Hyperscale Hardware Optimized Neural Architecture Search

Sheng Li
Google
USA
lsheng@google.com

Garrett Andersen
Google
USA
garrettaxel@google.com

Tao Chen
Google
USA
taoc@google.com

Liqun Cheng
Google
USA
liquncheng@google.com

Julian Grady
Google
USA
jpg@google.com

Da Huang
Google
USA
dahua@google.com

Quoc V. Le
Google
USA
qvl@google.com

Andrew Li
Google
USA
andrewyli@google.com

Xin Li
Google
USA
xinlix@google.com

Yang Li
Google
USA
yangliyl@google.com

Chen Liang
Google
USA
crazydonkey@google.com

Yifeng Lu
Google
USA
yifenglu@google.com

Yun Ni
Google
USA
yunn@google.com

Ruoming Pang*
Apple
USA
ruoming@gmail.com

Mingxing Tan*
Waymo
USA
tanmingxing@waymo.com

Martin Wicke
Google
USA
wicke@google.com

Gang Wu
Google
USA
wgang@google.com

Shengqi Zhu
Google
USA
sqzhu@google.com

Parthasarathy Ranganathan
Google
USA
parthas@google.com

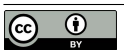
Norman P. Jouppi
Google
USA
jouppi@google.com

ABSTRACT

Recent advances in machine learning have leveraged dramatic increases in computational power, a trend expected to continue in the future. This paper introduces the first Hyperscale Hardware Optimized Neural Architecture Search (H_2O-NAS) to automatically design accurate and performant machine learning models tailored to the underlying hardware architecture. H_2O-NAS consists of three key components: a new massively parallel "one-shot" search algorithm with intelligent weight sharing, which can scale to search spaces of $O(10^{280})$ and handle large volumes of production traffic; hardware-optimized search spaces for diverse ML models on heterogeneous hardware; and a novel two-phase hybrid performance model and a multi-objective reward function optimized for large-scale deployments.

H_2O-NAS has been implemented around state-of-the-art machine learning models (e.g. convolutional models, vision transformers, and deep learning recommendation models) and deployed at zettaflop scale in production. Our results demonstrate significant improvements in performance (22% ~ 56%) and energy efficiency (17% ~ 25%) at same or better quality. Our solution is designed for large-scale deployment, streamlining privacy and security processes and reducing manual overhead. This facilitates a smooth and automated transition from research to production.

*Work done while at Google



ASPLOS '23, March 25–29, 2023, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9918-0/23/03.
https://doi.org/10.1145/3582016.3582049

CCS CONCEPTS

• **Computer systems organization → Neural networks**; • **Computing methodologies → Machine learning**.

KEYWORDS

Hyperscale Hardware, Accelerator, TPU, GPU, Machine Learning, Deep Learning, Neural Architecture Search, Pareto Optimization

ACM Reference Format:
Sheng Li, Garrett Andersen, Tao Chen, Liqun Cheng, Julian Grady, Da Huang, Quoc V. Le, Andrew Li, Xin Li, Yang Li, Chen Liang, Yifeng Lu, Yun Ni, Ruoming Pang, Mingxing Tan, Martin Wicke, Gang Wu, Shengqi Zhu, Parthasarathy Ranganathan, and Norman P. Jouppi. 2023. Hyperscale Hardware Optimized Neural Architecture Search. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3582016.3582049

1 INTRODUCTION

Datacenter machine learning (ML) accelerators (e.g. GPUs and TPUs) have been fueling rapid progress in ML research and applications. As larger and more complex ML models are demanding even faster and cheaper compute [39, 52], it has become increasingly important to tap into the full potential of datacenter ML hardware. However, ML model architectures are currently not optimized for the heterogeneity and specialization in hardware accelerators, stranding signficant performance benefits.

Concurrently, neural architecture search (NAS) [16, 31, 44, 59, 60] has shown significant success in automatically designing ML model architectures that rival the best human-designed models. Recent studies on multi-objective and hardware-aware NAS [6, 7, 15, 17, 19, 22, 23, 29, 30, 32, 34, 50, 51, 53, 55, 58] have further shown promising results in designing accurate and performant ML models in research settings. However, significant challenges remain, preventing NAS from being deployed in production environments with heterogeneous hardware accelerators, diverse ML model populations, and evolving deployment constraints.

To overcome these challenges, we present ***H**yperscale **H**ardware* **O**ptimized **N**eural **A**rchitecture **S**earch (H_2O-NAS). Our approach improves on traditional NAS in the following ways:

- We design a massively parallel single-step reinforcement learning (RL) search algorithm to learn policies and model weights simultaneously from real-time production traffic, eliminating the need for lengthy retraining and fine-tuning for model deployment.
- We co-design the RL-based NAS search algorithm with hardware optimized search spaces for various ML domains on heterogeneous hardware accelerators and introduce a first-of-a-kind search space with weight-sharing for Deep Learning Recommendation Models (DLRMs).
- We propose a simple yet effective multiple objective reward function and a novel two-phase hybrid performance model for accurate and cost-effective performance prediction for production-scale NAS.
- We open-source two new model families: CoAtNet-H (a vision transformer, or VIT) and EfficientNet-H (a convolutional neural network, or CNN), which demonstrate up to more than 1.8X performance and energy efficiency gains over state-of-the-art research models with neutral model accuracy.
- We present our deployment of H_2O-NAS at zettaflop scale for Google production workloads, without manual interventions, resulting in significant performance (22% ∼ 56%) and energy efficiency (17% ∼ 25%) gains at scale with neutral or better model accuracy/quality.

Section 2 introduces a taxonomy of the state-of-the-art NAS and the challenges in hyperscale production environments. We provide an overview of H_2O-NAS in Section 3. We then describe the design details of the three pillars of H_2O-NAS: search algorithms with the in-memory data pipeline in Section 4, search spaces with weight-sharing super-networks in Section 5, and multidimensional search objectives in Section 6. In Section 7, we discuss the results of H_2O-NAS on both the state-of-the-art ML models and deployment at scale. Related work is discussed in Section 8.

2 NEURAL ARCHITECTURE SEARCH AND THE CHALLENGES OF HYPERSCALE

Designing highly accurate and performant ML models is a multi-disciplinary challenge that necessitates a deep understanding of machine learning algorithms, computer architecture, and system design. Neural Architecture Search (NAS) approaches this challenge as an optimization problem. NAS conducts an automated multi-objective design space exploration with a given high level design constraints to find Pareto-optimized ML models. This section provides an introduction to the design of state-of-the-art NAS, and outlines the challenges in designing, optimizing, and deploying NAS at hyperscale.

2.1 Taxonomy of the State-of-the-Art NAS

A significant body of research has been dedicated to understanding the design of NAS. At a high level, the design of NAS can be characterized by four overarching dimensions: 1) search strategies, 2) search algorithms, 3) search spaces, and 4) search objectives. The center greyscale area in Figure 1 illustrates a traditional/generic one-shot NAS with its search algorithm, search space, and search objectives.

Search Strategies refer to the general method of how NAS explores its model architecture search space. There are two major strategies: *multi-trial* and *one-shot*. Multi-trial NAS [50, 51] explores the search space by sampling and evaluating different model architectures in the search space through separate and independent trials. While multi-trial NAS is straightforward to implement, it can be cost-prohibitive if the individual trials are large in scale. In contrast, one-shot NAS [4] constructs a single super-network that encompasses all possible model architectures in the search space as sub-graphs. The different candidates are then sampled and evaluated as different parts of the super-network in each training/searching step. By designing the super-network such that weights and graph structure are effectively shared between sub-graphs, one-shot NAS can be made highly efficient.

Search algorithms govern the progress of NAS, by determining the model architecture candidates to be considered in each step based on rewards consisting of performance and quality metrics

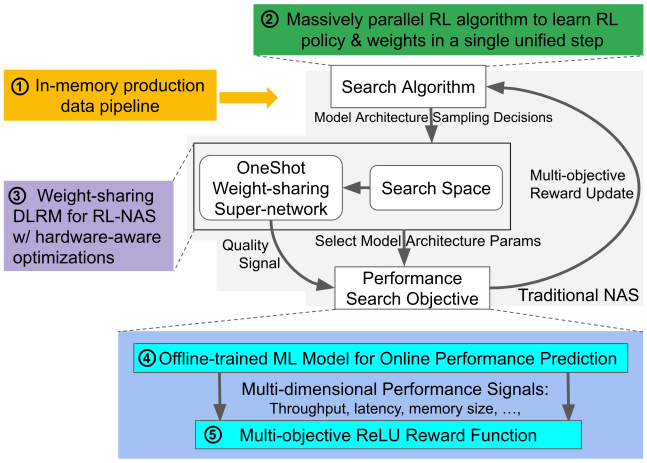


Figure 1: System Design of Hyperscale Hardware Optimized NAS (H_2O-NAS). Our contributions are highlighted in the colored areas, while the traditional one-shot NAS is shown in the center area in greyscale.

from the architecture candidates evaluated the current step. An effective search algorithm must quickly find the Pareto-front with respect to the search objectives by sampling a small portion of the candidates in the search space. There are three primary search algorithms for NAS: *reinforcement learning (RL) based search algorithms*, *gradient-based search algorithms*, and *evolution based algorithms*. RL-based search algorithms are highly flexible and can work with all kinds of rewards. Gradient-based search algorithms eliminates the need for an RL controllers by making the reward differentiable with a softmax layer over all model candidates. Evolution-based search algorithms allow for model architecture candidates to mutate and evolve to find the optimal model architectures, similar to gene mutations in a biological environment. However, evolution based search algorithms cannot be applied to one-shot NAS, because they require the rewards to be comparable across steps. In contrast, for one-shot NAS, the rewards depend on the amount of data the model has already trained on, and thus, are only comparable within each training step.

