100 GPU. Table 4-bottom lists the select configurations from this DSE and a homogeneous configuration with a resource profile of 20K PEs for further evaluation.

We measure GPU system performance using one instance of the A100 GPU and maximize performance using batch size appropriate for an input length of 512 tokens. For the TPU system

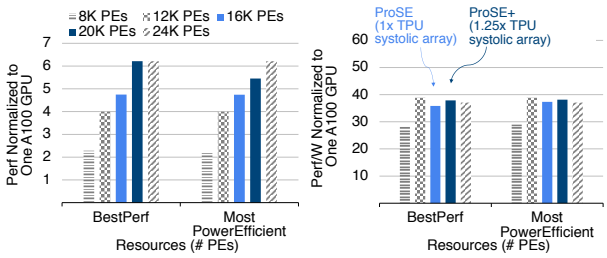


**Figure 17: Our DSE explores PEs counts ranging from 8K to 20K. We find both 16K and 20K PEs counts are good balance points for performance and efficiency when utilizing emerging links [12, 39].**

measurements, we use one instance of the TPUv3 from Google Cloud. Similarly, we maximize inference performance using appropriate batch size. Due to the competitive nature of deep learning software, the state-of-the-art Huggingface library for Transformers used here is heavily optimized to meet industry standards and minimizes time spent with low operational intensity (e.g. transposes). For the ProSE performance and power efficiency evaluation, we use an input sequence length of 512 tokens and a batch size of 128.

Figure 18 plots the ProSE speedup with respect to one NVIDIA A100 GPU on the left and one TPUv3 instance (4 chips/8 cores with hardware resource totaling 262K PEs) on the right with varying host-accelerator communication bandwidths. The `BestPerf` and the `MostEfficient` designs achieve a speedup of 3.9–4.7× over the A100 and a speedup of 3.1–3.8× over TPUv3 with NVLink 2.0. The `BestPerf+` and the `MostEfficient+` designs demand faster links as they have more compute resources and do not become compute-bound until the bandwidth reaches 360 GB/s, as shown in Figure 20. The homogeneous designs, however, cannot deliver the desired level of performance even at infinite bandwidth as it suffers from large overhead of starting and draining large systolic arrays when performing smaller matrix operations. In addition, homogeneous design do not have enough SIMD ALUs and special function units to compete with heterogeneous designs when performing inference on a BERT-style model with long input lengths.

Power efficiencies of the ProSE designs follow similar trends shown in Figure 19, gaining one to two orders of efficiency over GPU and TPU systems. Even with our design choice of streaming systolic arrays, because the intermediate data mostly fit within the large L3 cache on the host and can be stored and retrieved without frequent trips to host-side memory, DRAM is mostly accessed during cold misses to the caches, providing power efficiency over GPU and TPU platforms.

# 5 RELATED WORK

***Modified Systolic Arrays.*** It has been shown many small systolic arrays may increase the utilization (thus efficiency) at the cost of performance [23]. Maestro [24, 34] showed as much but only for short inputs only on BERT-style models. However, even when scaled to 7 nm, Maestro does not compete with modern accelerators like A100 or TPUs. Other streaming systolic arrays make use of emerging high-bandwidth host- accelerator external interfaces (e.g.

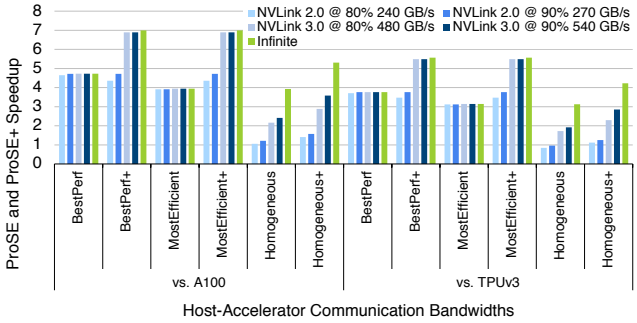


**Figure 18: Speedup of ProSE over NVIDIA A100 GPU and TPUv3 when evaluated with different link bandwidths. `BestPerf` and `BestPerf+` designs can be seen plateauing at different maximal speedups as they approach fully compute-bound performance.**

