IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 8 August 2022, accepted 30 September 2022, date of publication 6 October 2022, date of current version 17 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3212767

SURVEY

Neural Architecture Search for Transformers: A Survey

KRISHNA TEJA CHITTY-VENKATA[1], MURALI EMANI[2], VENKATRAM VISHWANATH[2], AND ARUN K. SOMANI[1], (Life Fellow, IEEE)

[1]Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, USA
[2]Argonne National Laboratory, Lemont, IL 60439, USA

Corresponding author: Krishna Teja Chitty-Venkata (krishnat@iastate.edu)

This work was supported in part by the Philip and Virginia Sproul Professorship, in part by the National Science Foundation at Iowa State University under Grant MRI 1726447 and Grant MRI 2018594, and in part by the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DEAC02-06CH11357.

ABSTRACT Transformer-based Deep Neural Network architectures have gained tremendous interest due to their effectiveness in various applications across Natural Language Processing (NLP) and Computer Vision (CV) domains. These models are the de facto choice in several language tasks, such as Sentiment Analysis and Text Summarization, replacing Long Short Term Memory (LSTM) model. Vision Transformers (ViTs) have shown better model performance than traditional Convolutional Neural Networks (CNNs) in vision applications while requiring significantly fewer parameters and training time. The design pipeline of a neural architecture for a given task and dataset is extremely challenging as it requires expertise in several interdisciplinary areas such as signal processing, image processing, optimization and allied fields. Neural Architecture Search (NAS) is a promising technique to automate the architectural design process of a Neural Network in a data-driven way using Machine Learning (ML) methods. The search method explores several architectures without requiring significant human effort, and the searched models outperform the manually built networks. In this paper, we review Neural Architecture Search techniques, targeting the Transformer model and its family of architectures such as Bidirectional Encoder Representations from Transformers (BERT) and Vision Transformers. We provide an in-depth literature review of approximately 50 state-of-the-art Neural Architecture Search methods and explore future directions in this fast-evolving class of problems.

INDEX TERMS Neural architecture search, NAS, transformers, BERT, vision transformers, multi-head self-attention, hardware-aware NAS.

I. INTRODUCTION

Deep Learning has achieved remarkable progress in the last decade due to its powerful automatic representation capability for a variety of tasks, such as Image Recognition [1], Speech Recognition [2], and Machine Translation [3]. This success is associated with network design, which is crucial to feature representation, leading to many innovative architectures such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Graph Neural Network (GNN) and Transformers.

The associate editor coordinating the review of this manuscript and approving it for publication was Frederico Guimarães.

Transformer networks [4] have attracted significant interest over the last few years due to its effectiveness in handling long-range dependencies in data, and attained state-of-the-art (SOTA) performance on several Natural Language Processing (NLP) tasks. The architecture and its family of networks rely on the Self-attention mechanism to model the global dependency in the input sequence data. The Transformer, based on Self-attention, can exhibit high scalability, model capacity and data parallel processing as opposed to its predecessor, Long short-term memory (LSTM) [3].

The original Transformer model [4] was primarily developed for NLP applications. Later, Vision Transformer (ViT) [5], a type of Transformer network, has been developed for many Computer Vision applications such as Image

classification [6], Object Detection [7], Semantic Segmentation [8]. The ViT model establishes interdependence among the patches of an image using the Self-attention mechanism. Recent studies have shown that Vision Transformers demonstrate superior performance over traditional CNNs in terms of model size scaling and training time. The variants built on top of ViT, such as DeiT [9] and ConViT [10], are able to learn visual representations better than CNNs. In this paper, we refer to the original Transformer architecture [4] for language tasks as Vanilla Transformer, and for Computer Vision applications, we use the term Vision Transformer.

Neural Architecture Search (NAS), a subset of Automatic Machine Learning (AutoML) [11], is a method to automate the design process of Neural Network architecture on a given task and dataset without significant human intervention. The NAS method is an intelligent algorithm to automatically search for an efficient neural architecture to save the researcher's manual effort and computation time. Hardware-aware Neural Architecture Search (HW-NAS) is a class of problems whose goal is to search for networks that are not only accurate on the given dataset but also hardware-efficient in terms of latency. The resulting searched models outperform manually designed networks in several aspects, such as model performance and inference latency on the actual hardware. NAS and HW-NAS have been very successful in searching for efficient models that achieve SOTA performance on many tasks such as image classification [12], object detection [13], machine translation [14], etc.

The success of NAS methods for CNNs motivated researchers to develop similar methods for the highly successful Transformer [4] and Vision Transformer [5] architectures, giving rise to a new research direction in NLP and Computer Vision research. Hence, a review paper focusing on NAS methods for Transformers and their family of architectures is essential to summarize several key contributions, including efficient search spaces and search algorithms. This is the *first paper* to provide an in-depth review of SOTA Neural Architecture Search methods targeting Vanilla Transformers, BERT model, and Vision Transformer for language, speech, and vision tasks, along with exploring future directions.

A. LIMITATIONS OF STATE-OF-THE-ART SURVEY PAPERS

In the past, several survey papers on Transformers and Vision Transformers have been published. The previous AutoML, NAS and HW-NAS survey papers emphasized the theoretical concepts of the search methods, with a greater focus on CNNs than Transformers. However, there is no dedicated review paper for Transformer-based architecture search methods. The existing surveys on Vanilla Transformers, Vision Transformers and Neural Architecture Search are outlined in Table 1. As far as we know, our work is the first dedicated review paper on Neural Architecture Search for Transformers and their family of architectures.

TABLE 1. Transformer & NAS surveys.

Transformer Surveys	Transformers [15]–[21] Vision Transformers [22]–[26]
NAS Surveys	AutoML [27]–[31], AutoDL [32] NAS Theory [33]–[38] Hardware-aware NAS [39]–[43] Reinforcement Learning NAS [44] Evolutionary Learning NAS [45], [46] Weight Sharing [47], SuperNetwork [48] Gradient Descent NAS [49] Surrogate-assisted NAS [50] Federated Learning NAS [51], [52] Generative Adversarial NAS [53], [54] Graph Neural Network NAS [55], [56] Quantum NAS [57]

B. PAPER ORGANIZATION

The detailed composition of this survey paper and classification of several Transformer NAS methods is given in Table 2. The remainder of this paper is arranged as follows: Section II briefly reviews the fundamentals of the Self-attention mechanism and Vanilla Transformer. The basics of Convolution and Vision Transformer are summarized in Section III, while Section IV presents several primitive attention and Convolution search spaces for Transformers. Section V reviews different search algorithms for Transformers, followed by Hardware-aware NAS methods in Section VI. Sections VII, VIII, and IX discuss different NAS methods for Vanilla Transformer, BERT model, and Vision Transformer, respectively. Section X briefly describes Hardware-aware NAS methods for Transformers. Section XI provides future directions to explore and Section XII concludes this survey paper.

II. SELF-ATTENTION AND TRANSFORMER MODEL

The Transformer network [4], one of the most remarkable topologies, was initially designed for NLP tasks and has outperformed the predecessor RNNs in terms of model performance. The key feature of this architecture is the modeling of global dependencies by learning relationships between elements of the input through a self-attention mechanism by pairwise correlation. RNNs recursively process the input sequence, fetching only short-term information, but the Transformers can establish the global dependencies by learning long-range relationships. The Vanilla Transformer (illustrated in Fig. 1) for a sequence-to-sequence application is built using the following basic modules: **(1)** Input and Output Embedding, **(2)** Positional Embedding, **(3)** Multi-Head Self-Attention, **(4)** Pointwise Feed-Forward Networks, **(5)** Residual Connection, and **(6)** Normalization. On a macro-level, the Transformer architecture stacks "N" identical segments of the following two units: **(1)** Encoder and **(2)** Decoder.

A. BASIC MODULES

1) INPUT AND OUTPUT EMBEDDING

Transformers cannot directly process text, hence the input is converted to embedding vectors, similar to word2vec [58].

TABLE 2. Organization of the paper and classification of transformer NAS methods.

II	Self-Attention and Transformer Model			VI	VI-B	HW-NAS Methods	
	II-A	Basic Modules				VI-B1	RL HW-NAS
		II-A1	I/p and O/p Embedding			VI-B2	Evolutionary HW-NAS
		II-A2	Positional Embedding			VI-B3	Differentiable HW-NAS
		II-A3	Self-Attention		VI-C	NAS for Pruning	
		II-A4	MHSA		VI-D	NAS for Quantization	
		II-A5	Feed Forward Network	VII	NAS for Vanilla Transformers		
	II-B	Encoder and Decoder			VII-A	Evolved Transformer	VII-B Primer
		II-B1	Encoder		VII-C	DARTSformer	VII-D AutoTrans
		II-B2	Decoder		VII-E	TextNAS	VII-F KNAS
	II-C	Family of Transformers			VII-G	SemiNAS	VII-H AutoTransformer
		II-C1	BERT		VII-I	Speech-Transformer	VII-J DARTS-Conformer
		II-C2	GPT		VII-K	Improved Conformer	VII-L BM-NAS
III	Convolution and Vision Transformer			VIII	NAS for BERT Models		
	III-A	Convolution Modules			VIII-A	AdaBERT	VIII-B NAS-BERT
		III-A1	Spatial Convolution		VIII-C	AutoBERT-Zero	VIII-D MAGIC
		III-A2	Depthwise Convolution		VIII-E	LightHuBERT	VIII-F Fast Search
		III-A3	MBConv		VIII-G	AQ-BERT	VIII-H AutoRC
	III-B	Attention for Convolutions		IX	NAS for Vision Transformers		
		III-B1	Squeeze and Excitation		IX-A	AutoFormer	IX-B ViTAS
		III-B2	SA for Images		IX-C	TF-TAS	IX-D ViT-ResNAS
	III-C	Vision Transformer			IX-E	As-ViT	IX-F S3-NAS
		III-C1	Patch Embedding		IX-G	NASformer	IX-H GLiT
		III-C2	Position Embedding		IX-I	HR-NAS	IX-J BossNAS
		III-C3	Transformer Encoder		IX-K	NASViT	IX-L BurgerFormer
	III-D	Family of Vision Transformers			IX-M	UniNet	IX-N SPViT
IV	Transformer Search Space				IX-O	Auto-regressive NAS	IX-P VTCAS
	IV-A	Primitive Elements			IX-Q	Alpha NAS	IX-R FBNetV5
	IV-B	Search Space Challenges			IX-S	T2IGAN	IX-T TNASP
	IV-C	Search Space Types		X	Hardware-aware NAS for Transformers		
		IV-C1	Cell-level Search		X-A	HAT	X-B ShiftAddNAS
		IV-C2	Layer-wise Search		X-C	LINAS	X-D LightSpeech
	IV-D	Transformer Search Spaces			X-E	LiteTransformerSearch	X-F SuperShaper
		IV-D1	MHSA-only Space		X-G	Compiler-Aware BERT	X-H AutoTinyBERT
		IV-D2	Hybrid Search Space		X-I	MobileBERT-EdgeTPU	X-J AE-BERT
V	Neural Architecture Search Methods				X-K	AutoDistill	X-L Style Transfer ViT
	V-A	Reinforcement Learning NAS		XI	Future Work		
	V-B	Evolutionary Learning NAS			XI-A	Search Space	XI-B Topologies
	V-C	One-Shot NAS			XI-C	Perf. Estimation	XI-D HW-NAS Methods
	V-D	Performance Evaluation			XI-E	Automated Sparse and Mixed Precision Search	
VI	Hardware-aware NAS				XI-F	Co-search	XI-G Applications
	VI-A	Hardware Acceleration Methods			XI-H	NAS Benchmarks	XI-I NAS Frameworks

The first step in Vanilla Transformers is to feed the input sequence to an input embedding layer to convert the word tokens into a vector format. Transformers employ a learnable parameter model so that the network learns in such a way that each word is mapped to the corresponding vector that characterizes that word.

2) POSITIONAL EMBEDDING

The Transformer is ignorant of the positional information as it does not contain any Convolution operation or Recurrent modules, and thus an additional positional vector is required to model the input word tokens. Positional Embedding, an encoding that denotes the position of words in the input sequence, is added to the embedded input vector to denote the position of each word. The pre-processed embedding step is performed before the encoder module combines input and the positional embedded vector. The Positional Embedding proposed by Vaswani et al. [4] in the original work is a d-dimensional vector of encoding information of each word based on the position index "i" in the sequence. For each

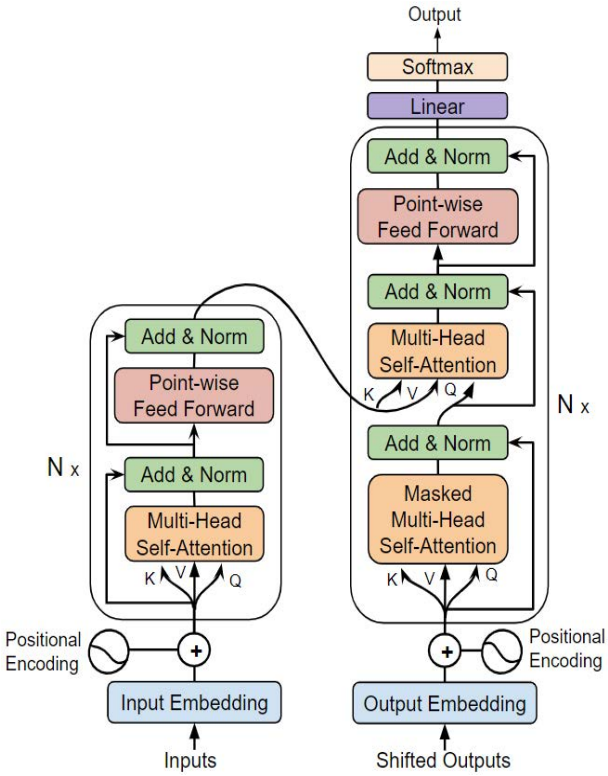


FIGURE 1. Transformer architecture [4].

position in the token embedding, the positional encoding is added as per Eq. 1.

$$PE(t) = \begin{cases} \sin\left(\frac{pos}{10000^{\frac{t}{d}}}\right), & if\ t = 2i \\ \cos\left(\frac{pos}{10000^{\frac{t}{d}}}\right), & if\ t = 2i+1 \end{cases} \quad (1)$$

where *pos* is the index of time step in the token embedding and d is the vector dimension.

3) SELF-ATTENTION

The attention mechanism is the core operation of a Transformer to establish long-range dependencies within a data sequence. The process is similar to a human's perceptual system of selectively focusing on the object's most crucial part. There are two types of attention methods: **(1)** Attention: between the input and output elements, and **(2)** Self-Attention: within the input elements only. The Self-attention mechanism in the transformer draws its inspiration from the human visual system, which manages quantification of interdependence to amplify the essential information by associating each word to other words and lessen the noisy information. The Self-attention [15] nature is attained by feeding the input sequence after Positional Embedding to three distinct Fully Connected (FC)/Linear layers to generate Query (Q), Key (K), and Value (V) matrices, as shown in Fig. 2. The individual nodes in the Fully Connected layer produces a unique weighted sum of all the inputs, thereby learning the interdependence differently. Given the matrix depiction of the Query Matrix ($Q \in \mathbb{R}^{NxD_k}$), Key Matrix ($K \in \mathbb{R}^{MxD_k}$), and Value Matrix ($V \in \mathbb{R}^{MxD_v}$), the scaled dot-product attention is given in Eq. 2.

$$\text{Self-attention } (\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right)V \quad (2)$$

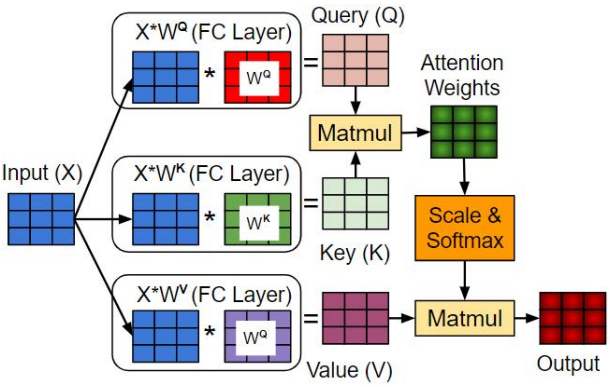


FIGURE 2. Self-attention.

The N and M notations in the dimension of Q-K-V matrices denote the length of Query, Keys, and Value. D_q, D_k, and D_v represent the output dimensions of the Query, Key, and Value FC layer, respectively. The Query matrix and Key matrix undergo an element-by-element dot product multiplication to generate a score matrix, which is further divided by $\sqrt{D_k}$ (square root of the matrix dimensions of Query and Key). The softmax function is applied to the scaled score to generate the attention weights, boost the high score elements and lessen lower score values. $\mathbf{A} = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right)$ is known as Attention matrix, where the softmax function is applied row-wise to the matrix. The dot products of queries and keys are divided by $\sqrt{D_k}$ to alleviate gradient vanishing problem of softmax function as the dot product produces high magnitude values. The score matrix and attention matrix determine the compatibility between the Query and Key to measure the importance of one word over the other words in the sequence.

4) MULTI-HEAD SELF-ATTENTION (MHSA)

Multi-Head Self-Attention is a group of "*h*" unconnected Self-attention modules, where each individual Self-attention process is called a head. The same input to the MHSA unit is replicated and passed to all parallel heads simultaneously, as depicted in Fig. 3. The input (X) to a head (head_i) is passed through three separate Fully Connected layers (W^{Q_i}, W^{K_i}, W^{V_i}) to produce one set of Query (Q_i), Key (K_i), and Value (V_i) matrices on each head, as per Eq. 3.

$$Q_i = XW^{Q_i}, K_i = XW^{K_i}, V_i = XW^{V_i} \quad (3)$$

Similar to the Self-attention operation, the output (Z_i) is computed from the respective set of Q_i, K_i, and V_i matrices, as per Eq. 4.

$$head_i = \text{Self-attention}(Q_i, K_i, V_i), \text{ i = 1,2,..h} \quad (4)$$

The outputs from all heads $\{\text{head}_1, \text{head}_2, \ldots \text{head}_i\}$ are concatenated and passed through a single FC layer of dimensions

W^O, as per Eq. 5. The output of this FC layer is the output of the Multi-head Self-Attention, which is further processed in next layer.

$$\text{MHSA}(Q, K, V) = [head_1; \ldots head_h] * W^O \tag{5}$$

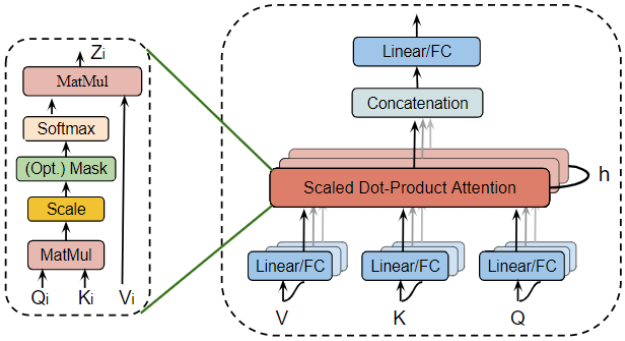


FIGURE 3. Multi-head self-attention (MHSA).

5) POINTWISE FEED FORWARD NETWORK (FFN)

The Pointwise Feed-Forward Network (FFN) unit in the Transformer is a series of two Fully Connected layers with ReLU [59] or GELU [60] activation and dropout function [61] in between the two FC layers. The output vector of Multi-Head Self-Attention is fed to the successive Pointwise FFNs. The Feed-forward layers can be treated as a point-wise correlation where each position in the input tensor is treated equally but learns the weighed sum differently with a different set of weight parameters. The input dimension of the first FC layer and output dimension of the second FC layer in the FFN module is denoted by Dim, as shown in Fig. 4. The output dimension of the first FC layer and input dimension of the second FC layer is denoted by MLP_Dim (typically the search element).

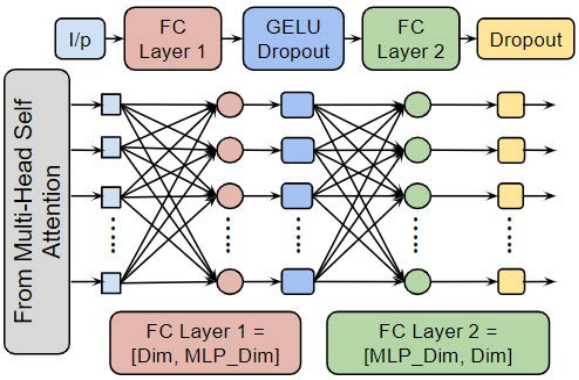


FIGURE 4. Feed forward network in transformer.

B. ENCODER AND DECODER

1) ENCODER

The encoder and decoder block is predominantly built using Multi-Head Self-Attention unit and Position-wise Feed-Forward Network. The first sub-layer is an MHSA module to allow the encoder to remain focused on the most relevant information of sequence. The MHSA output vector (Y_{MHSA}) is added to the Positional Embedding tensor (X_{PE}) through a residual connection [1] and passed through a Layer Normalization [62] layer for further processing, as given in Eq. 6.

$$\begin{aligned} Y_{MHSA} &= MHSA(X_{PE}) \\ Z_{MHSA} &= LayerNorm(Y_{MHSA} + X_{PE}) \end{aligned} \tag{6}$$

The normalized residual MHSA output vector (Z_{MHSA}) is fed to a Position-wise Feed-Forward Network to enhance the expressiveness of the Transformer. The output of the FFN module (Y_{FFN}) is added to the MHSA output vector or input of FFN (Z_{MHSA}) as a residual connection and normalized to produce output of the encoder unit i, as given in Eq. 7.

$$\begin{aligned} Y_{FFN} &= FFN(Z_{MHSA}) \\ Encoder &= LayerNorm(Y_{FFN} + Z_{MHSA}) \end{aligned} \tag{7}$$

The Transformer encoder is constructed by repeatedly stacking "N" such two sub-layered capsules of MHSA and FFN, along with residual connection and normalization.

