A Full-Stack Search Technique for Domain Optimized Deep Learning Accelerators

Dan Zhang
dazh@google.com
Google Brain
Mountain View, CA, USA

Safeen Huda
safeen@google.com
Google
Sunnyvale, CA, USA

Ebrahim Songhori
esonghori@google.com
Google Brain
Mountain View, CA, USA

Kartik Prabhu
kprabhu7@stanford.edu
Stanford University
Stanford, CA, USA

Quoc Le
qvl@google.com
Google Brain
Mountain View, CA, USA

Anna Goldie
agoldie@google.com
Google Brain
Mountain View, CA, USA

Azalia Mirhoseini
azalia@google.com
Google Brain
Mountain View, CA, USA

ABSTRACT

The rapidly-changing deep learning landscape presents a unique opportunity for building inference accelerators optimized for specific datacenter-scale workloads. We propose *Full-stack Accelerator Search Technique* (FAST), a hardware accelerator search framework that defines a broad optimization environment covering key design decisions within the hardware-software stack, including hardware datapath, software scheduling, and compiler passes such as operation fusion and tensor padding. In this paper, we analyze bottlenecks in state-of-the-art vision and natural language processing (NLP) models, including EfficientNet [91] and BERT [19], and use FAST to design accelerators capable of addressing these bottlenecks. FAST-generated accelerators optimized for single workloads improve Perf/TDP by 3.7× on average across all benchmarks compared to TPU-v3. A FAST-generated accelerator optimized for serving a suite of workloads improves Perf/TDP by 2.4× on average compared to TPU-v3. Our return on investment analysis shows that FAST-generated accelerators can potentially be practical for moderate-sized datacenter deployments.

CCS CONCEPTS

• **Hardware → Electronic design automation**; • **Computer systems organization → Parallel architectures**.

KEYWORDS

machine learning, tensor processing unit, hardware-software codesign, design space exploration, operation fusion

ACM Reference Format:
Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. 2022. A Full-Stack Search Technique for Domain Optimized Deep Learning Accelerators. 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, 16 pages. https://doi.org/10.1145/3503222.3507767

ASPLOS '22, February 28 – March 4, 2022, Lausanne, Switzerland

ACM ISBN 978-1-4503-9205-1/22/02.
https://doi.org/10.1145/3503222.3507767

1 INTRODUCTION

The deep learning landscape is constantly evolving. Neural networks nowadays may serve millions or even billions of daily users. Examples include language and image processing models that are used in search engines [66] and social networks [31]. The increasing application of deep learning across different industries suggests that this large-scale adoption trend will only continue to grow. Therefore, we see a potential opportunity for building inference accelerators optimized for specific datacenter-scale workloads.

Enabling automatic hardware optimization requires a search space definition encompassing beyond simple hardware accelerator families such as those used in prior work [18, 41, 59, 79, 86, 93, 101, 103, 112]. Hardware accelerator architectures can be described in terms of their datapath and schedule, where the datapath comprises the hardware components (compute units, scratchpad memories, connectivity, etc.) on which neural network operations are run, and the schedule comprises the compiler scheduling and hardware control logic that maps these operations onto the datapath. Common datapath designs use grids of processing elements (PEs), including scalar [14, 59, 79, 86, 101], vector [84, 93, 103, 104, 112], or matrix [39] compute units. In addition, many compiler optimization passes have major performance impact on production accelerators [113] and should be included in the search space.

We therefore propose FAST, a Full-stack Accelerator Search Technique that takes one or more neural networks as input, jointly optimizes key decisions within the hardware-software stack including compiler decisions, and outputs an optimized inference accelerator for the input (see Figure 1). FAST can optimize for desired objectives such as performance measured in inference queries per second (QPS) or performance per Total Cost of Ownership (TCO), a key optimization metric for designing datacenter accelerators [39]. Unfortunately, TCO is highly sensitive proprietary information; we instead evaluate performance per Thermal Design Power (TDP), known to highly correlate with Perf/TCO [37]. Our ROI analysis suggests FAST-generated accelerators can be practical for even moderate-sized datacenter deployments, potentially enabling accelerators optimized for single workloads.

Deep learning models are changing rapidly, necessitating similar rapid changes to accelerator architecture designs. State-of-the-art production models, such as EfficientNet [91] (an image classification model that uses depth-wise separable convolutions) and BERT [19] (a language model that uses the attention mechanism), introduce a

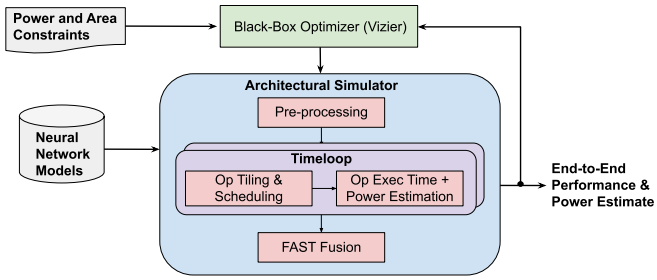


Figure 1: Full Stack Accelerator Search Technique overview.

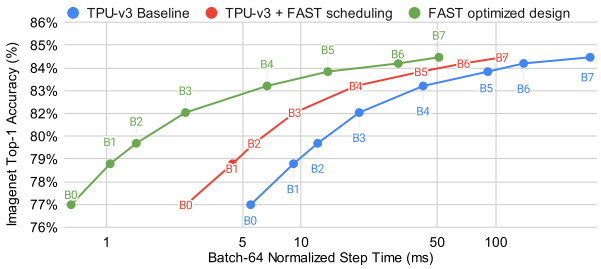


Figure 2: FAST-Large (see Table 5) inference step time vs. ImageNet top-1 accuracy running the EfficientNet model family. Faster hardware accelerators can run larger, more accurate models with the same latency budget, or significantly reduce inference latency and throughput given a fixed accuracy budget. FAST does not affect model accuracy; quantization can bring further gains but is outside the scope of this paper.

new set of computational and memory bottlenecks that previously did not exist. Due to the production impact of these models, we focus on their performance characterization in Section 4. Our analysis demonstrates that EfficientNet has poor operation intensity and efficiently mapping its depth-wise separable convolutions to existing hardware is challenging. We also describe BERT attention layer inefficiencies, and show that the attention layer becomes a substantial portion of total execution time as the model's sequence length increases. By designing our search space such that our identified bottlenecks can be properly addressed, we are able to achieve significantly higher Perf/TDP improvements relative to prior work not only on EfficientNet and BERT, but also on older models such as ResNet-50 [32]. Figure 2 shows the performance of FAST on EfficientNets with scheduling-only updates applied to a fixed hardware configuration, with even larger speedups possible when running full SW/HW co-optimization for each model.

Comprehensively addressing our identified bottlenecks requires a large search space $O(10^{2300})$ with parameters defining the hardware datapath, software scheduling, and compiler passes such as operation fusion and tensor padding (see Section 5.3). We extend FAST's datapath template to be an *approximate superset* of existing accelerator families capable of expressing scalar, vector, and matrix processing elements with a versatile memory hierarchy search space. Our datapath template also includes a TPU-like *vector processing unit* (VPU) [38] within the PEs which enables efficient execution of a wide range of vector ops, such as exponential and reduction ops, required for workloads such as BERT. We also discovered that a key limitation of prior work is the inability to generate high-performance accelerators for workloads with low operational intensity, such as EfficientNet. To address this, we devised a flexible and general *integer linear programming* (ILP)-based op fusion technique called *FAST fusion* which can determine the best set of activation and weight tensors to move from DRAM into on-chip scratchpads to maximize overall performance. FAST fusion enables our search tool to unlock significant speedups that are otherwise impossible due to memory bandwidth bottlenecks (see Section 6.2.7) by allowing our search tool to increase scratchpad space to improve fusion efficiency and reduce memory traffic, resulting in high-performance and well-balanced designs (see Table 5). FAST also considers various optimizations including tensor padding and a new softmax computation approach to the search space, which reduces the memory bottleneck at the expense of additional compute (see Section 5.6).

A flexible simulator is key to evaluating full-stack accelerator performance for a given neural network. We describe our fast and accurate simulation platform capable of modeling a wide range of hardware datapaths and schedules on unmodified XLA HLO graphs by leveraging Timeloop [68] and addressing its key limitations as discussed in Section 6.1. Our simulator also contains an analytical power and area model correlated to production designs on an industry sub-10nm process.

In summary, our contributions are as follows:

- We propose FAST, an automated framework for jointly optimizing hardware datapath, software schedule, and compiler passes, with a combined search space of up to $O(10^{2300})$ to design optimized inference accelerators for one or a set of input neural networks.
- We perform detailed performance characterization of state-of-the-art ML models, identify their bottlenecks, and propose several optimizations to address these bottlenecks.
- We propose a FAST fusion, a novel and flexible ILP-based op fusion technique, enabling FAST to fully address memory bottlenecks in low operational intensity workloads.
- We analyze the relationship between ROI, number of deployed hardware, and the Perf/TDP of accelerators to provide guidelines on exploring trade-offs between specialization and performance in future accelerators. Our ROI analysis demonstrates that FAST-generated inference accelerators can potentially be practical for even moderate-sized datacenter deployments (Section 6.2).
- We evaluate FAST on a comprehensive set of models, including the EfficientNet family [91], BERT [19], ResNet50v2 [32] and production OCR workloads [74].
- FAST's custom designs demonstrate an average of 3.7× Perf/TDP improvement across all the benchmarks when compared against TPU-v3, including a 6.4× and 2.7× improvement for EfficientNet and BERT respectively.
- FAST's general-purpose designs optimized for serving a set of important vision and NLP benchmarks show an average of 2.4× improvement in Perf/TDP vs. TPU-v3.

2 RELATED WORK

Accelerator design space exploration: Hardware ML accelerators can be described in terms of their hardware datapath and software schedule. Datapath designs often use grids of uniform processing elements comprised of scalar [14, 18, 59, 79, 86, 101], vector [84, 93, 103, 104, 112], or matrix [39] compute units. Previous work

focuses on optimizing families of accelerator designs with scalar or vector PEs and fixed memory hierarchies by mutating datapath hyperparameters, such as the number of PEs and buffer capacities [18, 41, 59, 79, 86, 93, 101, 103, 112], as well as the mapping of convolutions onto the datapath [18, 33, 41, 42, 52, 79, 86, 93, 101].

As described in Section 5.4, our datapath template is designed to be an approximate superset of popular designs capable of expressing scalar, vector, and matrix processing elements with varying memory hierarchies beyond variations on accelerator families. Our PEs containing both vector and systolic units share similarity to heterogeneous PE designs such as Plasticine [71]. Our search space also includes scheduling and other compiler optimizations, such as op fusion and tensor padding, enabling us to cover a much broader set of architectures. We also optimize for state-of-the-art models including EfficientNet and BERT, and demonstrate that our large co-optimization space allows for significant improvements over existing datacenter accelerators.