Search spaces provide the fundamental components for constructing the desired ML models. Search spaces must be co-designed with search algorithms. For example, if one-shot is chosen as the search strategy, a weight-sharing super-network is required. However, such a super-network cannot be easily constructed for an arbitrary search space. Inclusion of all the options in an enormous search space usually leads to a suboptimal outcome, as the search space becomes too large for even state-of-the-art search algorithms to explore [4, 43]. Therefore, in order to find architectures that perform well on specific hardware, the search spaces must be augmented to include hardware-friendly operations. This is particularly important as search spaces that are tailored for hardware are able to take advantage of the unique features and optimizations provided by that hardware, which leads to significant improvements in performance and efficiency.

Search objectives determine the optimization targets, which in turn determine the Pareto-optimization of the ML models. NAS has evolved from single-objective optimization [59] on quality (model accuracy) to multi-objective optimization that considers both quality and performance. Combining quality and performance objectives is a major challenges for all NAS systems. Reward functions, which combine quality and performance objectives, have evolved from the multiplicative form [29, 50] to the more stable additive form [4]. While the quality objective is always some form of model accuracy, performance objectives vary widely and include model size (as a proxy for hardware memory usage) [7], model FLOPs (as a proxy of hardware execution performance) [51], and true hardware performance (since performance proxies have been shown to have poor accuracy and correlation [7, 29]).

The center greyscale area in Figure 1 illustrates an example of traditional and generic one-shot NAS design. The search begins by constructing a one-shot super-network that represents the search space, consisting of various categorical parameters to govern sub-network selection. Iteratively, model architecture candidates are sampled, and their corresponding sub-networks are trained to generate quality signals. Additionally, the categorical parameters, representing the evaluated sub-networks, are also used to produce performance signals. By merging the quality and performance signals, a multi-objective reward is formed, which is then utilized by a reinforcement learning (RL) controller to adjust the probabilities of the categorical parameters. This process continues until the end of the search, at which point the parameters with the highest probability are used to construct the optimal neural architecture.

2.2 Challenges of Hyperscale Hardware

Despite the promising results from academic research, significant challenges hinder the use of NAS to benefit production-grade ML models on hyperscale hardware accelerators.

Challenge of Deployment: Dynamic and heterogeneous ML production environments impose significant challenges when deploying NAS. Deploying new model architectures into production systems requires additional optimizations, including lengthy retraining and fine-tuning, to bridge the gaps between the training datasets and live production data. Moreover, production environments are subject to various constraints, including model size, training and serving costs, end-to-end service latency, and more. These constraints are essential for the deployability of a ML model, yet they are difficult to consider during model research and development. Due to the difficulty in imposing these constraints directly on search spaces, NAS has been unable to directly improve the quality or performance of production models. Models discovered using NAS usually need signficant manual intervention to meet production constraints, often losing quality or efficiency in this complex process.

Challenge of Efficiency: NAS can consume large amounts of hardware resources. This is especially true when the optimization targets are state-of-the-art giant production models training on tens of thousands of TPUs or GPUs [35, 42]. Efficiency for NAS at hyperscale is multifaceted, including *NAS efficiency* (total computation needed for the neural architecture search itself), and *ML model efficiency* (performance/efficiency of the NAS optimized models on

target hardware). Datacenter accelerators such as TPUs and GPUs have complex and sometimes eclectic hardware, often requiring non-obvious tuning for peak performance. In particular, their performance behavior is often highly non-linear, with many interacting bottlenecks exposing performance cliffs that are difficult to predict and optimize around. Therefore, improving *ML model efficiency* requires sophisticated search space designs to ensure the final models are composed with hardware-friendly operations/layers and hardware-specialized optimizations that allow them to make full use of the available hardware.

Challenge of Scale: With hyperscale ML accelerators and models, NAS must be designed to leverage the unprecedented amount of computation resources. Many techniques that work effectively in a smaller scale research environments often fail in production environments where the scale is much larger. For instance, obtaining accurate and fast performance search objectives directly from hardware works well in a (smaller and stabler) research setting but becomes unmanageable for continuous NAS optimizations on giant ML models running on hundreds of ML accelerators. Additionally, to scale with the number of tasks, search algorithms must be specifically designed to make use of the massive parallelism available in hyperscale accelerators.

3 SYSTEM DESIGN

In response to these challenges, we re-design and optimize NAS holistically for hyperscale ML accelerators, as highlighted by the colored regions in Figure 1. We follow a few key high-level design principles to tackle distinct challenges and constraints from production environments:

Design for Deployment: To avoid a lengthy journey from model research to production caused by the significant training data shift from research to production, we design H_2O-NAS to use real-time production traffic directly as training data. Because of data privacy and security regulations, production traffic cannot be persisted in non-volatile media nor examined by humans. We therefore implement a pure in-memory data pipeline to comply with these regulations, shown as ① in Figure 1. We then design a new RL search algorithm to learn both ML model weights and the architecture choices in a single step. This allows us to use production traffic without creating and curating a separate evaluation data stream, shown as ② in Figure 1. Optimizing for deployment also involves dealing with different models, hardware, and ML product launch constraints. To address this challenge, we design a new multi-objective reward function (shown as ⑤ in Figure 1). This new reward function enables our H_2O-NAS to generate models for various target hardware, optimized for training or serving, while performing a complex constrained multi-objective optimization that takes into account launch criteria such as quality, throughput, latency, and model size.

Design for Efficiency: As described in Section 2.2, the efficiency of NAS has two main aspects: *NAS efficiency* and *ML model efficiency*. We have found that only one-shot optimization can perform at the scale required. We therefore use a one-shot NAS with an RL-based search algorithm as shown in ② in Figure 1. The RL-based search algorithms use significantly fewer resources than gradient-based search algorithms, because RL-based approaches only need to activate the sub-network under consideration in each step, while gradient-based approaches have to compute gradients for all sub-networks to compute a full set of gradients.

However, designing the search space and weight-sharing super-network for RL-based NAS is complex, and there was no existing design for DLRM. We thus design the first such search space and weight-sharing super-network, shown as ③ in Figure 1. We further optimize the search spaces for DLRM, VIT, and CNN, so that these search spaces contains hardware-friendly ops and layers to compose the final models with high *ML model efficiency*.

Design for Scale: To ensure our design can support giant production models, we optimize the aforementioned new single-step RL algorithm to train policies and model weights on hyperscale ML accelerators in parallel, shown as ② in Figure 1. Next, we design an ML-driven hardware performance model using two-phase (pre-training and fine-tuning) training, shown as ④ in Figure 1. This new scalable performance model provides accurate and low-cost performance signals, improving the efficiency of architectural design space exploration by orders of magnitude.

4 HYPERSCALE HARDWARE OPTIMIZED SEARCH ALGORITHM WITH IN-MEMORY DATA PIPELINE

Search algorithms govern the neural architecture search process. It is important to co-design the search algorithm and data pipeline for maximum efficiency.

4.1 A Unified Single Step Search Algorithm

As described in Section 3, we develop an in-memory data pipeline to comply with data privacy and security regulations. Because of the large volume of production data, it is prohibitively expensive to splice production data streams into two independent and statistically stable datasets for training and validation within the in-memory pipeline. Unfortunately, all existing NAS designs [4, 31, 59, 60], regardless of their choices on search strategies, algorithms, or objectives, require two datasets – the training dataset (for learning model weights W) and the validation dataset (for learning model architecture choices α) – to prevent over-fitting and ensure generalization. Figure 2 illustrates this requirement using the TuNAS [4] algorithm as an example. This requirement stems from the nature of NAS, where training shared model weights W is similar to vanilla training to optimize training loss, while learning the choice of model architecture α is similar to model evaluation on validation loss. To prevent information leak from evaluation to training, separating training and validation sets are necessary. This is especially crucial for NAS with smaller datasets such as Imagenet [45], where the same data is re-used multiple times to train a model. Without separate datasets, NAS trains shared model weights W with the same data used for evaluating the choices of α to determine the future decisions from the RL controller, resulting in over-fitting and poor generalization.

In H_2O-NAS, We co-design the RL search algorithm to leverage production traffic and eliminate the need for separate training and validation datasets. With vast amount of production traffic data, it is feasible to use each data sample only once. Moreover, our in-memory data pipeline is designed to ensure that learning

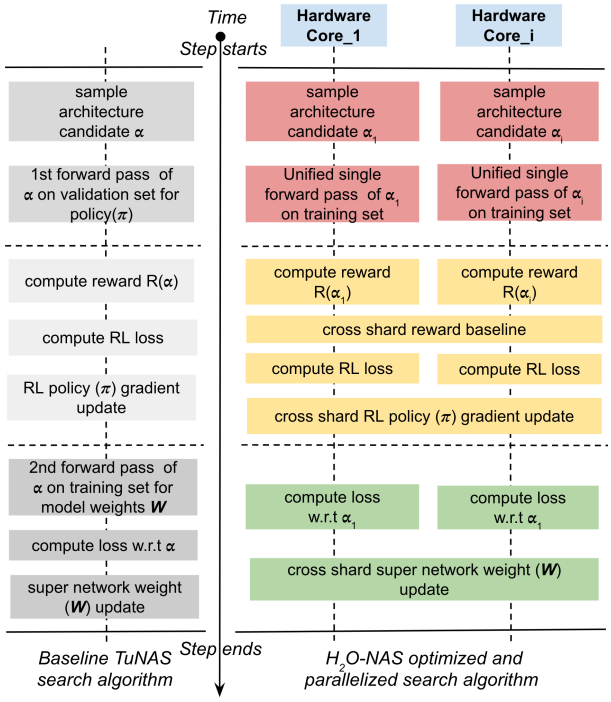


Figure 2: Baseline TuNAS Search Algorithm with Alternating Steps to Learn Policy π and Model Weights W (at Left with Greyscale) Vs. H_2O-NAS Massively Parallel Search Algorithm with Unified Single Step to Learn policy π and Model Weights W (at Right with Colors)

model architecture choices α always precede training shared model weights W in each step of NAS, as illustrated in Figure 2. Specifically, we guarantee that every incoming data is initially used by learning model architecture choices before it can be used by training model weights. As a result, H_2O-NAS always learns the choices of α using fresh data that has never been used for training shared weights W before, preventing over-fitting and ensuring generalization without the need for separate training and validation datasets.