2) DECODER

Similar to an encoder, the decoder is also composed of "N" identical blocks of a concatenation of three sub-layers. The first sub-layer is a masked Multi-Head Self-Attention, where the operation is similar to the MHSA module discussed earlier, except that the future positions in the sequence are masked. The attention mechanism is restricted in such a way that i^{th} position in the Query matrix can attend only up to $(i\text{-}1)^{th}$ positions. The future positions in the MHSA module of the encoder are not limited as the input sequence is fully available. However, the decoder cannot access the future positions because they are yet to be predicted by the network. The masking is done using a mask matrix, an element-by-element dot product of the mask matrix and the unnormalized attention matrix. This kind of masked scheme is often referred to as autoregressive or cross-attention. The second sub-layer is the cross-attention MHSA module, where an encoder-decoder mixer blends the output of the encoder and the output of the masked MHSA unit. This scheme allows the decoder to utilize previously generated data from the encoder block and focus only on relevant data in the sequence. The third sub-layer is a Pointwise FFN unit which takes the input vector from the cross-attention MHSA unit. The output vectors of all three sub-layers are coupled with residual connection and Normalization, similar to the encoder. Even though it is common for Transformer architecture to have the same number of encoder and decoder blocks, we can adjust their depth independent of each other.

C. FAMILY OF TRANSFORMER ARCHITECTURES

A wide variety of Transformer-based large self-supervised and pre-trained models are proposed for language applications such as Bidirectional Encoder Representations from Transformer (BERT) [63] and Generative Pre-trained Transformer

(GPT) [64]. These networks are designed to learn universal language representation from large unlabeled text datasets and pass the knowledge to a downstream application on labeled data. The pre-trained BERT or GPT is further fine-tuned on a specific task to avoid training from scratch [21].

1) BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS

Traditionally, the language models are designed to process input text data sequentially and in a single direction: right to left or left to right. However, the BERT model [63] predicts missing data from both previous and following words in an input sequence and is made up of only the encoder stack of the Vanilla Transformer. Hence, the bidirectional encoder name is associated with BERT. It uses masked language modeling by masking 15% of the words in input data, as shown in Fig. 5. The hyperparameters of BERT are the number of Transformer encoder layers (L), hidden size (H), and the number of Self-attention heads (h). The hidden size represents the input and output dimensions of MHSA and FFN units. BERT-base and BERT-large were the initial models developed, whose parameters are {L = 12, H = 768, h = 12}, and {L = 24, H = 1024, h = 16}, respectively. The success of the original BERT opened the way to develop efficient architectures, such as Roberta [65], Albert [66], Electra [67], etc. The NAS methods aiming BERT models are summarized in Section VIII.

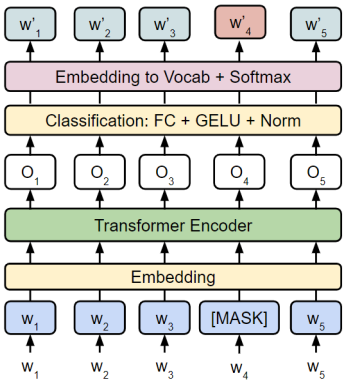


FIGURE 5. Encoder-only BERT network.

2) GENERATIVE PRE-TRAINED TRANSFORMER (GPT)

While BERT retains only the encoder of Vanilla Transformer, GPT [64] exploits its decoder. GPT can perform a wide range of language tasks with millions of parameters. The usual cross-attention connection from the encoder to the decoder in the Vanilla Transformer is removed as the encoder module is absent. Therefore, a GPT consists of only positional encoding, masked MHSA, FFN, and normalization units. As far as we know, there are no GPT-based NAS methods in the literature due to their size and computational complexity.

III. CONVOLUTIONS AND VISION TRANSFORMER

Convolutional Neural Networks (CNNs) have absolutely dominated Computer Vision applications for a long time, with successful networks from AlexNet [68], VGGNet [69], ResNet50 [1], to the State-of-the-art EfficientNet [70]. Inspired by the success of Vanilla Transformers in language and speech applications, its variant, Vision Transformer (ViT), is heavily utilized on vision tasks and has become a dominant player in this arena. In ViT and its children, the Self-attention mechanism is either used as the main operation to extract features or combined with Convolution in a hybrid manner. In our paper, a Neural Network falls under the category of Vision Transformer if a Self-attention module is used in the network and applied only on a Computer vision task such as Image Classification, Object Detection, etc. The Convolution operation is not limited to vision applications but is also explored and is now indispensable in many language and speech applications. For example, ConvBERT [71] replaces a few linear operations with Convolution to better learn local dependencies in language tasks. In this section, we first review basic Convolution operations, followed by examining Vision Transformer architecture.

A. CONVOLUTION OPERATIONS

A Convolution operation is the most basic unit in a CNN, whose primary job is to extract features and model local representations from the input image or intermediate feature map. Even though Convolutions are predominantly used in vision applications, they are significantly utilized in language and speech tasks alongside Self-attention.

1) STANDARD SPATIAL CONVOLUTION

In a Spatial Convolution, the weight matrix of size (k, k, I, O) is divided into a set of "O" filters, where each kernel has a uniform size of (k, k, I). The input kernel of size (k×k) convolves with a region equal to kernel size in the input activation/feature map to produce one pixel in the output feature map, as shown in Fig. 6. Each kernel of size (k, k, I) performs Convolution with all the channels of input activation map to generate one output pixel in a single channel of the output feature map. Pointwise Convolution is a special type of standard Convolution whose kernel size equals 1 × 1.

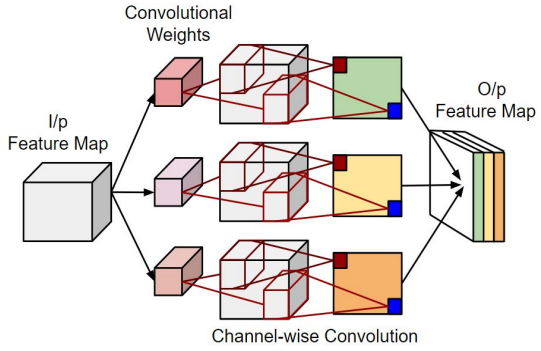


FIGURE 6. Standard spatial convolution.

2) DEPTHWISE CONVOLUTION

In a Depthwise Convolution, each individual filter is convolved with only one channel in the input feature

map/activation map to produce one output pixel in a single layer, unlike in spatial Convolution, where a filter is convolved with all channels in the feature map. The weight matrix of dimension (k, k, 1, O) is divided into a set of "O" filters, where each kernel is of size (k, k, 1). The (k×k) kernel convolves only with one channel to produce one pixel in the output feature map, as shown in Fig. 7. Therefore, the complexity of the Convolution operation is reduced by a factor of "I," where I is the input channel size. Depthwise-Separable Convolution, first introduced in Xception [72] and MobileNet [73], is a sequence of a Depthwise Convolution followed by a Pointwise Convolution to combine the outputs in the depth dimension. The Depthwise Convolution operation captures spatial correlations within individual channels, while the Pointwise operation is for cross-channel correlations.

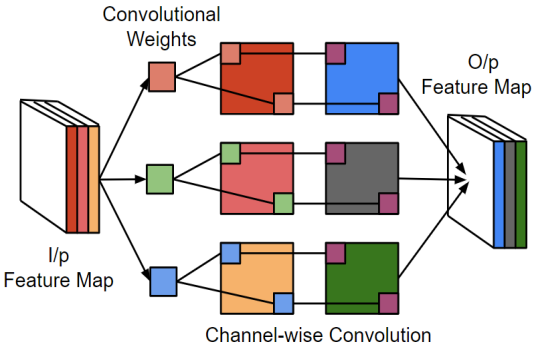


FIGURE 7. Depthwise convolution.

3) MOBILE INVERTED BOTTLENECK CONVOLUTION (MBConv)

MBConv, introduced in MobileNetV2 [74], is an encapsulation of three distinct Convolution operations. This unit is a straightforward extension to Depthwise Separable Convolution to improve model performance and efficiency over the predecessor MobileNet [73]. The input feature map to MBConv is a low-dimensional compressed tensor of Channel size "I," fed to a 1 × 1 Pointwise Convolution to enlarge the depth dimension of the feature map by a factor of expansion ratio "e." The expanded feature map, whose channel size is "I*e," is applied to a Depthwise Convolution for spatial correlation between pixels. The final operation is another 1 × 1 Pointwise Convolution to generate a low-dimension activation map for the next module. The MBConv module forms a fundamental unit in many Convolution-only and Hybrid Attention-Convolution search spaces.

B. ATTENTION FOR CONVOLUTIONS

1) SQUEEZE AND EXCITATION (SE) CONNECTION

The concept of channel attention is utilized in Squeeze and Excitation Network (SE-Net) [75], even before using the Self-attention style computation (Q-K-V style) for vision applications. SE-Net introduced feature recalibration to model interdependency between different channels of a feature map at a low computing cost, as shown in Fig. 8. The first step is to perform a regular Convolution operation (F_{tr}) on the input feature map (W) of size (H'×W'×C') to produce an output (U) of size (H×W×C). The second step involves applying a global average pooling operation ($F_{sq}(.)$) to squeeze the channel-wise elements to produce a 1D tensor of size (1 × 1 × C). The squeezed tensor, which condenses the information of each channel into a single dimension to reduce the computation cost, is fed through two FC layers to establish interdependence. The output of the second FC layer, a 1D weighted tensor, is multiplied with the unsqueezed feature map (U) to produce a weighted output feature map (Z). This way, useful features are carefully enhanced while suppressing the less important ones.

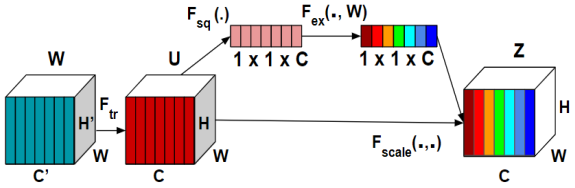


FIGURE 8. Squeeze and excitation [75].

2) SELF-ATTENTION FOR IMAGE FEATURE MAPS

The Self-attention mechanism can be used directly on the image feature maps, as shown in Fig. 9, instead of token embeddings described in the previous section. The input feature map is passed through three different 1 × 1 Convolutions to produce a set of Query, Key and Value feature maps [76], similar to Vanilla Transformers. Even though this kind of implementation is computationally very expensive, the global information in a Convolution operation is greatly enhanced.

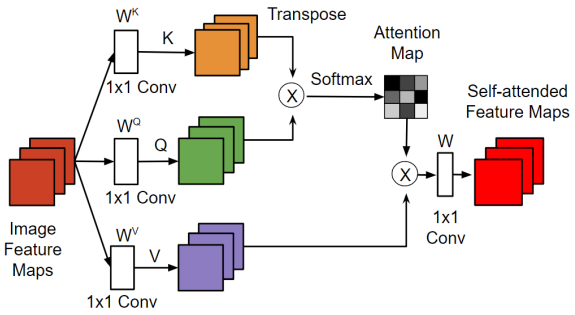


FIGURE 9. Self-attention for image feature maps.

C. VISION TRANSFORMER (VIT)

Vision Transformer (ViT) [5] paved the way for a new research direction of using Self-attention based modules for vision applications. ViTs have great flexibility and exhibit great characteristics in handling vision tasks such as a large receptive field, high model capacity, less inductive bias, grouping effect, etc. Similar to Vanilla Transformers, ViTs are also efficient in modeling long-range interdependencies and can process multi-modal data such as image, video, speech, and text. The original Vision Transformer [5] does not use any Convolutional layer, instead utilizes the encoder module

of Vanilla Transformer to perform image processing. ViT divides the input image (H, W, 3) into several patches, which are fed as input to the Transformer encoder. ViT attains better performance than traditional CNN models on many standard Image Classification benchmarks such as ImageNet [77], CIFAR-10 [78], and CIFAR-100 [78] by pre-training the network on a large-scale private dataset such as JFT-300M [79]. The initial Vision Transformer (Fig. 10) consisted of the following three important modules: **(1)** Patch Embedding, **(2)** Position Embedding, and **(3)** Transformer Encoder.

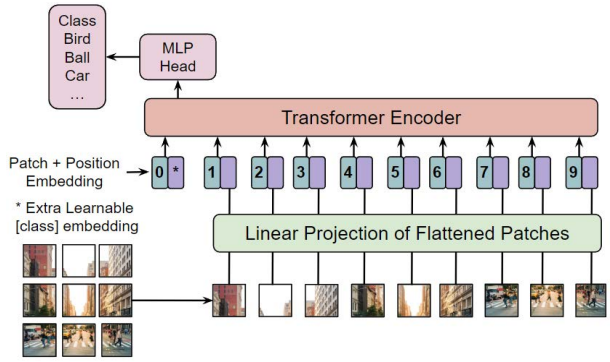


FIGURE 10. Vision transformer [5].

1) PATCH EMBEDDING

The Vanilla Transformer receives a 1D vector sequence of token embeddings as input for language modeling. However, input to a Vision Transformer is an image of dimension (H×W×3), which is transformed into a sequence of flattened 2D patches. ViT splits the input image into several non-overlapped patches of size p×p and treats them as token embeddings. The input sequence is a flattened vector (2D to 1D) of image pixel values, as illustrated in Fig. 11. Consider an ImageNet input image of dimension (B, H, W, C) = (B, 224, 224, 3), where B, H, W, and C indicate batch size, the height of the image, width of the image, and input channel size, respectively. The patch size (p) must be chosen in such a way that the height H and width W of the input image must be divisible by the patch size. The resolution of each image patch is (p, p), and effectively, the input sequence length is given as $N = H*W/P^2$. The patch height p_h (patch width p_w) is equal to image height H (width W) divided by the patch size p, as given in Eq. 8.

$$\begin{aligned} \textit{Patch Height } (p_h) &= H/p \\ \textit{Patch Width } (p_w) &= W/p \end{aligned} \tag{8}$$

The input image (B, H, W, C) = (B, (h*p_h), (w*p_w), C) is transformed into a vector of size (B, h*w, p_h*p_w*C). Therefore, for a patch size p = 16, the ImageNet image (B, 224, 224, 3) = (B, 14*16, 14*16, 3) is flattened to (B, 14*14, 16*16*3) = (B, 196, 768). The flattened 2D projected vector is passed through an FC layer to produce output activation, termed as Patch Embedding.

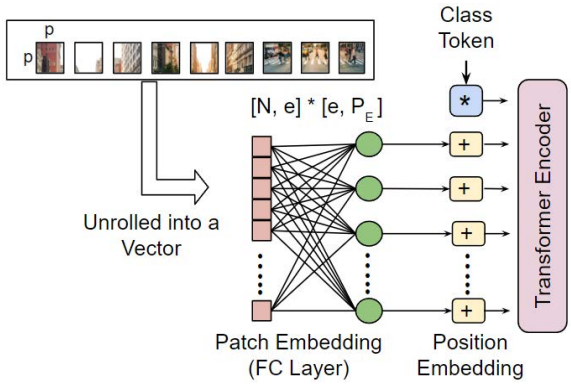


FIGURE 11. ViT patch and position embedding. p: patch size, e: Embedding dimension, P_E: Output dimension.

2) POSITION EMBEDDING

The Transformers receive a set of vector elements as input, and therefore the Neural Network learns irrespective of the input sequence order. Nevertheless, the position of an element in the vector or the sequence order is very important. This holds true for both text for language modeling, and for a pixel in an image for Computer Vision. Hence, a 1D learned parameter for positional encoding is linearly added to every patch in the flattened vector to preserve spatial information, similar to the Vanilla Transformer. ViT inserts BERT's style of learned [class] embedding to inject information about the relative or absolute position of the pixels in the patches of the sequence. Dosovitskiy et al. [5] perform 2D interpolation of pre-trained position embeddings based on pixel location in the input image. Other Vision Transformer-based networks replace non-parametric positional encoding in the original ViT with parametric encoding [80] or Fourier-based kernelized versions [81] or relative position encoding [82].

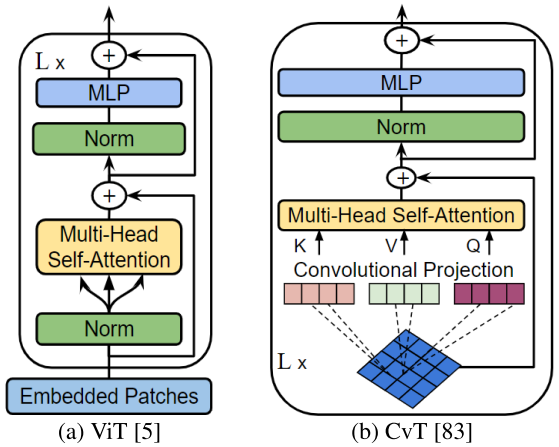


FIGURE 12. Encoder architectures.

3) TRANSFORMER ENCODER

The Vision Transformer employs only the encoder unit of Vanilla Transformer for input activation feature extraction. The Patch and Position embedded patches of split images are passed directly to the Transformer encoder for further processing. The encoder is a sequence of "L" identical units

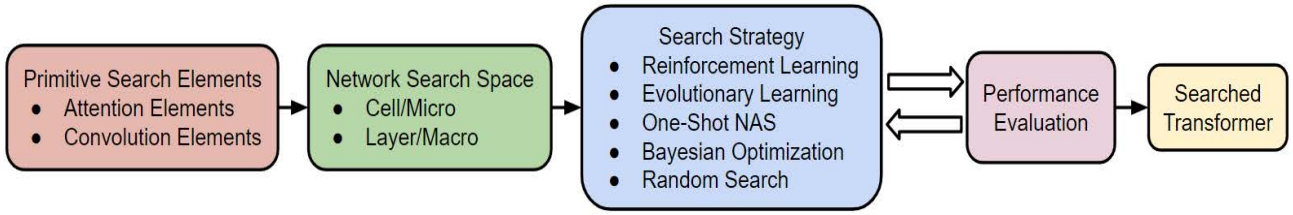


FIGURE 13. Pipeline of neural architecture search methodology.

of Multi-Head Self-Attention and Multi-Layer Perceptron (MLP) units. The MLP discussed here is same as the FNN module in the Vanilla Transformer. Unlike the encoder in the original Transformer, the normalization layer is applied before MHSA and MLP modules, as shown in Fig. 12a. The output tensor of MHSA is added to the input vector of the encoder through a residual connection. The MLP units consist of two FC layers with GeLU activation function whose output is added to the tensor before the second normalization layer through residual connection. The final layer in Vision Transformer is a simple FC layer, similar to the FC layer in a traditional CNN, to predict output probabilities. CvT [83] improves the ViT architecture by replacing the linear projection by a Convolution projection, as shown in Fig. 12b. The output of the Convolution projection is flattened to pass it as a token. DeepViT [84] introduced a Re-attention unit in the place of MHSA to resolve the attention collapse issue.

D. FAMILY OF VISION TRANSFORMERS

The remarkable success of the original Vision Transformer inspired researchers to improve it with respect to architectural structure. The advancement is accomplished either by modifying the QKV Self-attention layout or introducing a Convolution operation along with the MHSA segment in a hybrid manner to extract better local texture cues. The cascade hybrid ViTs only replace the Fully Connected layer in the initial Embedding layer [85] or linear projection [83] with a Convolution and do not disturb the rest of the architecture. On the other hand, the parallel hybrid ViT inserts a series of Convolution blocks between different MHSAs or replaces a few MHSA modules completely to explore the advantages of both modules. Some of the notable MHSA-only ViT improved architectures include Transformer in Transformer (TNT) [86], Swin [6], Twins [87], etc. TNT employs an inner and outer Transformer module to minimize the intrinsic pixel loss caused due to dividing the input image into patches. Orthogonal to efficient Vision Transformer network design, a few researchers enhanced training methodologies for higher model performance. For example, DeiT [9] solves the data-efficient issue by modifying the Transformer topology and adopting the Knowledge Distillation strategy to train ViT using a teacher-student method.

IV. TRANSFORMER SEARCH SPACE

The pivotal factors for an efficient end-to-end Neural Architecture Search (NAS) pipeline for any task, as depicted in Fig. 13, are **(i)** a set of primitive search elements, **(ii)** a well-designed search space, **(iii)** a search algorithm, and **(iv)** a performance evaluation strategy to pick the optimal network architecture from the pool of searched networks.

A. PRIMITIVE ELEMENTS

The primitive element set typically comprises manually designed Attention and Convolution units and their hyper-parameters. The discrete components' size and dimension impact the architecture's computation complexity and model performance. The search components within a Multi-head Self-Attention block are Q-K-V dimensions (W^Q, W^K, W^V matrix dimensions in Fig. 2), head number (h in Fig. 3), and hidden size. The search elements in the Feed Forward Network are MLP dimension/MLP ratio/MLP expansion ratio (MLP_Dim in Fig. 4). MLP ratio is the ratio of Input dimension (Dim in Fig. 4) and MLP dimension (MLP_Dim). The search components outside MHSA and FFN include depth or number of encoder/decoder layers (N in Fig. 1). Additionally, the Vision Transformer considers Patch Size (p) and Embedding dimension (P_E in Fig. 11). The search elements for a Spatial or Depthwise Convolution are kernel size (k), number of Channels/Filters (C). The search constituents of the Mobile Inverted Bottleneck (MBConv) block [74] are kernel size (k) and expansion ratio (e).

B. CHALLENGES IN BUILDING A SEARCH SPACE

The first step is to thoroughly curate a search space consisting of a general structure of the Transformer architecture from which the search method finds an acceptable model. The design of fundamental Neural Network operations (such as an FC or Convolution) in the search space is combinatorial, and the search method's computational complexity increases with the size of operations and search space. The larger the search space in terms of the fundamental units' size and dimension, the higher the computation time as the search algorithm requires evaluating more options. Also, similar to the traditional Machine Learning methods, the search method is also prone to overfitting, as the NAS algorithm relies on the standard ML techniques. As seen from the experiments of AutoFormer [88] (Fig. 14), increasing the depth (d), Embedding dimension (e), MLP ratio (r), and the number of heads (h) increases the accuracy on the ImageNet dataset till a peak value and overfits after attaining the maximum accuracy. Hence, there lies a tremendous challenge in choosing the

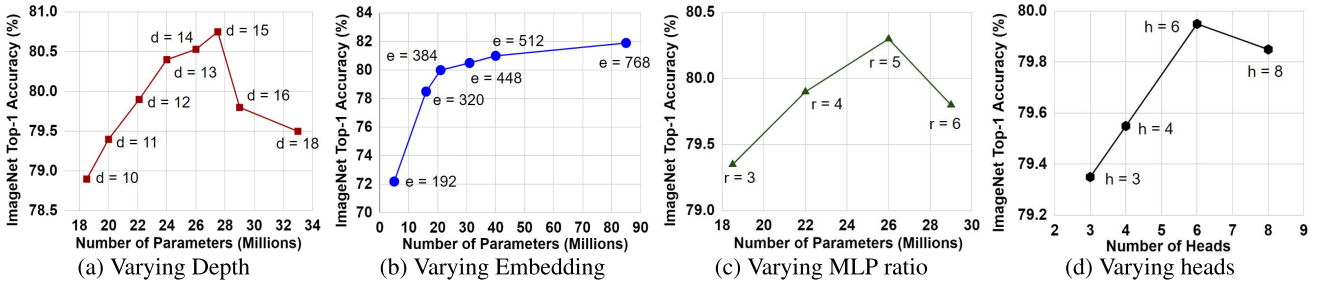


FIGURE 14. Top-1 Accuracy on ImageNet dataset [77] vs Varying dimensions of vanilla transformer hyperparameters: Depth (d), Embedding dimension (e), MLP ratio (r), and Number of heads (h) in the FFN unit. For a given plot, only the mentioned hyperparameter is varying, and the other parameters are kept constant.