A flexible scheduler is key to evaluating accelerator performance for a given neural network. Timeloop [68] and MAESTRO [47] use random search to optimize accelerator schedules given a datapath and layer definition. However, they only evaluate single layers and only consider convolution operation performance, thereby limiting utility for end-to-end performance evaluation and optimization (e.g., operator fusion, parameter prefetching). Compared to Timeloop, the MAESTRO datapath design space is more restrictive, assuming only NVDLA accelerator variants with private L1 and global L2 scratchpads, and can only fuse ReLU and pooling operations. Interstellar [101] uses Halide [77], a domain-specific programming language, to generate and analyze inference accelerators. Although Interstellar can perform many blocking and spatial optimizations, its datapath search space is limited to grids of scalar PEs and reduction trees with global buffers. dMazeRunner [18] optimizes only convolution and FC layers with a pruned search of the schedule space. ZigZag [64] has a flexible datapath design space, and can perform heuristic-based schedule search targeting a scalar PE architecture, but does not perform fusion. We use Timeloop while addressing its limitations in our simulator, as described in Section 6.1. Our comprehensive search space led to significantly larger speedups compared to prior work; for example, MAGNet's reported best result [93] only improved Perf/W by 1.75x (43% energy reduction) compared to our best reported result of 6x Perf/TDP. To the best of our knowledge, FAST is also the first to optimize designs across multiple workloads.

Recent work such as ASIC Clouds [62, 99] has used design space exploration to optimize directly for datacenter total cost of ownership in the context of bitcoin mining, video transcoding, and machine learning accelerators. FAST extends this by considering return-on-investment (ROI) and using ROI to demonstrate production feasibility of FAST-generated designs.

Accelerator search on Reconfigurable Hardware: Several recent efforts have targeted the acceleration of neural networks on reconfigurable hardware including FPGAs and spatial arrays, which unlike ASICs enable flexible hardware reconfiguration. These prior works primarily focused on automation tools and design space exploration for one particular neural network [7, 30, 61, 71, 75, 85, 88, 96, 105, 106, 108–110]. However, the flexibility of FPGAs comes at the cost of reduced performance and higher energy consumption [46]. Unlike prior work, our framework enables the exploration of a broad range of datapaths, schedule, and fusion. We believe it would be simple to adapt our work to target reconfigurable hardware.

Co-optimization of neural networks and hardware: More recently, co-optimizing neural networks and accelerators has gained significant attention [4, 30, 36, 54–56, 100, 112]. The design space contains both the neural network architecture and hardware components, while jointly optimizing for both accuracy and performance. While our framework does not currently allow modifications to the model architecture, it would be straightforward to extend. However, even without model changes, FAST already delivers significantly higher performance than previous work through the larger search space covering datapath, schedule, and fusion.

Operation fusion: We also developed FAST fusion, an efficient ILP-based multi-layer fusion technique for inference which significantly improves memory bandwidth usage efficiency and thus inference execution time. Most production compilers such as cuDNN [2] can only fuse simple pre-defined templates such as Conv2D+Bias+Add. Although XLA [27] can create large fusions, each XLA-generated HLO fusion region contains at most one matrix operation (Conv2D, einsum, matmul, etc). There is also a growing body of work on more elaborate fusion [7, 11, 43, 60, 80, 111, 113]. [5] presents an RL-based approach for op fusion for training. [95] designs a framework for efficiently utilizing FPGA on-chip memory. FAST fusion is a secondary pass that fuses existing XLA-generated HLO fusion regions by assigning intermediate tensors from DRAM to on-chip SRAM. Compared to prior approaches, FAST fusion considers weight tensor pinning as part of the fusion problem, and uses ILP to directly minimize total execution time as modeled through simulation rather than indirect metrics such as total memory accesses.

3 BACKGROUND

3.1 Mapping Convolutions onto Accelerators

Since convolutions dominate the overall runtime in convolutional neural networks (CNNs), considerable effort has been expended on software [49] and hardware [14, 20] acceleration of these operations. A standard Conv2D can be represented as a 7-dimensional nested loop over batch size (B), output tensor height and width (OH, OW), number of input and output features (IF, OF), and kernel height and width (KH, KW). Since these loop iterations are commutative, compilers can freely modify loop traversal order, allowing for arbitrary transformations in tensor layout format, loop blocking, and spatial vectorization [47, 68]. Recent work has exploited these properties to build efficient high-performance accelerators [41, 86, 101].

Systolic arrays combine parallel operations with local communication, making them well-suited for matrix computations [45]. To multiply two matrices, one matrix is latched into internal registers, while the other is streamed through the array. Double-buffering is typically employed to mask the latency of latching a new set of parameters into the systolic array [39] Accelerators such as Google's TPU family [38] exploit the dense compute enabled by systolic arrays to accelerate training and inference. Under a *weight stationary* mapping [17], the systolic array will not be fully utilized unless IF, OF, and B are multiples of the dimensions of the systolic array. Alternative mappings, such as *output stationary* and *row stationary* [14], may achieve higher utilization by selecting alternative dimensions to be spatially unrolled, but are still limited by dimensional constraints.

Therefore, although larger systolic arrays improve area-density and power-efficiency per FLOP, they tend to have lower utilization. Each workload has different problem shapes, thus having different optimal systolic array dimensions which can be found through FAST.

3.2 EfficientNet Overview

Convolutional neural networks (CNNs) are often over-parameterized [29, 35]. A popular method for reducing model size and compute cost is replacing Conv2D with a *depthwise-separable convolution*: a depthwise convolution combined with a 1x1 point-wise convolution [15, 82, 90]. For example, a 3x3 depthwise-separable convolution uses 8-9x less compute than a standard Conv2D with only a slight reduction in accuracy [34]. EfficientNet [91], a CNN based on *inverted residual* (MBConv) [82] blocks, demonstrated that depthwise-separable convolutions were viable outside of compute and storage-constrained settings. However, depthwise-separable convolutions do not map well onto TPUs due to poor systolic array utilization and operational intensity. Depthwise convolutions allow significant parameter and compute reduction by reducing kernel filter depth (*IF*) to 1, but number of FLOPS is not an accurate proxy for performance on state-of-the-art accelerators such as Google TPUs or NVIDIA GPUs [13]. Common mappings unfortunately depend on large *IF* for good utilization. For example, assuming a depthwise convolution with a 3x3 kernel, maximum utilization for a 128x128 systolic array is only $KH * KW = 9$ out of 128. To address this, EfficientNet-X replaces some depthwise-separable convolutions with Conv2Ds to improve accuracy and latency [53]. However, the poor performance of depthwise convolutions remains a challenge. As shown in Table 5, FAST optimized for EfficientNet automatically generates hardware with smaller systolic arrays, improved scheduling, and reduced memory bottlenecks, enabling EfficientNet inference at high efficiency.

3.3 BERT Overview

Transformer-based models outperform traditional recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) on natural language processing tasks by replacing sequential computation with the self-attention mechanism [92]. BERT [19] is a Transformer-based model that achieves state-of-the-art results on both word-level and sentence-level tasks, and is the inspiration for a number of NLP models, including XLNet[102], GPT-2 [76], GPT-3 [12], ALBERT [48], and RoBERTa [57].

BERT is composed of multiple transformer encoder layers, where each layer consists of a self-attention layer, softmax operation, feed-forward layer, residual connection, and layer normalization. An important hyperparameter is the sequence length, which controls the size of the input token sequence. Increasing sequence length generally improves task accuracy at the cost of computation, with some operations scaling more efficiently than others, as discussed in Section 4.3. Although most operations are matrix-matrix multiplications, vector operations such as softmax and layernorm cannot be ignored.

4 WORKLOAD PERFORMANCE ANALYSIS

In the following section, we analyze various contributing factors to EfficientNet and BERT performance on TPU-v3. We first characterize EfficientNet and BERT in terms of operational intensity and discuss the impact of op fusion. We then analyze the implications of TPU-v3 architecture and compute scheduling strategy on EfficientNet. Finally, we examine BERT performance as a function of sequence length. These characterizations motivated us to build a comprehensive hardware and software search space for FAST able to deliver significant performance improvements.

Table 1: EfficientNet on-chip storage requirements (bfloat16). Working set sizes are shown for the op with the largest memory footprint at batch size 1. The storage requirements of larger EfficientNets exceed on-chip memory capacity, requiring more advanced op fusion techniques.

Model	Max Working Set	Weights
EfficientNet-B0	2.87 MiB	12.7 MiB
EfficientNet-B1	3.3 MiB	22.1 MiB
EfficientNet-B2	3.9 MiB	26.1 MiB
EfficientNet-B3	5.1 MiB	36.8 MiB
EfficientNet-B4	12.4 MiB	61.4 MiB
EfficientNet-B5	17.8 MiB	101 MiB
EfficientNet-B6	31.9 MiB	146 MiB
EfficientNet-B7	41.2 MiB	231 MiB

4.1 Operational Intensity and Op Fusion

ML model graphs are executed on accelerators as a series of kernels, or *operations*, where each op reads its inputs from device memory (DRAM), transfers these inputs to on-chip memory, performs the computation, and writes the output back to DRAM. This results in unnecessary DRAM reads and writes for intermediate values which are usually performed in parallel with computation, but may cause slowdowns with insufficient bandwidth. To determine if a model is compute or memory bandwidth-bound, one can calculate a model's *operational intensity*, defined as the ratio of compute operations (in FLOPS) to DRAM accesses (in bytes). For example, a TPU-v3 chip supports 123 TFLOPS/s of bfloat16 compute and 900GB/s memory bandwidth [38]. Therefore, a model that can operate at full compute utilization must have an operational intensity of at least 137 FLOPS/B to avoid becoming memory-bound. Note that it is cheaper to scale compute performance than memory bandwidth due to the *memory wall* [98]. The latest NVIDIA A100 GPU supports 312 TFLOPS bfloat16 with 1.5TB/s bandwidth [16], requiring an operational intensity of 208 FLOPS/B to prevent bandwidth bottlenecks.

Compilers such as TensorFlow XLA [27] mitigate this issue with *operation fusion*, merging multiple ops into one large op to avoid DRAM accesses of intermediate results, resulting in greater operational intensity and improved performance [38]. Most prior work has focused on training, where intermediate results must be preserved for the backwards pass [5, 58]. In this work, we focus on inference, which does not require a backwards pass, meaning that intermediate results may be immediately discarded after use.

Figure 3 shows that EfficientNet has low operational intensity due to its heavy use of depthwise-separable convolutions. Without op fusion, EfficientNet operational intensity ranges from 13 to 35 FLOPS/B, far below the level required to run without memory bottlenecks on TPU-v3 or A100. Using batching to amortize weight accesses across multiple inferences is effective for ResNet-50 and moderately effective for BERT, but not for EfficientNet due to its lower parameter count. As such, these workloads present a significant challenge to architects, since provisioning greater memory bandwidth can result in exorbitant incremental costs. Current fusion approaches are

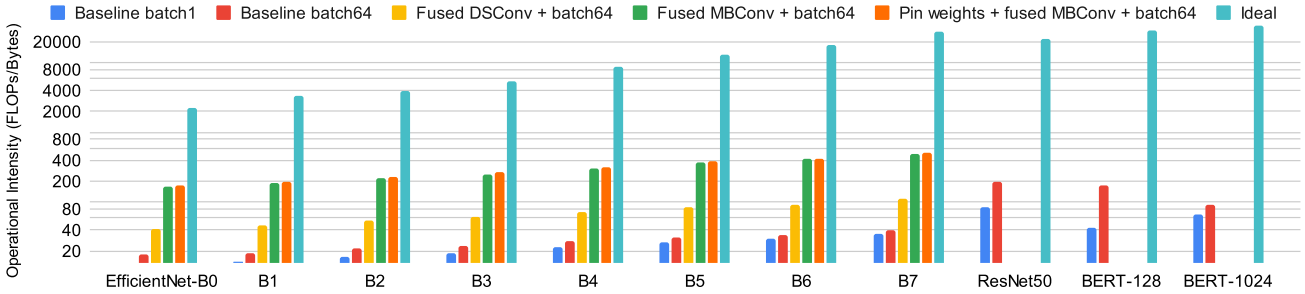


Figure 3: The impact of op fusion on operational intensity. Models with op intensity below 200 are memory bandwidth-bottlenecked on current accelerators. BERT and ResNet-50 do not contain depthwise-separable convolution (DSConv) or inverted residual (MBConv) blocks. Increasing batch size is effective for ResNet-50 and BERT-seq128, but not for EfficientNet and BERT-seq1024. Supporting future accelerators with op intensity over 400 requires more advanced fusion techniques.