As mentioned in Section 3, we opt for the one-shot search strategy and RL search algorithm to maximize H_2O-NAS efficiency. To the RL search algorithm, the search space consists of a set of categorical decisions, where each decision controls a different aspect of the network architecture. During a search, the RL algorithm learns a policy π, a probability distribution over a collection of independent multinomial variables. Each variable controls a decision of the search space. The RL algorithm samples from π to get the search space decisions to compose high quality architectures α. The RL algorithm also trains a set of shared neural network weights W, which are then used to compute the quality of candidate architectures sampled.

4.2 Parallelized Single Step Search Algorithm

H_2O-NAS uses a single-step search algorithm to learn both shared weights W and the policy π used to decide model architecture candidates α. To tap into the full potential of the hyperscale ML hardware, we enhance the RL search algorithm to run on hundreds of accelerators in parallel, where each search step has the following three stages as shown in Figure 2:

(1) Each accelerator (e.g., a TPU core or a GPU card) first samples a different neural architecture α_i from π, then runs a forward pass with the shared weights W to estimate the quality $Q(\alpha_i)$ of the sampled architecture using a single batch of examples from the training data.
(2) The accuracy $Q(\alpha_i)$ and performance $T(\alpha_i)$ jointly determine the reward $R(\alpha_i)$ (See Section 6 for objective and reward details) on hardware core i[1]. Then all cores conduct cross-shard updates on policy π using REINFORCE [54].
(3) In parallel with the policy π update, all cores also update the shared weights W with a cross-shard gradient update w.r.t. all the architecture candidates $Q(\alpha_i)$ on the same batch of examples from the training set.

At the end of the search, the final architecture is obtained by independently selecting the most probable value for each categorical decision in π.

Figure 2 compares H_2O-NAS with TuNAS, a state-of-the-art RL search algorithm, in detail. TuNAS alternates between learning the shared weights W on training data and learning the policy π using REINFORCE [54] on validation data. The policy π determines the choices of model architectures α. TuNAS was also not built for hyperscale deployments, and therefore lacks parallelism which could leverage hyperscale ML hardware. Our single-step learning algorithm unifies learning of policy π and shared weights W with the same dataset for the first time. Note that this unified learning approach needs access to a large amount of data. Two-step learning is still needed for small-scale research datasets such as Imagenet [45].

5 HARDWARE OPTIMIZED SEARCH SPACE

A search space includes primitive operations (e.g., layers, ops, activation functions), a weight-sharing super-network if applicable, and rules for NAS to compose ML models. It is also the key link to connect neural architectures with hardware architectures. A hardware-optimized search space contains hardware-friendly ML ops/layers to build final models that are inherently optimized for target hardware for high efficiency. Modern ML accelerators including TPUs and GPUs are massively parallel machines, where matrix/tensor units (called MXUs in TPUs [11, 25, 36] and Tensor Cores in GPUs [9, 37]) are the most important components. To fully utilize the capabilities of hardware platforms, search spaces need to maximize the parallelism within tensor units and the parallelism among tensor units and the rest of the hardware subsystems, including vector processing units (VPU), memory, and networking.

The design of a search space and the associated super-network also affect NAS efficiency significantly. Unfortunately, despite deep learning recommendation models (DLRMs) being a critical ML model domain, prior work on NAS for DLRM is scarce[2]. Therefore, we build the first DLRM search space with a carefully designed

[1]For simplicity of exposition, we assume that each model candidate uses a single core. This is not a restriction of the algorithm, as each model can use many cores.
[2]The only related work is the DRLM search space and super-network design for **gradient-based** one-shot NAS [27], which is unsuitable for the RL-based one-shot NAS that we choose for its better NAS efficiency.

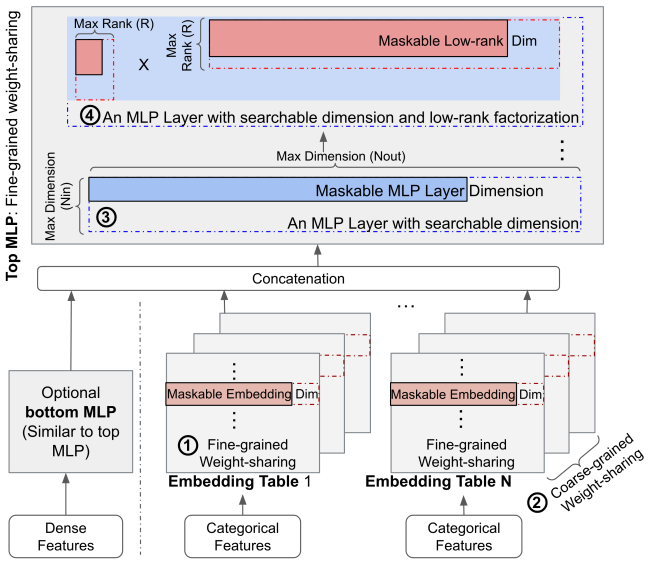


Figure 3: DLRM Model Diagram and H_2O-NAS Search Space Design with Hybrid Fine-grained and Coarse-grained Weight-sharing Super-network. It is the first design for DLRM on RL-based one-shot NAS.

super-network specifically for RL-based one-shot NAS, and optimize it for model efficiency for hardware. For CNN and VIT, we augment existing search spaces with hardware-specific optimizations.

5.1 Design of DLRM Search Space

As shown in Figure 3, modern deep learning recommendation models (DLRMs) [3, 21, 35] handle both dense (continuous) features and sparse (categorical) features describing higher-level attributes (e.g. users and products). The dense features are processed by the optional bottom MLP (multi-layer perceptron) layers, while the sparse features are first processed by the embedding layers. The processed dense and sparse features are then concatenated and processed by the top MLP layers before the final sigmoid layer produces a prediction.

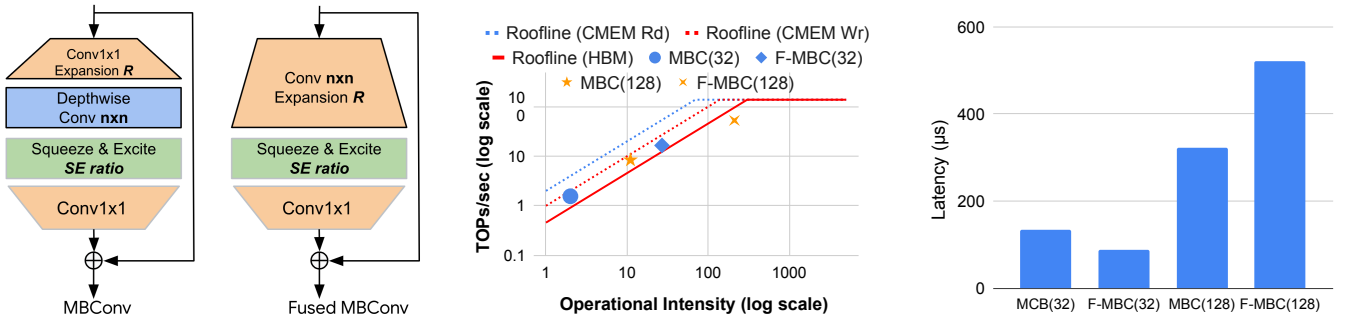
5.1.1 Search Space for MLP and Embedding Layers. Figure 3 and Table 5 (in Appendix A) show the search space designed for DLRM. For the embedding layers, we sweep both embedding width and vocabulary size for each embedding table to strike a balance between accuracy and efficiency. The larger the embedding tables are, the higher the DLRM model quality will be, at the cost of higher memory capacity and bandwidth requirements. For the MLP layers, our search space supports searching for layer width, layer depth, and low-rank matrix-factorization options. Low-rank matrix factorization in MLP layers has the potential to reduce compute load but may cause quality losses. Low-rank matrix factorization in deep learning [47] is different from that in traditional data science, because both the rank and the weights of the low-rank matrices in deep learning can be directly learned without the existence of the original full-rank matrix. However, it is challenging to find the appropriate rank to balance the quality and performance. Our search space enables searching for the best rank to reduce total compute while maintaining sufficient parallelism on all tensor dimensions to maximize parallelism within the hardware tensor units. These performance optimizations are always done together with model quality optimizations.

The search space size for jointly optimizing embedding and dense layers is $O(10^{282})$ as shown in Table 5. While MLPs run on matrix/tensor units and are commonly compute-bound, embedding layers do not run on matrix/tensor units and are memory-bound. Since embedding layers are usually distributed across ML accelerators [35], they are also network-bound. Balancing the load between embedding and MLP computing plays an important role in maximizing the parallelism across compute, memory and network, which impacts model performance significantly. The embedding layers and MLP layers are responsible for model memorization and generalization, respectively. Thus, the balance between embeddings and MLP also changes the balance between model memorization and generalization, which affects model quality significantly. With its search space supporting all dimensions of embedding and MLP layers, H_2O-NAS for DLRM can perform Pareto-optimization of quality and performance by jointly searching for embedding and MLP architectures.

5.1.2 Weight-Sharing Super-Network for DLRM. A key enabler of one-shot NAS is a super-network with many redundant weights, with different sub-networks representing all the neural architecture candidates in the search space. During a search, NAS samples and trains a different sub-network each time a candidate is evaluated. A super-network relies on weight sharing to keep the total memory size of the network in check and ensure that each path of trainable weights in the super-network gets a sufficient gradient signal [4]. However, sharing weights among model candidates also causes significant interference between different sub-networks/candidates, making the design of the weight-sharing super-network critical to the success of one-shot NAS.