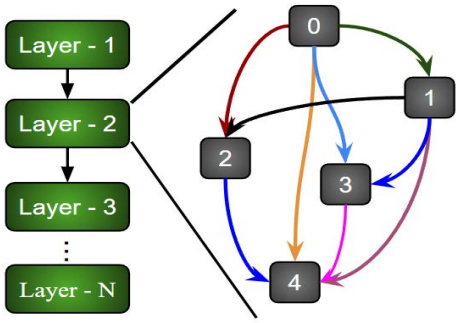


FIGURE 15. Cell-level search space.

correct dimensions of a Transformer and a robust search space that can guarantee optimal model performance for a given dataset. The search algorithm can only find average networks with average search space.

C. SEARCH SPACE TYPES

There exist two types of widely used search spaces through which the primitive elements are connected to each other to form an end-to-end Transformer neural architecture: **(i)** Micro/Cell-level Search space and **(ii)** Micro/Layer-wise Search. These search spaces are applicable to any kind of Neural Network, not limited to the Transformer model.

1) MICRO/CELL-LEVEL SEARCH

The Micro search works on the principle that a small Directed Acyclic Graph (DAG) or a cell structure is searched instead of searching for the entire network end-to-end. The searched DAG is stacked N times sequentially to form the final architecture, thereby reusing the same structure repeatedly throughout the model. A cell is constructed using M nodes and edges, as shown in Fig. 15, where each node is a tensor, and the edge represents an operation that transforms data between any two nodes.

The task boils down to finding the optimal operation for each individual edge from the pre-defined set of primitive elements. Although the same topology is replicated across different layers in the network, the dimensions of the primitive elements, such as filter size, vary with each cell. Examples of NAS methods applying the cell-based search space are Evolved Transformer [89], DARTS-Conformer [90], etc.

2) MACRO/LAYER-WISE SEARCH

The identical cell structure throughout the network in the micro-search process offers limited flexibility and performs poorly on multiple hardware platforms due to its unstructured computation. Layer-wise/Macro search space can resolve this problem by first constructing a chain-type macro-architecture and searching for different operations/configurations at each layer to obtain more robust models for hardware-friendly inference and better model performance. The majority of NAS methods discussed in this survey paper, such as GLiT [91], rely on layer-wise search.

D. TRANSFORMER SEARCH SPACE TYPES

Previously, we divided the search spaces based on the interconnection of several primitive operations, irrespective of the Self-attention or Convolution operation. There are two categories of search spaces with reference to the type of operations in the primitive element set: **(i)** Self-Attention only search space and **(ii)** Hybrid Attention-Convolution search space. Both the search spaces can be employed in any architecture in the Transformer family; Vanilla Transformer, BERT, and Vision Transformer depending on the application and requirement. We outline several key Attention and Convolution search spaces which act as the backbone for searching different neural architectures at the respective subsections while discussing the method. The summarized search spaces help understand the different components of a Transformer architecture search space. Also, researchers willing to start researching Transformer NAS-related problems can pick up the search spaces mentioned.

1) SELF-ATTENTION (SA) ONLY SEARCH SPACE

The SA-only search space is limited to the elements in Vanilla Transformer, such as head number, FFN hidden size, etc., simply a Convolution-free search space. The early NAS methods such as Evolved Transformer [89] and AutoFormer [88] on language and vision tasks relied only on the encoder or decoder of Vanilla Transformer to search for better hyperparameters. We outline a few commonly used search spaces and their components to get a sense of the problem.

a: AUTOFORMER [88]

The search space, illustrated in Table 3, consists of key components of a Transformer such as Embedding dimension, number of heads, Q-K-V dimension, MLP ratio, and network depth. The authors propose three network setups (tiny, small, and base) to operate under different constraints and model complexities that vary across layers. The dimensions of search elements can take any value between the highest (x) and lowest (y) value with a step size of z, which is represented as a tuple (x, y, z) in the table.

TABLE 3. AutoFormer search space.

Parameters	Tiny	Small	Base
Embedding Dimension	(192, 240, 24)	(320, 448, 64)	(528, 624, 48)
Q-K-V Dimension	(192, 256, 64)	(320, 448, 64)	(512, 640, 64)
MLP Ratio	(3.5, 4, 0.5)	(3, 4, 0.5)	(3, 4, 0.5)
Head Num	(3, 4, 1)	(5, 7, 1)	(8, 10, 1)
Depth Num	(12, 14, 1)	(12, 14, 1)	(14, 16, 1)
Parameters Range	4-9 M	14-34 M	42-75 M

b: TWINS SEARCH SPACE [87]

ViTAS [92] extended the tiny, small, large, and base backbones of Twins Transformer [87] to search for operation type (Op.), patch size, head number, and dimensions of MHSA and MLP. The backbone architecture is divided into four stages, as shown in Table 4, where each stage is defined with maximum dimensions and number of repeatable blocks. The local and global in the Table refers to Locally-grouped Self-attention (LSA) and Global sub-sampled attention (GSA), respectively, proposed by Twins [87]. LSA is used to capture the short-distance and fine-grained information, and GSA is utilized to fetch long-distance and global information. The Max_a specifies the maximum dimension of MHSA layer, also the size of patch embedding layer, and Max_m indicates maximum dimension of MLP layer. The ratio {i/10} denotes the reduction ratio from the maximum output dimension i.e., the search space includes MHSA dimensions of $\frac{i*Max_a}{10}$ for i ={1,2..10}.

TABLE 4. Twins transformer search space.

Num.	Op.	Type	Patch Size #Heads	Max_a	Max_m	Ratio i = {1,2..10}
1	Embedding		4	192	–	{i/10}
4	TBS	LSA GSA	{2, 4, 8, 16}	480	768	{i/10}
1	Embedding		4	384	–	{i/10}
4	TBS	LSA GSA	{2, 4, 8, 16}	960	1536	{i/10}
1	Embedding		4	768	–	{i/10}
20	TBS	LSA GSA	{2, 4, 8, 16}	1920	1920	{i/10}
1	Embedding		2	1563	–	{i/10}
4	TBS	LSA GSA	{2, 4, 8, 16}	3840	3840	{i/10}

c: DeiT SEARCH SPACE [9]

ViTAS [92] also extended the tiny and small backbones of the Data-efficient Image Transformer (DeiT) [9], whose search space is delineated in Table 5. DeiT retains the skeleton of the original Vision Transformer [5] by searching for patch size, number of heads, and output dimensions of MHSA and MLP. The "Max Dim." in the table indicates the maximum output dimensions of MHSA and MLP blocks, and the ratio specifies the reduction ratio from the max dimension.

TABLE 5. DeiT [9] -tiny and -small transformer space.

Num.	Op.	Type	Patch Size #Heads	Max Dim. (tiny/Small)	Ratio (i) {1,2..10}
14	TBS	MHSA	{3, 6,12, 16}	1440/2880	{ i/10 }
		FFN	—	1440/2880	{ i/10}
1	Linear/FC		{14, 16, 32}	384/768	{ i/10}

2) HYBRID ATTENTION-CONVOLUTION SEARCH SPACE

There is a growing interest in integrating the Self-Attention mechanism and Convolution operations (Spatial and Depth-wise Convolutions) in ViTs to combine the strengths of both worlds [94]. For instance, the Convolutional vision Transformer (CvT) [83] introduced a Convolution-based Embedding and QKV projection instead of an FC network in ViT. MobileViT [95] adopted a macro-architecture similar to MobilenetV2 [74] and replaced a few MBConv blocks with proposed attention-based modules. The blended Attention-Convolution search space is not limited to vision applications but also tremendously used in NLP and speech tasks.

a: NASViT SEARCH SPACE [93]

The search space of NASViT [93] is inspired by the architecture of LeViT [96], where the first four layers are Convolution operations for better efficiency in processing high-resolution feature maps, followed by MHSA operations in the remaining part of the network to handle low-resolution embeddings for modeling global representation. The initial three layers of NASViT consist of MBConv units, and the remaining four layers are occupied by Transformer modules, as shown in Table 6. The search elements in MBConv are output channel width (C), kernel size (k) of Depthwise Convolution, expansion ratio (e), and the number of repeat MBConvs (depth). The primitive elements in the Transformer block are the number of windows (a technique in Swin Transformers [6]), width (hidden dimension features), number of MHSAs (depth), and MLP ratio in FFN. The shifted windowing method in Swin Transformer limits the Self-attention to a non-overlapping local window for better efficiency while permitting cross-windowing in a few cases.

V. NEURAL ARCHITECTURE SEARCH METHODS

The pivotal aspect to the success of Neural Networks is the efficient design of architecture for a given application. The manual design of a neural architecture is a labor-intensive procedure, requiring domain expertise and significant computing infrastructure to train and evaluate many networks. Furthermore, prior knowledge restricts researchers/engineers from developing out-of-the-box and innovative architectures

TABLE 6. NASViT search space [93].

Dynamic Blocks	Width	Depth	Kernel Size	Exp. Ratio	SE	Stride	No. of Windows
Conv	{16, 24}	–	3	1	-	2	–
MBConv-1	{16, 24}	{1, 2}	{3, 5}	1	N	1	–
MBConv-2	[24, 32]	{3, 4, 5}	{3, 5}	{4, 5, 6}	N	2	–
MBConv-3	{32, 40}	{3, 4, 5, 6}	{3, 5}	{4, 5, 6}	Y	2	–
Transformer-4	{64, 72}	{3, 4, 5, 6}	–	{1, 2}	–	2	1
Transformer-5	{112, 120, 128}	{3, 4, 5, 6, 7, 8}	–	{1, 2}	–	2	1
Transformer-6	{160, 168, 176, 184}	{3, 4, 5, 6, 7, 8}	–	{1, 2}	–	1	1
Transformer-7	{208, 216, 224}	{3, 4, 5, 6}	–	{1, 2}	–	2	1
MBPool	{1792, 1984}	–	1	6	–	–	–
Input Resolution	{192, 224, 256, 288}						

to attain higher model performance than state-of-the-art designs. Neural Architecture Search (NAS) aims to discover best performing neural architecture s* from a given Search Space S on a given Dataset D. The search algorithm describes the methodology used to find the optimal model from a pool of all possible Neural Networks. Although there exist several types of NAS methods in the literature, we discuss preliminaries and basic outline of **(1)** Reinforcement Learning (RL), **(2)** Evolutionary Learning NAS, and **(3)** One-Shot NAS methods on which Transformer NAS strategies rely.

A. REINFORCEMENT LEARNING NAS (RL-NAS)

Reinforcement Learning (RL) is a computational paradigm for learning a policy that takes optimal actions in various states of a task environment to maximize the cumulative rewards received by the acting agent, as shown in Fig. 16.

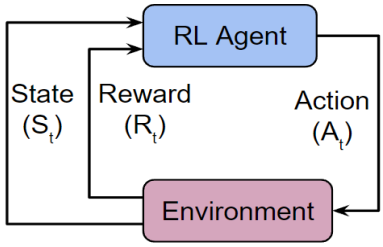


FIGURE 16. Reinforcement learning methodology.

The Reinforcement Learning-based Neural Architecture Search methodologies [12], [97] learn to discover the best-performing model architecture, similar to traditional RL methods, whose objective is to learn the best action. The environment denotes the pool of all possible distinct Neural Networks, built using a pre-defined search space and primitive elements. The NAS methods using RL framework use RNNs as RL controllers whose job is to sample high-quality neural architectures from of all possible models, as shown in Fig. 17. The RNN predicted model is trained end-to-end to obtain validation accuracy or relevant model performance metric. The validation accuracy is the feedback or reward which guides and updates the RNN controller to predict the next best candidate model. In RL terminology, neural architecture moves from the current state to the next best state, where the reward maximizes.

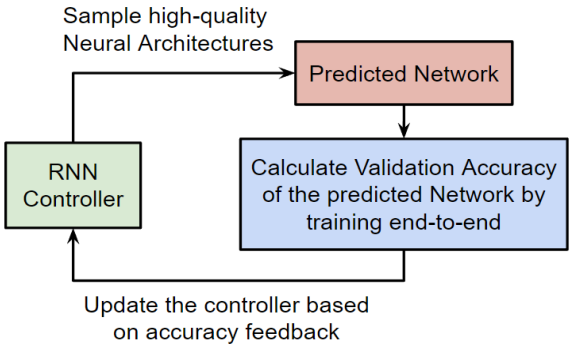


FIGURE 17. Reinforcement learning NAS method.

B. EVOLUTIONARY LEARNING NAS (EL-NAS)

Evolutionary Neural Architecture Search (EL-NAS) [98] is a generic population optimization algorithm that draws its inspiration from natural selection and genetics [45]. Although this search method comes in various forms, the most commonly used technique is the Genetic Algorithm (GA) which simulates the evolution of species. GA is an iterative process of evaluating selected individuals according to a fitness function and generating a new set of individuals using characteristics of best-performing models from the previous generation. The basic component of GA is population, also called individuals, each of which describes a distinct architecture in the predefined search space. An initial population is generated by randomly sampling different networks from a large pool of networks. The individuals, each representing a specific neural architecture, are trained end-to-end on the target task to determine fitness (typically model performance such as Validation Accuracy). Generating new individuals or populations involves mutation of primitive operations within a network or crossover of top individual models in the current generation of networks.

The Genetic Algorithm utilizes the Tournament Selection method to choose the fittest candidates from the current generation of architectures [89]. The K-way tournament selection method selects the top k best performing individuals to run a tournament among themselves. The fittest candidate among chosen k candidates is picked for further processing in iteration. In general, there exist several tournaments to pick the best architecture for a given dataset and application.

The weaker candidates have less chance of survival in case of a large tournament as it competes with candidates of a higher fitness function. The summary of Evolutionary Neural Architecture Search is depicted in Fig. 18.

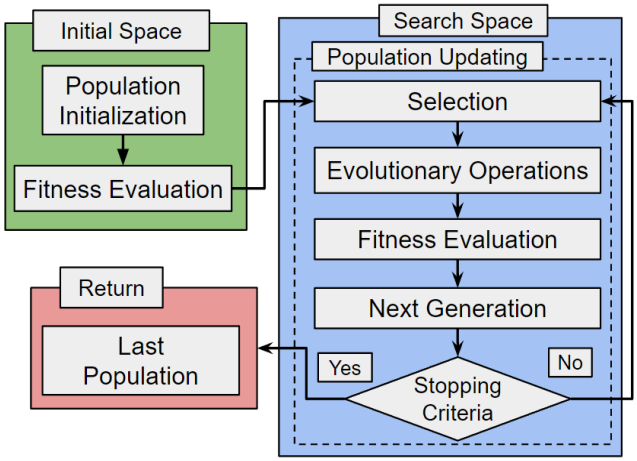


FIGURE 18. Evolutionary learning NAS [45].

C. ONE-SHOT NAS

The RL search methods' resource, time, and computational demand make it very difficult to find efficient neural architectures as it requires the method to train and evaluate several networks. The One-shot NAS algorithm [100] lowers the search burden on limited computing devices by representing or encoding all possible distinct models in the search space in the form of a single Supernetwork. The key idea is to couple network training and neural architecture sampling into a supernet to reduce the search cost. The discrete set of candidate architectures is replaced by one continuous supernetwork, as opposed to the RL or EL-NAS method. Therefore, only one Supernetwork is trained instead of discrete training of individual networks by leveraging the weight sharing concept as the same operation can be used to evaluate several neural architecture combinations. Nevertheless, the search cost increases with the size of search elements as more parameters are trained in the Supernetwork. In the One-shot NAS approach, the term SuperTransformer is a substitute for supernetwork consisting of all possible combinations on the Transformer space and subTransformer refers to the subnetwork sampled from the SuperTransformer.

1) DARTS: DIFFERENTIABLE ARCHITECTURE SEARCH

The pioneering work in One-Shot NAS is **DARTS** [99], which formulates the search problem in a differentiable manner, unlike RL or Evolutionary search. DARTS uses the softmax function to relax the discrete search space and gradient descent to train Supernetwork and search for architecture. The pivotal idea in this search method is to construct a Supernetwork that encodes all possible distinct neural architectures based on the predefined primitive search elements. For example, DARTS defines a cell structure with four nodes (numbered 0, 1, 2, 3) and six edges (each edge corresponds to an operation). The distinct possible paths with four nodes are as follows: (0→1), (0→2), (0→3), (1→2), (1→3), and (2→3), as shown in Fig. 19a. The task is to search for the best operation at each edge such that overall model performance is as high as possible. Each node is populated with all possible operations in the set of search elements to form a supernode, as illustrated in Fig. 19b. A set of auxiliary parameters called architectural weight parameters $\{\alpha_1, \alpha_2, \ldots \alpha_N\}$ are defined for every individual node which guides the search process at respective nodes. The input activation at each node is broadcasted to all distinct operations, and the output activation map is a softmax weighted sum of operations. If $(\alpha_1, \alpha_2, \ldots \alpha_N)$ are designated to operations $(\mathrm{Op}_1, \mathrm{Op}_2, \ldots \mathrm{Op}_N)$ respectively in a supernode, activation feature map O_{fmap} at output node for input feature map I is given as per Eq. 9.

$$O_{fmap} = \sum_{i=1}^{N} Op_i(I) * \Omega_i \tag{9}$$

where $\{\Omega\}_N$ is the set of softmax weighted sum over the architectural parameters $\{\alpha_1, \alpha_2, \ldots \alpha_N\}$, as shown in Eq. 10.

$$\Omega_i = \frac{e^{\alpha_i}}{\sum_{n=1}^{N} e^{\alpha_n}} \tag{10}$$

The entire Supernetwork is trained end-to-end using standard gradient descent, similar to traditional Neural Network training. The Convolution or Attention weights $W*$ of Supernetwork are updated on the training dataset $\mathcal{D}_{train}$ and freezing the architectural weights $\alpha*$. On the other hand, architectural weight parameters $\alpha*$ are updated on the validation dataset $\mathcal{D}_{val}$ and freezing operation weights $W*$. The best operation at each edge of the trained Supernetwork is sampled by choosing the operation with the highest architectural weight α from the respective set of architectural weights $\{\alpha_1, \alpha_2, \ldots \alpha_N\}$. For example, the blue-colored path is chosen between nodes (0, 2), as shown in Fig. 19c. The final cell architecture is formed by repeatedly sampling operations at each edge, as depicted in Fig. 19d.

Although One-shot NAS training eases the search burden, it is prohibitive in case of a huge search element set and search space. This leads to high GPU memory consumption and eventually running out on a large dataset. Also, training a Supernetwork is non-trivial due to interference caused by child networks; thus, rank correlation is not preserved. Several follow-up methods, such as ProxylessNAS [101] and Single-path NAS [102], addressed these issues and enhanced the differentiable search algorithms.

D. PERFORMANCE EVALUATION

The performance evaluation phase in a search algorithm specifies the estimation methodology of performance of a predicted neural architecture in the search space. It compares different Neural Networks generated by search algorithms and guides the search process to find optimal models. A few examples of performance evaluation strategies include full

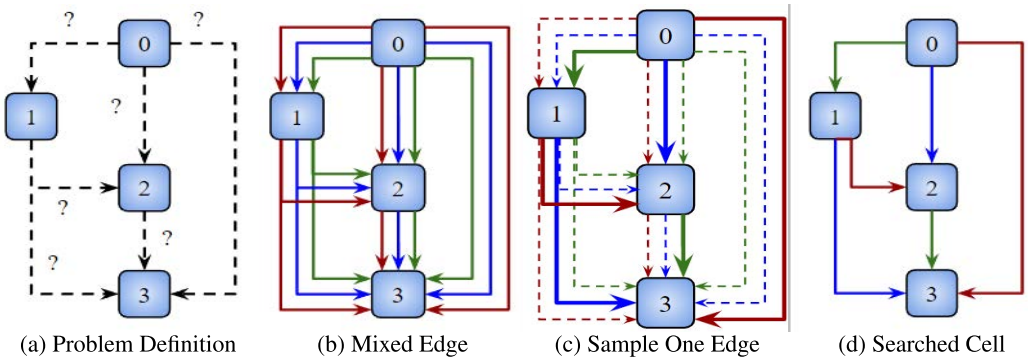


FIGURE 19. DARTS: differentiable architecture search [99].

training of architectures or partial training on the target/proxy datasets. The seminal RL-NAS method [12] trains every network predicted by the RNN controller till convergence. Hence, this process is highly expensive, requiring 60 years of GPU computation time, even on a proxy dataset (CIFAR-10). In order to curtail expensive performance estimation, several methods have been introduced, such as early stopping [103], use of NAS Benchmarks [104], low-resolution dataset, and network search with fewer parameters and cells [99]. The One-shot methods can alleviate the performance estimation burden by sharing weight parameters of all networks within a single graph and evaluating a sub-graph from a Supernetwork. A few examples of performance evaluation in Transformer-based NAS methods are as follow: **(i)** GLiT [91] uses the validation dataset for evaluating a subnetwork during the search process, and **(ii)** subnetworks in AutoFormer [88] are evaluated according to the manager of the evolution algorithm.

VI. HARDWARE-AWARE NAS (HW-NAS)

Hardware-Aware Neural Architecture Search fully automates the Neural Network design procedure to search for models that are not optimized for accuracy but also for efficient performance on the underlying hardware platform. The conventional hardware-agnostic NAS methods aim to search for the best neural architecture that maximizes only task performance or accuracy. However, HW-NAS finds promising networks which are optimized for a wide range of hardware performance metrics such as latency [105], [106], FLOPS [70], power consumption, energy [107], and memory usage [108]. This section first reviews efficient methods for hardware inference, followed by a discussion on injecting hardware awareness in RL, One-shot, and EL-NAS methods.