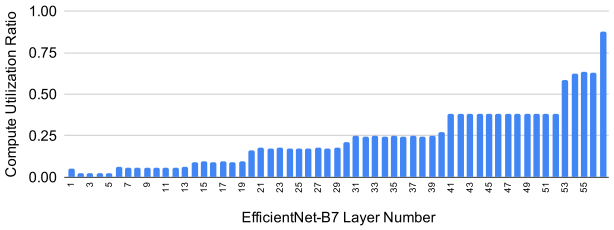


Figure 4: EfficientNet-B7 per-layer performance as a fraction of peak FLOPS on TPU-v3. Earlier layers have low utilization due to having few channels. A good utilization ratio should exceed 0.7. Smaller EfficientNets have worse utilization due to having fewer channels.

based on templates comprising specific compiler-defined sequences of ops [2]; we consider hypothetical depthwise-separable and MBConv fusion templates. By fusing entire MBConv blocks, we are able to achieve an operational intensity greater than 200 FLOPS/B. However, writing custom block fusion templates is not scalable. There is also considerable operational intensity headroom remaining, as shown by the ideal case in which all model weights are pinned [17] and only the input and final output results require off-chip accesses. These insights motivate *FAST fusion*, an fusion technique for inference capable of fusing arbitrary sequences of ops as described in Section 5.5, addressing the memory bottleneck.

Aggressive op fusion and weight pinning can come at the cost of significant on-chip storage capacity, as shown in Table 1. An op's working set size is the size of its input activations and outputs, and a model's working set size is the working set size of its largest op. Since working sets scale linearly with batch size, fusion tends to perform better at smaller batch sizes since more tensors will fit into SRAM. However, larger batch sizes can improve systolic array utilization, resulting in higher overall performance. Determining the best resource allocation between compute and memory depends on the specific operational intensity, memory footprint, and batch size for a target workload. FAST can automatically explore this space through datapath, scheduling, and fusion co-optimization.

4.2 EfficientNet Resource Utilization

We profiled EfficientNet-B7 performance on TPU-v3. Figure 4 shows the performance of each MBConv block as a fraction of peak TPU-v3 compute (FLOPS). Initial layers have poor utilization, with utilization

Table 2: EfficientNet-B7 per-op performance as a fraction of total execution time on TPU-v3. Depthwise convolutions consume the majority of execution time, due to their poor mapping efficiency on TPU-v3.

Op Type	FLOP Percentage	Runtime Percentage
DepthwiseConv2dNative	5.00%	65.30%
Conv2D	94.67%	34.20%
Other	0.33%	0.50%

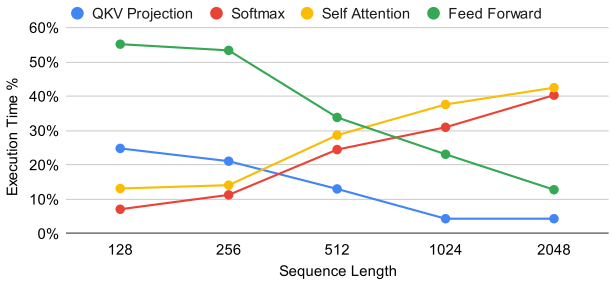


Figure 5: BERT per-op performance on TPU-v3. Softmax and self-attention ops, which run inefficiently on TPU-v3, dominate execution time at longer sequence lengths.

improving as the number of input/output channels increases. Overall TPU-v3 utilization on EfficientNet-B7 is only 14.8%, suggesting a potential 6.75x performance upside with an improved datapath and scheduler with similar peak FLOPS that can reach full utilization.

To identify the cause of low average utilization, we examined EfficientNet-B7 operation performance as a fraction of total execution time on TPU-v3 as shown in Table 2. The culprit is clear: depthwise convolutions comprise the majority of overall runtime, but only utilize a small fraction of total compute. An accelerator design that balanced depthwise convolution and regular convolution performance would therefore see significant speedups on EfficientNet. We discuss how this can automatically be achieved through FAST.

4.3 BERT Resource Utilization

We profiled BERT-Base [19] performance on TPU-v3 with default hyperparameters, sweeping sequence length from 128 to 2048. Since each BERT layer is identical, we broke a single layer into its subcomponents: Query/Key/Value matrix projection, softmax, self-attention, and feed-forward. The QKV projection and feed-forward ops already

run efficiently on TPU-v3, at 65% and 75% compute utilization respectively. Softmax is comprised of vector operations and thus must entirely execute on the TPU-v3 vector unit instead of its systolic array, so has exceptionally low compute utilization as defined as a fraction of peak throughput of less than 1%. Self-attention performs an *activation* × *activation* matrix multiply instead of *activation* × *weight* such that the cost of latching a matrix into the systolic array cannot be fully amortized over the batch dimension, resulting in lower utilization.

As shown in Figure 5, at low sequence lengths the efficient QKV projection and feed-forward ops dominate execution time, resulting in overall efficient execution. However, QKV projection and feed-forward computationally scale linearly with sequence length, whereas softmax and self-attention scale quadratically $O(N^2)$. Therefore, the inefficient softmax and self-attention ops dominate execution time at longer sequence lengths, resulting in poor overall performance. Self-attention performance can be addressed with smaller systolic arrays automatically discovered through FAST. We discuss techniques for addressing softmax performance in Section 5.6.

5 FULL-STACK ACCELERATOR SEARCH

FAST is a full-stack accelerator search technique for automatically designing custom accelerators optimized for a given set of ML workloads and subject to constraints as shown in Figure 1. We first consider if such techniques are practical, before describing the framework in detail in the following sections.

5.1 The Economics of Specialized Accelerators

It is well-known that increased hardware specialization improves performance [39, 73, 81, 107]. However, given that specialized accelerators target fewer workloads than general accelerators and building custom chips is expensive, it is less clear whether such specialization is economically viable. We analyze this question by examining *Return on Investment (ROI)*, a common profitability metric measuring an investment's return relative to its cost [22]; an ROI exceeding 1 is profitable. Companies typically aim to reach a predefined ROI threshold for their projects. A proper ROI calculation should be based on a company's specific circumstances. The following analysis is hypothetical, is based only on publicly-available data, and is intended to be used for illustrative purposes.

A simple method for estimating investment return is based on the savings from deploying a more cost-efficient accelerator relative to the baseline, typically either the currently-deployed accelerator or a next-generation design under consideration. Suppose we design an accelerator optimized for EfficientNet with a higher Perf/TCO relative to the baseline, and plan to offload all datacenter traffic currently running EfficientNet onto this new accelerator, i.e., aggregate QPS served by the new accelerator will be the same as the current accelerator. The ROI for this accelerator can be estimated as:

$$TCO_{old} = C_{cap}(n) + t_D \cdot C_{op}(n) \quad (1)$$

$$ROI = \frac{TCO_{old} \cdot (S-1)}{(t_{design} \cdot C_{eng} + C_{mask} + C_{IP}) \cdot S} \quad (2)$$

where $C_{cap}(n)$ and $C_{op}(n)$ are the capital and operational costs, respectively, to deploy n accelerators, t_D is the accelerator deployment lifetime in years, S is the Perf/TCO improvement relative to the baseline accelerator, t_{design} is the aggregate engineering-years to design the accelerator and its system software, C_{eng} is the corporate cost per engineer per year including compensation, benefits, and all overhead, C_{mask} is the wafer mask cost, and C_{IP} is the IP licensing cost such as the DRAM PHY. All pricing and power numbers for the baseline and new accelerator should include shared system server components, including the fractional host machine, networking, and rack infrastructure amortized between several accelerators.

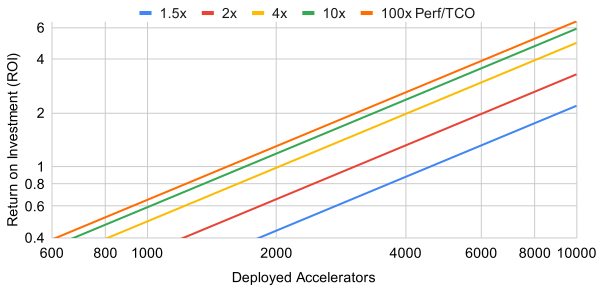


Figure 6: Accelerator Return on Investment (ROI) vs. deployment volume for hypothetical specialized accelerators with improved Perf/TCO relative to NVIDIA A100 baseline. An ROI exceeding 1 is profitable. Modern datacenters typically deploy thousands of accelerators, enabling initial engineering and manufacturing costs to be amortized.

Our example ROI calculation assumes a NVIDIA DGX A100 320GB platform baseline containing 8x A100 accelerators with a manufacturer's suggested price (MSRP) of $199,000 [87]. We assume the May 2021 average price of electricity for the US Commercial sector ($0.1084/kWh) from the US Energy Information Administration [1], an accelerator deployment lifetime of 3 years [89], the cost per engineer based on the reported median total compensation for a SWE working in the San Francisco bay area ($240,000) [3] with a 65% salary overhead [99], and all other values from previous work [99]. Since our experimental results assume a sub-10nm process technology, we extrapolate wafer mask and PHY IP costs using exponential scaling as observed in [99].

Estimating aggregate engineering years is more difficult, since it varies widely on a project methodology and company basis. Modern chip design has significantly reduced the engineering effort required for custom accelerators using techniques including HLS [44, 78] and agile design methodologies [50]. Custom accelerators can further reduce design time by leveraging existing hardware and software infrastructure. Simba [84] was built by 5-10 engineers in 20 months (12.5 engineer-years) to go from architecture to tape-out [44]. The Tesla Full Self Driving (FSD) SoC was built by 100 engineers [83] in 14 months [8] (117 engineer-years). Since Simba is a research test chip, and FSD is a full SoC containing a custom ML inference accelerator, we average the two designs to estimate the effort for a dedicated ML inference accelerator (65 engineer-years).

To approximate accelerator deployment volumes, a naive approach may be to divide a workload's QPS by the accelerator's throughput to estimate the total number of accelerators required to serve a certain amount of traffic. However, datacenters in practice are heavily over-provisioned to lower response latency and account for issues including traffic spikes, reliability, and projected future user growth [9]. These provisioning calculations are highly confidential; we therefore looked at public examples in industry. Microsoft

deployed 1,632 Catapult FPGAs for a medium-scale pilot study to accelerate a portion of the Bing search ranking pipeline [73]; currently, more than a million Catapult FPGAs are deployed in Microsoft datacenters [72]. Google has deployed tens of thousands of servers with video transcoding accelerators [78]. Facebook trains its Facer model on thousands of servers [31]. Large language models such as Meena [6], GShard [51], Switch Transformer [21], and GPT-3 [12] take 1024 to 10,000 accelerators to train [70]; a McKinsey study showed that datacenter inference demand typically exceeds training [10].

Figure 6 shows ROI as a function of the number of deployed accelerators, assuming hypothetical specialized accelerators capable of improving Perf/TCO from 1.5x to 100x relative to the NVIDIA DGX A100 baseline. There are several key takeaways. Firstly, a large deployment volume is the most important factor: all accelerators with positive Perf/TCO relative to the baseline become ROI-positive with sufficient volume. Secondly, there are diminishing returns to improving Perf/TCO under our strict definition of ROI. For example, deploying 8000 accelerators with 1.5x Perf/TCO has higher ROI than deploying 2000 accelerators with 100x Perf/TCO.