Weight-sharing for super-networks can be implemented at varying granularities. Coarse-grained weight-sharing keeps the sub-networks on parallel paths independent, because all sub-network candidates must be evaluated during each search step and thus cannot share weights. Fine-grained weight-sharing [4, 49] shares weights among common blocks and kernels on critical paths among parallel sub-networks. Fine-grained weight sharing is more efficient in both memory utilization and provides more gradient updates for all weights. On the other hand, coarse-grained weight sharing causes less interference among different model candidates, at the cost of insufficient training on the weights.

Our weight sharing super-network design for DLRM shown in Figure 3 uses a hybrid of fine-grained and coarse-grained weight-sharing to balance the trade-off between NAS efficiency and search quality. To support weight-sharing in embedding tables, we create a single embedding vector with the largest possible embedding width for each row in a given embedding table, shown as ① in Figure 3. This is fine-grained weight-sharing, because smaller embedding widths are simulated by masking all but the first D embeddings (reusing the weights from candidates with larger embeddings).



(a) MBConv and F-MBConv Architectures, with searchable parameters boldened. Activation functions are also searchable but omitted for simplicity.

(b) Rooflines of MBConv (MBC) and F-MBConv (F-MBC) on TPUv4i [24], showing fused MBConvs always have better throughput (FLOPS)

(c) Latency of MBConv (MBC) and F-MBConv (F-MBC) on TPUv4i [24], showing the dependence on both throughput (FLOPS) and total computing load (total FLOPs).

Figure 4: Hardware Implications of Model Architectures, Roofline, and Latency on MBConv (MBC) and Fused MBConv (F-MBC) on TPUv4i [24]. The #s in MBC(#)/F-MBC(#) refer to the input and output depth. Input depth always equals to output depth.

When searching for an embedding table with a different vocabulary size, we use coarse-grained weight-sharing, shown as ② in Figure 3. Each vocabulary size is represented with a different embedding table to avoid harmful interactions between candidates. We use fine-grained weight-sharing for MLP layers. Our super-network creates only one matrix of weights with the largest possible input (Nin) and output (Nout) size for each MLP layer. Smaller Nin and Nout sizes are simulated by retaining only the upper left sub-matrix of weights and masking out the rest, shown as ③ in Figure 3. For MLP layers with low-rank factorization, fine-grain weight-sharing is also used by low-rank weight matrices to search for the best rank, shown as ④ in Figure 3.

5.2 Search Spaces for CNN and VIT

Table 5 shows our search spaces for CNN and VIT models. These search spaces are built atop previous designs for CNN [4, 29] and VIT [8]. The full details of CNN and VIT architectures can be found in Table 5 in Appendix A. We use the dynamically fused MBConv in both, showcasing how the search spaces support maximizing hardware parallelism across the compute and memory subsystems.

As shown in Figure 4a, an MBConv is a macro structure, consisting of an expansion layer of 1x1 convolutions, a depthwise convolution, and a projection layer of 1x1 convolutions, together with activation, batch normalization, and skip-connections. A fused MBConv (F-MBConv) combines depthwise convolutions with the expansion or projection layer as a vanilla convolution. While MBConv has less total compute (i.e., FLOPs), it also has lower operational intensity (see Figure 4b), which leads to a lower compute rate (FLOPs/s). F-MBConv has a higher operational intensity and thus higher throughput (FLOPs/s), but comes with more total compute workload. Thus, MBConv may be faster or slower than F-MBConv depending on the model architecture details. For example, as shown in Figure 4b and 4c, while fused MBConv with *32 input/output channel depth* (F-MBC(32)) has higher operational intensity and *better* latency than MBConv (MBC(32)), fused MBConv with *128 input/output channel depth* (F-MBC(128)) has higher operational intensity but *worse* latency than MBConv (MBC(128)). Moreover, MBConv and F-MBConv contribute differently to the final model accuracy. By including both MBConv and F-MBConv in the search space, our NAS can dynamically fuse MBConv at different layers to Pareto-optimize model quality and performance.

6 SEARCH OBJECTIVES FOR HYPERSCALE HARDWARE

A reward function combines the quality and performance results of sampled architecture candidates for the RL search controller to select the next batch of candidates.

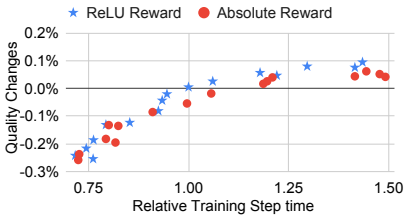
6.1 A Single-Sided Multi-Objective Reward Function

Since hyperscale optimized NAS must handle heterogeneous hardware (including different generations of TPUs and GPUs) and various model launch criteria (including throughput, latency, and memory capacity), we propose a new single-sided ReLU multi-objective reward function as shown in Equation 1.

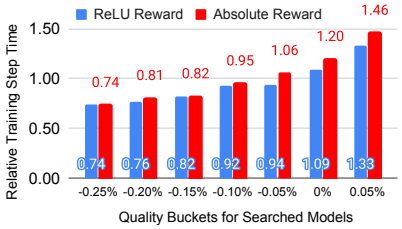
$$R(\alpha) = Q(\alpha) + \sum_i \beta_{T_i} \times ReLU(\frac{T_i(\alpha)}{T_{i0}} - 1). \quad (1)$$

where $Q(\alpha)$ indicates the quality (accuracy) of a candidate architecture α, and T_i is a performance (e.g., latency) objective. The ReLU reward function accepts multidimensional performance objectives with different weights β_{T_i}. $\beta_{T_i} < 0$ is a finite negative scalar that controls how much penalty to give when the candidate's performance deviates from the target T_{i0}. Normalizing by T_{i0} ensures that the reward is scale-invariant w.r.t. performance.

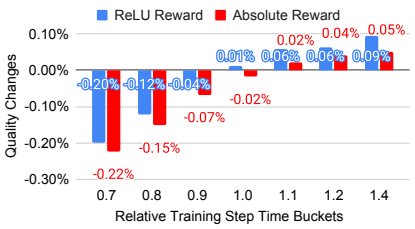
The ReLU reward function applies a linear penalty for model architecture candidates that are above the performance target, but no penalty for those at or below the performance target. This avoids unnecessarily penalizing model candidates with comparable quality but better performance. This is an important design point for NAS over multiple performance objectives: it helps us optimize both



(a) ReLU Reward Achieves Better Pareto-front of Quality and Performance Than Absolute Reward. The better Pareto-front is on the upper left.



(b) ReLU Reward Achieves Better Training Step Time. Results (lower is better) are bucketized by quality and then averaged within a bucket.



(c) ReLU Reward Achieves Better Quality. Results (higher is better) are bucketized by training step time and then averaged within a bucket.

Figure 5: New ReLU Reward Function (ReLU Reward) vs. Baseline Absolute Value Reward Function (Absolute Reward) in NAS for Production DLRMs. Training performance is the primary search objective because DLRM is training cost dominated. Model size is used as the secondary performance objective. System, model, and experiment configurations can be found in Table 2.

training/serving performance (e.g., throughput and latency) and memory capacity simultaneously for large-scale DLRM models [3]. The more constraints we have, the sparser the search space is—because there are much fewer candidates that can meet all targets simultaneously. Therefore, the RL algorithm needs to favor the overachieving models with the same quality but better performance, so as to navigate through the sparse search space quickly and effectively. The closest design to our ReLU reward function is the absolute value reward function proposed by TuNAS [4]. Equation 2 shows the TuNAS absolute value reward function with multiple performance objectives

$$R(\alpha) = Q(\alpha) + \sum_i \beta_{T_i} \times |\frac{T_i(\alpha)}{T_{i0}} - 1|, \tag{2}$$

where $|\cdot|$ denotes the absolute value function. The difference between our ReLU reward function and the TuNAS absolute value reward function is the use of the absolute value function instead of ReLU in generating individual performance reward signals. Unlike our ReLU reward function, the TuNAS absolute reward function penalizes model architecture candidates whose performance is worse *but also those whose performance is better* than the target performance. Thus, it will miss overachieving models with the same quality but better performance. While this design difference does not result in different optimization results when using only one performance objective, our ReLU reward function achieves much better results in the presence of multiple performance objectives.

Figure 5 compares our new ReLU reward function with prior state-of-the-art absolute value reward function in searching for large scale Pareto-optimized DLRM models[3]. Our ReLU reward function delivers a better Pareto-front than the prior art of absolute reward function as shown in Figure 5a. Figure 5b zooms into the Pareto-fronts by clustering the searched models into quality buckets and compares the average training step time of the searched models within each quality bucket, where our ReLU reward function achieves up to 13% better training step time than the absolute reward function with comparable quality. Similarly, Figure 5c demonstrates that the ReLU reward function achieves up to 0.4% better quality than the absolute reward function, with comparable training step time. While both the ReLU and absolute value reward functions meet the neutral serving memory target, the DLRMs found by using the ReLU reward function have, on average, 1.6% smaller model serving memory size than the models found by using the absolute reward function.

[3]The training step latency target is set to vary from 0.75X to 1.5X of training step latency of the baseline DLRMs, while the memory size (i.e., model size) target to remains the same as baseline.

6.2 Scalable ML-Driven Performance Model

As shown in Equation 1, our multi-objective NAS needs quality and performance objectives for ML model Pareto-optimization. While model accuracy is the search objective for quality, performance objectives are complicated and heterogeneous, including throughput, latency, and memory capacity for both training and serving on heterogeneous hardware of TPUs and GPUs. Hardware-agnostic performance objectives such as FLOPs have been demonstrated to be a poor performance objective for NAS because of their high correlation error (>400%) to actual performance [29].