A. HARDWARE ACCELERATION METHODS

Pruning [109], [110], Quantization [111], [112], [113], Knowledge Distillation [114], Tensor Decomposition [115], Neural Architecture Search, etc., are some of the techniques to design lightweight, memory-efficient, and hardware-friendly methods for inference on a variety of devices such as CPU, GPU, ASIC, FPGA. These methods achieved remarkable success in compressing large networks into smaller models with negligible accuracy or task performance loss. Neural Network Pruning refers to removing redundant or unimportant weights/nodes/neurons/filters parameters which do not significantly hinder model performance, thereby reducing the size and computational complexity of a model. Network Quantization converts the high precision model weights/parameters (Floating point 32) to low precision (Integer 8, Integer 4). Quantization methodology has attracted much attention in academia and industry as inference of a model can be performed at a low precision with a negligible drop in accuracy, as opposed to training where a model is trained at high precision. Knowledge distillation is a type of network compression method where a relatively small model is learned to mimic the behavior of a large pre-trained network.

B. HW-NAS METHODS

We discuss three HW-NAS methods, namely, **(i)** RL-HW-NAS, **(ii)** EL-HW-NAS, and **(iii)** Differentiable-HW-NAS, where the core search algorithm remains similar to RL-NAS, EL-NAS and Differentiable-NAS, respectively.

1) REINFORCEMENT LEARNING BASED HW-NAS (RL-HW-NAS)

In RL-HW-NAS, the Controller samples networks that are not only accurate but also less computationally expensive. For instance, MNasNet [105] added hardware latency characteristics as constraints in formulating the NAS problem. Given a Neural Network model M, Acc(M), Lat(M), and T stand for the validation accuracy, the latency of model M on the hardware (or any other constraints), and target latency, respectively. The NAS optimization can be reformulation to include the hardware constraints, as per Eq. 11.

$$\max_{M} \; Acc(M); \quad s.t \quad Lat(M) \leq T \tag{11}$$

A weighted product of accuracy and latency can be used to approximate Eq. 11 along a tunable parameter γ for the trade-off between these two metrics, as shown in Eq. 12. A higher γ value prefers latency minimization over accuracy

maximization and vice versa.

$$\max_{M} \; Acc(M) * \left(\frac{Lat(M)}{T}\right)^{\gamma} \tag{12}$$

Jiang et al. [116] upgrade the RL-HW-NAS by avoiding training sampled networks that do not meet the target constraint. In this case, the controller receives a zero signal to update and predict the next best network in the search process, thereby decreasing the search cost.

2) EVOLUTIONARY HW-NAS (EL-HW-NAS)

EL-HW-NAS modifies the fitness function (fit(Acc, Lat, t)), similar to RL-HW-NAS, to include both accuracy and latency metrics along with a tunable parameter (t ∈ [0, 1]) for tradeoff models, as shown in Eq. 13.

$$fit(Acc, Lat, t) = (1 - t) * Acc(M) + t * Lat(M) \tag{13}$$

3) DIFFERENTIABLE HW-NAS

Differentiable HW-NAS methods, such as Proxyless-NAS [101], add hardware cost function directly to the loss function of Supernetwork, thereby formulating it as a multi-objective optimization function, as shown in Eq. 14

$$Loss = Loss_{CE} + \lambda * C(Lat(\alpha)) \tag{14}$$

where C(Lat(α)) is the latency cost function of both architectural parameters (α) and actual hardware latencies. If $\{\alpha_{i1}, \alpha_{i2}, .. \alpha_{iN}\}$ correspond to architectural parameters of layer "i" and $\{Lat_{i1}, Lat_{i2}, .. Lat_{iN}\}$ represent latencies of operations $\{Op_{i1}, Op_{i2}, .. Op_{iN}\}$ in the same layer, the latency cost function C(Lat(α)) is formulated as per Eq. 15.

$$C(Lat(\alpha)) = \sum_{i=1}^{L} \sum_{j=1}^{N} Lat_{ij} * \alpha_{ij} \tag{15}$$

A similar bi-level optimization of alternate weights (w*) and architectural parameters (α) update is performed to search for a subnetwork that minimizes validation loss and latency cost function.

Once-For-All (OFA) [117] combines the idea of One-shot Supernetwork training and Evolutionary search to reduce the search cost. OFA for CNN search first trains a large over-parameterized network of maximum dimensions (kernel size and channel width) and samples different sized models to specialize for different hardware platforms. The main advantage is avoiding fine-tuning of sampled networks for different cases as weights are retained from the Supernetwork. Several Transformer NAS methods described in this paper, such as AutoFormer and ViT-ResNAS [118], rely on the Once-for-all methodology by first training a large Transformer Supernetwork and applying Evolutionary search to find the optimal submodel. For instance, an MHSA/FC dimension 256 is encapsulated within a linear layer of dimension 512 in SuperFormer. During the Evolutionary search, various sub-networks within the size of 512 are sampled for evaluation.

C. NAS FOR PRUNING

Automatic pruning applies the search methods to automatically prune the redundant weights/neurons in a network instead of manually designed pruning algorithms. The automatic pruning methods, such as APQ [119] and NetAdapt [120], significantly reduce the size of a CNN model. Several BERT-based automatic pruning methods, like AdaBERT [121] and NAS-BERT [122], have been developed to compress the large pre-trained BERT model on a huge dataset into a small model for downstream tasks.

D. NAS FOR QUANTIZATION

Uniform Quantization (UQ) is a process of lowering the precision of Neural Network parameters to the same precision. For example, Uniform Integer 8 quantization is a simple and promising method to quantize without any accuracy loss. On the other hand, Mixed Precision Quantization (MPQ) quantizes layers within a model or tensors (weight and activation) within a layer to different precisions to attain additional speedup compared to uniform Int8. For a given network of "L" layers and "P" precisions to choose from, there exist P^L distinct configurations. Hence, the mixed precision quantization can be transformed into an architecture search problem, where the operations in the search space are replaced by different precisions such as Int2, Int4, Int8, etc. There have been many CNN-based works, such as HAQ [123] and DNAS [124], which apply NAS principles to solve the problem. There is a growing interest in applying the search methods to Transformers, such as AQ-BERT [125].

VII. NAS FOR VANILLA TRANSFORMERS

This section reviews Vanilla Transformer-based NAS methods for language and speech Applications. As a part of summarizing various methods, we highlight the following three components: **(i)** search space design, **(ii)** search algorithm, and **(iii)** model performance of the searched network. The Vanilla Transformers are used on numerous language modeling and speech tasks. Sequence-to-Sequence (seq2seq) is a Machine Translation task to convert a text sequence in one language (Eg: English) to another (Eg: French). The commonly used datasets for machine translation tasks are WMT 2014 English-German, WMT 2014 English-French (En-Fr), WMT 2014 English-Czech (En-Cs) [126], and One Billion Words Benchmark (LM1B) [127]. BLEU (BiLingual Evaluation Understudy) [128] is a metric for automatically evaluating machine-translated text. The BLEU score is a number between zero and one that measures the similarity of the machine-translated text to a set of high-quality reference translations. Text Classification trains a model to assign a label to input text as per its content. Natural Language Inference determines whether the given hypothesis text is entailment (true), contradiction (false), or neutral. Text to speech (TTS) synthesizes natural speech audio to convert it into a text, as the name suggests. Automatic Speech Recognition

(ASR) is the task of processing human speech and extracting useful information in the form of a text.

A. EVOLVED TRANSFORMER

Evolved Transformer [89] is one of the early methods for efficient Transformer search intended for sequence-to-sequence tasks. It adopts a cell-based search space, inspired by NASNet [103], using Convolution and Self-attention elements as primitive operations to leverage the strengths of two layers. The search space comprises two stackable cells, as shown in Fig. 20, each for an encoder and decoder unit. The cell topology is repeated "*number of cells*" times to form the overall architecture, where each cell consists of multiple blocks: six unique blocks for the encoder and eight blocks per cell for the decoder. The block receives two hidden states as input and applies transformation individually to inputs, referred to as branches (right and left branches). The Evolved Transformer method builds a large search space focusing on many primitives in the network, as shown in Fig. 20, which is described as follows:

1) **Branch-level Primitives (left and right): (a) Input:** Indicates specific hidden state to be fed to the branch in a cell, **(b) Normalization:** Layer Normalization [62], None, **(c) Layer:** Standard Spatial Convolution, Depthwise Separable Convolution, Lightweight Convolution [129], Head Attention $h \in \{4, 8, 16\}$, Gated Linear Unit [130], Attend to encoder (available only to decoder), Identity layer, Dead branch, **(d) Output:** The Output dimension, **(e) Activation function:** Swish [131], [132], ReLU, Leaky ReLU [133], and None.
2) **Block-level Primitive:** Combiner function chosen from Addition, Concatenation, and Multiplication
3) **Cell-level Primitive:** Number of cells: {1, 6}

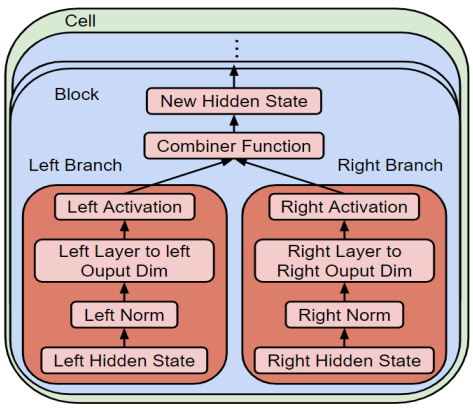


FIGURE 20. Search space in evolved transformer [89].

The authors devised Progressive Dynamic Hurdles (PDH) within the evolution-based architecture search [134]. It dynamically allocates more computational resources to the networks with higher fitness function values and allows them to train for more epochs than the models with lower fitness function values. The Evolved Transformer [89] finds better feedforward configurations by alternating between Convolution and Self-attention modules. The searched Transformer architecture exhibits improvement over the original Transformer on WMT 2014 En-Ger, WMT 2014 En-Fr, WMT 2014 En-Cs, and 1 Billion Word Language Model Benchmark dataset. The Evolved Transformer matches the performance of the Vanilla Transformer with only 78% of its parameters. Nevertheless, the search process is computationally very expensive, requiring training and evaluating several architectures in the search space.

B. PRIMER

Primer [135] introduces the Multi-DConv-Head Attention (MDHA) unit by adding a 3×1 Depthwise Convolution after Q, K, and V projection in Self-attention heads, as shown in Fig. 21. Also, it squares the ReLU output activation in MHSA module. The search space includes basic operations, such as add, multiply, and cosine, and key Spatial and Depthwise Convolution parameters like kernel and filter size.

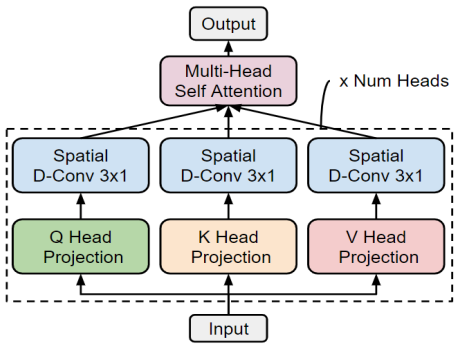


FIGURE 21. Primer [135].

Primer's search strategy is similar to the Evolutionary search method used in Evolved Transformer [89], which evolves close to 25k distinct architectures by mutating one network at a time through inserting, deleting, and swapping, or randomly changing operation and parameter values. The search algorithm's constraint is finding models that offer low validation loss for a fixed training budget (24 TPUv2 hours). The search is performed on the One Billion Words Benchmark, and searched Primer architecture matches the performance of Vanilla Transformer with 4.2x less compute time on the language modeling task.

C. DARTSFORMER

DARTSformer [136] first motivates the problem of directly applying DARTS [99] to the Transformer search space. The memory consumption of Transformer Supernetwork increases with hidden size and runs out even with small values. Therefore, DARTSformer combines DARTS methodology with Reversible Networks [137] to search without running out of memory. The idea is to reconstruct the input of a reversible network layer from its output during backpropagation, hence requiring storing only the

output of the last layer. This relieves the memory burden on Transformer Supernetwork and allows to include operations with higher hidden sizes and more candidate choices. The searched network for Machine Translation task consistently performs better than Vanilla Transformer and is comparable to the large Evolved Transformer with a significant reduction in search cost.

D. AUTOTRANS

AutoTrans [138] automates the end-to-end design process of a Vanilla Transformer by not only including parameter values such as number of heads (1, 2, 4, 8, 16), number of encoder/decoder layers (1, 2, 3, 4, 6, 9), activation function (ReLU, Leaky ReLU, ELU, Swish, GELU, GELU new), relative dimension of hidden size (0.5, 1, 2, 4, 8), but also searching for where to put encoder attention, how to set layer-norm, whether to scale, etc., in the search space. It relies on Efficient Neural Architecture Search (ENAS) [97] to sample and evaluate several models. The network is searched and evaluated directly on three machine translation tasks using the following three benchmarks: Multi-30K [139], WMT-14 (English-German), and CoNLL2003 [140]. The AutoTrans-searched efficient Transformer produces a better BLEU score than manually designed networks across different datasets.

E. TEXTNAS

TextNAS [141] proposed a multi-path novel search space by using a mix of Convolution (filter sizes of 1, 3, 5, and 7), Pooling (max and average), Recurrent (bi-directional GRU), and Self-attention layers. The macro search space can be depicted as an Acyclic Directed Graph (DAG), as shown in Fig. 22a. Each layer in the DAG can choose any of the previously mentioned operations to form the searched DAG, as depicted in Fig. 22b.

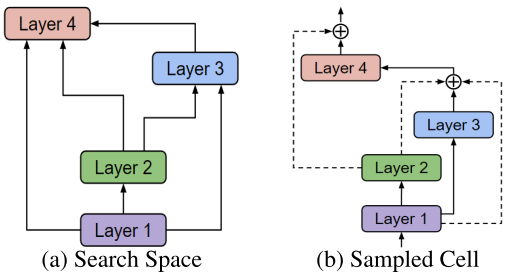


FIGURE 22. TextNAS [141] search space.

The authors of TextNAS directly utilize Efficient Neural Architecture Search (ENAS) [97] to find better Transformer models for Text Classification and Natural Language Inference tasks. ENAS is an enhanced version of RL-NAS where the common weight tensors in all the predicted child models are shared in the same iteration to avoid individual training and save computation time. The network is searched on the Stanford Sentiment Treebank dataset [142], and the performance is measured on eight Text Classification datasets [143]. The searched Transformer, consisting of Convolution and Attention operations, outperforms the previous Convolution-only networks searched using DARTS [99], ENAS [97], etc., by at least a 0.5 significant score.

F. KNAS

KNAS [144] can be dubbed as Green Neural Architecture Search, meaning that candidate networks in the search process are evaluated without training the actual architecture. The authors use gradients as a coarse-grained proxy to extract essential features from random-initialized models, and these key features are directly used to evaluate the selected network. The Text Classification task search space comprises 12 encoder and 12 decoder layers, which also share the same hyperparameters. The searched encoder-decoder network outperforms the baseline RoBERTa-large [65] on two text classification datasets.

G. SEMINAS

SemiNAS [145] is a semi-supervised search method that leverages an accuracy predictor to evaluate networks at a low cost. The accuracy predictor is initially trained on a small set of network-accuracy pairs and uses the trained predictor to estimate the accuracy of unseen architectures. The predicted architecture-accuracy data pairs are further added to the accuracy predictor training dataset to enhance the surrogate model's performance. The end-to-end accuracy predictor framework comprises an LSTM-based encoder-decoder network and an MLP network-based predictor. The backbone for Text-to-Speech application is a multi-layer encoder-decoder network with the following three operations in the search space: a standard Convolution layer with kernel sizes {1, 5, 9, 13, 17, 21, 25}, Transformer blocks with the number of heads {2, 4, 8} in MHSA, and an LSTM operation. SemiNAS explores a Self-attention-based architecture on the LJSpeech dataset [146] in four GPU days, and the searched model achieves a better Intelligibility Rate (%) and Diagonal Focus Rate (%) than the manually designed Transformer TTS network [147] by 9% and 4%, respectively.

H. AUTOTRANSFORMER

AutoTransformer [148] first designs a customized search space for Time Series Classification (TSC) task and applies Gradient-based Differentiable Architecture Sampler (GDAS) [149], an improved DARTS method, to search for an efficient Transformer. The search space incorporates several structures and operations within the Transformer backbone, which can extract global and local features from the time series. Fig. 23 illustrates the overall search space of AutoTransformer, where an operation choice is selected from the pool of following elements: 1D standard and dilated Convolutions of kernel size {1, 3, 5, 7, 9, 11, 13, 17, 21}, max and average pooling, LSTM [3] and MHSA. The solid black line indicates fixed data flow, red dotted lines are the candidate choices for input, and blue represents choices for residual input choice. The input to a given layer or residual connection

within a layer can be chosen from any of its predecessor layers, including the original input. For example, layer 3 can independently choose any tensor from the set of {original input, layer 1 output, layer 2 output}. This type of multi-layer connection is crucial as initial layers capture low-level information and deep layers capture high-level information in the time series input. AutoTransformer utilizes GDAS [149] to sample and train only sub-graph from Supernetwork in each training iteration to reduce search computation time. The searched AutoTransformer model achieves SOTA results on the UCR TSC Archive dataset [150], thereby outperforming the Vanilla Transformer.

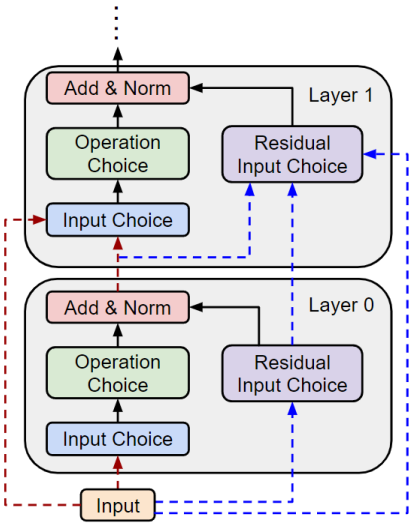


FIGURE 23. AutoTransformer [148].

I. EVOLVED SPEECH-TRANSFORMER

Evolved Speech-Transformer [151] is a straightforward extension of Evolved Transformer [89] for Automatic Speech Recognition (ASR) task, utilizing the same primitive elements, cell-level search space (Fig. 20), and PDH-based Evolutionary search algorithm. However, the search method is modified, so only models with less than three million parameters are considered in the trimmed search space to improve search time. Evolved Speech-Transformer is searched on Librispeech [152], a dataset of 80 hours of read English speech, and Zeroth [153], a dataset for Korean speech recognition. The searched model outperforms the baseline Speech-Transformer [2] in terms of Word Error Rate (WER %) with considerably fewer parameters. The search time is less than Evolved Transformer as the search space includes only 150 models as networks below three million parameters are considered.

J. DARTS-CONFORMER

DARTS-Conformer [90], as the name suggests, is a DARTS-based search method on the Conformer network [154]. The Conformer is a modified version of the Vanilla Transformer, formed by replacing the final FFN layer with a half-step FFN

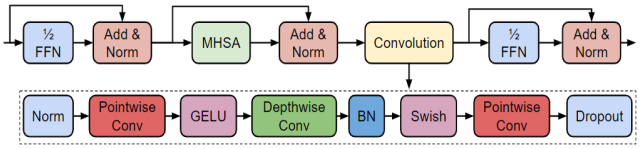


FIGURE 24. Conformer architecture [154].

and a series of Convolution modules, as shown in Fig. 24. The search elements within each module are: (i) $\frac{1}{2}$ FFN: Pointwise FFN, Skip, (ii) Convolution: Depthwise Separable Convolution of kernel sizes {15, 23, 31}, and (iii) MHSA: number of heads from the pool of {2, 4, 8}. The authors pointed out that the original DARTS algorithm can cause collapse issues in the early stages of Supernetwork training. This problem in the DARTS-Conformer is resolved by freezing architectural parameters (αs) for the first e_w epochs. The results on the AISHELL-1 dataset [155] show that the searched DARTS-Conformer model performs better than Conformer by at least 4.7% in terms of Character Error Rate (CER).

K. IMPROVED CONFORMER

Improved Conformer [156] is parallel work to DARTS-Conformer using the same Conformer as the backbone, but more primitive elements and larger in terms of size of each operation in the search space. The set includes the following: (i) $\frac{1}{2}$ FFN: inner dimension of {1024, 512, 256}, (ii) Convolution: Skip, Spatial and Dilated Convolution, each of kernel sizes {7, 11, 15}, and (iii) MHSA: number of heads from the pool of {4, 8, 16}. Improved Conformer also proposed a Dynamic Search Schedule method to alleviate the start-up problem caused by DARTS. To directly compare DARTS-Conformer and Improved Conformer, the latter achieves a CER score of 7.5% with 28.89M parameters, while the former attains a score of 6.3% with only 26.68M parameters. Hence, DARTS-Conformer is a more robust model than Improved Conformer.

L. BM-NAS

Bilevel Multimodal Neural Architecture Search (BM-NAS) [157] makes the architecture of multimodal fusion models fully searchable via a bilevel searching scheme. Fig. 25 depicts the DAG of multimodal fusion network search space in BM-NAS, where cells $\{\text{Cell}^{(1)}, \text{Cell}^{(2)}, \text{Cell}^{(3)}\}$ receive inputs either from previous cell $\text{Cell}^{(i)}$ or from two pre-trained Unimodal networks A and B. The bilevel scheme has two levels of search: **(i)** the high-level search space finds inter-modal/intra-modal features via a pre-trained unimodal backbone, and **(ii)** the low-level searchable components space includes primitive operations (Self-attention, Zero, Activation functions) at Step^1 and Step^2 along with connectivity of different operations inside each cell. BM-NAS is adjustable to various multimodal tasks with different scales in such a way that the number of cells and steps are chosen as hyperparameters. The search process is similar to DARTS [99], where a Supernetwork with all possible operations and

connectivity is constructed, and the search process is guided by three sets of architectural search parameters $\{\alpha, \beta, \gamma\}$, corresponding to cell connectivity among $\{\text{Cell}^{(1)}, \text{Cell}^{(2)}, \text{Cell}^{(3)}\}$, connectivity inside each cell and primitive operation search (Step^1 and Step^2), respectively. The similar bilevel optimization followed in DARTS is used to train network weights and architectural parameters. The results on three multimodal tasks using the BM-NAS framework show an improvement over baseline multimodal NAS methods with less search time and model parameters in the searched model.