However, our ROI calculation is conservative since it only captures the returns from switching to a more cost-effective platform. Improving inference latency can increase revenue [97]; a 500ms delay in the Bing search engine reduced revenue per user by 1.2% [94]. A custom accelerator may also enable larger models that are currently infeasible for deployment on current platforms. Finally, a new accelerator deployment should not just replace the current accelerator baseline, but account for future application growth; Facebook DL inference server demand increased by 3.5x over less than two years [69]. Therefore, even accelerators that simply break-even in ROI while enabling significantly lower latency or larger models can potentially be economically viable with sufficient justification.

5.2 Problem Definition

Our objective is to find an optimized set of hyperparameters h for the hardware datapath, scheduler, and op fusion, given user-defined workloads w, objective function f (i.e., minimizing any function of power, area, and latency/throughput), subject to cost constraints (e.g., maximum area a or thermal design power p). Our optimization problem may be described by:

$$\min_{h,w} f(h, w) \tag{3}$$

$$\text{s.t.} \quad Area(h) \leq a, \quad TDP(h) \leq p, \tag{4}$$

$$ScheduleFailures(h, w) = 0, \tag{5}$$

The constraint $ScheduleFailures(h, w) = 0$ ensures that workload w can be successfully mapped onto the architecture described by the hyperparameters h.

5.3 FAST Framework Overview

The FAST framework explores the hardware datapath configuration, software schedule, and compiler operations for a combined search space up to $O(10^{2300})$. This estimate takes the product of the fully unconstrained mapspace [68] sizes for each layer in a moderately sized model like ResNet-50 (~10^{2000}), combined with the 10^{13} datapath and 10^{300} op fusion search spaces, rounded down.

As shown in Figure 1, the FAST framework uses a three-phase approach for each trial. Firstly, FAST uses Google Vizier [26], a black-box optimizer, to propose new choices of hyperparameters that define candidate hardware datapaths. To make exploring the schedule space more tractable, Vizier constrains the software schedule mapspace to known-good mapping schemes such as weight and output-stationary [14]. Secondly, our architectural simulator, described in Section 6.1, simulates the mapping and execution of target workloads on the candidate architecture. Compute-intensive ops such as Conv2D are optimized via pre-processing passes, such as tensor padding optimization, before calling Timeloop [68] with Vizier-provided constraints to determine the best schedule and predicted op performance. Finally, the per-op performance statistics are passed to our FAST fusion ILP solver to determine the best op fusion configuration. Our simulator then estimates op post-fusion performance and outputs final execution time and power for the target workloads. This cycle then repeats for thousands of trials until convergence.

Table 3: Accelerator datapath search space with 10^{13} possible values. When combined with scheduling and op fusion search spaces, the FAST total search space exceeds 10^{2300}. Memory technologies besides GDDR6 can easily be modeled.

Parameter Name	Type	Potential Values
PEs_x_dim	int	1 to 256, powers of 2
PEs_y_dim	int	1 to 256, powers of 2
Systolic_array_x	int	1 to 256, powers of 2
Systolic_array_y	int	1 to 256, powers of 2
Vector_unit_multiplier	int	1 to 16, powers of 2
L1_buffer_config	enum	Private, Shared
L1_input_buffer_size	int	1KB to 1MB, powers of 2
L1_weight_buffer_size	int	1KB to 1MB, powers of 2
L1_output_buffer_size	int	1KB to 1MB, powers of 2
L2_buffer_config	enum	Disabled, Private, Shared
L2_input_buffer_multiplier	int	1x to 128x, powers of 2
L2_weight_buffer_multiplier	int	1x to 128x, powers of 2
L2_output_buffer_multiplier	int	1x to 128x, powers of 2
L3_global_buffer_size	int	0MB to 256MB, powers of 2
GDDR6_channels	int	1 to 8, powers of 2
Native_batch_size	int	1 to 256, powers of 2

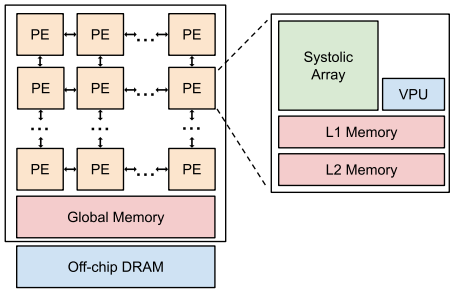


Figure 7: Accelerator datapath configuration. PEs are connected by a mesh on-chip network. PE systolic arrays perform a matrix-vector multiply each cycle. Vector and scalar MACs can be modeled by setting systolic array X and/or Y dims to 1. PEs also contain a VPU for non-MAC vector ops. L2 and Global Memory structures are optional.

5.4 Architectural Search Parameters

As shown in Figure 7, we target a highly-parameterized and general ML accelerator template capable of modeling a wide range of previously proposed architectural designs. Unlike prior work which

targets specific families of accelerators, we enlarged our datapath search space to cover an approximate superset of popular accelerator families based on grids of processing elements (PEs), as described in Table 3. In addition, to run models like BERT which require high performance non-MAC operations for layers such as softmax and layernorm, we add a fully general Vector Processing Unit (VPU) similar to TPU-v3 [38]. VPU width, as a multiple of systolic array width, is added to our search space. The TPU family of accelerators instantiate large systolic arrays coupled with two levels of shared memory. This can be represented in our framework by setting the systolic array dimensions to the appropriate values, setting L1_buffer_config to Shared, and L2_buffer_config to Disabled. Many accelerators such as Eyeriss [14] use flexible scalar PEs with per-PE buffers for input activations, weights, and output activations. This design can be reached by setting systolic array X and Y dimensions to 1, and L1_buffer_config to Private. Several edge accelerators proposed in industry such as Simba [84] and EdgeTPU [112] use vector PEs, which can be represented by setting the systolic array X dimension to 1. While our datapath search space cannot perfectly cover all possible designs, it is still much larger than those used in previous work [86, 93, 101]. We plan to further extend the search space in future work.

5.5 FAST Fusion

Modern neural network models pose a challenge due to their poor operational intensity, as discussed in Section 4.1. We propose *FAST fusion*, an aggressive fusion technique designed specifically for inference. FAST fusion leverages leftover Global Memory capacity unused by Timeloop to store activations and weights. Activations have short lifetimes, typically until the next op. Weights can be stored to reuse across multiple inference requests, a technique called *weight pinning*. FAST fusion must balance the benefits of fusing activations with weights while focusing on memory-bound ops; there is typically no performance benefit for fusing a compute-bound op when latency can be hidden. We express this constrained optimization problem as an *integer linear program* (ILP) minimizing cycle count using simulator performance metrics. We implemented FAST fusion as a secondary pass that fuses XLA-generated HLO fusion regions. This improves ease of implementation and greatly reduces ILP problem size, at the cost of potentially suboptimal fusions. FAST fusion conservatively assumes that entire tensors are stored in memory; schedulers can use inter-op blocking to reduce tensor working set sizes.

Next, we describe the problem formulation. We are given an input graph $G(V, E)$ representing an $n-$layer, partially fused[1] neural network which we wish to optimize, where each vertex $v \in V$ is a layer of the network, while each edge $e = (u, v)$ represents an activation dependency from layer u to v (that is, the output activation of u is an input to v). Let $F_{in}(v)$ and $F_{out}(v)$ represent the fan-in and fan-out sets, respectively, of some vertex $v \in V$. We assume that G has the property that while $0 \leq |F_o(v)| \leq n-1, 0 \leq |F_i(v)| \leq 1$. To simplify notation, let $D_t := \{I, O, W\}$ represent the set of data types used to annotate variables, where I, O, and W represent input activations, output activations, and weights, respectively. Given a known execution order $o : v \in V \longrightarrow \{0, \ldots, n-1\}$ for each network layer, we express the optimization problem in Figure 8.

$$
\begin{aligned}
\min_{p_i^k} \quad & \sum_{i \in V} T_i \\
\text{s.t.} \quad & T_i \geq T_i^{min} \\
& T_i \geq T_i^{max} - \sum_{k \in D_t} t_i^k \cdot p_i^k \\
& C_{GM} \geq B_i + \sum_{k \in D_t} d_i^k \cdot p_i^k + \sum_{j \in V, j \neq i} W_j \cdot p_j^W \\
& p_i^O \geq p_j^I \quad \forall j \in F_{out}(i) \\
& \sum_{j \in F_{out}(i)} p_j^I \geq p_i^O \\
& M \cdot (1 - p_i^I) \geq o(i) - o(F_{in}(i)) - 1 \\
& p_i^k \in \{0, 1\} \; \forall k \in D_t
\end{aligned}
\tag{6}
$$

Figure 8: Optimization problem for FAST fusion.

The variable p_i^k is a binary decision variable indicating whether the tensor of type $k \in D_t$ for layer i is to be placed in the Global Memory (if equal to 1), while the variable T_i represents the optimized execution time for layer i as a function of p_i^k. T_i^{min} and T_i^{max} are the execution times for layer i when the inputs and outputs of the layer are pinned exclusively in the Global Memory and DRAM, respectively (these are obtained from Timeloop evaluation of the layer). The parameter t_i^k is time to access layer i's tensor of type k (where $k \in D_t$) from DRAM, C_{GM} is the capacity of the Global Memory in bytes, B_i is the nominal global buffer usage of layer i, d_i^k is the difference between the size of layer i's tensor of type k and the corresponding tile size allocated on the global buffer if we were to assume the tensor is being streamed from/to DRAM, W_j is the size of layer j's weight tensor, and $M \geq n-1$ is an arbitrarily large constant. Note that the constraints imply that activations are only stored in the global buffer if the op consuming an activation executes immediately after the op which produces the activation. This also means that in cases where a node has multi-fanout (e.g., skip connections), at most only one op in the fanout cone will benefit from reading its input activation from global memory. These constraints – which limit the maximum potential upside of the technique – were imposed because of some limitations in our simulation infrastructure. Future work will address these limitations, thereby potentially allowing for further performance gains.

5.6 Two-Pass Softmax

The softmax operation is challenging on the TPU-v3 and other existing ASICs for several reasons. Firstly, calculating an exponential operation requires significant hardware resources. Typically, a look-up table is used with a Taylor series expansion, resulting in a large latency and area [67]. Secondly, numerically-stable softmax requires 3 passes over the input vector, as shown in Algorithm 1. Due to the size of the vector in most models, these 3 passes usually involve reading and writing the values to and from DRAM.

In order to reduce the number of memory accesses, [65] proposes a mathematically equivalent algorithm that performs the first 2 passes

[1] That is, $G(V, E)$ is derived from an original $m-$layer network that has been optimized such that combinations of data formatting, element-wise, and matrix operations have been grouped in fused computations [38].

Algorithm 1 Numerically-Stable Softmax

```
maxVal ← −∞
for i ← 1 to N do
    maxVal ← max(maxVal, V[i])
end for
sum ← 0
for i ← 1 to N do
    tempVec[i] ← exp(V[i] − maxVal)
    sum ← sum + tempVec[i]
end for
for i ← 1 to N do
    out[i] ← tempVec[i]/sum
end for
```

Algorithm 2 Two-Pass Softmax

```
runningMax ← −∞
runningSum ← 0
for i ← 1 to N do
    newMax ← max(runningMax, V[i])
    runningSum ← runningSum∗exp(runningMax−newMax)+
exp(V[i] − newMax)
    runningMax ← newMax
end for
for i ← 1 to N do
    out[i] ← exp(V[i])/runningSum
end for
```

together, as seen in Algorithm 2. The two-pass softmax eliminates N memory accesses relative to Algorithm 1, but increases the number of exponential calculations by up to $2N$. Therefore, the benefit of the two-pass approach is dependent on the accelerator's memory bandwidth and vector unit throughput. We add the option to enable the two-pass softmax as a hyperparameter within our FAST search space.