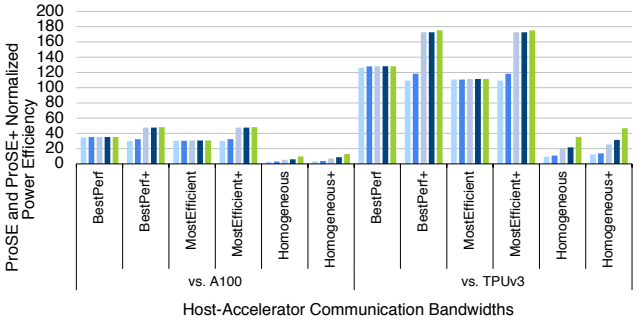


**Figure 19: Power efficiency of ProSE over NVIDIA A100 GPU and TPUv3 when evaluated with different link bandwidths. ProSE is a lot more power-efficient than TPUv3 (two orders of magnitude more efficient) due to the elimination of the large and power-hungry Unified Buffer.**

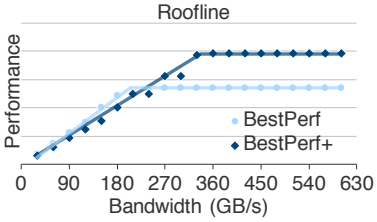


**Figure 20: Empirical roofline for BestPerf and BestPerf+ designs. As the different heterogeneous components of ProSE become compute-bound, the BestPerf and BestPerf+ designs creep towards saturation.**

NVLink [12, 32]), e.g. DUET [29], which only functioned for sparse CNN and DNNs, not BERT-style NLP models (Transformers).

More generally, sparse GEMM accelerators require significant sparsity and similarly accelerate only the matrix multiplication portions of a Transformer [42]. SIGMA's specific gains on BERT-style models are also limited to short input lengths as it does not target long input lengths and tiling efficiency gains are amortized on longer inputs. Multi-tenant systolic-array-based accelerators [15] do not accelerate BERT-style models, and most of their gains are CNN-specific.

***Accelerators for NLP Models.*** A handful of competitive works accelerate BERT-style NLP models, but these optimize only for isolated operations (e.g. softmax) [18, 19, 31, 52]. Our approach is distinct as it can accelerate the bulk of the model across multiple layers, something past work cannot handle. Namely, our three accelerated dataflows capture 80 to 95% of operations, as shown in Figure 3.

***Non-NLP Accelerators.*** A plethora of specialized accelerators deliver performance and efficiency gains when performing inferences on CNNs and DNNs [2, 6, 7, 14, 26, 37, 40]. These models exhibit fundamentally different computational patterns to BERT-style models (Transformers), which do not benefit from the significant resources dedicated to accelerating convolutional filters (not present in most Transformers). As such, these architectures cannot efficiently predict protein binding affinity for drug discovery.

## 6 CONCLUSION

As the realizable benefits of AI increase, it is imperative to increase the efficiency of transfer learning not just for natural language processing but also for drug design and discovery. To increase the performance, power, and area efficiencies of producing more accurate drug candidates, we present ProSE, a systolic engine for protein discovery. By swapping out the transformer model weights being accelerated (e.g., adding decoder layers for language translation) or adding different fine-tuning downstream models, ProSE is easily applicable to a multitude of other protein and NLP-related tasks.

With ProSE, we present a power- and area-efficient design of a collection of output-stationary streaming heterogeneous systolic arrays of varying sizes and counts. These systolic arrays are equipped with capabilities to execute SIMD ALU operations, SIMD GELU functions, and SIMD Exp functions. ProSE performs Protein BERT inference at up to 6.9× speedup over one NVIDIA A100 GPU and up to two orders of magnitude power efficiency gain over the latest GPU and TPU platforms. Our work demonstrates that identifying optimization opportunities from common computational patterns across model layers allows the efficiency gains of specialization without sacrificing generality.

## 7 ACKNOWLEDGEMENT

We thank the reviewers for their insightful comments. This work was supported in part by a Facebook SysML Research Award, in part by an National Science Foundation CAREER award CCF-2045974, and in part by the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation program co-sponsored by DARPA.

## REFERENCES

[1] Ethan C. Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M. Church. 2019. Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods* 16, 12 (01 Dec 2019), 1315–1322. https://doi.org/10.1038/s41592-019-0598-1

[2] Bahar Asgari, Ramyad Hadidi, Hyesoon Kim, and Sudhakar Yalamanchili. 2019. LODESTAR: Creating Locally-Dense CNNs for Efficient Inference on Systolic Arrays. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. https://doi.org/10.1145/3316781.3322472

[3] Surojit Biswas, Grigory Khimulya, Ethan C. Alley, Kevin M. Esvelt, and George M. Church. 2020. Low-N protein engineering with data-efficient deep learning. *bioRxiv* (2020). https://doi.org/10.1101/2020.01.23.917682 arXiv:https://www.biorxiv.org/content/early/2020/08/31/2020.01.23.917682.full.pdf