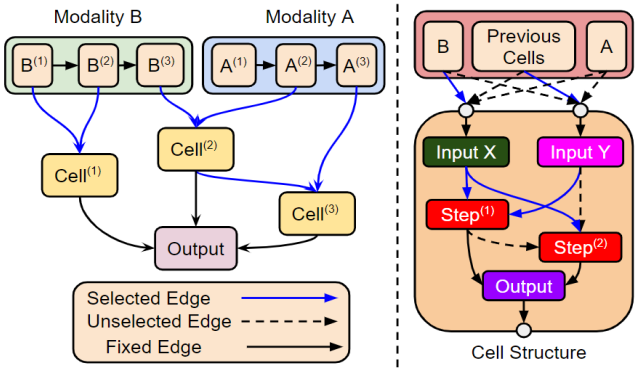


FIGURE 25. BM-NAS [157].

VIII. NAS FOR BERT MODELS

Large pretrained models such as BERT have shown their capabilities on several NLP tasks. A BERT is essentially made up of only an encoder from the Transformer. This section reviews search methods specific to BERT models on tasks such as Question answering, Sentence similarity, and Relation Classification (RC). The Stanford Question Answering Dataset (SQuAD) [158] is a widely used comprehension dataset of question and answer pairs, consisting of queries on a set of Wikipedia articles for Question answering tasks. General Language Understanding Evaluation (GLUE) [159] is an extensive collection of training and validation datasets for multiple NLP applications on Single-Sentence, Similarity and Paraphrase, and Inference tasks.

A. ADABERT

AdaBERT [121] leverages differentiable NAS to compress a pre-trained BERT model into smaller ones in a task-dependent manner, unlike previous works which compress the network in an application-independent manner. AdaBERT adopts a cell-based architecture in the search process and integrates task-specific knowledge distillation to impart hints on the application. The search algorithm also incorporates an efficiency-aware loss (Eq. 16) to fetch models with better efficiency and performance. The searched AdaBERT model is 11.5x to 17.0x more compressed, and 12.7x to 29.3x inferred faster than task-agnostic compressed models on GLUE benchmark datasets as task-specific redundant parameters in the network are further pruned.

$$\mathcal{L}_E = \frac{\mathcal{K}}{\mathcal{K}_{max}} \sum_{(o_{i,j}) \in \alpha_c} SIZE(o_{i,j}) + FLOPs(o_{i,j}) \qquad (16)$$

B. NAS-BERT

NAS-BERT [122] is a task-independent, NAS-based compression strategy to automatically compress a pre-trained BERT model. The NAS-BERT training is performed on a standard self-supervised pre-training task and does not rely on a specific downstream application. The search space is a chain-like structure of MHSA, FFN, and Separable Convolution units. Table 7 illustrates the NAS-BERT search space and each element's dimensions. The Separable Convolution mentioned in the search space is a sequential cascading of two Depthwise Separable Convolution operations (Depthwise + Pointwise Convolution). NAS-BERT operates on block-wise search [160], where the whole Supernetwork is divided into multiple blocks of a few layers, and each block is trained from the blocks present in a pre-trained teacher model. The block-wise search scheme improves the search space and, combined with a performance approximation policy, improves the search efficiency. The NAS-BERT-searched model on GLUE and SQuAD [158] datasets outperform the manually designed BERT model and previously designed searched networks such as the AdaBERT method in terms of model performance metric and number of parameters.

TABLE 7. NAS-BERT candidate operations.

Hidden Size	128	192	256	384	512
MHSA: #Heads	2	3	4	6	8
FFN: (Size of Intermediate Layer)	512	768	1024	1536	2048
SepConv	{3, 5, 7} Kernel Sizes				

C. AUTOBERT-ZERO

AutoBERT-Zero [161] deviates from the convention design of a Transformer encoder (Fig. 26a) by introducing Inter-layer search space to search for an optimal sequence of Self-attention and Convolution operations in a chain-like structure (Eg: Fig. 26c) to learn better local dependency. Also, it differs from the traditional design of a Self-attention module (Fig. 26b) by proposing an Intra-layer search Space of most fundamental math operations to search for an optimal inter-connection of primitive operations (Eg: Fig. 26d). The pool of Self-attention primitive operations in the intra-layer search space can be divided into the following two types: **(i) Unary:** neg, transpose, scale, softmax, logsigmoid, softsign, **(ii) Binary:** add, matmul, cosine similarity, euclidean distance. The Convolution search space consists of the following kernel sizes: $\{3 \times 1, 5 \times 1, 7 \times 1, 9 \times 1, 15 \times 1, 31 \times 1, 65 \times 1\}$. AutoBERT-Zero also proposed an operation-priority (OP) acquisition method to enhance the search performance of the Evolutionary search algorithm, as standard EL-NAS is prone to the risk of being trapped by a local optimum in a large search space. The searched model achieves a 1.4 higher GELU score than the BERT-base network on the SQuAD dataset [158] with fewer parameters.

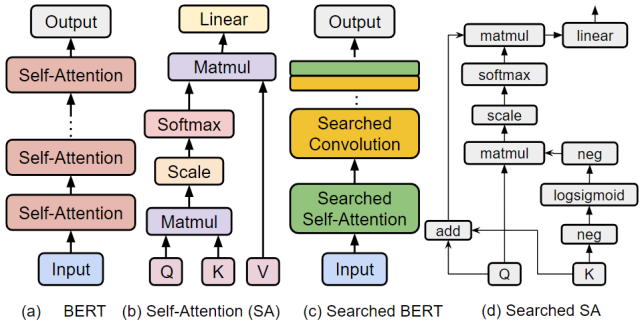


FIGURE 26. Comparison between BERT & searched BERT.

D. MAGIC

MitigAtinG InterferenCe (MAGIC) [162], as the name indicates, proposes two methods, MAGIC-T and MAGIC-A, to alleviate interference problems caused due to sampling different child models during the search process. The authors first investigate the reasons for this behavior and conclude that interference in a shared operation strongly correlates with the number of different operations in child models. This is due to receiving different gradient directions from sampled child networks with different topologies, even with the same set of training samples. Also, this phenomenon is particularly severe when multifaceted architectures exist in the search space, such as Convolution and Self-attention. MAGIC-T minimizes interference by sampling a child network that differs from the previously sampled child module with only one change in the architecture structure. In MAGIC-A, the neural architecture with the best validation accuracy among all child networks is selected as an anchor model to orient the child model's inputs and outputs together. The proposed methods are analyzed on Single-path NAS [163] using the following search space: MHSA with the number of heads {6, 8}, Convolution with kernel sizes {3, 5} and FFN. The searched model performs better than baselines RoBERTa-base [65] and ELECTRA-base [67] by 0.6 and 1.1 performance metric points, respectively, on GLUE test dataset.

E. LIGHTHUBERT

LightHuBERT [164] is a Once-for-all BERT search and compression framework to find desirable architectures by automatically pruning the model parameters of a large pre-trained network. Specifically, a Transformer-based Supernetwork is constructed to host several weight-sharing subnetworks followed by a two-stage distillation training using HuBERT [165] as a teacher model. The search elements are Embedding dimension, the number of heads, FFN ratio, and model depth, as detailed in Table 8. The Once-for-all student Transformer is the network with maximum dimensions on a given search space and trained using the pre-training distillation loss function. The student Transformer's trained weights are used in the second step, where subnetworks are randomly sampled at each forward pass during the Supernetwork training. Many subtransformers are selected and evaluated without fine-tuning, and a network that achieves the best validation metric within a given constraint is selected as the final model. LightHuBERT attains comparable performance to the baseline teacher model HuBERT on several ASR tasks with 29% fewer parameters.

TABLE 8. LightHuBERT [164] once-for-all search space.

	Small	Base
Embedding Dimension	{256, 384, 512}	{512, 640, 768}
Head Number	{4, 6, 8}	{8, 10, 12}
FFN ratio	{3, 3.5, 4}	{3.5, 4}
Network Depth	{10, 11, 12}	{12}
Params range	11-45 M	41 - 95 M
Total subnetworks	9.5×10^{11}	6.5×10^{9}

F. FAST BERT SEARCH

Tsai et al. [166] first profiled the computation time of basic components of the BERT-base [63] and MiniBERT [167] networks on a TPU V2 device. The authors also derived several useful conclusions about various operations impacting inference time. A novel sampling-based one-shot NAS method is developed to search for a fast and efficient model. The miniBERT model is 1.7× faster than the SOTA distilled BERT model with a negligible performance drop.

G. AQ-BERT

Automatic Mixed-Precision Quantized BERT (AQ-BERT) [125] is a Differentiable NAS method to assign different precisions to different encoder layers of a BERT and different bit-widths to different sub-groups with a single layer. AQ-BERT first constructs a Supernetwork of BERT, similar to DARTS, where each layer is divided into sub-groups, and each sub-group is populated with all possible bit-widths in the search space. The bilevel optimization follows DARTS, except that the architectural weights (α*) are updated after N_1 epochs during the Supernetwork training. The best sub-group and best precision choice from the best sub-group are chosen to be in the final searched network, thereby conducting pruning alongside quantization. The experiments on four NLP tasks demonstrate that the searched AQ-BERT model performs better than the SOTA method Q-BERT [168].

H. AUTORC

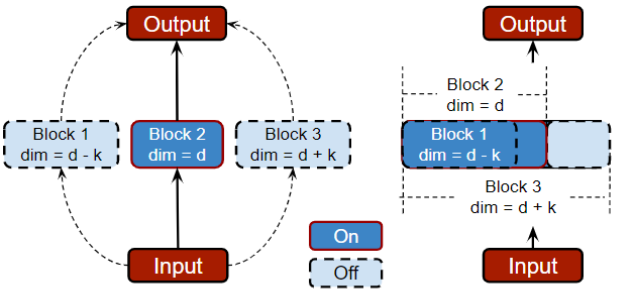
AutoRC [169], for Relation Classification (RC) application, exploits NAS for efficient BERT network design. RC task predicts the relation between any two entities in a sentence. The search space consists of common elements for BERT-based RC tasks, how to distinguish entities, how to use entity positional embedding, and entity pooling operations, and there exists 1.64 x 10^8 unique architectures in the entire search pool. The standard Reinforcement Learning (RL) search strategy is employed to search for an optimal BERT architecture on seven distinct datasets. The experiments on the benchmark dataset show that the searched BERT-RC model performs better than the classical BERT-based RC model quite significantly.

IX. NAS FOR VISION TRANSFORMERS

This section reviews prominent NAS methods for Vision Transformers on several vision applications such as Image Classification and Object Detection. Top-1 and top-5 accuracies are widely used model performance metrics for Image Classification. Top-1 accuracy indicates the percentage of testing samples correctly predicted by the network and top-5 percentage signifies the percentage of samples lying in the top-5 predictions of the network. While most methods target Image Classification on the ImageNet dataset, we also summarize a couple of ViTs for other applications. We classify any network as ViT if the network has at least one Self-attention layer and used for Computer vision application.

A. AUTOFORMER

AutoFormer [88] is a simple yet powerful technique based on the mix of One-shot NAS and Evolutionary search methods to find efficient Vision Transformer architectures. The large AutoFormer search space (Table 3) considers key dimensions of a Transformer, such as the number of heads, Embedding dimension, Q/K/V value, MLP ratio, and network depth. The traditional weight-sharing/One-shot NAS methods [99] decouple the weights of different operations in the search space into different paths, as shown in Fig. 27a. Therefore, this requires forward propagation through all the distinct paths in the Supernetwork and updating all individual weights in the backpropagation. However, the proposed Weight Entanglement in AutoFormer forces all weights to share in the form of a superweight of the highest dimension, as illustrated in Fig. 27b, similar to OFA [117]. Hence, the memory footprint is significantly reduced, and the Supernetwork requires only one forward and backward propagation through a single large weight matrix.



(a) Traditional Weight Sharing (b) Weight Entanglement

FIGURE 27. AutoFormer [88].

After training the Supernetwork, several subnetworks are obtained, and the Evolutionary search is performed to pick the optimal model. The objective function of the search process is to maximize accuracy while minimizing the network size. The searched AutoFormer-tiny/small/base architectures attain 74.7% /81.7% /82.4% accuracy on the ImageNet dataset with 5.7M /22.9M /53.7M model parameters, respectively. The searched models outperform the manually-designed Transformers (ViT [5] and DeiT [9]) and automatically searched CNN model (EfficientNet [70]) in terms of accuracy, number of parameters, and FLOPs.

B. VITAS

Vision Transformer Architecture Search (ViTAS) [92] is a search method based on ensemble contributions in search space, ViT token structure, and training methodology. The search space backbone is a pure Self-attention network, an extension of Twins [87] and DeiT [9] Transformer. The primitive search elements include main Vision Transformer dimensions such as the number of heads, patch size, output dimension of MHSA and FFN, and network depth. A detailed explanation of Twins and DeiT search spaces is provided in detail in Section IV-D1 (Tables 4 and 5). In original ViT, a class token of size p (equal to patch size) is appended to the patch tokens before passing it to the Transformer encoder. However, ViTAS introduces a private class token for each patch size p individually to avoid interaction between class tokens. The authors developed a new cyclic weight sharing method for token embedding to ensure more channel balance and contribute evenly towards all candidate architecture choices. ViTAS uses an Evolutionary search algorithm to find optimal Transformer architecture at a given target budget using weak augmentation and regularization for more steady training. The best-searched model achieved 84.0% accuracy on ImageNet on Twins Transformer search space, outperforming manually designed baselines Swin [6], Twins [87], and DeiT [9] Transformers.

C. TRAINING-FREE TRANSFORMER ARCHITECTURE SEARCH (TF-TAS)

Zero-Cost NAS Proxies [170] are substitutes for accuracy prediction to speed up the search process. TF-TAS [171] introduces a zero-cost proxy, DSS-indicator, to evaluate several ViT architectures at a low cost. The indicator is based on two theoretical components called synaptic diversity, synaptic saliency of MHSA, and MLP modules in the network. As input propagates through the depth of the ViT network, outputs gradually become the same, and rank converges to one. The synaptic diversity estimates the degree of rank collapse in an MHSA module. The synaptic saliency measures the amount of most important weights/parameters in an MLP unit. On AutoFormer search space of pure Self-attention modules, TF-TAS attains an acceptable performance and accelerates the NAS process by searching an optimal network in less than 0.5 GPU from 24 GPU days.

D. VIT-RESNAS

ViT-ResNAS [118] first manually designs a residual spatial reduction (RSR) module to reduce sequence length and increase the Embedding dimension for deeper layers. The search space is formed by adding an RSR unit at each stage of the ViT Transformer encoder, hence the name ViT-ResNAS. The residual spatial reduction unit reshapes a 2D feature map into a sequence using a multi-branch connection of Average pooling and Norm-Convolution operations. At each stage of ViT-ResNAS, the search algorithm finds the Embedding dimension and number of Transformer blocks. The search

elements for each block include the number of heads in MHSA and hidden size in FFN. Although the backbone network contains a Convolution operation (in the RSR module), the search space of ViT-ResNAS cannot be directly placed in the Hybrid Convolution-Attention search space as it does not search for Convolution operation and its dimensions. The second step in the search process involves two sub-steps: One-shot Supernetwork training multi-architectural sampling and the Evolutionary search method. The ViT-Res Supernetwork is enlarged to contain all possible combinations in the search space. For each ViT-Res Supernetwork iteration, multiple subnetworks are randomly sampled and trained with a batch of training samples with only one forward and backward propagation, thereby decreasing the training complexity of Supernetwork. The Evolutionary search method evaluates subnetworks from Supernetwork weights without further finetuning. The best network found from the evolutionary search process is trained till convergence. The experiments on ImageNet show that searched ViT-ResNAS model outperforms DeiT and ViT in terms of accuracy and MACs.

E. AS-VIT

Scaling up CNNs or ViTs is a process of designing a small base network on a limited resource budget and progressively increasing the size in terms of channel width, network depth, hidden dimension, or the number of attention modules to get better accuracy or model performance. For example, ResNet [1]/EfficientNet [70] families scaled from ResNet-18/EfficientNet-B0 to ResNet-200/EfficientNet-B7 to attain better accuracy. **As-ViT** [172] is a NAS framework to automatically search and scale up ViTs without training in a well-defined manner. While EfficientNet used NAS to search for a base CNN network, which is further scaled up manually, As-ViT proposed NAS for both searching the "seed" network and scaling up the base ViT network automatically. The search space consists of four stages and searchable elements: Kernel size, attention splits, and FFN expansion choice are listed in Table 9. The RL-NAS method is utilized to find the seed/base network, where best performing networks are iteratively sampled from search space. The controller is updated based on the fast-estimating Length Distortion parameter ($\mathcal{L}^E$) and Neural Tangent Kernel condition (k_Θ) instead of ViT accuracy, thereby avoiding full training of the sampled network. The seed network is scaled up in the training-free As-Vit search process using the same parameters for estimating the scale-up network's importance. The process starts with considering the seed model as the initial network, with only one attention block per stage and a hidden size of 32. The search process once again calculates $\mathcal{L}^E$ and k_Θ values after adding each choice (width, depth, and expansion ratio) to the base network. The network with the highest $\frac{\mathcal{L}^E}{k_\Theta}$ value is chosen as the best model in the current iteration. This process is repeated until the desired number of parameters in the scaled-up network are attained. The end-to-end process of searching the seed network and automatically scaling it up for a specific parameter count requires only 12 V100 GPU hours (5 for seed search and 7 for scaling up). As-ViT delivers strong top-1 accuracy (83.5%) on ImageNet for Image Classification and mAP (52.7%) on the COCO dataset for Object Detection, outperforming several manually designed and automatically searched ViT architectures.

TABLE 9. As-ViT search space [172].

Stage	Sub-space	Choices
1	Kernel K1	4, 5, 6, 7, 8
	Attention Splits S1	2, 4, 8
	FFN Expansion E1	2,3,4,5,6
2	Kernel K2	2, 3, 4
	Attention Splits S2	1, 2, 4
	FFN Expansion E2	2, 3, 4, 5, 6
3	Kernel K3	2, 3, 4
	Attention Splits S3	1, 2
	FFN Expansion E3	2, 3, 4, 5, 6
4	Kernel K4	2, 3, 4
	FFN Expansion E4	2, 3, 4, 5, 6
-	Number of Heads	16, 32, 64

F. SEARCHING FOR SEARCH SPACE (S3-NAS)

S3-NAS [173], as the name implies, automates both search space and search process, unlike previous methods whose search space is manually constructed. The overall search space of ViT, which includes Embedding dimension, network depth, MLP expansion ratio, number of heads, window size, and QKV FC size, is decomposed into several dimensions. The search space progressively evolves in each dimension, guided by a measurement called E-T Error. The search process on searched space is a two-step process of first training a Supernetwork without any constraints, similar to AutoFormer, and applying Evolutionary search, similar to SPOS [163], with accuracy, model size, and FLOPs as constraints. The best S3-NAS-searched network achieves 84.7% accuracy on ImageNet and outperforms ViT [5], Swin [6], and DeiT [9] in terms of model performance, size and FLOPs. The searched network also shows great performance when transferred to other tasks such as Object Detection, Semantic Segmentation, Visual Question Answering.

G. NASFORMER

NASformer [174] first proposed a structural change to Vision Transformer by introducing a dilated window-based Self-attention, inspired by Dilated Convolution [175], to fetch hierarchical feature representations in a parameter-free approach. The windowing approach presented in Swin Transformer [6] could exchange information only from the neighboring windows. However, the dilated window is designed to cover distant tokens as far as possible rather than only local tokens. The backbone network for searching an architecture is a pure Self-attention-based ViT with the dilated window technique. The end-to-end network is divided into different stages where searchable parameters are the number of blocks and the Embedding dimension. The block-level search space includes window size, number of MHSA heads, and QKV dimensions. The search process is similar to One-for-all [117], where a Transformer Supernetwork with maximum

dimensions is trained, followed by an Evolutionary search to find an optimal submodel. The best NASformer searched model attains an accuracy of 83.9% on ImageNet, performing better than ViT [5], Swin [6], Twins [87], and DeiT [9].

H. GLOBAL LOCAL IMAGE TRANSFORMER (GLIT)

GLiT [91] is the first work to apply the principles of NAS on hybrid Attention and Convolution search space for Vision applications. GLiT designed a new module called Global-local (GL) block, a combination of **(i)** Self-Attention: to model the global dependencies and **(ii)** Convolution bundle: to capture the local representations. Similar to standard ViT, where Self-Attention modules are stacked to form final architecture, the GL blocks are stacked "*M*" times to form an end-to-end GLiT network, as shown in Fig. 28.

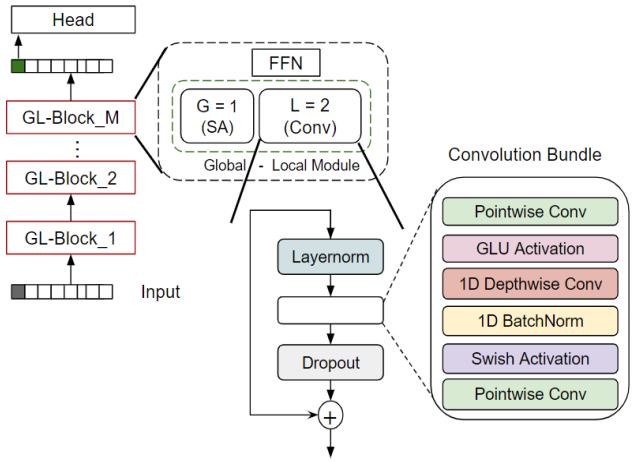


FIGURE 28. GLiT backbone [91].

The SA and Convolution bundle operate in parallel, equivalent to SAs in a conventional MHSA module, and total number of SA units (*G*) and Convolution bundles (*L*) for any given layer cannot exceed four. The search task now boils down to finding optimal combination of SA and Convolution bundle in each layer from the following pool of (G, L) choices: {(0,3), (1,2), (2,1), (3,0)}. The total search space is summarized in Table 10. The two-level search process, i.e., searching for optimal number of SA and Convolution bundles (G, L) and their respective hyperparameters (k, e, d_k, d_z), is broken down using hierarchical search method of a separate Evolutionary process. GLiT-Small and GLiT-Base achieve up to 80.5% and 82.3% accuracy, respectively, and outperform the baselines DeiT [9] and ViT [5], with an increase in the number of parameters and FLOPs reduction.