While direct performance measurement on hardware is the best and most accurate approach for multi-trial NAS [29, 50], it is not feasible for one-shot NAS. One-shot NAS requires accurate performance signals at each search step (with latencies of 10-100 milliseconds) to compute the reward (consisting of quality and performance for the RL controller in time). However, with one-shot NAS, individual sub-networks do not exist physically to directly measure performance on hardware during search. Similarly, while performance simulators can be used for multi-trial NAS [20], their cost and latency makes their use in one-shot NAS infeasible.

An *ML-driven* approach [7] trains a performance model offline and uses it in a neural architecture search to provide performance signals for sampled architecture candidates. However, existing ML-driven approaches are too expensive to employ for the size of search space that we are using. For example, for the ML-driven approaches in [7, 53], even for a search space orders of magnitude smaller than ours, the performance models require thousands of performance measurements from hardware. For our search space sizes of $O(10^{280})$, millions of performance samples would be needed to train an adequate model for performance prediction. Moreover, a

new performance model is needed when the target hardware platform and/or search space change. Unlike the small models running on mobile devices, our target production-scale models run on hundreds of expensive TPUs/GPUs in parallel, making data gathering at this scale impractical.

6.2.1 Performance Model Using MLP Neural Networks. In order to address these challenges, we design a scalable two-stage performance model training methodology including *pre-training* and *fine-tuning* phases. The performance model is an MLP with variable layers and neurons per layer depending on target ML models to be optimized. The inputs of the performance model are the model architecture hyper-parameters as shown in Table 5. The performance model is integrated in H_2O-NAS: the RL controller injects the model architecture parameters of the sampled model architecture candidates in each search step as shown in Figure 1. The output of the performance model are performance metrics (e.g., throughput and latency) of the target ML model on the target hardware. The performance model has dual heads, to predict both training and serving performance for the same target ML model. The performance model also has a analytical objective output to predict model size. Model size prediction can be calculated directly from architecture candidates without simulation. While model size is not usually a primary performance objective by itself, it is an important objective given constraints of the hardware platforms [3]. The RL controller in H_2O-NAS chooses to use either the training or serving performance of the target ML model as the primary performance search objective depending on which one consumes more hardware resources in deployment, in conjunction with the model size as the secondary performance objective.

6.2.2 Two-Phase Training of the Performance Model. The MLP-based performance model is trained in two phases:

Pre-training: For a given search space and target hardware platforms, our system first samples millions of model architecture candidates from the search space. It then uses an in-house ML performance simulator (see Section 6.2.3 for more details) to simulate the performance of the sampled neural architecture candidates. Finally, using the simulated results as training data, the pre-training phase trains the MLP-based performance model.

Fine-tuning: While the simulation results (as training data) have reasonable fidelity, it is critical to ensure that the performance model generates accurate performance predictions w.r.t. real hardware performance. Thus, in the fine-tuning step, our system samples $O(20)$ candidates from the training data, launches these full-size ML models on target ML hardware, and collects performance measurements. The measured performance metric for training is throughput (i.e., reciprocal of training step time). The measured performance metric for serving is the serving throughput under P99 target latency over O(n) serving accelerators. Our system then uses the $O(20)$ real measurements as training data to fine-tune the performance model.

Table 1 shows the quality and two-stage training details of our MLP-based performance model. This performance model achieves very high quality w.r.t. the real hardware performance with $1\% \sim 3\%$ normalized root-mean-square error (NRMSE). The pre-training stage learns the non-convex performance behavior, and the fine-tuning stage effectively reduces the final NRMSE by 10X as shown in Table 1. The scalable two-stage training methodology makes the best use of simulations and measurements on real hardware to achieve high accuracy and low cost simultaneously for real-time performance prediction in our one-shot NAS. Table 1 also highlights the importance of using both phases. We have 14%-47% average relative error with the pre-trained model, versus a final 1-3% after fine-tuning. Reusing a single pre-trained model for all domains also leads to significant accuracy loss. It might be possible to construct a larger, universal model for all domains, and then fine-tune for each domain. This single pre-trained model approach is beyond the scope of this paper, and we leave it to future work.

Table 1: Quality and Two-stage Training Details of a 2-layer MLP Performance Model with 512 Neurons Per Layer. The model predicts DLRM training performance, as training performance prediction is the most challenging because of the hardware system scale and model sizes. System, model, and experiment configurations are in Table 2.

Search Space Size	$O(10^{282})$
Number of Pretraining Samples	1 *Million*
NRMSE of Pretrained Models on the Pretraining Samples	$0.31\% \sim 0.47\%$
Number of Finetuning Samples	20
NRMSE of Pretrained Models on Production Measurements	$14.7\% \sim 42.9\%$
NRMSE of Finetuned Models on Production Measurements	$1.05\% \sim 3.08\%$

6.2.3 Performance Simulator Details. The ML performance simulator used in generating the training data is an in-house simulator. It takes a set of inputs including: 1) hardware configurations to specify the target ML hardware architecture, 2) a TensorFlow graph [1] or a high level operation (HLO) graph [46] of the target ML model, 3) runtime statistics for the target ML model such as loop/branch counts and embedding table access counts, and 4) model architecture configurations such as the embedding layout that are not present in the model TensorFlow/HLO graph.

The ML performance simulator models hardware architecture in detail for tensor/matrix units, vector units, and memory (including on-chip memory) and network subsystems. The simulator does not model host CPUs, assuming that ML models run on accelerators completely. When the TensorFlow-graph is used as input, The simulator simulates compiler optimizations such as op/layer fusion, memory management including on-chip memory management, and model sharding and partitioning. Since HLO graphs usually contain sufficient compiler optimization annotations, the simulator need not simulate compiler optimizations when using HLO as the input. With the aforementioned inputs, the simulator walks through a TensorFlow/HLO graph, simulates run-time of each operator, and finally sums the total run-time on the critical path as the execution time of the target ML model on the target ML accelerator.

Table 2: Model Characteristics and Hardware Configurations for the Three Key Domains. Baseline CoAtNet and EfficientNet-X are model families and show a range of parameter counts (Params) and FLOPs.

	VIT	DLRM	CNN
Baseline	CoAtNet [51]	Internal	EfficientNet-X [29]
Params (Million)	25~688	O(1000)	7.6~199
FLOPs (Billion)	8.4~1060	O(100)	1.8~186
Training HW	128 TPUv4	128 TPUv4	128 TPUv4
Serving HW	1 TPUv4i	1 TPUv4i	1 TPUv4i
DominantCost	Training	Training	Training

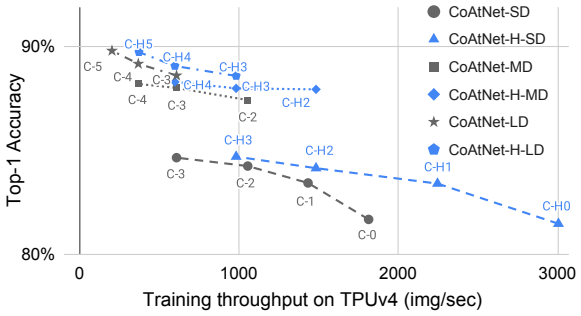


Figure 6: Pareto-front Results on Accuracy and Training Performance of the H_2O-NAS Designed CoAtNet-H and the Baseline CoAtNet, at Varying Dataset Sizes (Small, Medium, and Large). Both model families pretrain on ImageNet1K (SD), ImageNet21K (MD), and JFT-300M (LD), while evaluating on ImageNet1K. C-H-# and H-# denote different models in the CoAtNet-H family and the baseline CoAtNet family, at varying model sizes. The better Pareto-front direction is towards the upper right, representing faster and more accurate models

7 DEPLOYMENT AT GOOGLE SCALE WITH HIGH EFFICIENCY

In this section, we first show the Pareto-optimization results from H_2O-NAS on large-scale ML models in important ML domains, including vision transformers (VITs), deep learning recommendation models (DLRMs), and convolution neural networks (CNNs). We then present the results from H_2O-NAS deployment on large-scale live ML production at Google.

7.1 H_2O-NAS Pareto-Optimization for SoTA Models

Table 2 shows the model and hardware information for the three focus domains of VIT, DLRM, and CNN. All models are trained on 128 TPUv4 [11] chips and served on a single TPUv4i chip [24]. During model Pareto-optimization, H_2O-NAS always targets better performance, with neural or better quality. Moreover, because all models are training cost dominated as shown in Table 2, H_2O-NAS targets to optimize training performance, with neural or better serving performance (serving throughput under p99 target latency) and model size. Table 2 also lists the baseline state-of-the-art models for each domain. CoAtNeT [13] and EfficientNet-X [29] model families were designed by previous NAS efforts, making them very strong baselines. The internal business-critical DLRM model has also been continuously and heavily optimized.

7.1.1 Pareto-Optimization for VIT. When searching for better vision transformer (VIT) models, we use CoAtNet [13] as the baseline and the ViT search space as shown in table 5. Figure 6 shows the results of the H_2O-NAS designed CoAtNet-H model family, on the target training platform of TPUv4 [11]. Compared with the baseline CoAtNet model family, the CoAtNet-H model family improves the Pareto-front, with 1.54X better training throughput and neutral quality and serving performance. Baseline CoAtNet is a hybrid network with both convolution and transformer sections. To optimize quality and performance, H_2O-NAS increases model depth, while decreasing image resolution for the convolution section and replacing ReLU with Squared ReLU as the activation function in the transformer section. Trading off image resolution for model depth makes the model friendlier to TPUs, while other changes of the network increase the models' capacity and non-linearity to increase quality. The ablation study in Table 3 shows how the quality and performance changes with each change on the model

Table 3: Breakdowns of Model Architecture Change Impact to Training Throughput and Accuracy of the H_2O-NAS Designed CoAtNet-H Over the Baseline CoATNet. The deeper convolution change is an increase from 12 to 16 layers of the convolution part of the model, and the resolution pretraining shrink is from 224 to 160 pixels. †Top-1 accuracy is on Imagenet. §The training throughput is images/sec per chip, with per chip batch size of 64 on TPUv4.