[4] Jenny Bostrom, Lauric Haber, Patrick Koenig, Robert F. Kelley, and Germaine Fuh. 2011. High Affinity Antigen Recognition of the Dual Specific Variants of Herceptin Is Entropy-Driven in Spite of Structural Plasticity. *PLoS ONE* 6, 4 (April 2011), e17887. https://doi.org/10.1371/journal.pone.0017887

[5] N. Burgess, J. Milanovic, N. Stephens, K. Monachopoulos, and D. Mansell. 2019. Bfloat16 Processing for Neural Networks. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. 88–91. https://doi.org/10.1109/ARITH.2019.00022

[6] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[7] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308. https://doi.org/10.1109/JETCAS.2019.2910232

[8] Ratul Chowdhury, Nazim Bouatta, Surojit Biswas, Charlotte Rochereau, George M. Church, Peter K. Sorger, and Mohammed AlQuraishi. 2021. Single-sequence protein structure prediction using language models from deep learning. *bioRxiv* (2021). https://doi.org/10.1101/2021.08.02.454840

[9] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 2978–2988. https://doi.org/10.18653/v1/P19-1285

[10] DeepMind. 2022. AlphaFold: Using AI for scientific discovery. https://deepmind.com/blog/article/AlphaFold-Using-AI-for-scientific-discovery.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[12] Denis Foley and John Danskin. 2017. Ultra-Performance Pascal GPU and NVLink Interconnect. *IEEE Micro* 37, 2 (2017), 7–17. https://doi.org/10.1109/MM.2017.37

[13] David H. Freedman. 2020. Hunting for New Drugs with AI. *Scientific American Digital Issue* (February 2020). https://doi.org/10.1038/d41586-019-03846-0

[14] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 751–764. https://doi.org/10.1145/3037697.3037702

[15] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, Cliff Young, and Hadi Esmaeilzadeh. 2020. Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 681–697. https://doi.org/10.1109/MICRO50266.2020.00062

[16] Matthew R. Guthaus, James E. Stine, Samira Ataei, Brian Chen, Bin Wu, and Mehedi Sarwar. 2016. OpenRAM: An open-source memory compiler. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–6. https://doi.org/10.1145/2966986.2980098

[17] Mark A. Hallen, Jeffrey W. Martin, Adegoke Ojewole, Jonathan D. Jou, Anna U. Lowegard, Marcel S. Frankel, Pablo Gainza, Hunter M. Nisonoff, Aditya Mukund, Siyu Wang, Graham T. Holt, David Zhou, Elizabeth Dowd, and Bruce R. Donald. 2018. OSPREY 3.0: Open-source protein redesign for you, with powerful new features. *Computational Chemistry* 39, 30 (October 2018), 2494–2507. https://doi.org/10.1002/jcc.25522

[18] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W Lee, and Deog-Kyoon Jeong. 2020. A3: Accelerating Attention Mechanisms in Neural Networks with Approximation. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.

[19] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, Soosung Kim, Hyunji Choi, Sung Jun Jung, and Jae W. Lee. 2021. ELSA: Hardware-Software Co-design for Efficient, Lightweight Self-Attention Mechanism in Neural Networks. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 692–705. https://doi.org/10.1109/ISCA52012.2021.00060

[20] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. 2020. A Domain-Specific Supercomputer for Training Deep Neural Networks. *Commun. ACM* 63, 7 (jun 2020), 67–78. https://doi.org/10.1145/3360307

[21] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, aron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law ans Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiana, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.

[22] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* (15 Jul 2021). https://doi.org/10.1038/s41586-021-03819-2

[23] H.T. Kung, Bradley McDanel, and Sai Qian Zhang. 2019. Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 821–834. https://doi.org/10.1145/3297858.3304028

[24] H. T. Kung, Bradley McDanel, Sai Qian Zhang, Xin Dong, and Chih Chiang Chen. 2019. Maestro: A Memory-on-Logic Architecture for Coordinated Parallel Use of Many Systolic Arrays. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Vol. 2160-052X. 42–50. https://doi.org/10.1109/ASAP.2019.00-31

[25] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 754–768. https://doi.org/10.1145/3352460.3358252

[26] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. *SIGPLAN Not.* 53, 2 (mar 2018), 461–475. https://doi.org/10.1145/3296957.3173176

[27] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1eA7AEtvS

[28] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joseph E. Gonzalez. 2020. Train large, then compress: rethinnking model size for efficient training and inference of transformers. *arXiv* (February 2020). https://arxiv.org/abs/2002.11794.