6 EVALUATION

6.1 Experimental Setup

Methodology and Simulator: We use TPUs as the baseline because TPUs are well-characterized as the most popular dedicated datacenter ML accelerator; TPUs vs GPUs have also been well-studied in prior work [39] [37] [63] [28]. We modified an internal TPU performance simulator to enable modeling of a wide range of architectures as described in Section 5.4. The baseline simulator is well-correlated: on our benchmark suite, simulator accuracy is on average within 8.2 ± 2.7% of profiled TPU-v3 performance. Because our simulator tends to produce slightly optimistic results, we evaluated against a simulated rather than measured TPUv3 baseline to take optimistic simulator assumptions into account.

Our simulator takes unmodified XLA HLO graphs [27] as input and is modified to employ Timeloop [68] to evaluate the performance of Conv2D, DepthwiseConv2D, Einsum, and MatMul operations. Since Timeloop cannot handle problem dimensions that do not factorize cleanly into hardware datapath dimensions, we added a padding preprocessing step to improve utilization. All other ops, such as vector ops used in softmax, are modeled using our simulator's custom cost models. Simulated design points with Timeloop scheduling failures are considered invalid. To estimate area and power consumption, we built analytical models correlated to production designs on an industry sub-10-nm manufacturing process; for a fair comparison, the TPU-v3 baseline is also modeled using this same process technology. TDP is estimated as *power virus power*, in which each component is assumed to be accessed at 100% utilization. FAST fusion's ILP is solved using `SCIP v7.0.1` [23], and is configured with a 20 minute time-out; if an optimal solution is not found in that time the solver returns the best incumbent solution. Because we use prefused XLA HLO graphs as input, our FAST fusion implementation fuses XLA-generated fusion regions instead of individual ops.

Workloads: FAST is evaluated on a range of state-of-the-art workloads in both computer vision and natural language processing domains. The entire suite of EfficientNets is evaluated, from B0 to B7 [91]. BERT-Base [19] is evaluated for both short (128) and long (1024) sequence lengths. ResNet50v2 [32] is one of the most popular CNN-based models. We also evaluated two components of a production OCR pipeline described in [74]. OCR-RPN is the first stage in a standard Mask R-CNN implementation used to propose candidate text regions of interest. OCR-Recognizer is an LSTM-based model within the recognizer pipeline. These workloads were selected based on their range of performance characteristics on TPU-v3. EfficientNets currently run less efficiently on TPUs due to their use of depthwise-separable convolutions (see Section 4.2). BERT runs efficiently on TPUs at short sequence lengths, but is less efficient at longer sequence lengths (see Section 4.3); we capture both by evaluating BERT with sequence lengths 128 and 1024. ResNet50v2 runs much more efficiently than EfficientNet by using standard Conv2D operations. OCR-RPN and OCR-Recognizer are already optimized to run efficiently on TPUs, and represent a worst-case scenario: models that already run efficiently on our TPU-v3 baseline will benefit less from FAST.

Optimization framework: We used Google Vizier [26] enabling LCS optimization [25] and safe search [24], disabling transfer learning, with 5000 trials per experiment. Each trial takes 10 minutes to 2 hours wall clock time based on model size and datapath constraints.

6.2 Experimental Results

6.2.1 Overall Speedup. Figure 9 shows overall performance improvement from FAST-generated custom accelerators on each workload relative to a simulated TPU-v3 baseline, in which performance is measured in processed inference queries per second (QPS). FAST is given a power and area budget similar to the current-generation TPU-v3, but on a new process technology, emulating the methodology used by accelerator architects to design next-generation products. The purpose of this experiment is to evaluate if FAST can fully utilize the available power and area headroom to create high-performance designs. We evaluate FAST optimizing for individual workloads as well as across multiple workloads. Our multi-workload optimized FAST finds a single hardware design evaluated on the geometric mean across EfficientNet-B7, ResNet50v2, OCR-RPN, OCR-Recognizer, and BERT-1024 achieving a 3.1× speedup over TPU-v3 baseline. Overall speedups are much higher on EfficientNets due to their use of depthwise separable convolutions. When provided with pure performance as the objective, FAST successfully finds large designs that come close to our maximum area and TDP constraints. OCR-RPN and OCR-Recognizer are already well-optimized for TPU-v3 and thus have the lowest gains as expected. Utilizing FAST-specified scheduling and fusion on the TPU-v3 datapath provides a substantial 1.7× speedup; however, this is optimistic since implementing the generated schedules and achieving the projected speedup may require hardware changes. Tuning an architecture

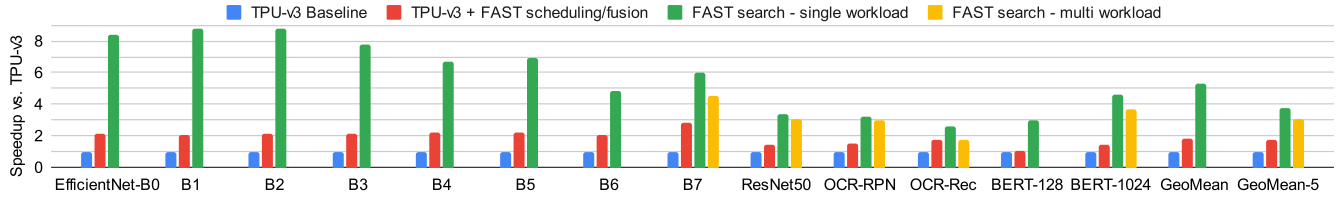


Figure 9: Modeled inference throughput relative to TPU-v3. "FAST scheduling/fusion" optimization offers large speedups over existing TPU-v3. Speedups are much larger when FAST is also allowed to search over the datapath. "FAST search - single workload" are optimized designs for a specific workload. "FAST-search - multi workload" is a single design optimized across the 5 workloads (i.e., EfficientNet-B7, ResNet50, OCR-RPN, OCR-Rec, BERT-1024). GeoMean and GeoMean-5 results correspond to the geometric mean across all workloads, and across the aforementioned 5 workloads respectively.

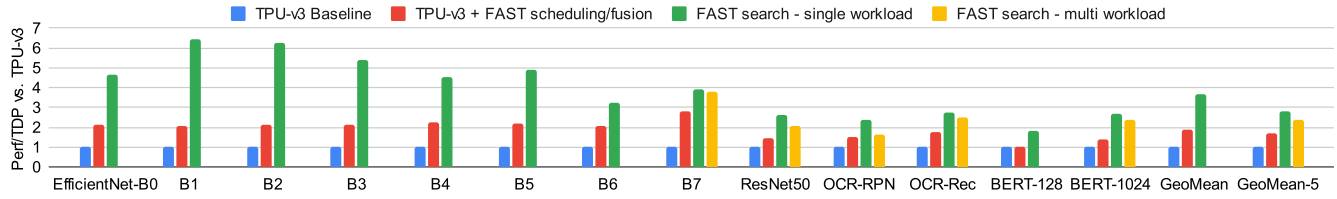


Figure 10: Modeled inference throughput per TDP (peak power draw) relative to TPU-v3, normalized to the same manufacturing process technology. FAST demonstrates large Perf/TDP wins across all workloads.

Table 4: FAST-generated accelerator deployment volume required to reach a specific ROI target based on Perf/TDP speedups from Figure 10. An ROI above 1 is profitable. "Multi-Workload" is optimized across a set of workloads: EfficientNet-B7, ResNet50, OCR-RPN, OCR-Rec, BERT-1024.

Target Workload	Perf/TCO	1x ROI	2x ROI	4x ROI	8x ROI
EfficientNet-B7	3.91x	2,164	4,327	8,655	17,309
ResNet50	2.65x	2,588	5,177	10,354	20,707
OCR-RPN	2.34x	2,810	5,620	11,241	22,482
OCR-Rec	2.72x	2,548	5,096	10,192	20,385
BERT-128	1.84x	3,534	7,069	14,138	28,276
BERT-1024	2.7x	2,558	5,115	10,231	20,462
Multi-Workload	2.82x	2,792	5,584	11,167	22,335

across multiple workloads results in slightly reduced, but still substantial improvements over the baseline. FAST search achieves a 3.8× average speedup when optimizing for single workloads.

Absolute performance numbers can be misleading since different hardware designs vary in cost. A common metric for evaluating datacenter accelerators is to normalize for these differences by considering performance per TCO, which includes initial capital expenses and recurring operating costs such as electricity (Section 5.1). While due to sensitivity of data, TCO numbers are not published, TDP can be used as a proxy for TCO [39]. Figure 10 shows Perf/TDP numbers relative to a hypothetical TPU-v3 die-shrunk to the same sub-10-nm process technology. When optimizing for Perf/TDP, FAST finds balanced designs that are smaller than our maximum area and TDP constraints, but achieve high compute utilization with minimal memory bandwidth bottlenecks. Designs found by FAST tend to have smaller systolic arrays, smaller L1 scratchpads, and larger Global Memories than the TPU-v3 baseline. Two-pass softmax was not useful when fusion was enabled. Overall, FAST individually optimized for each workload improves Perf/TDP on average by 3.7× across all workloads and 2.8× on the reduced workload suite, whereas FAST optimized for multiple workloads still improves Perf/TDP by 2.4×.

6.2.2 ROI Discussion. To determine the practicality of building FAST-generated ML accelerators optimized for single workloads, we estimate FAST accelerator ROI relative to our TPUv3 baseline in Table 4, using the methodology and parameters as described in Section 5.1. We estimate Perf/TCO speedup relative to TPUv3 using the Perf/TDP speedups shown in Figure 10. Prior work shows that Perf/TCO and Perf/TDP are highly correlated [37]. TPUs are not available for sale to the general public and TPU TCO is confidential; we instead used the NVIDIA DGX A100 320GB platform to approximate TPUv3 TCO. This example is intended as an estimate for illustrative purposes only; entities planning to build their own custom accelerators should calculate ROI using costs and baselines specific to their own situation.

Our analysis shows that the ROI 1x break-even point can be reached with a deployment volume of 2,164 to 3,534 accelerators. However, breaking even is typically not enough to justify building custom accelerators; the goal is typically to turn a profit. Many corporations make business decisions based on whether a planned project reaches a pre-determined ROI threshold [22]. Due to ROI diminishing returns from increasing Perf/TCO, it is typically better to target larger deployment volumes, thus suggesting that the FAST-generated design optimized for 5 different workloads (Multi-Workload) may be a more profitable design since it can likely be deployed in a much larger volume without much impact on Perf/TCO. Even single-workload designs can easily reach profitability with typical deployment sizes discussed in Section 5.1.

6.2.3 Search Convergence Rate. We evaluated several black box optimizer heuristics as provided by Vizier. In Figure 11, we compare the convergence rate of Vizier's default Bayesian algorithm against Linear Combination Swarm (LCS) [25] and random sampling when optimizing for Perf/TDP on EfficientNet-B7. We show the mean and 90% confidence interval across each heuristic, across 5 runs per heuristic. LCS outperforms the other heuristics when trials exceed 2000.