Model	Top-1† Accuracy	#Param (Million)	FLOPs (Billion)	Training§ Throughput
CoAtNet-5 [51]	89.7%	688	1012	101
+DeeperConv	90.3%	697	1060	97
+ResShrink	88.9%	697	474	186
+SquaredReLU (CoAtNet-H5)	89.7%	697	476	186

architecture, quantitatively. Further details on the model architectures of the CoAtNet-H family can be found in its opensource code repository [40]

Detailed hardware analysis on CoAtNet-5 and CoAtNet-H5 (the largest in the respective families) in Figure 7 further reveals the reason behind the performance gains with neutral quality. Both CoAtNet-H5 and CoAtNet-5 have high operational intensity and are compute bound. While remaining compute-bound, CoAtNet-H5 is designed by H_2O-NAS to re-balance the load on compute and memory to maximize parallelism and reduce bottlenecks on TPUv4. As shown in Figure 7, the compute rate (FLOPS, i.e., FLOPs/s) of CoAtNet-H5 drops by 14%. To compensate for the drop in the compute rate, CoAtNet-H5 reduces total compute load (FLOPs) by 53%, which, together with the more hardware-friendly model architecture changes, leads to an 1.84X speedup despite the compute rate

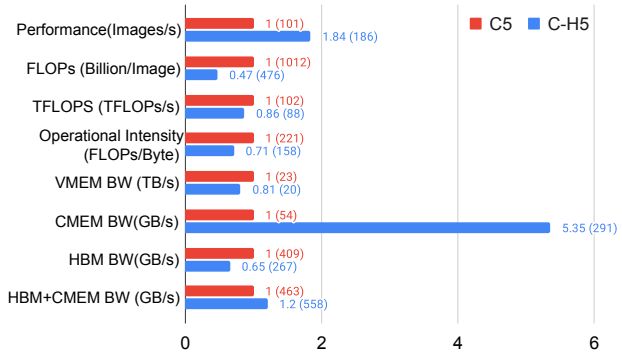


Figure 7: Training Performance Analysis for CoAtNet-H5 (C-H5) and Baseline CoAtNet-5 (C5) on TPUv4. C-H5 stats are normalized to those of C5, with raw numbers included in the parentheses after the ratios.

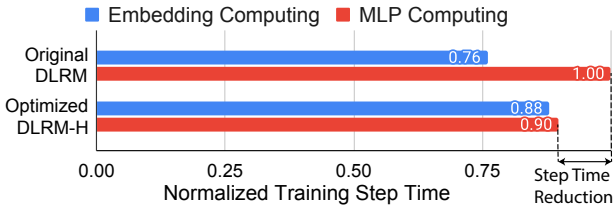


Figure 8: Training Step Time of H_2O-NAS designed DLRM-H, normalized to the original DLRM. The training step time is MAX(Embedding computing time, DNN computing time). The optimized DLRM-H has a quality gain of 0.02%.

drop. Total memory bandwidth increases by 20%, compensating for potential quality loss from the the compute load (FLOPs) drop.

The CoATNet model family is a state-of-the-art VIT model family, including advanced transformer in its model architectures. The VIT model family in our study and transformer-based NLP models have similar transformer architectures and search space with weight-sharing designs. Thus, the strong performance gains from the H_2O-NAS Pareto-optimizations on CoAtNet provide confidence in the effectiveness of the Pareto-optimizations of H_2O-NAS on transformer-based NLP models as well.

7.1.2 Pareto-Optimization for DLRM. When searching for better DLRM models, we use an internal production DLRM model and the DLRM search space as shown in Table 5. Figure 8 shows the performance of the H_2O-NAS designed DLRM-H, in comparison with the baseline DLRM. Although the performance gains over baseline DLRM are smaller than that for CoATNet, it is important to note that the baseline (business-critical) DLRM model is extensively optimized, making it a very competitive comparison. The 10% performance gain from H_2O-NAS within days is equivalent to improvements historically achieved by a team of 10+ experts over months.

Table 4: Geometric Mean Speedup of H_2O-NAS designed EfficientNet-H family over the baseline Efficientnet-X family. Geometric Mean Speedup of B5 ~ B7 models is also reported in the parentheses, as each model family has 8 models (B0 ~ B7) with B0 ~ B4 models of EfficientNet-H being the same as those of the baseline.

Training Speedup on TPUv4	Serving Speedup on TPUv4i	Serving Speedup on GPUv100
5% (14%)	6% (16%)	6% (17%)

Moreover, despite having been optimized heavily by many prior efforts including generic NAS, the original DLRM still has significant load-imbalance between embedding processing and MLP processing, as the MLP compute time is much longer than the embedding computing time. This load imbalance not only causes sub-optimal performance as shown in Figure 8 but also skews the model towards generalization without sufficient memorization, which in turn hurts quality. However, this trade-off among embedding, MLP, quality, and performance is very hard for humans to optimize but can effectively be handled by H_2O-NAS. Concretely, H_2O-NAS enables the previously impossible end-to-end Pareto-optimizations on quality and performance over both embedding layers and MLP layers, which balances embedding compute for memorization and MLP compute for generalization. The end-to-end automated optimization and load-balancing reduce the total embedding layer size and increase the total MLP layer size, improving DLRM end-to-end performance by 10+% with 0.02% better quality and neutral serving performance and memory.

7.1.3 Pareto-Optimization for CNN. When searching models over the baseline of EfficientNet-X, we use the convolutional search space as shown in table 5 in Appendix A. Table 4 demonstrates the performance improvements of the H_2O-NAS designed EfficientNet-H model family over the baseline EfficientNet-X [29] family. The performance improvements over EfficientNet-X are smaller than those of CoAtNet-H, because the baseline EfficientNet-X family has already been heavily optimized including through previous NAS [29] efforts and is much smaller in model size and total FLOPs as demonstrated in Table 2. As a result, the EfficientNet-X family is already very close to its Pareto-front in its search space.

Nonetheless, H_2O-NAS can further Pareto-optimize the relatively bigger models (B5 ~ B7) in the family to achieve about 6% family-wide average performance gains and about 15% average speedup for B5 ~ B7 on both training (on TPUv4 [11]) and serving (on TPUv4i [24] and GPUv100 [9]), with neutral quality. The smaller models in the family (B0 ~ B4) do not exhibit any changes and thus do not lead to performance gains over the baseline. Unlike the VIT/CoAtNet [13] and DLRM models which have different settings between training and serving, the CNN EfficientNet-X [51] model family has the same settings between training and serving, making its training and serving performance well correlated. The performance gains of EfficientNet-H come from the changes on the expansion factors **R** inside the dynamic fused MBConv (as shown in Figure 4) of models B5-B7 from uniformly 6 to a mixture of 4 and

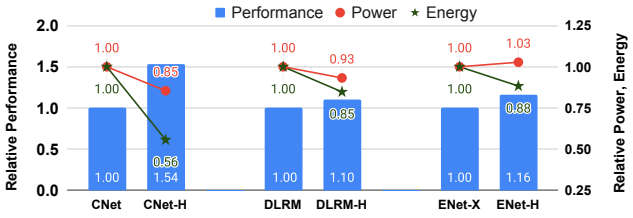


Figure 9: Performance, power and energy of H_2O-NAS designed EfficientNet-H (ENeT-H), CoAtNet-H (CNet-H), and DLRM-H. Results are normalized to respective baselines. Results are geometric mean for model families.

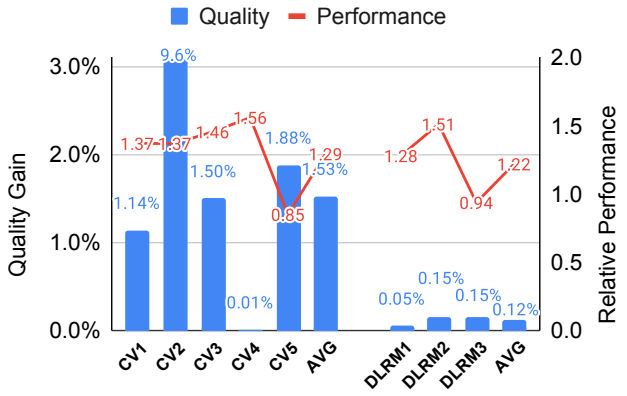


Figure 10: Quality and Performance Gains of H_2O-NAS designed Datacenter-scale Production-grade ML Models. CV# and DLRM# refer to different production scale computer vision and DLRM models. Performance gains are normalized by the baseline production scale models, while quality gains are the raw improvements as quality itself is measured by percentage.

6. Further details on the model architectures of the EfficientNet-H family can be found in its opensource code repository [41].

7.2 Energy Benefits of H_2O-NAS

To study energy implications of H_2O-NAS on designing new models, we measure the power consumption in $(Watts)$ for target accelerators running these ML models. We then compute $Energy(Joules) = ExecutionTime(seconds) \times Power(Watts)$. Figure 9 shows the power and energy results of H_2O-NAS designed models. All of the H_2O-NAS designed models have significant energy savings. Contrary to the general intuition that faster models require more power, the fast models of CoAtNet-H and DLRM-H consume much less power than their slower baselines. For example, the CoAtNet-H family is 1.54X faster in training, but still consumes 15% less power (and thus 46% less energy) than the baseline CoAtNet family. DLRM-H is 10% faster in training, but consumes 7% less power (and thus 15% less energy) than DLRM.