TABLE 10. GLiT search space [91].

(G, L)	(0, 3), (1, 2), (2, 1), (3, 0)	
Local Submodule (Convolution bundle)	Kernel Size (k)	17, 31, 45
	Expansion Ratio (e)	1, 2, 4
Global Submodule (Self-Attention block)	Feature Dimension (d_k)	96, 192, 384
Feed-Forward Network (FFN)	Expansion Ratio (d_z)	2, 4, 6

I. HIGH-RESOLUTION NAS (HR-NAS)

HR-NAS [176] aims to search for a model that takes a high-resolution image as input, using a multi-branch search space incorporating both Convolutions and Self-attention components. The search space, inspired by HRNet [177], contains multi-scale features and global contexts at the same time while preserving high-resolution representation through the model. Each branch in the HR-NAS Supernetwork is a chain-like search for searchable blocks of Convolution and attention at different resolutions. The same search space can be used for several dense prediction tasks, such as Semantic Segmentation and Instance Segmentation, of different granularities as each branch is specialized for a specific feature resolution. The authors designed a lightweight Transformer, shown in Fig. 29, whose computational complexity is dynamically adjustable for different budget constraints. The Transformer module consists of an encoder and a decoder, similar to Vanilla Transformer. In the projector unit, the input feature map is concatenated with a 2D positional map instead of a sinusoidal encoding. The feature map is transformed into a set of *n* tokens and fed to the MHSA unit for Self-attention.

The Supernetwork is a multi-branch model, where each branch is composed of a series of searchable blocks, where each block can take either a MixConv [178] or lightweight Transformer unit. The proposed resource-aware search process, a blend of DARTS-like method and progressive shrinking strategy, explores optimal combinations at each stage of the network. The Supernetwork jointly learns network weights and architectural parameters (α), which guides the search process. The search algorithm also discards Convolution channels, and Transformer unit queries progressively from the Supernetwork. As a result, HR-NAS [176] is able to search for networks for various tasks such as Image Classification, Semantic Segmentation, Human Pose Estimation, and 3D Detection. The searched model of HR-NAS performs better than several CNN based searched models such as ProxylessNAS and FBNet.

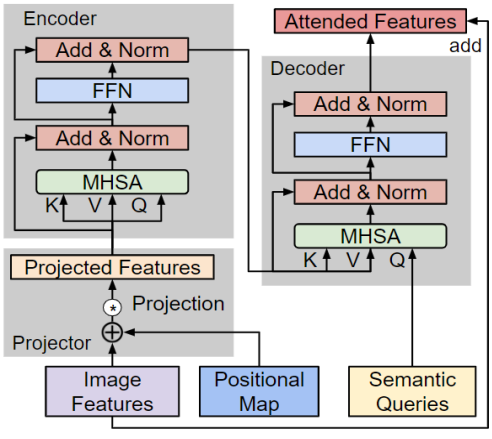


FIGURE 29. Lightweight transformer in HR-NAS [176].

J. BOSSNAS

Block-wisely **s**elf-supervised **N**eural **A**rchitecture **S**earch (BossNAS) [179] is an unsupervised NAS method on hybrid

Convolution-Transformer (HyTra) search space. The Unsupervised NAS (UnNAS) methods [180] search for an efficient network without using any human-annotated labels and therefore relying only on the input data (Eg: Images). Block-wise search weight-sharing methods divide the end-to-end network into blocks to individually optimize, thereby reducing the weight-sharing search space. The authors propose ensemble bootstrapping, a self-supervised scheme to optimize each block, where sampled networks are trained to estimate the probability of all sampled models. The search space consists of two fundamental units: Residual bottleneck (ResConv) from ResNet [1] and ResAtt, a modified BoTBlock module [181], formed by replacing the relative position encoding branch with a lightweight Depthwise Separable Convolution. The searched BossNAS model on the hybrid search space attains 82.5% top-1 accuracy on ImageNet, exceeding EfficientNet [70] by 2.4%.

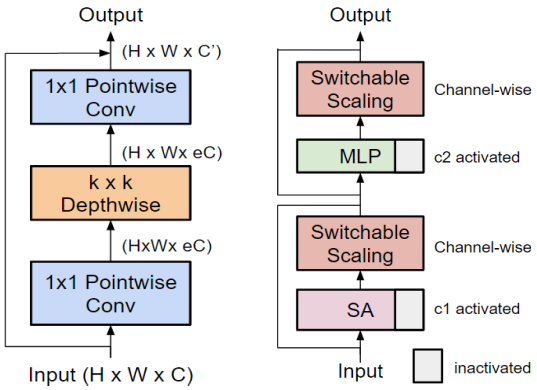


FIGURE 30. NASViT [93].

K. NASVIT

NASViT [93] is a One-shot based NAS method for efficient hybrid Convolution-Attention network search. The main contribution of this technique lies in resolving an issue caused by the Supernetwork training. The issue is that the gradients of different subnetworks within a Supernetwork conflict with each other more strongly in ViTs than in CNNs, leading to early saturation and slow convergence. NASViT overcomes this issue by using a projection gradient algorithm to identify and remove the sub-network gradient component causing gradient conflict. A switchable channel-wise scaling is also added in the Transformer layer (Fig. 30b) to reduce overlapping among different Self-attention blocks and lessen gradient conflict. Finally, a weak data augmentation method reduces the training regularization and optimization difficulty to reduce gradient conflicts. The search space backbone is a sequential interconnection of Mobile Inverted Residual block (MBConv) and MHSA unit (Self-attention + MLP). While the positions for MBConv units (Fig. 30a) and Transformer blocks (Fig. 30b) are fixed, their hyperparameters such as depth and block dimensions are searched, as summarized in Table 6 and explained in detail in Section IV-D2. The experiments on several vision tasks such as Image Classification and Semantic Segmentation show that the searched NASViT models outperform several hand-designed CNNs such as ResNet50 in terms of accuracy and FLOP count.

L. BURGERFORMER

Yang et al. proposed BurgerFormer [182], a novel search space built in three hierarchy levels: Micro, Meso, and Macro levels. As shown in Fig. 31, the backbone network is divided into four stages, and each stage consists of a varying number of Transformer-like blocks. Each stage is a series of several primitive operations, including Norm-Op-Norm-Act to be searched from the search space, where Op comprises Convolution and Self-attention operations. The Micro, Meso, and Macro classification corresponds to the granularity of operation type, Transformer style block structure, and composition of each stage, respectively. The micro (low) level space includes operations (OP), norm and activation functions, while the meso (intermediate) level finds the optimal interconnection or combination of these three elements.

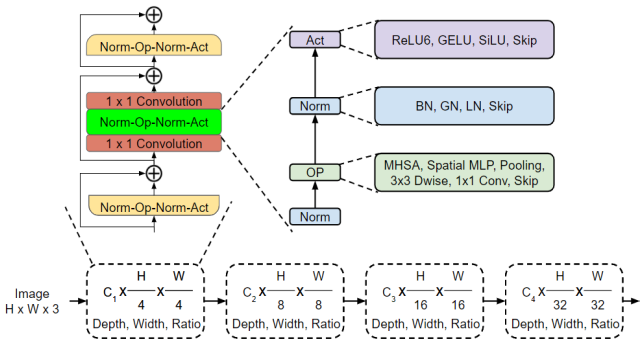


FIGURE 31. BurgerFormer search space [182].

On the macro (high) level, each stage's depth, the activation's channel width, and expansion ratio between the two 1×1 Convolutions are searched as a whole, as shown in Table 11. The definition of Micro and Macro used in BurgerFormer differs from the traditional definitions we defined in the search space section (Section IV-C). BurgerFormer utilizes Single path One-shot search [163] and Sandwich sampling [183] methods to train Supernetwork and Evolutionary Search to evaluate subnetworks. The best BurgerFormer searched model achieves 83% accuracy on ImageNet, thereby outperforming AutoFormer, GLiT, and ViTAS.

TABLE 11. BurgerFormer search space.

Stage	Depth	Width	Ratio
1	{1, 2, 3, 4}	{32, 64, 96}	{1,2, 4, 6}
2	{1, 2, 3, 4}	{64, 96, 128}	{1,2, 4, 6}
3	{1, 2, 3, 4, 5, 6, 7, 8}	{128, 192, 256, 320}	{1,2, 4, 6}
4	{1, 2, 3, 4}	{128, 256, 384, 512}	{1,2, 4, 6}

M. UNINET

UniNet [184] is an RL-based NAS method to jointly search the optimal combination of Convolution, Self-attention, and MLP layer, along with their depth and channel dimension on a macro search space. It considers the three operations in its search space to leverage the advantages of each type at different stages of the network, such as Convolutions for capturing local dependency, MHSA to extract global and long-range representation, and MLPs for high utilization and efficiency. The MLP layer refers to the MLP-Mixer [185] style module to capture spatial features. The backbone of UniNet is a pile of five stages, where each stage consists of General Operators (GOPs) and Down Sampling Modules (DSMs) units, as depicted in Fig.32a. The authors designed three DSM candidates, namely, (i) Local-DSM (L-DSM) of only 2D Convolution, (ii) Local-Global-DSM (LG-DSM) with MHSA and 2D Convolution, and (iii) Global-DSM (G-DSM) with only MHSA and a 1D Convolution. Each stage of the initial network is initialized with a certain repeat number (r), Convolution channel size (c), and initial expansion ratio (r) and progressively increased or decreased during the search process with choices shown in Fig. 32 (a).

The search algorithm jointly optimizes both model accuracy and FLOPs by formulating the reward function as $r(m) = a(m) \times \frac{t^{\alpha}}{f(m)}$, where $a(m)$ and $f(m)$ are the predicted accuracy and FLOPs of model m, respectively, t is the target FLOPs, and α is the weight factor to balance accuracy and FLOP cost. Even though a network is searched directly on a large-scale ImageNet dataset, a proxy setting is implemented by training the sampled architecture for only five epochs, and validation accuracy for reward function is calculated on the partially trained network. Fig. 32b shows the base searched network with its dimensions, which is scaled up using EfficientNet's [70] compound scaling method. The experiments on Image Classification, Object Detection, and Semantic Segmentation show that the searched model outperforms manually designed CNNs and Self-attention based ViTs.

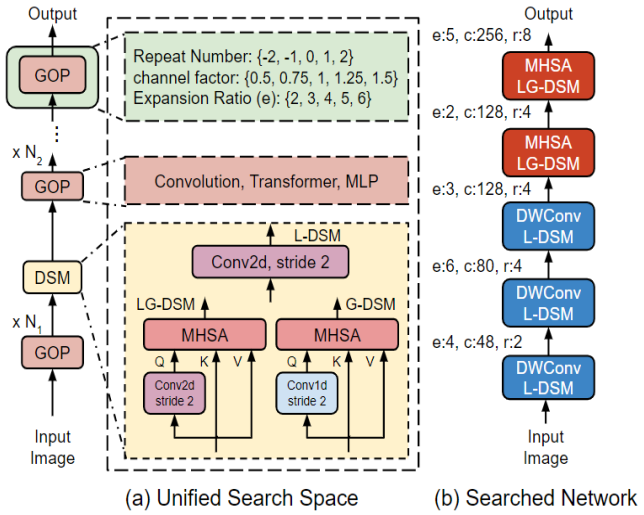


FIGURE 32. UniNet search space [184].

N. SPVIT

Single-**P**ath **Vi**sion **T**ransformer pruning (SPViT) [186] is a One-shot method to automatically compress a pre-trained ViT model. SPViT prunes the costly MHSA into a lightweight operation and searches for MLP expansion ratios under different performance constraints. The main contribution of this work lies in designing a novel weight-sharing technique between Convolution and MHSA to encode all candidate choices into a single MHSA layer instead of a multi-path space, as shown in Fig. 33. The output of Convolution can be derived by indexing the intermediate results of MHSA, thereby aiding search process to ease computation burden with less trainable parameters. Also, learnable binary gates ($g_{|p|}$, $g_{|p|-1}$, ... g_1) are introduced to encode candidate choices, which are jointly trained along with model parameters to find the configuration of each layer. The gates determine the significance of each FNN dimension and prune trivial dimensions. The automatic pruning experiments on the DeiT model [9] show that 56.2% of the FLOPs can be reduced with only a 0.3% of loss in accuracy on ImageNet.

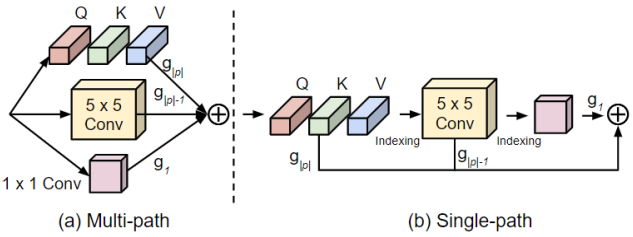


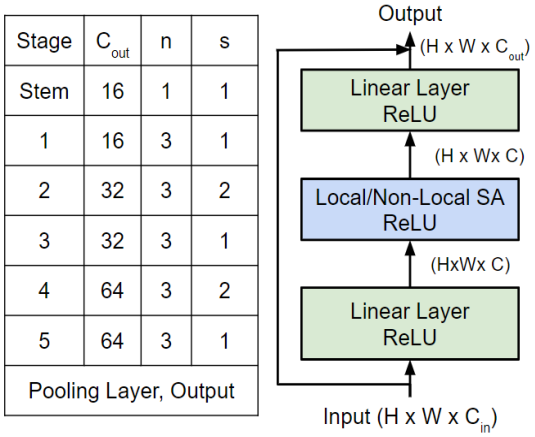
FIGURE 33. SPViT [186].

O. AUTO-REGRESSIVE NAS

Zhou et al. [187] proposed a fully Self-attention based, self-supervised search method with context auto-regression. The layer-wise search space consists of five stages with fixed stride, input, and output channels, and each stage searches for either Local Multi-head Self-attention (Local-MHSA) [188] or Non-local Self-attention [189]. The search elements of Local-MHSA are spatial extent: {3, 5, 7} and the number of heads: {4, 8}. The backbone network and search space are detailed in Fig. 34. The search process is formulated as a differentiable/One-shot NAS where a Supernetwork is constructed with all possible combinations of Local Self-attention modules and trained in a bi-level fashion to find a good network. The search phase is accomplished by a self-supervised search method with context auto-regression to guide architecture search, followed by a fine-tuning phase on the target task. The best model attained 73.3% top-1 accuracy on ImageNet with 4M parameters, better than DeiT [9].

P. VTCAS

Vision Transformer with Convolutions Architecture Search (VTCAS) [190] is a One-shot NAS method based on Progressive DARTS (P-DARTS) [191] on a hybrid search space. The DARTS method [99] first finds a base cell architecture on



$\mathbf{C}_{out}$: Output Channels, **n**: Stage depth, **s**: Stride

FIGURE 34. Auto-regressive NAS [187] $\mathbf{C}_{out}$: Output channels, n: stage depth, s: Stride.

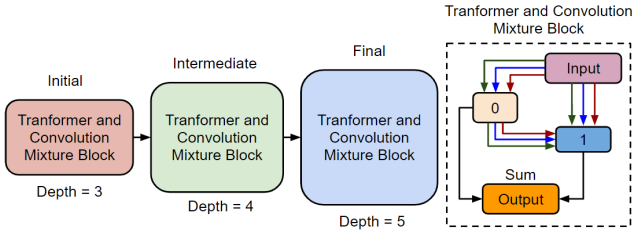


FIGURE 35. VTCAS [190].

a small-scale proxy dataset such as CIFAR-10 and transfers the same searched topology on a large-scale dataset such as ImageNet by increasing the network depth. This straightforward transfer of searched architecture on the large-scale data causes a performance gap when increasing overall network depth. P-DARTS resolves this issue by simultaneously dropping candidate choices in the search space and progressively increasing the network depth in a series of steps, unlike DARTS, which drops all candidate choices in Supernetwork in a single training step on proxy task. The initial Supernetwork consists of three Transformer and Convolution Mixture blocks, inspired by Swin Transformer [6], and seven candidate choices on each node. The depth of the mixture block is gradually increased to four and five in three successive steps, as shown in Fig. 35, while the number of candidate operations is reduced to five, three, and one in those three steps. The searched VTCAS network achieves 82.0% accuracy on ImageNet and 50.4% mAP on the COCO dataset, surpassing many manually designed CNN and Transformer architectures.

Q. αNAS

αNAS [192] is an Evolutionary search-based approach where the novel property-guided synthesis procedure directs network mutations. The authors developed a principled way of overcoming the limitations posed by prior Evolutionary search algorithms. The previous techniques apply stochastic mutations to neural architectures to produce a new model. Each such mutated network is trained and validated to insert into the population that acts as the beginning for future mutations. Contrary to traditional methods, αNAS considers a significantly large search space and make high-quality changes in each mutation by inferring from a set of program properties. A random submodule is considered from a large network, and a stochastic mutation is applied to the properties of the submodule. The new submodule is synthesized that meets the mutated properties and adds the new submodule in place of the old one. αNAS is applied on the original ViT-based search space and attains better accuracy than ViT with 30% fewer FLOPS and parameters.

R. FBNETV5

FBNetV5 [193] is a framework to search for architectures on diverse Computer vision tasks such as Image Classification, Semantic Segmentation, and Object Detection in a single run. Specifically, the authors first designed a flexible and easily transferable search space followed by a search process that is adaptable to different tasks by not adhering to a single task's training pipeline. The search method is applied on a Transformer style model only for the Semantic Segmentation application. FBNetV5 proposed Lite MaskFormer, a modified version of MaskFormer [194], to use as the backbone network in the search process. The results on the ADE20K dataset [195] show that the searched network outperforms manually designed architectures such as ResNet-50 [1] and Swin Transformer [6].

S. T2IGAN

Generative Adversarial Networks (GANs) [196] are unsupervised Neural Networks that automatically discover input data patterns to generate new realistic natural images. A typical GAN has a Convolution-based Generator and a Discriminator network playing against each other. Transformers are being used to replace the Convolution operation in the traditional Generator and Discriminator for better performance [197]. T2IGAN [198] is the first method to use NAS principles for efficient Transformer-based GAN architecture design on Text-to-Image (T2I) task to generate a realistic image from the input text description. It uses a cell-based search space and finds an efficient topology using the RL-based search strategy on a Convolution and Lightweight Transformer MHSA pool of primitive elements. The results on CUB-200 Birds [199], Oxford-102 Flowers [200], and COCO [201] datasets show that the searched model surpasses many hand-designed GAN models.

T. TRANSFORMER-BASED NAS PREDICTOR (TNASP)

Previously, we discussed how different NAS methodologies were developed to search for efficient Transformer models for different applications. Now, we explore the opposite context, i.e, employing Transformers to aid NAS methods in performance estimation of CNN models. TNASP [202], a Transformer model-based NAS predictor, utilizes MHSA

TABLE 12. Summary of vision transformer NAS methods on ImageNet dataset.

Searched Model	Top-1 Accuracy (%)	Top-5 Accuracy (%)	Number of Parameters (M)	FLOPs (G)	Search Space	Search Method	Ref.
AutoFormer-tiny	74.7	92.6	5.7	1.3	Transformer	SuperNetwork Training + Evolutionary Search	[88]
AutoFormer-small	81.7	95.7	22.9	5.1			
AutoFormer-base	82.4	95.7	54	11			
ViTAS-Twins-tiny	79.4	94.8	13.8	1.4	Transformer	SuperNetwork Training + Evolutionary Search	[92]
ViTAS-Twins-small	82.0	95.7	30.5	3.0			
ViTAS-Twins-base	83.5	96.5	66.0	8.8			
ViTAS-Twins-large	84.0	96.9	124.8	16.1			
ViTAS-DeiT-A	75.6	92.5	6.6	1.4			
ViTAS-DeiT-B	80.2	95.1	2.3	4.9			
TF-TAS-tiny	75.3	92.8	5.9	1.4	Transformer	Zero-Cost Proxy	[171]
TF-TAS-small	81.9	95.8	22.8	5.0			
TF-TAS-base	82.2	95.6	54.0	12.0			
ViT-ResNAS-tiny	80.8	–	41	1.8	Transformer	SuperNetwork Training + Evolutionary Search	[118]
ViT-ResNAS-small	81.7	–	65	2.8			
ViT-ResNAS-medium	82.4	–	97	4.5			
As-ViT-small	81.2	–	29.0	5.3	Transformer	Reinforcement Learning	[172]
As-ViT-base	82.5	–	52.6	8.9			
As-ViT-large	83.5	–	88.1	22.6			
S3-NAS-tiny	82.1	95.8	28.1	4.7	Transformer	SuperNetwork Training + Evolutionary Search	[173]
S3-NAS-small	83.7	96.4	50	9.5			
S3-NAS-base	84.0	96.6	71	13.6			
NASformer-tiny	82.3	95.8	31	4.6	Transformer	SuperNetwork Training + Evolutionary Search	[174]
NASformer-small	83.7	96.4	50	9.5			
NASformer-base	83.9	96.6	71	13.6			
GLiT-tiny	76.3	–	7.2	1.4	Hybrid Transformer & Convolution	SuperNetwork Training + Evolutionary Search	[91]
GLiT-small	80.5	–	24.6	4.4			
GLiT-base	82.3	–	96.1	17			
HR-NAS-1	75.7	–	7.2	5.5	Hybrid Transformer & Convolution	SuperNetwork/ One-Shot	[176]
HR-NAS-2	76.5	–	24.6 96.1	6.4			
BossNAS-T0 w/o SE	80.5	95.0	–	–	Hybrid Transformer & Convolution	One-shot+ Unsupervised NAS	[179]
BossNAS-T0	81.6	95.6	–	–			
BossNAS-T1	82.2	95.8	–	–			
NASViT-1	78.2	–	–	0.20	Hybrid Transformer & Convolution	SuperNetwork Training + Evolutionary Search	[93]
NASViT-2	79.7	–	–	0.30			
NASViT-3	80.5	–	–	0.42			
NASViT-4	81.4	–	–	0.59			
NASViT-5	81.8	–	–	0.75			
NASViT-6	82.9	–	–	1.88			
BurgerFormer-tiny	78.0	93.7	10	1.0	Hybrid Transformer & Convolution	SuperNetwork Training + Evolutionary Search	[182]
BurgerFormer-small	80.4	95.0	14	2.1			
BurgerFormer-base	82.7	96.2	26	3.9			
BurgerFormer-large	83.0	96.8	36	6.5			
UniNet-1	80.8	–	11.5	1.1	Hybrid Transformer, Convolution & MLP	Reinforcement Learning	[184]
UniNet-2	82.5	–	16.2	2.2			
UniNet-3	83.5	–	24	4.3			
SPViT-Swin-tiny	80.1	95.0	25.8	3.4	Hybrid Transformer & Convolution	One-shot/ Differentiable	[186]
SPViT-Swin-small	82.4	96.0	39.2	6.1			
SPViT-Swin-base	83.2	96.3	68.0	11.4			
Auto-regressive NAS	73.3	91.6	4	–	Hybrid Transformer & Convolution	One-shot/ Differentiable	[187]
VTCAS	82.0	–	31.2	5.2	Hybrid Transformer & Convolution	One-shot/ Differentiable	[190]

to map discrete architectures to a relevant feature representation and applies Laplacian matrix linear transformation as the positional encoding to strengthen the representation of topology information. This method consists of an encoding phase to encode the operations and positions in a DAG into a continuous representation, accompanied by three Transformer encoder layers (MHSA layers) and a regressor to obtain the final prediction using the output features of Transformer layers. The performance estimator learns the relation between a network in the search space and its validation

accuracy, which is used to predict the model accuracy of an unseen network in the same search space. The advantages of using Self-attention styled modules are listed by the authors as follows: (i) the MHSA can explore better features from the structured graph data, and (ii) also, due to its nature of modeling global dependencies, MHSA can enhance the encoding capability at different distant positions. TNASP performs better than previous encoding methods on NAS-Bench-101 [104], NAS-Bench-201 [203], and DARTS [99] search spaces.