6.2.4 Pareto Frontier. To evaluate our search space coverage, in Figure 12 we characterize the relationship between performance, TDP

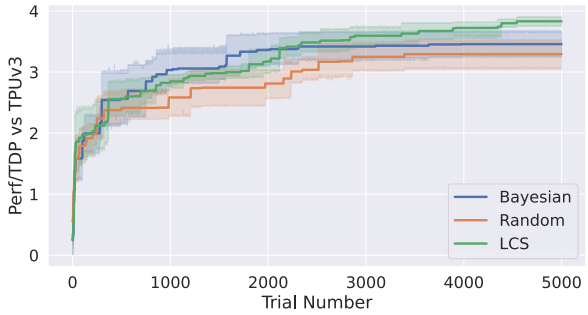


Figure 11: Search convergence rate on EfficientNet-B7 for Bayesian, random, and Linear Combination Swarm [25].

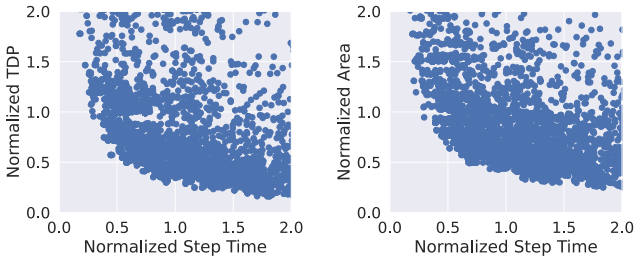


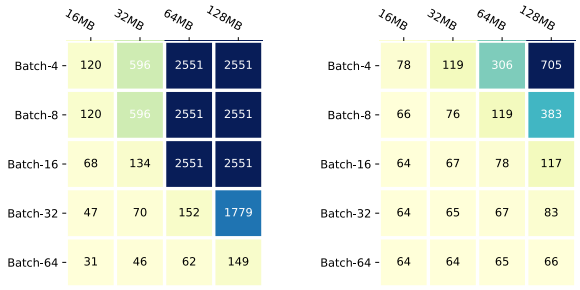
Figure 12: EfficientNet-B7 step time vs. TDP and area relative to TPU-v3 on the same process technology.

Table 5: Two example designs found by FAST optimized for EfficientNet-B7 with similar overall Perf/TDP. Area and power are normalized to threshold constraints.

	Modeled TPU-v3	FAST-Large	FAST-Small
Normalized TDP	0.5x	0.4x	0.15x
Normalized Area	0.6x	0.7x	0.3x
Peak Compute	123 TFLOPS	131 TFLOPS	32 TFLOPS
Peak Bandwidth	900 GB/s	448 GB/s	448 GB/s
Batch Size	2x64	8	64
Num PEs	2x2	64	8
PE Systolic Array Dims	128x128	32x32	64x32
PE Vector Width	512	32	64
PE L1 Buffer Size	2x64KiB	8 KiB	8 KiB
PE L1 Buffer Config	Shared	Shared	Shared
PE L2 Buffer Config	Disabled	Disabled	Disabled
Global Buffer Size	2x16 MiB	128 MiB	8 MiB
Compute Utilization	0.14	0.61	0.74
Pre-fusion Mem Stall %		63%	21%
Fusion Efficiency		85%	0%
OpInt Ridgepoint	137	292	73
Fused Model OpInt	63	383	63
B7 Performance (QPS)	210 (aggregate)	733	241
B7 Inference Latency	609ms	11ms	265ms
Normalized Perf/TDP	1	3.9	3.9

and area on EfficientNet-B7. Each figure is normalized to a hypothetical TPU-v3 built with the same sub-10nm process technology at (1.0, 1.0), and points located towards the lower left are Pareto-optimal. FAST is able to find a range of designs significantly better than the baseline, suggesting that FAST can be easily applied to other domains besides datacenters, such as targeting embedded systems.

6.2.5 Example Designs Found by FAST. Table 5 shows two example designs found with FAST when optimizing Perf/TDP on EfficientNet-B7, compared to TPU-v3 normalized to the same sub-10nm process



(a) EfficientNet-B0 **(b) EfficientNet-B7**

Figure 13: FAST-Large post-fusion operational intensity, sweeping Global Memory (columns) and batch size (rows). Operational intensity increases with larger Global Memory and smaller batch size. Higher is better, but there is no more performance benefit after reaching the ridgepoint (292).

technology. TPU-v3 is a dual-core design, in which each core is treated as a separate accelerator; our EfficientNet-B7 QPS results show aggregate QPS when using both cores. Each TPU-v3 core contains two 128x128 systolic arrays and a 1024-wide vector unit. FAST-generated designs are all single-core, in which each core contains one or more PEs. EfficientNet-B7 when executed on TPU-v3 is both compute-bound (low compute utilization of 0.14) and memory-bound (low operational intensity of 63), both of which are addressed by FAST designs with different approaches. Overall, FAST preferred small shared L1 buffers with no L2 buffer; although L2 buffers may reduce dynamic power from improved blocking, they increase overall TDP when assuming maximum buffer accesses per cycle. To improve mapping efficiency for depthwise-separable convolutions, both designs have PEs with smaller systolic array dimensions resulting in significantly higher compute utilization. Despite FAST-Large having similar peak compute performance as TPU-v3 with half the peak memory bandwidth, the design is not bandwidth-bottlenecked due to its 128MiB Global Buffer, enabling aggressive FAST fusion to improve operational intensity from 63 to 383. Overall, idle time spent waiting for DRAM transfers to complete is reduced by 85%, from 63% pre-fusion to 9% post-fusion. FAST-Small avoids fusion entirely and achieves high efficiency through a low compute to memory bandwidth ratio. Although both designs have similar Perf/TDP, FAST-Large is preferred for datacenter environments because it meets MLPerf image classification latency requirements (15ms) [40], enabling EfficientNet-B7 for latency-sensitive applications.

6.2.6 Evaluating FAST Fusion. We evaluate FAST fusion performance in Figure 13 by measuring its impact on operational intensity as we sweep Global Memory and batch size in an otherwise-fixed FAST-Large design. Increasing Global Memory capacity enables FAST fusion to assign more activation and weight tensors from DRAM to Global Memory, resulting in higher operational intensity. Decreasing batch size decreases tensor activation size (see Table 1), increasing operational intensity since more tensors can be kept in Global Memory. However, decreasing batch size also potentially reduces compute utilization due to decreased parallelism. Therefore, the goal of FAST fusion is to select the largest batch size in which post-fusion operational intensity meets or exceeds the accelerator ridgepoint (292 for FAST-Large). EfficientNet-B0 has small activation

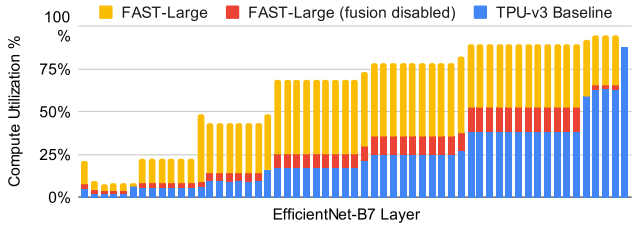


Figure 14: FAST-Large EfficientNet-B7 per-layer performance as a fraction of peak FLOPS. Changing from TPU-v3's 128x128 systolic arrays to FAST-Large's 32x32 systolic arrays improves compute utilization, but remains bottlenecked by memory bandwidth until FAST fusion is enabled.

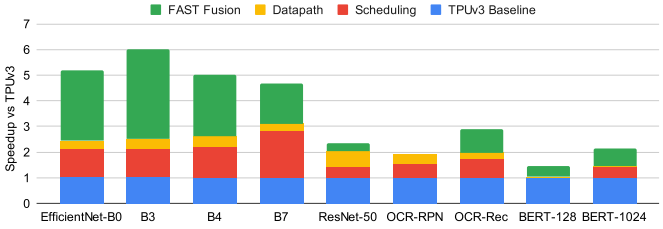


Figure 15: Performance breakdown of each component of FAST relative to a modeled TPU-v3 single core baseline. Improvements are additive; for example, FAST fusion includes both datapath and scheduling improvements.

and weight tensor sizes, making it easy for FAST fusion to exceed 292. The largest EfficientNet model, B7, represents a worst-case scenario for fusion, but FAST fusion can still achieve sufficiently high operational intensity at batch size 8 to overcome the memory bottleneck.

6.2.7 Performance Characterization. In Figure 15, we show the contributions of improved scheduling from Timeloop, datapath improvements, and FAST fusion. We start with the TPU-v3 baseline, and then incrementally add improvements from FAST-Large. Since TPU-v3 is a dual-core design whereas FAST-Large is single-core, we compare a single TPU-v3 core against a halved FAST-Large design with 32 PEs. The scheduling component shows the potential speedup from better mappings discovered by Timeloop; implementing these better mappings may require changes to the hardware. In the datapath component, the TPU-v3's 128x128 systolic arrays are replaced with 32x32 systolic arrays, keeping peak FLOPS constant. We also replace TPU-v3's 16MB global memory with the FAST-discovered 128MB global memory. Datapath improvements without FAST fusion result in significantly lower speedups since performance is a function of both compute and memory, and increasing compute utilization results in no further improvements once the memory bandwidth limit is reached. There is no performance benefit from increasing global memory size when fusion is disabled. Enabling FAST fusion removes the memory bandwidth bottleneck, allowing the improved datapath to realize its utilization improvements. Scheduling, datapath, and fusion all work in synergy to achieve FAST's projected speedups, thereby demonstrating the criticality of including fusion when performing hardware datapath + scheduling co-optimization to address both memory and compute bottlenecks.

In Table 6, we performed an *ablation study* to evaluate FAST-Large performance relative to TPU-v3. An ablation study characterizes system performance by removing individual components to understand

Table 6: FAST-Large ablation study measuring Perf/TDP relative to a die-shrunk TPUv3 baseline. Numbers in parentheses show Perf/TDP relative to FAST-Large. The first row shows an unmodified FAST-Large baseline. Subsequent rows show FAST-Large with a single component reverted to the TPUv3 baseline with no other changes.

	EfficientNet-B7	ResNet50	BERT-Seq1024
FAST-Large	4.27x (1.00)	2.95x (1.00)	2.39x (1.00)
With 16MB Global Mem	2.26x (0.53)	2.20x (0.75)	1.22x (0.51)
Without FAST Fusion	1.91x (0.45)	1.74x (0.59)	1.05x (0.44)
With 128x128 systolic arrays	2.69x (0.63)	1.41x (0.48)	1.35x (0.56)
With 32KB L1 scratchpads	3.20x (0.75)	2.26x (0.77)	1.83x (0.77)

each component's contribution to the overall system. The first row shows unmodified FAST-Large Perf/TDP relative to our die-shrunk TPU-v3 baseline. Subsequent rows evaluate FAST-Large with a single component replaced with what was used in the original TPU-v3 design with no other changes. For example, the second row evaluates FAST-Large with its 128MB Global Memory replaced with the 16MB Global Memory used in TPU-v3. The resulting performance loss of each row relative to the first row represents the feature's importance towards achieving good overall performance. Moving from 32KB to 8KB L1 scratchpads has a minimal impact on performance with lowered power, resulting in improved Perf/TDP. Removing each component optimization results in substantial Perf/TDP degradation, thereby demonstrating their importance for FAST-Large.