Detailed hardware level analysis shown in Figure 7 reveals the reason for the power efficiency gain. The 14% drop on compute rate (FLOPS, i.e., FLOPs/s) of CoAtNet-H5 improves model power efficiency significantly. Although total memory bandwidth increases by 20% to compensate for potential quality loss from the the compute load (FLOPs) drop, most of the memory bandwidth increase is inside the 128MB on-chip memory (CMEM) of the TPUv4 [11] as evidenced by the 5.3X bandwidth increase for CMEM. As a result, the total offchip traffic to HBM is actually reduced by 35%. Since the on-chip CMEM is much more power-efficient than the off-chip HBM, the increased total memory bandwidth does not cause a corresponding power increase.

Similar reasons lead to the power consumption improvement for DLRM-H. However, EfficientNet-H consumes similar power as the baseline, because both model families are mostly memory-bound, which makes the TPUv4 hardware run at low utilization with system power dominated by idle power. Thus, EfficientNet-H's better energy consumption is due to its better performance than the baseline.

These counter-intuitive results not only highlight the benefits of H_2O-NAS's capability to change ML model architectures for optimizations but also shed light on important ML optimization principles. Many believe that, to achieve high training/serving speed, ML models need to achieve the highest operational intensity possible and stay closest to the hardware peak performance. But this is unnecessary, as long as the models have reasonable operational intensity and stay compute-bound. Redesigning the model architectures to re-balance compute and memory processing according to the underlying hardware resources is more effective at improving performance and even power efficiency. Such architectural optimizations are very hard for humans, but H_2O-NAS can effectively capture the interaction between model and hardware architectures to achieve optimal performance and quality.

7.3 H_2O-NAS in Production at Scale

H_2O-NAS in production aims at *zero-touch* Pareto-optimizations on quality, performance, and energy efficiency for many production-grade ML models in diverse domains. We apply H_2O-NAS to accomplish this goal. Figure 10 shows results of H_2O-NAS for production-grade computer vision (CV) models and DLRM. Quality is always the first priority during the optimizations, which is the reason that all the optimized production models have neutral or better quality. In some cases, to achieve quality gains, H_2O-NAS designs the models to allow a reasonable performance degradation (e.g., CV5 and DLRM3 in Figure 10). All production-grade ML models use training performance as the primary performance objective, with model size and serving performance as the secondary objectives. Overall, for computer vision production scale models, H_2O-NAS improves performance by 1.29X and model quality by 2.83%. For DLRM models, H_2O-NAS improves training performance by 1.22X and model quality by 0.12% on average. All models achieve neutral or better secondary performance objectives. Products using these models have benefited significantly from the seemingly small quality gains. These higher performance models also bring significant energy savings (similar to Figure 9), with 15% and 27% energy savings at datacenter-scale from the optimized CV models and DLRM models, respectively.

H_2O-NAS at scale in production is efficient: Because the training data for the performance model is primarily from simulations, the machine hours for performance model building are mostly on CPUs and negligible in cost compared to training the target model. Then, when performing architecture search, the search cost is ~1.5X that of regular model training. After a candidate architecture has been identified, it has to be retrained without the one-shot model overhead, making the total cost of H_2O-NAS about ~2.5X of a vanilla model training. However, H_2O-NAS runs only once, exploring a space of $O(2^{280})$ model architectures, using orders of magnitude less compute and human effort than achieving the same results manually (if such human optimizations were feasible at all). Moreover, large scale downstream serving and continued research (e.g., training for feature research on DLRMs) benefits from the H_2O-NAS designed models. The total accelerator machine hours used on H_2O-NAS is $< 0.03\%$ of the total accelerator machine hours used for downstream serving or research training jobs.

8 RELATED WORK

Datacenter accelerators, including TPUs [11, 14, 24, 25, 36] and GPUs [9, 37, 38], have been fueling the rapid growth in ML by providing unprecedented computing power for both training and inference at scale. Powered by these accelerators, state-of-the-art ML models have grown ever larger. For example, computer vision models [57], language models [2, 5, 10, 18, 28], and deep learning recommendation models (DLRMs)[35] have hundreds of billions to tens of trillions of parameters and require up to 2000+ Zettaflops for training. These powerful models and hardware provide both challenges and opportunities for NAS.

Neural Architecture Search (NAS) aims to improve machine learning models with reinforcement learning [59, 60], evolutionary search [44], differentiable search [16, 31], and other methods [26, 33]. In addition to the popular convolutional models, NAS has also been used in searching for transformer [8] and vision transformer [53] models and DLRMs [27]. Recent work on efficient NAS [6, 56] have also explored the opportunity to directly use weights learned during the search without retraining, especially for mobile models. Neural architecture search for low-level primitives [48] has also shown its success in finding alternative architectures, but combining the search for high level model architectures and low level primitives remains a challenge for NAS.

Recent work in NAS has also used multi-objective search [6, 7, 15, 17, 19, 22, 23, 29, 30, 32, 34, 50, 51, 53, 55, 58] to optimize accuracy and performance. These multi-objective hardware-aware NAS optimizers use FLOPs [51], performance measured on target hardware [29, 50], or predictions from performance models [4, 7, 53] as a search objective for performance. However, prior work has not studied use of multiple performance objectives.

9 CONCLUSIONS

We presented the first hyperscale hardware optimized neural architecture search (H_2O-NAS) at scale, demonstrating an impressive zero-touch capability for Pareto-optimizations. We have shown significantly improved performance, quality, and energy efficiency for both state-of-the-art research models and live production ML models at Google, with seamless integration with datacenter-scale production deployments.

As hardware architects continue to innovate around new accelerators, they face a challenge in forecasting the models that will run on such future machine: Hardware designs must often be committed years before models will run on the machine! H_2O-NAS enables late binding of model architectures to hardware architectures. This empowers architects to focus more on optimizing hardware for peak performance, silicon area, and power constraints, while H_2O-NAS can later optimizes future model to run on the hardware. As a result, H_2O-NAS can significantly enhance the value we can derive from hardware over its lifetime. We believe that it will be a key tool in the computer architect's arsenal as we continue to codesign across hardware and software for more innovation to accelerate machine learning.

ACKNOWLEDGMENTS

We would like to acknowledge the Brain, Visual Semantic Search, Ads, Platforms, and ML system co-design & optimization teams for making H_2O-NAS a reality. We would like to acknowledge Amin Vahdat, Carrie Grimes Bostock, Danner Stodolsky, David Patterson, Jeff Dean, and Robert Hundt as well as the anonymous reviewers for their valuable feedback.

A SEARCH SPACE DETAILS FOR VIT AND CNN

Table 5 shows the hardware-optimized search spaces for DLRM (Section 5.1), CNN, and VIT. Our transformer search space and weight-sharing super-network is similar to [8, 53], with augmentation on performance-aware transformer layers such as funnel transformer [12]. Our transformer search space can be used isolation to search for pure VIT or transformer based NLP models. The CNN search space is similar to recent factorized search spaces [7, 29, 50, 51]. By combining both CNN and transformer search space, our search space supports search for hybrid vision transformer models with both convolution and transformer layers. The search dimensions in the search spaces are designed to be hardware-friendly.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Daniel Adiwardana, Minh-Thang Luong, David R. So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, and Quoc V. Le. 2020. Towards a Human-like Open-Domain Chatbot. *CoRR* abs/2001.09977 (2020). arXiv:2001.09977 https://arxiv.org/abs/2001.09977

[3] Ehsan K. Ardestani, Changkyu Kim, Seung Jae Lee, Luoshang Pan, Valmiki Rampersad, Jens Axboe, Banit Agrawal, Fuxun Yu, Ansha Yu, Trung Le, Hector Yuen, Shishir Juluri, Akshat Nanda, Manoj Wodekar, Dheevatsa Mudigere, Krishnakumar Nair, Maxim Naumov, Chris Peterson, Mikhail Smelyanskiy, and Vijay Rao. 2022. Supporting Massive DLRM Inference Through Software Defined Memory. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*. IEEE, 302–312. https://doi.org/10.1109/ICDCS54860.2022.00037

[4] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc Le. 2020. Can Weight Sharing Outperform Random

Table 5: Search space for H_2O-NAS. [2,3,4]$\mathcal{X}, \mathcal{Y}, and \mathcal{Z}$ are model dependent variables. [1,3]SE ratio or embedding table width of zero means removing the corresponding SE layer or embedding table, respectively.

Convolutional Models	**Searchable Dimensions**
Block Choices	*Block types*: MBConv, Fused MBConv, *Kernel size*: 3×3, 5×5, 7×7 *Stride*: 1, 2, 4 (Stride-2/4 are only allowed in the first layer of a stage, if chosen.) *Expansion Ratio (for MBConv and FusedMBConv)*: 1, 3, 4, 6 *Activation function*: ReLU, swish
Tensor reshaping	Space-to-depth with or without stride-N Conv2D_N×N Space to batch with or without memory-copy-reshape ops
Squeeze & Excite Ratio	0^1, 1.0, 0.5, 0.25. 0.125
Skip connections	none, identity with pool and/or Conv2D-1x1 when feature map tensors mismatch
Block depth (# of Layers)	-3, -2, -1, 0, +1, +2, +3 w.r.t. baseline depth
Block Width (Output Filters)	$[-5, +5] \times \mathcal{X}^2$, *excluding zero* w.r.t. baseline width
Initial Resolution	$224 \times 224 \sim 600 \times 600$, with total 8 choices
Search Space Size (7 blocks):	$(302400)^7 * 8 \approx O(10^{39})$
DLRM Models	**Searchable Dimensions**
Embedding	*Width*: $[-3, +3] \times \mathcal{Y}^3$, w.r.t baseline w/ a minimal increment of 8 *Vocabulary size*: 50%, 75%, 100%, 125%, 150%, 175%, 200% of baseline
DNN	*Width*: $[-5, +5]\mathcal{Z}^4$, *excluding zero* w.r.t baseline w/ a minimal increment of 8 *Low rank*: $\frac{1}{10}, \frac{2}{10}, ..., 1$ of layer width w/ a minimal increment of 8 *Depth (# of layers)*: -3, -2, -1, 0, +1, +2, +3 w.r.t. baseline depth
Search Space Size:	$7^{O(300)} * (7 \times 10 \times 10)^{O(10)} \approx O(10^{282})$
Vision Transformer Models	**Searchable Dimensions**
Self-Attention	*Hidden Size*: Multiples of 64 up to 1024 *Low rank*: $\frac{1}{10}, \frac{2}{10}, ..., 1$ of layer width w/ a minimal increment of 8
Activation function	ReLU, swish, GeLU, Square ReLU
Sequence pooling layers	w/ or w/o pooling to reduce sequence length after each block
Primer transformer options	w/ or w/o channel-wise depth convolutions after projection in self-attention
# of Layers per TFM block	-3, -2, -1, 0, +1, +2, +3 w.r.t. baseline depth
Transformer Search Space Size:	$(17920)^2 \approx O(10^8)$ (2 multi-layer TFM Blocks)
Conv Stem	*Convolutional stem*: Search with conv search space *Patch Size*: 4×4, 7×7, 8×8, 14×14, 16×16, 28×28, 32×32 *Initial Resolution*: 112 × 112 448 × 448, with total 21 choices
Hybrid ViT Search Space Size:	$17920^2 * 21 * 302400^2 * 7 \approx O(10^{21})$ with 2 TFM and 2 Convolution blocks