Vision Transformer NAS Summary. A detailed overview of all Vision Transformer NAS Methods on the ImageNet dataset is provided in Table 12. As we reviewed several ViT-based NAS strategies, we observed that the search space plays an equal and important role as the search method in finding efficient Transformer models. The majority of NAS methods draw inspiration from methods developed for CNNs, such as One-shot NAS and Evolutionary search, and are tweaked to customize for Transformers. However, the search elements and backbone search space we reviewed in this paper differ in most ViT NAS methods. A given NAS method may perform better on a given search space and may not be well transferable to other search spaces. For a fair comparison between two search algorithms, the search space should be identical, including the total number of search choices. The initial Reinforcement or Evolutionary NAS algorithms are expensive for model search as every network sampled by the controller needs to be trained end-to-end until convergence. Also, the Supernetwork-based technique requires significant GPU memory to load and train all the operations. A few methods like As-ViT [172] and UniNet [184] use approximation or proxy task settings by utilizing the properties of the Vision Transformer, thereby making the search faster. Hence, in the future, priority should be given to zero-cost proxy methods for efficient network design to make NAS approachable.

X. HARDWARE-AWARE NAS FOR TRANSFORMERS

In the previous sections, we extensively discussed NAS methods pertaining to Vanilla Transformers, BERT models, and Vision Transformers without considering a target hardware platform. This section reviews several Hardware-aware multi-objective Transformer NAS methods for efficient inference on several devices.

A. HARDWARE-AWARE TRANSFORMERS (HAT)

HAT [14] is one of the early methods for Hardware-aware Transformer search, targeting NLP tasks on Raspberry Pi ARM CPU, Intel Xeon CPU, and Nvidia TITAN Xp GPU. The search space of HAT breaks the traditional Vanilla Transformer in the following ways: **(i) Arbitrary Encoder-Decoder Connection:** The Transformer is searched in such a way that the decoder block can choose connections from one or multiple encoder layers; unlike the Vanilla Transformer, where only one encoder output is connected to the decoder, and **(ii) Heterogeneous Transformer Layers:** The encoder/decoder is more elastic in the sense that each encoder or decoder layer can have different dimensions (key, head number). The traditional CNN-based Hardware-aware NAS methods, such as ProxylessNAS [101] and FBNet [106], train a Supernetwork specific to the target hardware platform and sample an optimal architecture. However, HAT trains only one SuperTransformer (or Supernetwork) and derives different networks suitable for different devices (MCU, CPU, GPU). HAT performs an Evolutionary search with the target latency constraints on the trained SuperTransformer, thereby picking only the models with latency smaller than the target constraints. The HAT-searched model for Raspberry Pi-4 on the WMT'14 translation task attains thrice speed with 3.7x fewer parameters over Vanilla Transformer. The key takeaway from HAT is that a model specialized for one hardware platform may not be optimal for other devices. For example, Table 13 shows that the model searched with respect to GPU runs faster on GPU than CPU and vice versa at the same BLEU score.

TABLE 13. HAT latency evaluation on GPU and CPU.

Network Specialized for	BLEU Score	GPU Latency (ms)	ARM CPU Latency (ms)
HAT- GPU	28.10	**147**	6491
HAT- ARM CPU	28.15	184	**6042**

B. SHIFTADDNAS

Multiplication-free Neural Network boosts hardware performance by replacing the expensive matrix multiplication such as Convolutions or Transformers with the addition-only operation in AdderNet [204], or a mix of bitwise shift and addition in ShiftAddNet [205]. However, merely replacing the matrix multiplications with multiplication-free operations can hurt model performance despite an increase in hardware performance. Therefore, ShiftAddNAS [206] proposed to search for a hybrid multiplication-based (Vanilla Convolution and Self-attention) and multiplication-free (Shift and Add) network by incorporating both kinds of blocks in the search space. Table 14 summarizes the search space for the NLP task, consisting of block type and key Self-attention dimensions. The vision task search space is given in Table 15, consisting of the type of block to be searched and the depth of each block.

The Convolution weights follow Gaussian distribution, whereas the weight parameters in multiplication-free addition operation follow Laplacian. Hence, naively including both types of operations leads to an inconsistent architecture ranking in the One-shot NAS algorithm. Therefore, the authors developed a novel search method with improved weight sharing capability by including KL-divergence loss in addition to the traditional cross-entropy loss function. The searched-ShiftAddNAS models on WMT'14 En-Fr and WMT'14 En-De NLP datasets outperform the baseline Vanilla Transformer, Evolved Transformer, and HAT networks in terms of latency, energy, and BLEU score on the

TABLE 14. ShiftAddNAS: search space for NLP tasks SA: self-attention, Conv: convolution.

Encoder Type	[SA, SA + Conv, SA + Shift] [SA + Add, Conv, Shift, Add]
Decoder Type	[SA, SA + Conv] [SA + Shift, SA + Add]
#Decoder blocks	[6, 5, 4, 3, 2, 1]
Elastic Embedded dim.	[1024, 768, 512]
Elastic head dim.	[16, 8, 4]
Elastic MLP dim.	[4096, 3072, 2048, 1024]
Arbitrary SA	[3, 2, 1]

TABLE 15. ShiftAddNAS: search space for CV tasks.

Block Type	[SA, Conv, Shift, Add]
# 56^2 x 128 blocks	[1, 2, 3, 4]
# 28^2 x 256 blocks	[1, 2, 3, 4]
# 14^2 x 512 blocks	[3, 4, 5, 6, 7]
# 7^2 x 1024 blocks	[4, 5, 6, 7, 8, 9]

Eyeriss accelerator [207]. Also, the searched-ShiftAddNAS network performs better than ResNet50 and other searched ViTs on ImageNet.

C. LIGHTWEIGHT ITERATIVE NAS (LINAS)

Lightweight Iterative NAS [208] is designed to accelerate the subnetwork search phase post the Supernetwork training, as the validation component in One-shot NAS methods comes with a huge computational cost, especially on large datasets. Besides, the main contribution of LINAS is the a generalizable framework to offer support for various search methods and model performance predictors in a multi-objective setting across multimodal environments. LINAS also showed that the Evolutionary search algorithm and Sequential Model-based Optimization (SMBO) methods pair well with One-shot/weight-sharing search spaces. In a nutshell, the validation phase of LINAS is summarized as follows:

1) Randomly sample subnetworks as initial population
2) Validate the subnetworks to fetch accuracy
3) Train the predictors with the validated networks
4) Run the search algorithm with trained predictors
5) Pick only the best subnetworks from the updated pool
6) Validate a few subnetworks and retrain the predictor
7) Repeat steps 2 to 6 until LINAS finds the best network

The Supernetwork for the proposed method evaluation in LINAS is inspired by HAT [14]. Finally, various performance predictors, such as Ridge Regression, Support Vector Machine regression, and stacked regression, are compared with each other, showing the effectiveness of LINAS.

D. LIGHTSPEECH

LightSpeech [209] is an HW-NAS framework for lightweight Transformer search targeting Text to Speech models. The search space is adopted from FastSpeech [210], with four feed-forward Transformer (FFT) blocks in both encoder and decoder. As shown in Fig. 36, each FFT unit is made up of MHSA and Separable Convolution (SepConv) operation [211]. The search candidates for LightSpeech are {2, 4, 8} for number of heads in MHSA and {1, 5, 9, 13, 17, 21, 25} for kernel size in SepConv. The search process relies on the Gradient Boosting Decision Tree (GBDT) method [212], where an accuracy predictor is leveraged to train on pre-collected samples of architecture-accuracy pairs. The searched Transformer comprises an encoder with SepConv of kernel sizes 5, 13, and 25, and a decoder with kernel sizes 9, 13, and 21. The evaluation results on Intel Xeon CPU E5-2690 v4 show that the searched network is 6.5x faster than the baseline FastSpeech model [210] while preserving voice quality metrics.

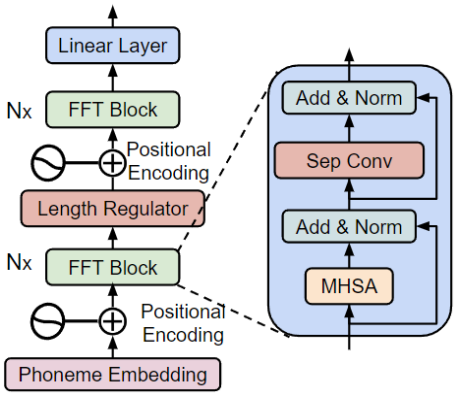


FIGURE 36. LightSpeech search space [209].

E. LITETRANSFORMERSEARCH

LiteTransformerSearch (LTS) [213] performs a zero-cost proxy or training-free Evolutionary search directly on constrained hardware. The authors empirically established a high correlation between final model validation accuracy and the number of parameters in the decoder layer (non-embedding parameter count), thereby utilizing decoder parameter count as a strong substitute in the search process. Also, 200 randomly sampled architectures from the search space are trained on WikiText-103 and (LM1B) datasets to verify correctness. LTS obtains the real latency values by measuring directly on the target hardware during the search process instead of using a Look-up-table or latency predictor. The search space can be expanded without constraints as the search method does not rely on any Supernetwork. It supports all the internal dimensions of a decoder in a heterogeneous manner, i.e., different hyperparameters for different layers in the network. The primitive element set comprises the number of decoder layers, the number of heads, output dimension of the decoder, inner dimension of FFN, embedding dimension, and division factor in adaptive input embedding. The search is performed over the entire Pareto-front, spanning across a wide range of latency and accuracy spectrum. At each search iteration, a set of models are sampled from the current Pareto line and fed to the Evolutionary search algorithm to predict a new set of networks. The accuracy is estimated

through proxy metrics and latency on the target hardware. The Pareto-frontier is updated based on a new set of neural architectures. This iterative process is repeated until an optimal model is found at different latencies. The search experiments on ARM CPUs, Intel Core i7, and Nvidia Titan Xp GPU show that the searched model is twice faster than Vanilla Transformer with comparable accuracy.

F. SUPERSHAPER

The standard BERT-base model has a fixed hidden size of 768 across all the layers of the Transformer encoder (Fig. 37a). However, the task of SuperShaper [214] is to find the optimal hidden dimensions of each encoder layer and sublayers separately, which is the only component in its search space. For example, Fig. 37b shows an example of a searched BERT model, where green, blue and yellow colors indicate the height and width of the searched Add & Norm, Feed Forward, and MHSA modules, respectively. The intuition behind choosing this parameter is that the different shape of each layer contributes uniquely to model performance. Also, for a given parameter count, a model that meets the requirement of a device, the shape is a free variable to be optimized for higher accuracy as latency on CPUs and GPUs are insensitive to the shape of the network. The authors adopt Evolutionary search algorithm to find the optimal shape of each layer that meet accuracy, the number of parameters, latency constraints on 1080Ti and K80 GPUs, and a server class single-core Xeon CPU. The SuperShaper Supernetwork is trained irrespective of a downstream task while fine-tuned on a specific application. The Supernetwork training and subnetwork evaluation process enabled specialization across tasks, hardware devices, and model sizes. The searched models using SuperShaper method finds networks several SOTA BERT compression techniques on GLUE benchmarks.

G. COMPILER-AWARE BERT

Niu et al. [215] proposed a compiler-aware architecture search framework to find a BERT network that achieves good accuracy and is latency-friendly on mobile CPU and GPU. The search process is identical to the RL-NAS method, where a controller predicts a BERT model with different hyperparameters. The predicted BERT is fine-tuned on downstream tasks to evaluate validation accuracy. The compiler code generation phase returns the latency information of the predicted model on the target device. The controller is updated based on the combined accuracy and latency metric to predict the next best model. The compiler-aware searched BERT model is 5.2× faster on CPU and 4.1× faster on GPU than BERT-base with only 0.5-2% accuracy loss. Also, it achieves 7.8× speedup over the TFLite compiler [216].

H. AUTOTINYBERT

AutoTinyBERT [217] is a method to search for efficient hyperparameters of pre-trained language models under different latency constraints. The process is similar to the Once-for-all search, where a large Supernetwork of maximum dimensions is trained in a One-shot manner, followed by the Evolutionary search of optimal architecture for a specific latency constraint. The Supernetwork is a proxy for all submodels, where sampled architectures can be evaluated without training from scratch. The searched AutoTinyBERT outperforms NAS-BERT regarding latency on a single CPU and GLUE score on the SQuAD dataset.

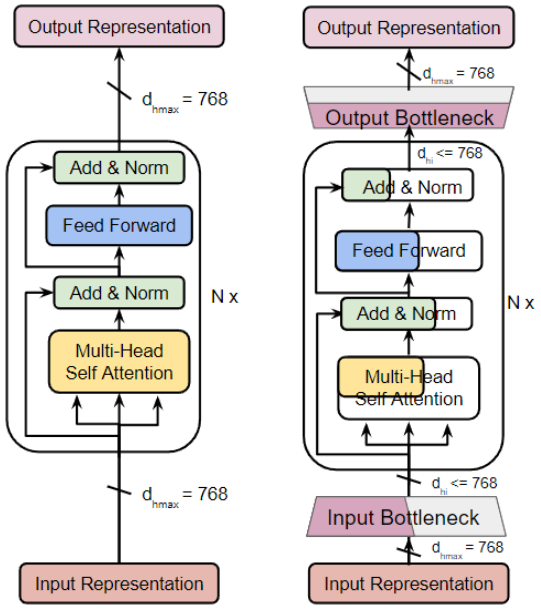


FIGURE 37. SuperShaper [214].

I. MOBILEBERT-EDGETPU

MobileBERT-EdgeTPU [218] is an efficient search algorithm for several vision and language tasks targeting On-device Deep Learning accelerators such as an EdgeTPU. The authors first designed a Performance Power Evaluation (PPE) service using the EdgeTPU compiler, a cycle-accurate simulator for latency estimation. TuNAS-based weight sharing approach [219] is utilized for vision tasks to search for tiny architectures. However, a multi-trial NAS approach is used for BERT due to the limitations of the weight-sharing method. The search space of MobileBERT-EdgeTPU is similar to MobileBERT [220]. The accuracy of MobileBERT-EdgeTPU on the SQuAD dataset is comparable to the BERT-large model and can be deployed to edge devices.

J. AE-BERT

AE-BERT [221] is an automatic BERT compression framework to select an optimal subnetwork from a pre-trained BERT model for a given target pruning ratio. The end-to-end process has three steps: (1) the first step randomly generates ''n'' pruning strategies to produce ''n'' pruned BERT candidates, such that each strategy results in the target pruning ratio, (2) the second step trains ''n'' candidates to pick the best subnetwork with the highest validation metric, (3) the chosen

model is fine-tuned from the previous stage to obtain the final trained and compressed BERT. The experiments on the GLUE benchmark show that AE-BERT outperforms SOTA manually designed pruning methods on BERT-base. Also, the compressed BERT-base is 1.83× faster on Xilinx Alveo U200 FPGA compared to Intel(R) Xeon(R) Gold 5218 (2.30GHz) CPU.

K. AUTODISTILL

AutoDistill [222] generates a task-agnostic BERT model for the target hardware platform by considering several objectives, constraints, design space, evaluation metrics, and hardware performance. The framework operates in a loop fashion, in a series of three steps: Model Exploration, Flash Distillation, and Evaluation. The Model exploration phase finds a good model based on the comprehensive set of inputs provided to the framework. The authors use Bayesian Optimization (BO) [223] to perform the multi-objective search for a student architecture by considering accuracy and latency. Flash distillation grows promising network candidates as a student architecture from pretraining dataset and teacher network. The Bayesian Optimization with Flash distillation significantly reduces the search cost. The network chosen in the Flash Distillation stage is evaluated in the Evaluation step for accuracy and hardware latency. AutoDistill searched model outperforms several previous works such as MobileBERT [220] and BERT-base [63] in terms of accuracy on the SQuAD dataset and latency on TPUv4.

L. REAL-TIME STYLE TRANSFER

Benmeziane et al. performed Real-time Style Transfer [224] on Xiaomi Redmi 10 mobile and Raspberry Pi 3 embedded platform by searching efficient ViTs. Style Transfer [225], [226] is a vision task of transferring the style of an image to the content of a different image. The ViT model used for Style Transfer can be divided into two modules: **(i)** The first style sub-network fetches the style from the style image using the Transformer encoder, and **(ii)** The second content sub-network obtains the content of the content image using a different Transformer encoder and transfers the extracted style using a Transformer decoder. The search space is composed of typical ViT hyperparameters such as patch size, number of heads, number of FFN layers, and number of encoder/decoder layers. The style sub-module is searched using random search, and the content sub-network is searched using Evolutionary search, where the search optimizes the Frames per second and model accuracy. The searched ViT is 3.5× and 2.1× faster than the CNN-only model AdaIN [227] and Self-attention-based network StyTr^2 [228], respectively.

XI. FUTURE DIRECTIONS

Transformer NAS research has become an important topic due to its capability of circumventing many hurdles while building a model. There have been nearly 50 research papers on Transformer architecture NAS algorithms in the last two years alone. However, we are still in the early stages, as there are only a few works compared to the gazillion methods on the Convolutional Neural Network search algorithm. The current solutions are only a first step towards developing more efficient search algorithms for the family of Transformer models. There is still room for improvement in search space, adaptability in various scenarios, and hardware performance. This section provides an overview of a few limitations and highlights open research challenges.

A. EFFICIENT SEARCH SPACE DESIGN

The primitive search element set and search space often limit NAS algorithms in finding efficient models for any backbone Neural Network, which is not just confined to Transformers. While the previously proposed CNN-based search methods directly inspire Transformer search algorithms, the search space is systematically developed for each search algorithm discussed in this paper. Hence, going into the future, the manual search space design will play a vital role in exploring out-of-the-box Transformer models.

B. TOPOLOGIES

The Transformer-style modules or their variants have been exploited in various classes of Neural Networks such as Unet [229], Graph Neural Networks (GNN) [230], Graph Convolutional Network (GCN) [231], 3D Convolutional Neural Networks [232], Point Clouds [233], Spiking Transformer Neural Network [234], etc. However, very little focus has been given in this direction to use NAS for efficient Transformer search for other topologies. Therefore, expanding the scope of Transformer NAS to other groups of neural architectures is crucial in advancing the field. For example, a Convolution, Self-Attention, and GCN hybrid search space followed by an efficient search algorithm can be studied for Node Classification tasks.

C. LOW-COST PERFORMANCE ESTIMATION

The increase in the size and complexity of search space calls for matur NAS algorithms to keep the search cost in check without compromising the quality. Effective Zero-cost proxy methods, such as TF-TAS [171], play an important role in quickly evaluating the model performance without expensive training. However, the current training-free methodologies operate only a small set of search elements without considering the hybrid Attention-Convolution search spaces. Therefore, more research is needed in this direction to make NAS accessible to a wider set of researchers and engineers.

D. HARDWARE-AWARE NAS METHODS

Orthogonal to efficient model and algorithm design lies enhancing hardware platforms to meet the computational demand of ever-growing size and magnitude of Neural Networks. The CNN-based Hardware-aware NAS has been investigated thoroughly on Microcontroller Unit [108], Mobile CPUs [105] high-end CPUs [101], ASIC [235], [236], FPGA [237], Resistive RAM (ReRAM) [238], Digital Signal Processor [239], and Vision Processing Unit (VPU) [240].

On the other hand, Transformer Hardware-aware NAS is not extensively examined on diverse platforms. More specifically, Vision Transformer Hardware-aware NAS is still fresh and unexplored on a large-scale dataset such as ImageNet. Although the current hardware-agnostic NAS algorithms on Transformer architectures are extremely useful for research purposes, hardware-aware search methods on actual or simulated hardware are critical for the real-time deployment of ViTs. Therefore, Hardware-aware search space and enhanced NAS algorithms for several hardware platforms are needed to improve accuracy and performance.

E. AUTOMATED SPARSE AND MIXED PRECISION QUANTIZED TRANSFORMER SEARCH

Neural Network Pruning [110] is an outstanding technique for optimized hardware inference as it removes redundant parameters. In order to complement the sparse nature of networks, numerous hardware platforms, such as Nvidia A100 GPUs [241], started offering explicit support for sparse calculation. The present NAS techniques only consider dense MHSA (FC) and Convolution layers while ignoring sparse matrix computing. Hence, future search methods should include sparsity features in the search space for higher performance gain. On the other hand, Mixed Precision Quantization [113] also plays an important role on top of pruning. Due to the low precision capability of several DNN models, accelerators such as Bitfusion [242] boost the performance of varying bit-width multiplication. As far as we know, AQ-BERT [125] is the only method to utilize NAS methods for bit-width assignment. However, this work is a hardware-agnostic method that does not evaluate their searched model on a mixed precision supporting hardware. There is scope to explore Hardware-aware Mixed Precision Quantization search methods, such as HAQ [123], for Transformer and its family. Therefore, future Transformer NAS algorithms should be aligned in the direction of sparse and mixed-precision to search for models smaller in size and faster on hardware platforms.