7 CONCLUSION AND FUTURE WORK

We presented FAST, a full-stack accelerator search technique that performs joint optimization of the hardware datapath, software scheduling, and compiler passes such as operation fusion and tensor padding. Our results demonstrate that FAST-generated inference accelerators can provide large Perf/TDP improvements on state-of-the-art computer vision and natural language processing models with compelling ROIs. For example, the FAST-Large design provides 3.9× Perf/TDP improvement over a die-shrunk TPU-v3 baseline through a combination of higher efficiency and lower optimal batch size, thus enabling not only substantial Perf/TCO improvements but also enables EfficientNet-B7 to be deployed for latency-sensitive applications. FAST-generated accelerators currently do not support the full feature set provided by designs like TPU-v3 optimized for not just single-chip inference, but also training across thousands of devices. However, key inference datacenter workloads may be sufficiently important or provide sufficient volume for substantial returns on investment. Specialized designs optimized for small sets of workloads are unlikely to completely replace general-purpose designs, but may still serve an important niche in production environments. By substantially enlarging the set of workloads, FAST may also be used to propose the design of future generations of general-purpose ML accelerators. We plan to extend FAST by further enlarging the search space and adding support for optimizing accelerators for training.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers, Herman Schmit, Norm Jouppi, Cliff Young, James Laudon, Ed Chi, Priyanka Raina, Milad Hashemi, Yanqi Zhou, and Kunle Olukotun for their valuable feedback.

REFERENCES

[1] 2020. Electric Power Monthly with Data for May 2021. https://www.eia.gov/electricity/monthly/current_month/july2021.pdf Accessed: 2021-08-09.

[2] 2021. Developer Guide - NVIDIA Deep Learning cuDNN. https://web.archive.org/web/20210520075036/https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html#op-fusion

[3] 2021. Software Engineer Salaries in San Francisco Bay Area. https://www.levels.fyi/Salaries/Software-Engineer/San-Francisco-Bay-Area/ Accessed: 2021-08-09.

[4] Mohamed S. Abdelfattah, Lukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D. Lane. 2020. Best of Both Worlds: AutoML Codesign of a CNN and Its Hardware Accelerator. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference* (Virtual Event, USA) *(DAC '20)*. IEEE Press, Article 192, 6 pages. https://dl.acm.org/doi/abs/10.5555/3437539.3437731

[5] Amirali Abdolrashidi, Qiumin Xu, Shibo Wang, Sudip Roy, and Yanqi Zhou. 2019. Learning to Fuse. *Workshop on ML for Systems at NeurIPS*.

[6] Daniel Adiwardana, Minh-Thang Luong, David R So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, et al. 2020. Towards a human-like open-domain chatbot. *arXiv preprint arXiv:2001.09977* (2020).

[7] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783725

[8] Pete Bannon, Ganesh Venkataramanan, Debjit Das Sarma, Emil Talpes, and Bill McGee. 2021. Compute and Redundancy Solution for the Full Self-Driving Computer. https://web.archive.org/web/20210413053454/https://old.hotchips.org/hc31/HC31_2.3_Tesla_Hotchips_ppt_Final_0817.pdf

[9] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. 2018. *The Datacenter as a Computer: Designing Warehouse-Scale Machines* (3rd ed.). Morgan & Claypool Publishers.

[10] Gaurav Batra, Zach Jacobson, Siddarth Madhav, Andrea Queirolo, and Nick Santhanam. 2018. Artificial-intelligence hardware: New opportunities for semiconductor companies. *McKinsey & Company, New York, NY, USA, Tech. Rep* (2018).

[11] Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Prithviraj Sen, Alexandre V. Evfimievski, and Niketan Pansare. 2018. On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML. *Proc. VLDB Endow.* 11, 12 (Aug. 2018), 1755–1768. https://doi.org/10.14778/3229863.3229865

[12] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[13] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. arXiv:1812.00332 [cs.LG]

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

[15] François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1800–1807. https://doi.org/10.1109/CVPR.2017.195

[16] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro* (2021), 1–1. https://doi.org/10.1109/MM.2021.3061394

[17] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Maleen Abeydeera, Logan Adams, Hari Angepat, Christian Boehn, Derek Chiou, Oren Firestein, Alessandro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Ahmad El Husseini, Tamas Juhasz, Kara Kagi, Ratna K. Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Brandon Perez, Amanda Rapsang, Steven Reinhardt, Bita Rouhani, Adam Sapek, Raja Seera, Sangeetha Shekar, Balaji Sridharan, Gabriel Weisz, Lisa Woods, Phillip Yi Xiao, Dan Zhang, Ritchie Zhao, and Doug Burger. 2018. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. *IEEE Micro* 38, 2 (2018), 8–20. https://doi.org/10.1109/MM.2018.022071131

[18] Shail Dave, Aviral Shrivastava, Youngbin Kim, Sasikanth Avancha, and Kyoungwoo Lee. 2020. dMazeRunner: Optimizing Convolutions on Dataflow Accelerators. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1544–1548. https://doi.org/10.1109/ICASSP40776.2020.9054275

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[20] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 92–104. https://doi.org/10.1145/2749469.2750389

[21] William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961* (2021).

[22] George T Friedlob and Franklin J Plewa Jr. 1996. *Understanding return on investment*. John Wiley & Sons.

[23] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. 2020. *The SCIP Optimization Suite 7.0*. Technical Report. Optimization Online. http://www.optimization-online.org/DB_HTML/2020/03/7705.html

[24] Michael A. Gelbart, Jasper Snoek, and Ryan P. Adams. 2014. Bayesian Optimization with Unknown Constraints. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence* (Quebec City, Quebec, Canada) *(UAI'14)*. AUAI Press, Arlington, Virginia, USA, 250–259. https://dl.acm.org/doi/10.5555/3020751.3020778

[25] Daniel Golovin, Greg Kochanski, and John Elliot Karro. 2017. Black box optimization via a bayesian-optimized genetic algorithm. In *10th NIPS Workshop on Optimization for Machine Learning*.

[26] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. 2017. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 1487–1495. https://doi.org/10.1145/3097983.3098043

[27] Google. 2018. *XLA: Optimizing Compiler for TensorFlow*. https://www.tensorflow.org/xla

[28] Suyog Gupta and Mingxing Tan. 2019. EfficientNet-EdgeTPU: Creating Accelerator-Optimized Neural Networks with AutoML. https://ai.googleblog.com/2019/08/efficientnet-edgetpu-creating.html

[29] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations*. http://arxiv.org/abs/1510.00149

[30] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen mei Hwu, and Deming Chen. 2019. FPGA/DNN Co-Design: An Efficient Design Methodology for IoT Intelligence on the Edge. In *1904.04421*. https://doi.org/10.1145/3316781.3317829

[31] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. 2018. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 620–629. https://doi.org/10.1109/HPCA.2018.00059

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[33] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W. Fletcher. 2021. Mind Mappings: Enabling Efficient Algorithm-Accelerator Mapping Space Search. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 943–958. https://doi.org/10.1145/3445814.3446762

[34] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[35] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[36] Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. 2020. Hardware/Software Co-Exploration of Neural Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (2020), 4805–4815. https://doi.org/10.1109/TCAD.2020.2986127

[37] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. 2021. Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1–14. https://doi.org/10.1109/ISCA52012.2021.00010

[38] 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 (June 2020), 67–78. https://doi.org/10.1145/3360307

[39] 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, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, 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 Samadiani, 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 44th Annual International Symposium on Computer Architecture*. https://doi.org/10.1145/3079856.3080246

[40] David Kanter and Vijay Janapa Reddi. 2021. MLPerf Inference Rules. https://github.com/mlcommons/inference_policies/blob/master/inference_rules.adoc

[41] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 622–636. https://doi.org/10.1109/MICRO50266.2020.00058

[42] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. https://doi.org/10.1145/3400302.3415639

[43] Samuel J. Kaufman, Phitchaya Mangpo Phothilimthana, Yanqi Zhou, and Mike Burrows. 2020. A Learned Performance Model for the Tensor Processing Unit. *ML for Systems Workshop at NeurIPS*, Article arXiv:2008.01040 (Aug. 2020), arXiv:2008.01040 pages.

[44] Brucek Khailany. 2019. Machine-Learning-Assisted Agile VLSI Design For Machine Learning. https://web.archive.org/web/20210810054054/http://crva.ict.ac.cn/documents/agile-and-open-hardware/khailany-sigarch-visioning-oahw2019.pdf

[45] Sun Yuan Kung. 1988. VLSI array processors: designs and applications. In *IEEE International Symposium on Circuits and Systems*. 313–320 vol.1. https://doi.org/10.1109/ISCAS.1988.14929

[46] Ian Kuon and Jonathan Rose. 2007. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 2 (2007), 203–215. https://doi.org/10.1109/TCAD.2006.884574

[47] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. 2020. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. *IEEE Micro* 40, 3 (2020), 20–29. https://doi.org/10.1109/MM.2020.2985963

[48] 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 *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=H1eA7AEtvS

[49] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4013–4021. https://doi.org/10.1109/CVPR.2016.435

[50] Yunsup Lee, Andrew Waterman, Henry Cook, Brian Zimmer, Ben Keller, Alberto Puggelli, Jaehwa Kwak, Ruzica Jevtic, Stevo Bailey, Milovan Blagojevic, Pi-Feng Chiu, Rimas Avizienis, Brian Richards, Jonathan Bachrach, David Patterson, Elad Alon, Bora Nikolic, and Krste Asanovic. 2016. An Agile Approach to Building RISC-V Microprocessors. *IEEE Micro* 36, 2 (2016), 8–20. https://doi.org/10.1109/MM.2016.11

[51] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. Gshard: Scaling giant models with conditional computation and automatic sharding. In *Ninth International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2006.16668

[52] Rui Li, Yufan Xu, Aravind Sukumaran-Rajam, Atanas Rountev, and P. Sadayappan. 2021. Analytical Characterization and Design Space Exploration for Optimization of CNNs. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 928–942. https://doi.org/10.1145/3445814.3446759

[53] Sheng Li, Mingxing Tan, Ruoming Pang, Andrew Li, Liqun Cheng, Quoc Le, and Norman P. Jouppi. 2021. Searching for Fast Model Families on Datacenter Accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/CVPR46437.2021.00799

[54] Yuhong Li, Cong Hao, Xiaofan Zhang, Xinheng Liu, Yao Chen, Jinjun Xiong, Wen mei Hwu, and Deming Chen. 2020. EDD: Efficient Differentiable DNN Architecture and Implementation Co-search for Embedded AI Solutions. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*. https://dl.acm.org/doi/10.5555/3437539.3437669

[55] Shengwen Liang, Cheng Liu, Ying Wang, Huawei Li, and Xiaowei Li. 2020. DeepBurning-GL: An Automated Framework for Generating Graph Neural Network Accelerators. In *Proceedings of the 39th International Conference on Computer-Aided Design* (Virtual Event, USA) *(ICCAD '20)*. Association for Computing Machinery, New York, NY, USA, Article 72, 9 pages. https://doi.org/10.1145/3400302.3415645

[56] Yujun Lin, Driss Hafdi, Kuan Wang, Zhijian Liu, and Song Han. 2019. Neural-Hardware Architecture Search. In *NeurIPS ML for Systems Workshop*.

[57] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[58] Guoping Long, Jun Yang, Kai Zhu, and Wei Lin. 2018. FusionStitching: Deep fusion and code generation for tensorflow computations on gpus. *arXiv preprint arXiv:1811.05213* (2018).