[29] Liu Liu, Zheng Qu, Lei Deng, Fengbin Tu, Shuangchen Li, Xing Hu, Zhenyu Gu, Yufei Ding, and Yuan Xie. 2020. DUET: Boosting Deep Neural Network Efficiency on Dual-Module Architecture. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 738–750. https://doi.org/10.1109/MICRO50266.2020.00066

[30] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis Luke Zettlemoyer, and Vaseline Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *ArXiv e-prints* (July 2019). arXiv:1907.11692

[31] Siyuan Lu, Meiqi Wang, Shuang Liang, Jun Lin, and Zhongfeng Wang. 2020. Hardware Accelerator for Multi-Head Attention and Position-Wise Feed-Forward in the Transformer. *ArXiv e-prints* (September 2020). arXiv:2009.08605

[32] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2020. Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects. In *Proceedings of the International Conference on Management of Data (SIGMOD)* (Portland, OR, USA). Association for Computing Machinery, New York, NY, USA, 1633–1649. https://doi.org/10.1145/3318464.3389705

[33] Mayler Martins, Jody Maick Matos, Renato P. Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. 2015. Open Cell Library in 15nm FreePDK Technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design* (Monterey, California, USA) *(ISPD '15)*. Association for Computing Machinery, New York, NY, USA, 171–178. https://doi.org/10.1145/2717764.2717783

[34] Bradley McDanel, Sai Qian Zhang, H. T. Kung, and Xin Dong. 2019. Full-Stack Optimization for Accelerating CNNs Using Powers-of-Two Weights with FPGA Validation. In *Proceedings of the ACM International Conference on Supercomputing* (Phoenix, Arizona) *(ICS '19)*. Association for Computing Machinery, New York, NY, USA, 449–460. https://doi.org/10.1145/3330345.3330385

[35] Joshua Meier, Roshan Rao, Robert Verkuil, Jason Liu, Tom Sercu, and Alexander Rives. 2021. Language models enable zero-shot prediction of the effects of mutations on protein function. *bioRxiv* (2021). https://doi.org/10.1101/2021.07.09.450648

[36] NC State University. 2022. FreePDK15. https://www.eda.ncsu.edu/freepdk15.

[37] NVIDIA. 2018. The NVIDIA deep learning accelerator. In *Hot Chips: A Symposium on High Performance Chips*.

[38] NVIDIA. 2022. NVIDIA Grace CPU. https://www.nvidia.com/en-us/data-center/grace-cpu/.

[39] NVIDIA. 2022. NVLink and NVSwitch. https://www.nvidia.com/en-us/data-center/nvlink/.

[40] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 27–40. https://doi.org/10.1145/3079856.3080254

[41] PyTorch. 2022. PyTorch: An open source machine learning framework that accelerates the path from research prototyping to production deployment. http://pytorch.org.

[42] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. 2020. SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 58–70. https://doi.org/10.1109/HPCA47549.2020.00015

[43] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Xi Chen, John Canny, Pieter Abbeel, and Yun S. Song. 2019. *Evaluating Protein Transfer Learning with TAPE*. Curran Associates Inc., Red Hook, NY, USA. https://doi.org/10.5555/3454287.3455156

[44] Berkeley Architecture Research. 2022. Chisel Hardware Construction Language. https://chisel.eecs.berkeley.edu/

[45] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and Rob Fergus. 2021. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences* 118, 15 (2021). https://doi.org/10.1073/pnas.2016239118

[46] Sarah Sirin, James R. Apgar, Eric M. Bennett, and Amy E. Keating. 2016. AB-Bind: Antibody binding mutational database for computational affinity predictions. *Protein science : a publication of the Protein Society* 25, 2 (Feb 2016), 393–409. https://doi.org/10.1002/pro.2829 26473627[pmid].

[47] Aaron Stillmaker and Bevan Baas. 2017. Scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm. *Integration* 58 (June 2017), 74–81. https://doi.org/10.1016/j.vlsi.2017.02.002

[48] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 2158–2170. https://doi.org/10.18653/v1/2020.acl-main.195

[49] Synopsys. 2022. Synopsys Silicon Design and Verification. https://www.synopsys.com.

[50] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long Range Arena : A Benchmark for Efficient Transformers. In *International Conference on Learning Representations*. https://openreview.net/forum?id=qVyeW-grC2k

[51] Jakob Uszkoreit. 2017. Transformer: A Novel Neural Network Architecture for Language Understanding. https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html.

[52] H. Wang, Z. Zhekai, and S. Han. 2021. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Purning. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*.

[53] Kevin K. Yang, Zachary Wu, and Frances H. Arnold. 2019. Machine-learning-guided directed evolution for protein engineering. *Nature Methods* 16 (July 2019), 687–694. https://doi.org/10.1038/s41592-019-0496-6