Architecture Search? An Investigation with TuNAS. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14323–14332. https://doi.org/10.1109/CVPR42600.2020.01433

[5] 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. In *Advances in Neural Information Processing Systems*, Vol. 33. 1877–1901. https://dl.acm.org/doi/abs/10.5555/3495724.3495883

[6] Han Cai, Chuang Gan, and Song Han. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *Proceedings of the International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.1908.09791

[7] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *Proceedings of the International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.1812.00332

[8] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. 2021. AutoFormer: Searching Transformers for Visual Recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. https://doi.org/10.48550/arXiv.2107.00651

[9] Jack Choquette, Olivier Giroux, and Denis Foley. 2018. Volta: Performance and Programmability. In *IEEE Micro*. https://doi.org/10.1109/MM.2018.022071134

[10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. https://doi.org/10.48550/ARXIV.2204.02311

[11] Google Cloud. 2022. System Architecture: TPU v4. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm#tpu_v4.

[12] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. 2021. Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. In *Advances in Neural Information Processing Systems*, Vol. 34. 6010–6022. https://dl.acm.org/doi/10.5555/3495724.3496083

[13] Zihang Dai, Hanxiao Liu, Quoc V. Le, and Mingxing Tan. 2021. CoAtNet: Marrying Convolution and Attention for All Data Sizes. In *Advances in Neural Information Processing Systems*, Vol. 34. 3965–3977. https://doi.org/10.48550/arXiv.2106.04803

[14] Jeffrey Dean. 2019. The Deep Learning Revolution and Its Implications for Computer Architecture and Chip Design. arXiv:1911.05289 [cs.LG] https://doi.org/10.48550/arXiv.1911.05289

[15] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. 2018. Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures. In *Proceedings of the European Conference on Computer Vision*. 517–531.

[16] Xuanyi Dong and Yi Yang. 2019. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1761–1770. https://doi.org/10.1109/CVPR.2019.00186

[17] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Efficient multi-objective neural architecture search via lamarckian evolution. In *Proceedings of the International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.1804.09081

[18] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. In *Proceedings of Machine Learning Research*. https://doi.org/10.48550/arXiv.2101.03961

[19] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *Proceedings of the European Conference on Computer Vision*. Springer, 544–560. https://doi.org/10.1007/978-3-030-58517-4_32

[20] Suyog Gupta and Berkin Akin. 2020. Accelerator-aware Neural Network Design using AutoML. In *On-device Intelligence Workshop, in conjunction with the 3rd SysML Conference*. https://doi.org/10.48550/arXiv.2003.02838

[21] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *Proceedings of the IEEE Symposium on High-Performance Computer Architecture*. 620–629. https://doi.org/10.1109/HPCA.2018.00059

[22] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. https://doi.org/10.1109/ICCV.2019.00140

[23] Chi-Hung Hsu, Shu-Huan Chang, Da-Cheng Juan, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Shih-Chieh Chang. 2018. MONAS: Multi-Objective Neural Architecture Search using Reinforcement Learning. *arXiv preprint arXiv:1806.10332* (2018). https://doi.org/10.48550/arXiv.1806.10332

[24] 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 *Proceedings of the IEEE/ACM International Symposium on Computer Architecture*. 1–14. https://doi.org/10.1109/ISCA52012.2021.00010

[25] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. 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, Richard C. 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, 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 IEEE/ACM International Symposium on Computer Architecture*. https://doi.org/10.1145/3079856.3080246

[26] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. 2018. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, Vol. 31. https://doi.org/10.48550/arXiv.1802.07191

[27] Ravi Krishna, Aravind Kalaiah, Bichen Wu, Maxim Naumov, Dheevatsa Mudigere, Misha Smelyanskiy, and Kurt Keutzer. 2021. Differentiable NAS Framework and Application to Ads CTR Prediction. *CoRR* abs/2110.14812 (2021). arXiv:2110.14812 https://arxiv.org/abs/2110.14812

[28] 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 *Proceedings of the International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.2006.16668

[29] Sheng Li, Mingxing Tan, Ruoming Pang, Andrew Li, Liqun Cheng, Quoc V. 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*. 8085–8095. https://doi.org/10.48550/arXiv.2102.05610

[30] Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. 2019. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9145–9153. https://doi.org/10.48550/arXiv.1903.03777

[31] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *Proceedings of the International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.1806.09055

[32] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. 2019. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 419–427. https://doi.org/10.1145/3321707.3321729

[33] Renqian Luo, Fei Tian, Tao Qin, and Tie-Yan Liu. 2018. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, Vol. 31. https://doi.org/10.48550/arXiv.1808.07233

[34] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. 2020. Fast Hardware-Aware Neural Architecture Search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 692–693. https://doi.org/10.48550/arXiv.1910.11609

[35] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie Amy Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, K. R. Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. 2022. High-performance, Distributed Training of Large-scale Deep Learning Recommendation Models. In *Proceedings of the IEEE/ACM International Symposium on Computer Architecture*. 993–1011. https://doi.org/10.48550/arXiv.2104.05158

[36] 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. In *Communications of the ACM*, Vol. 67. 67–78. https://doi.org/10.1145/3360307

[37] NVIDIA. 2020. NVIDIA A100 Tensor Core GPU Architecture. *White Paper* (2020).

[38] NVIDIA. 2022. NVIDIA H100 Tensor Core GPU Architecture. *White Paper* (2022).

[39] OpenAI. 2018. AI and Compute. https://openai.com/blog/ai-and-compute/.

[40] Opensource. 2023. CoAtNet-H. https://github.com/tensorflow/tpu/tree/master/models/official/coatnet/tpu.

[41] Opensource. 2023. EfficientNet-H. https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/tpu.

[42] David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. 2021. Carbon Emissions and Large Neural Network Training. *CoRR* abs/2104.10350 (2021). arXiv:2104.10350 https://arxiv.org/abs/2104.10350

[43] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Designing network design spaces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10428–10436. https://doi.org/10.1109/CVPR42600.2020.01044

[44] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*. https://doi.org/10.1609/aaai.v33i01.33014780

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. In *International Journal of Computer Vision*, Vol. 115. Springer, 211–252. https://dl.acm.org/doi/10.1007/s11263-015-0816-y

[46] Amit Sabne. 2020. XLA : Compiling Machine Learning for Peak Performance.

[47] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. 6655–6659. https://doi.org/10.1109/ICASSP.2013.6638949

[48] David R. So, Wojciech Manke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. 2021. Primer: Searching for Efficient Transformers for Language Modeling. In *Advances in Neural Information Processing Systems*, Vol. 34. 6010–6022. https://doi.org/10.48550/arXiv.2109.08668

[49] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. [n. d.]. Single-Path NAS: Device-Aware Efficient ConvNet Design. In *Joint Workshop on On-Device Machine Learning*

Compact Deep Neural Network Representations (ODML-CDNNR 2019). https://doi.org/10.48550/arXiv.1905.04159

[50] 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/CVF Conference on Computer Vision and Pattern Recognition*. https://doi.org/10.48550/arXiv.1807.11626

[51] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the International Conference on Machine Learning* (2019). https://doi.org/10.48550/arXiv.1905.11946

[52] Neil C. Thompson, Kristjan H. Greenewald, Keeheon Lee, and Gabriel F. Manso. 2020. The Computational Limits of Deep Learning. *CoRR* abs/2007.05558 (2020). arXiv:2007.05558 https://arxiv.org/abs/2007.05558

[53] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. HAT: Hardware-Aware Transformers for Efficient Natural Language Processing. In *Annual Conference of the Association for Computational Linguistics*. http://dx.doi.org/10.18653/v1/2020.acl-main.686

[54] R. J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, Vol. 8. 229–256. https://doi.org/10.1007/BF00992696

[55] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10734–10742. https://doi.org/10.1109/CVPR.2019.01099

[56] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. 2020. BigNAS: Scaling up Neural Architecture Search with Big Single-Stage Models. In *Proceedings of the European Conference on Computer Vision*. https://doi.org/10.48550/arXiv.2003.11142

[57] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. 2022. Scaling Vision Transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12104–12113. https://doi.org/10.48550/arXiv.2106.04560

[58] Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arık, Haonan Yu, Hairong Liu, and Greg Diamos. 2018. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912* (2018). https://doi.org/10.48550/arXiv.1806.07912

[59] Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. *Proceedings of the International Conference on Learning Representations* (2017). https://doi.org/10.48550/arXiv.1611.01578

[60] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. http://dx.doi.org/10.1109/CVPR.2018.00907

Received 2022-10-20; accepted 2023-01-19