[59] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. 2017. FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 553–564. https://doi.org/10.1109/HPCA.2017.29

[60] Chenhui Ma, Xiaodong Mu, and Dexuan Sha. 2019. Multi-Layers Feature Fusion of Convolutional Neural Network for Scene Classification of Remote Sensing. *IEEE Access* 7 (2019), 121685–121694. https://doi.org/10.1109/ACCESS.2019.2936215

[61] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17)*. 45–54. https://doi.org/10.1145/3020078.3021736

[62] Ikuo Magaki, Moein Khazraee, Luis Vega Gutierrez, and Michael Bedford Taylor. 2016. ASIC Clouds: Specializing the Datacenter. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 178–190. https://doi.org/10.1109/ISCA.2016.25

[63] Peter Mattson, Vijay Janapa Reddi, Christine Cheng, Cody Coleman, Greg Diamos, David Kanter, Paulius Micikevicius, David Patterson, Guenther Schmuelling, Hanlin Tang, Gu-Yeon Wei, and Carole-Jean Wu. 2020. MLPerf: An Industry Standard Benchmark Suite for Machine Learning Performance. *IEEE Micro* 40, 2 (2020), 8–16. https://doi.org/10.1109/MM.2020.2974843

[64] Linyan Mei, Pouya Houshmand, Vikram Jain, Sebastian Giraldo, and Marian Verhelst. 2021. ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators. *IEEE Trans. Comput.* 70, 8 (2021), 1160–1174. https://doi.org/10.1109/TC.2021.3059962

[65] Maxim Milakov and Natalia Gimelshein. 2018. Online normalizer calculation for softmax. arXiv:1805.02867 [cs.PF]

[66] Pandu Nayak. 2019. Understanding searches better than ever before. https://blog.google/products/search/search-language-understanding-bert/

[67] Peter Nilsson, Ateeq Ur Rahman Shaik, Rakesh Gangarajaiah, and Erik Hertz. 2014. Hardware implementation of the exponential function using Taylor series. In *2014 NORCHIP*. 1–4. https://doi.org/10.1109/NORCHIP.2014.7004740

[68] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[69] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Shanker Khudia, James Law, Parth Malani, Andrey Malevich, Nadathur Satish, Juan Miguel Pino, Martin Schatz, Alexander Sidorov, Viswanath Sivakumar, Andrew Tulloch, Xiaodong Wang, Yiming Wu, Hector Yuen, Utku Diril, Dmytro Dzhulgakov, Kim M. Hazelwood, Bill Jia, Yangqing Jia, Lin Qiao, Vijay Rao, Nadav Rotem, Sungjoo Yoo, and Mikhail Smelyanskiy. 2018. Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications. *CoRR* abs/1811.09886 (2018).

[70] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350* (2021).

[71] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. 2018. Plasticine: A Reconfigurable Accelerator for Parallel Patterns. *IEEE Micro* 38, 3 (2018), 20–31. https://doi.org/10.1109/MM.2018.032271058

[72] Andrew Putnam. 2017. FPGAs in the datacenter: Combining the worlds of hardware and software development. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*. 5–5. https://doi.org/10.1145/3060403.3066860

[73] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 13–24. https://doi.org/10.1109/MM.2015.42

[74] Siyang Qin, Alessandro Bissacco, Michalis Raptis, Yasuhisa Fujii, and Ying Xiao. 2019. Towards Unconstrained End-to-End Text Spotting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[75] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16)*. 26–35. https://doi.org/10.1145/2847263.2847265

[76] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[77] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines *(PLDI '13)*. Association for Computing Machinery, New York, NY, USA, 519–530. https://doi.org/10.1145/2491956.2462176

[78] Parthasarathy Ranganathan, Daniel Stodolsky, Jeff Calow, Jeremy Dorfman, Marisabel Guevara, Clinton Wills Smullen IV, Aki Kuusela, Raghu Balasubramanian, Sandeep Bhatia, Prakash Chauhan, Anna Cheung, In Suk Chong, Niranjani Dasharathi, Jia Feng, Brian Fosco, Samuel Foss, Ben Gelb, Sara J. Gwin, Yoshiaki Hase, Da-ke He, C. Richard Ho, Roy W. Huffman Jr., Elisha Indupalli, Indira Jayaram, Poonacha Kongetira, Cho Mon Kyaw, Aaron Laursen, Yuan Li, Fong Lou, Kyle A. Lucke, JP Maaninen, Ramon Macias, Maire Mahony, David Alexander Munday, Srikanth Muroor, Narayana Penukonda, Eric Perkins-Argueta, Devin Persaud, Alex Ramirez, Ville-Mikko Rautio, Yolanda Ripley, Amir Salek, Sathish Sekar, Sergey N. Sokolov, Rob Springer, Don Stark, Mercedes Tan, Mark S. Wachsler, Andrew C. Walton, David A. Wickeraad, Alvin Wijaya, and Hon Kwan Wu. 2021. Warehouse-scale video acceleration: co-design and deployment in the wild. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 600–615. https://doi.org/10.1145/3445814.3446723

[79] Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2017. A case for efficient accelerator design space exploration via Bayesian optimization. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6. https://doi.org/10.1109/ISLPED.2017.8009208

[80] Jared Roesch, Steven Lyubomirsky, Marisa Kirisame, Logan Weber, Josh Pollock, Luis Vega, Ziheng Jiang, Tianqi Chen, Thierry Moreau, and Zachary Tatlock. 2019. Relay: A High-Level Compiler for Deep Learning. arXiv:1904.08368 [cs.LG]

[81] Jason Sanders and Edward Kandrot. 2010. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional.

[82] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

[83] Zachary Shahan. 2020. Tesla Autopilot Innovation Comes From Team Of 300 Jedi Engineers — Interview With Elon Musk. https://web.archive.org/web/20210430195722/https://cleantechnica.com/2020/08/15/tesla-autopilot-innovation-comes-from-team-of-300-jedi-engineers-interview-with-elon-musk/

[84] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. 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, 14–27. https://doi.org/10.1145/3352460.3358302

[85] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783720

[86] Zhan Shi, Chirag Sakhuja, Milad Hashemi, Kevin Swersky, and Calvin Lin. 2020. Learned Hardware/Software Co-Design of Neural Accelerators. arXiv:2010.02075 [cs.LG]

[87] Ryan Smith. 2020. NVIDIA Ampere Unleashed: NVIDIA Announces New GPU Architecture, A100 GPU, and Accelerator. https://www.anandtech.com/show/15801/nvidia-announces-ampere-architecture-and-a100-products Accessed: 2021-08-09.

[88] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-Optimized OpenCL-Based FPGA Accelerator for Large-Scale Convolutional Neural Networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '16)*. Association for Computing Machinery, New York, NY, USA, 16–25. https://doi.org/10.1145/2847263.2847276

[89] Supermicro. 2021. Data Centers & the Environment, on the state of the green datacenter. https://www.supermicro.com/en/white-paper/datacenter-report

[90] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2820–2828. https://openreview.net/forum?id=rmAwMJQedpH

[91] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning*. 6105–6114.

[92] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[93] Rangharajan Venkatesan, Yakun Sophia Shao, Miaorong Wang, Jason Clemons, Steve Dai, Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Yanqing Zhang, Brian Zimmer, William J. Dally, Joel Emer, Stephen W. Keckler, and Brucek Khailany. 2019. MAGNet: A Modular Accelerator Generator for Neural Networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942127

[94] Ashish Vulimiri, P Godfrey, Sri Varsha Gorge, Zitian Liu, and Scott Shenker. 2013. A cost-benefit analysis of low latency via added utilization. *arXiv preprint arXiv:1306.3534* (2013).

[95] Xuechao Wei, Yun Liang, and Jason Cong. 2019. Overcoming Data Transfer Bottlenecks in FPGA-Based DNN Accelerators via Layer Conscious Memory Management. In *Proceedings of the 56th Annual Design Automation Conference 2019* (Las Vegas, NV, USA) *(DAC '19)*. Association for Computing Machinery, New York, NY, USA, Article 125, 6 pages. https://doi.org/10.1145/3316781.3317875

[96] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3061639.3062207

[97] Zhe Wu, Curtis Yu, and Harsha V. Madhyastha. 2015. CosTLO: Cost-Effective Redundancy for Lower Latency Variance on Cloud Storage Services. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 543–557. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/wu

[98] William A. Wulf and Sally A. McKee. 1995. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH Computer Architecture News* 23, 1 (1995), 20–24.

[99] Shaolin Xie, Scott Davidson, Ikuo Magaki, Moein Khazraee, Luis Vega, Lu Zhang, and Michael B Taylor. 2018. Extreme datacenter specialization for planet-scale computing: Asic clouds. *ACM SIGOPS Operating Systems Review* 52, 1 (2018), 96–108.

[100] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Weiwen Jiang, Liangzhen Lai, Yiyu Shi, Tushar Krishna, and Vikas Chandra. 2020. Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference* (Virtual Event, USA) *(DAC '20)*. IEEE Press, Article 163, 6 pages.

[101] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, Christos Kozyrakis, and Mark Horowitz. 2020. Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Lausanne, Switzerland).

[102] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).

[103] Amir Yazdanbakhsh, Christof Angermueller, Berkin Akin, Yanqi Zhou, Albin Jones, Milad Hashemi, Kevin Swersky, Satrajit Chatterjee, Ravi Narayanaswami, and James Laudon. 2021. Apollo: Transferable Architecture Exploration. arXiv:2102.01723 [cs.LG]

[104] Amir Yazdanbakhsh, Kiran Seshadri, Berkin Akin, James Laudon, and Ravi Narayanaswami. 2021. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. arXiv:2102.10423 [cs.LG]

[105] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '15)*. Association for Computing Machinery, New York, NY, USA, 161–170. https://doi.org/10.1145/2684746.2689060

[106] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. 2019. Caffeine: Toward Uniformed Representation and Acceleration for Deep Convolutional Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 11 (2019), 2072–2085. https://doi.org/10.1109/TCAD.2017.2785257

[107] Dan Zhang, Xiaoyu Ma, Michael Thomson, and Derek Chiou. 2018. Minnow: Lightweight offload engines for worklist management and worklist-directed prefetching. *ACM SIGPLAN Notices* 53, 2 (2018), 593–607. https://doi.org/10.1145/3296957.3173197

[108] Xinyi Zhang, Weiwen Jiang, Yiyu Shi, and Jingtong Hu. 2019. When Neural Architecture Search Meets Hardware Implementation: from Hardware Awareness to Co-Design. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 25–30. https://doi.org/10.1109/ISVLSI.2019.00014

[109] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2018. DNNBuilder: an Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1145/3240765.3240801

[110] Xiaofan Zhang, Hanchen Ye, Junsong Wang, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2020. DNNExplorer: A Framework for Modeling and Exploring a Novel Paradigm of FPGA-Based DNN Accelerator. In *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD '20)*. Article 61, 9 pages. https://doi.org/10.1145/3400302.3415609

[111] Jie Zhao and Peng Di. 2020. Optimizing the Memory Hierarchy by Compositing Automatic Transformations on Computations and Data. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 427–441. https://doi.org/10.1109/MICRO50266.2020.00044

[112] Yanqi Zhou, Xuanyi Dong, Berkin Akin, Mingxing Tan, Daiyi Peng, Tianjian Meng, Amir Yazdanbakhsh, Da Huang, Ravi Narayanaswami, and James Laudon. 2021. Rethinking Co-design of Neural Architectures and Hardware Accelerators. arXiv:2102.08619 [cs.LG]

[113] Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter Ma, Qiumin Xu, Hanxiao Liu, Phitchaya Phothilimtha, Shen Wang, Anna Goldie, Azalia Mirhoseini, and James Laudon. 2020. Transferable Graph Optimizers for ML Compilers. In *Advances in Neural Information Processing Systems*, Vol. 33. 13844–13855.