F. NETWORK-ACCELERATOR CO-SEARCH

Neural Network and Hardware Accelerator Co-search is an important class of Multi-objective co-design problem where two independent search algorithms are used to simultaneously find the optimal network parameters and accelerator configuration [40]. The co-design involves iteratively sampling a network-accelerator pair and evaluating validation accuracy and hardware performance metrics to guide the search process and converge at an optimal combination of network and hardware specifications. Many CNN-based co-search methods, such as DIAN [243], have reformulated the HW-NAS methods as a Co-search problem. Nonetheless, there are no such similar methods in the Transformer domain. Hence, in the future, Transformer model and hardware accelerator co-search can be utilized to achieve better accuracy-efficiency tradeoff than SOTA CNN-only co-designs.

G. APPLICATIONS AND PURPOSES

NAS and HW-NAS for CNN models have found their way from a simple task on a small dataset to a complex application on a gigantic dataset. However, the Transformer NAS methods have not been specialized for a wide variety of tasks and datasets, even though the manual design of Transformers has been used in diverse tasks such as Super Resolution [244], Semantic Segmentation [245], Medical Image Segmentation [246], etc. Also, the algorithmic breakthrough of search algorithms allowed the use for other purposes, such as Winograd Convolution Search [247], Mixed Precision Quantization Search [123], Fault-Tolerant network search [248], etc. Therefore, the scope should be extended beyond just architecture search for global dominance of Transformers and their kin, and associated current applications.

H. NEURAL ARCHITECTURE SEARCH BENCHMARKS

The computation demand for NAS methods is a hurdle for the development of novel NAS algorithms, as it requires training and evaluation of numerous architectures during the search process. Also, reproducibility is another challenge due to dissimilarity in search space and experiment setting, such as learning rate, initialization, etc., and therefore comparison between search methods and searched models becomes unfair. As a consequence, several Neural Architecture Search Benchmarks (NAS-Bench) have been released, acting as a Look-up-table (LUT) or dictionary of models and their appropriate performance metrics. For example, NAS-Bench-101 [104] is the first work in this context, where the dictionary consists of 423k distinct cell-based architectures and their validation accuracy and training time on the CIFAR-10 dataset. The users of benchmarks can obtain accuracy on the selected search space by querying the benchmark dictionary, thereby avoiding the expensive training process.

The hardware-agnostic benchmarks are equipped only with training and validation-related metrics and do not provide measured latency of networks on the hardware. Hence, these benchmarks are not helpful for non-hardware researchers in developing novel HW-aware NAS algorithms. There are a few attempts in this direction, such as LatBench [249] and HW-NAS-Bench [250], which provide hardware performance metrics for CNN networks on a wide range of CPUs, GPUs, and ASICs. Thus, moving forward in Transformer NAS research, NAS and HW-NAS Benchmarks will play a key role in development of better search methods. In fact, NAS-Bench-NLP [251], the first NAS benchmark dataset for language tasks based on RNNs, pointed out that their benchmarks are not as efficient as Transformers. Following the success of NAS Benchmarks, many surrogate models (ML models) have evolved, which are trained on a limited set of NAS Benchmarks to predict performance of unknown

networks in the search space. Therefore, Transformer NAS research requires such NAS Benchmarks and auxiliary ML models to engulf a large space.

I. INTEGRATING IN NAS FRAMEWORKS

The remarkable development of efficient NAS algorithms over the last few years lead the way for multiple frameworks such as Auto-keras [252], Microsoft Archai [253], etc. The next logical step is to integrate these easy-to-use NAS methods and Benchmarks in NAS toolkits. The current frameworks do not widely support Transformer based networks as there are a fairly recent success. Therefore, software engineers can work towards incorporating Transformers and Vision Transformers in the NAS frameworks.

XII. CONCLUSION

Transformer network design is a challenging problem with important applications in several tasks and hardware platforms. In the last few years, Neural Architecture Search and Hardware-aware NAS methods have significantly contributed to the automatic design of efficient Transformer, BERT, and Vision Transformer models. The automatically searched Transformers outperformed many manually designed Transformer architectures in terms of model and hardware performance. In this survey paper, we extensively reviewed recent advances related to NAS algorithms specific to Transformer and its family of architectures. We mainly summarized the search space, search strategy, and performance of the searched Transformer of state-of-the-art NAS methods. A diverse set of methods have developed over the last two years with innovations in architecture design and learning methodologies, which are analyzed in different sections. Although the performance of NAS algorithms has been greatly enhanced, the SOTA methods still have limitations, some of which are outlined in this paper. We hope our effort helps the reader understand the latest in Transformer NAS and ignites interest in developing novel and efficient methods.

ACKNOWLEDGMENT

All opinions, findings, and conclusions expressed are those of the authors.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[2] L. Dong, S. Xu, and B. Xu, "Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 5884–5888.

[3] A. Graves, "Long short-term memory," in *Supervised Sequence Labelling With Recurrent Neural Networks* (Studies in Computational Intelligence). Berlin, Germany: Springer, 2012, pp. 37–45.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.

[6] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical vision transformer using shifted Windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 10012–10022.

[7] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 213–229.

[8] R. Strudel, R. Garcia, I. Laptev, and C. Schmid, "Segmenter: Transformer for semantic segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 7262–7272.

[9] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 10347–10357.

[10] S. d'Ascoli, H. Touvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun, "ConViT: Improving vision transformers with soft convolutional inductive biases," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 2286–2296.

[11] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2013, pp. 847–855.

[12] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.

[13] N. Wang, Y. Gao, H. Chen, P. Wang, Z. Tian, C. Shen, and Y. Zhang, "NAS-FCOS: Fast neural architecture search for object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11943–11951.

[14] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "HAT: Hardware-aware transformers for efficient natural language processing," 2020, *arXiv:2005.14187*.

[15] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," 2021, *arXiv:2106.04554*.

[16] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, "Transformers in time series: A survey," 2022, *arXiv:2202.07125*.

[17] A. Ziyaden, A. Yelenov, and A. Pak, "Long-context transformers: A survey," in *Proc. 5th Sci. School Dyn. Complex Netw. their Appl. (DCNA)*, Sep. 2021, pp. 215–218.

[18] A. M. P. Brasoveanu and R. Andonie, "Visualizing transformers for NLP: A brief survey," in *Proc. 24th Int. Conf. Inf. Visualisation (IV)*, Sep. 2020, pp. 270–279.

[19] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," 2020, *arXiv:2009.06732*.

[20] Q. Fournier, G. M. Caron, and D. Aloise, "A practical survey on faster and lighter transformers," 2021, *arXiv:2103.14636*.

[21] K. S. Kalyan, A. Rajasekharan, and S. Sangeetha, "AMMUS: A survey of transformer-based pretrained models in natural language processing," 2021, *arXiv:2108.05542*.

[22] Y. Xu, H. Wei, M. Lin, Y. Deng, K. Sheng, M. Zhang, F. Tang, W. Dong, F. Huang, and C. Xu, "Transformers in computational visual media: A survey," *Comput. Vis. Media*, vol. 8, no. 1, pp. 33–62, Mar. 2022.

[23] Y. Liu, Y. Zhang, Y. Wang, F. Hou, J. Yuan, J. Tian, Y. Zhang, Z. Shi, J. Fan, and Z. He, "A survey of visual transformers," 2021, *arXiv:2111.06091*.

[24] J. Selva, A. S. Johansen, S. Escalera, K. Nasrollahi, T. B. Moeslund, and A. Clapés, "Video Transformers: A survey," 2022, *arXiv:2201.05991*.

[25] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, and Z. Yang, "A survey on vision transformer," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Feb. 18, 2022, doi: 10.1109/TPAMI.2022.3152247.

[26] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Comput. Surveys*, vol. 54, no. 10, pp. 1–41, Jan. 2022.

[27] D. J. Kedziora, K. Musial, and B. Gabrys, "AutonoML: Towards an integrated framework for autonomous machine learning," 2020, *arXiv:2012.12600*.

[28] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*. Berlin, Germany: Springer, 2019.

[29] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.
[30] Q. Yao, M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, and Y. Yu, "Taking human out of learning applications: A survey on automated machine learning," 2018, *arXiv:1810.13306*.
[31] E.-G. Talbi, "Automated design of deep neural networks: A survey and unified taxonomy," *ACM Comput. Surveys*, vol. 54, no. 2, pp. 1–37, Mar. 2022.
[32] X. Dong, D. J. Kedziora, K. Musial, and B. Gabrys, "Automated deep learning: Neural architecture search is not the end," 2021, *arXiv:2112.09245*.
[33] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1997–2017, 2019.
[34] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," 2019, *arXiv:1905.01392*.
[35] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 1–34, 2021.
[36] Y.-Q. Hu and Y. Yu, "A technical view on neural architecture search," *Int. J. Mach. Learn. Cybern.*, vol. 11, no. 4, pp. 795–811, Apr. 2020.
[37] G. Kyriakides and K. Margaritis, "An introduction to neural architecture search for convolutional networks," 2020, *arXiv:2005.11074*.
[38] D. Baymurzina, E. Golikov, and M. Burtsev, "A review of neural architecture search," *Neurocomputing*, vol. 474, pp. 82–93, Feb. 2022.
[39] K. T. Chitty-Venkata and A. K. Somani, "Neural architecture search survey: A hardware perspective," *ACM Comput. Surveys*, Apr. 2022.
[40] L. Sekanina, "Neural architecture search and hardware accelerator co-search: A survey," *IEEE Access*, vol. 9, pp. 151337–151362, 2021.
[41] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "A comprehensive survey on hardware-aware neural architecture search," 2021, *arXiv:2101.09336*.
[42] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "Hardware-aware neural architecture search: Survey and taxonomy," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, Aug. 2021, pp. 4322–4329.
[43] X. Zhang, W. Jiang, Y. Shi, and J. Hu, "When neural architecture search meets hardware implementation: From hardware awareness to co-design," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 25–30.
[44] Y. Jaafra, J. L. Laurent, A. Deruyver, and M. S. Naceur, "Reinforcement learning for neural architecture search: A review," *Image Vis. Comput.*, vol. 89, pp. 57–66, Sep. 2019.
[45] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 6, 2021, doi: 10.1109/TNNLS.2021.3100554.
[46] X. Zhou, A. K. Qin, Y. Sun, and K. C. Tan, "A survey of advances in evolutionary neural architecture search," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2021, pp. 950–957.
[47] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, L. Wang, Z. Chen, A. Xiao, J. Chang, X. Zhang, and Q. Tian, "Weight-sharing neural architecture search: A battle to shrink the optimization gap," *ACM Comput. Surveys*, vol. 54, no. 9, pp. 1–37, Dec. 2022.
[48] S. Cha, T. Kim, H. Lee, and S.-Y. Yun, "SuperNet in neural architecture search: A taxonomic survey," 2022, *arXiv:2204.03916*.
[49] S. Santra, J.-W. Hsieh, and C.-F. Lin, "Gradient descent effects on differential neural architecture search: A survey," *IEEE Access*, vol. 9, pp. 89602–89618, 2021.
[50] S. Liu, H. Zhang, and Y. Jin, "A survey on surrogate-assisted efficient neural architecture search," 2022, *arXiv:2206.01520*.
[51] H. Zhu, H. Zhang, and Y. Jin, "From federated learning to federated neural architecture search: A survey," *Complex Intell. Syst.*, vol. 7, no. 2, pp. 639–657, Apr. 2021.
[52] D. Liu and Y. Cao, "Federated neural architecture search evolution and open problems: An overview," in *Proc. Int. Conf. Bio-Inspired Comput., Theories Appl.* Singapore: Springer, 2021, pp. 330–345.
[53] V. V. V. Ganepola and T. Wirasingha, "Automating generative adversarial networks using neural architecture search: A review," in *Proc. Int. Conf. Emerg. Smart Comput. Informat. (ESCI)*, Mar. 2021, pp. 577–582.
[54] V. U. Buthgamumudalige and T. Wirasingha, "Neural architecture search for generative adversarial networks: A review," in *Proc. 10th Int. Conf. Inf. Autom. Sustainability (ICIAfS)*, Aug. 2021, pp. 246–251.
[55] Z. Zhang, X. Wang, and W. Zhu, "Automated machine learning on graphs: A survey," 2021, *arXiv:2103.00742*.
[56] B. M. Oloulade, J. Gao, J. Chen, T. Lyu, and R. Al-Sabri, "Graph neural architecture search: A survey," *Tsinghua Sci. Technol.*, vol. 27, no. 4, pp. 692–708, Aug. 2022.
[57] S. Poornachandra and S. Prapulla, "Neural architecture search in classical and quantum computers: A survey," *Int. Res. J. Eng. Technol.*, vol. 7, no. 6, pp. 1–6, 2020.
[58] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.
[59] A. F. Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv:1803.08375*.
[60] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," 2016, *arXiv:1606.08415*.
[61] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.
[62] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.
[63] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[64] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," Tech. Rep., 2018.
[65] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.
[66] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," 2019, *arXiv:1909.11942*.
[67] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training text encoders as discriminators rather than generators," 2020, *arXiv:2003.10555*.
[68] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1–9.
[69] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[70] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.
[71] Z.-H. Jiang, W. Yu, D. Zhou, Y. Chen, J. Feng, and S. Yan, "ConvBERT: Improving BERT with span-based dynamic convolution," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 12837–12848.
[72] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.
[73] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[74] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[75] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7132–7141.
[76] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7354–7363.
[77] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.
[78] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[79] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 843–852.
[80] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1243–1252.
[81] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4055–4064.
[82] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," 2018, *arXiv:1803.02155*.
[83] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, "CvT: Introducing convolutions to vision transformers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 22–31.
[84] D. Zhou, B. Kang, X. Jin, L. Yang, X. Lian, Z. Jiang, Q. Hou, and J. Feng, "DeepViT: Towards deeper vision transformer," 2021, *arXiv:2103.11886*.
[85] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi, "Escaping the big data paradigm with compact transformers," 2021, *arXiv:2104.05704*.
[86] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, "Transformer in transformer," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 15908–15919.
[87] X. Chu, Z. Tian, Y. Wang, B. Zhang, H. Ren, X. Wei, H. Xia, and C. Shen, "Twins: Revisiting the design of spatial attention in vision transformers," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 9355–9366.
[88] M. Chen, H. Peng, J. Fu, and H. Ling, "AutoFormer: Searching transformers for visual recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 12270–12280.
[89] D. So, Q. Le, and C. Liang, "The evolved transformer," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5877–5886.
[90] X. Shi, P. Zhou, W. Chen, and L. Xie, "Efficient gradient-based neural architecture search for end-to-end ASR," in *Proc. Companion Publication Int. Conf. Multimodal Interact.*, Oct. 2021, pp. 91–96.
[91] B. Chen, P. Li, C. Li, B. Li, L. Bai, C. Lin, M. Sun, J. Yan, and W. Ouyang, "GLiT: Neural architecture search for global and local image transformer," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 12–21.
[92] X. Su, S. You, J. Xie, M. Zheng, F. Wang, C. Qian, C. Zhang, X. Wang, and C. Xu, "ViTAS: Vision transformer architecture search," 2021, *arXiv:2106.13700*.
[93] C. Gong, D. Wang, M. Li, X. Chen, Z. Yan, Y. Tian, and V. Chandra, "NASViT: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–18.
[94] X. Pan, C. Ge, R. Lu, S. Song, G. Chen, Z. Huang, and G. Huang, "On the integration of self-attention and convolution," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 815–825.
[95] S. Mehta and M. Rastegari, "MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer," 2021, *arXiv:2110.02178*.
[96] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jegou, and M. Douze, "LeViT: A vision transformer in ConvNet's clothing for faster inference," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 12259–12269.
[97] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.
[98] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[99] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*.
[100] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 550–559.
[101] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*.
[102] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-Path NAS: Designing hardware-efficient convnets in less than 4 hours," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Cham, Switzerland: Springer, 2019, pp. 481–497.
[103] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
[104] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.
[105] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2820–2828.
[106] B. Wu, K. Keutzer, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, and Y. Jia, "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10734–10742.
[107] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–7.
[108] J. Lin, W.-M. Chen, Y. Lin, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 11711–11722.
[109] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv:1506.02626*.
[110] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
[111] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.
[112] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.
[113] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.
[114] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.
[115] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, Jan. 2017.
[116] W. Jiang, L. Yang, E. Sha, Q. Zhuge, S. Gu, Y. Shi, and J. Hu, "Hardware/software co-exploration of neural architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4805–4815, Dec. 2020.
[117] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," 2019, *arXiv:1908.09791*.
[118] Y.-L. Liao, S. Karaman, and V. Sze, "Searching for efficient multi-stage vision transformers," 2021, *arXiv:2109.00642*.
[119] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "APQ: Joint search for network architecture, pruning and quantization policy," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2078–2087.
[120] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "NetAdapt: Platform-aware neural network adaptation for mobile applications," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 285–300.
[121] D. Chen, Y. Li, M. Qiu, Z. Wang, B. Li, B. Ding, H. Deng, J. Huang, W. Lin, and J. Zhou, "AdaBERT: Task-adaptive BERT compression with differentiable neural architecture search," 2020, *arXiv:2001.04246*.
[122] J. Xu, X. Tan, R. Luo, K. Song, J. Li, T. Qin, and T.-Y. Liu, "NAS-BERT: Task-agnostic and adaptive-size BERT compression with neural architecture search," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2021, pp. 1933–1943.
[123] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 8612–8620.
[124] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of ConvNets via differentiable neural architecture search," 2018, *arXiv:1812.00090*.
[125] C. Zhao, T. Hua, Y. Shen, Q. Lou, and H. Jin, "Automatic mixed-precision quantization search of BERT," 2021, *arXiv:2112.14938*.

[126] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, and A. Tamchyna, "Findings of the 2014 workshop on statistical machine translation," in *Proc. 9th Workshop Stat. Mach. Transl.*, 2014, pp. 12–58.

[127] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," 2013, *arXiv:1312.3005*.

[128] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proc. 40th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, 2001, pp. 311–318.

[129] F. Wu, A. Fan, A. Baevski, Y. N. Dauphin, and M. Auli, "Pay less attention with lightweight and dynamic convolutions," 2019, *arXiv:1901.10430*.

[130] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 933–941.

[131] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, *arXiv:1710.05941*.

[132] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Netw.*, vol. 107, pp. 3–11, Nov. 2018.

[133] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, Atlanta, GA, USA, 2013, vol. 30, no. 1, p. 3.

[134] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[135] D. So, W. Mańke, H. Liu, Z. Dai, N. Shazeer, and Q. V. Le, "Searching for efficient transformers for language modeling," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 6010–6022.

[136] Y. Zhao, L. Dong, Y. Shen, Z. Zhang, F. Wei, and W. Chen, "Memory-efficient differentiable transformer architecture search," 2021, *arXiv:2105.14669*.

[137] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[138] W. Zhu, X. Wang, Y. Ni, and G. Xie, "AutoTrans: Automating transformer design via reinforced architecture search," in *Proc. CCF Int. Conf. Natural Lang. Process. Chin. Comput.* Cham, Switzerland: Springer, 2021, pp. 169–182.

[139] D. Elliott, S. Frank, K. Sima'an, and L. Specia, "Multi30K: Multilingual English-German image descriptions," 2016, *arXiv:1605.00459*.

[140] E. F. T. K. Sang and F. De Meulder, "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition," 2003, *arXiv:cs/0306050*.

[141] Y. Wang, Y. Yang, Y. Chen, J. Bai, C. Zhang, G. Su, X. Kou, Y. Tong, M. Yang, and L. Zhou, "TextNAS: A neural architecture search space tailored for text representation," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 5, pp. 9242–9249.

[142] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2013, pp. 1631–1642.

[143] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.

[144] J. Xu, L. Zhao, J. Lin, R. Gao, X. Sun, and H. Yang, "KNAS: Green neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 11613–11625.

[145] R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, and T.-Y. Liu, "Semi-supervised neural architecture search," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 10547–10557.

[146] K. Ito and L. Johnson. (2017). *The LJ Speech Dataset*. [Online]. Available: https://keithito.com/LJ-Speech-Dataset/

[147] N. Li, S. Liu, Y. Liu, S. Zhao, and M. Liu, "Neural speech synthesis with transformer network," in *Proc. AAAI Conf. Artif. Intell.*, Jul. 2019, vol. 33, no. 1, pp. 6706–6713.

[148] Y. Ren, L. Li, X. Yang, and J. Zhou, "Autotransformer: Automatic transformer architecture design for time series classification," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining*. Cham, Switzerland: Springer, 2022, pp. 143–155.

[149] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1761–1770.

[150] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The UCR time series archive," *IEEE/CAA J. Autom. Sinica*, vol. 6, pp. 1293–1305, 2019.

[151] J. Kim, J. Wang, S. Kim, and Y. Lee, "Evolved speech-transformer: Applying neural architecture search to end-to-end automatic speech recognition," in *Proc. Interspeech*, Oct. 2020, pp. 1788–1792.

[152] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 5206–5210.

[153] M. Ravanelli, T. Parcollet, and Y. Bengio, "The PyTorch-Kaldi speech recognition toolkit," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 6465–6469.

[154] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," 2020, *arXiv:2005.08100*.

[155] H. Bu, J. Du, X. Na, B. Wu, and H. Zheng, "AISHELL-1: An open-source Mandarin speech corpus and a speech recognition baseline," in *Proc. 20th Conf. Oriental Chapter Int. Coordinating Committee Speech Databases Speech I/O Syst. Assessment (O-COCOSDA)*, Nov. 2017, pp. 1–5.

[156] Y. Liu, T. Li, P. Zhang, and Y. Yan, "Improved conformer-based end-to-end speech recognition using neural architecture search," 2021, *arXiv:2104.05390*.

[157] Y. Yin, S. Huang, and X. Zhang, "BM-NAS: Bilevel multimodal neural architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2022, vol. 36, no. 8, pp. 8901–8909.

[158] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," 2016, *arXiv:1606.05250*.

[159] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," 2018, *arXiv:1804.07461*.

[160] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, "Block-wisely supervised neural architecture search with knowledge distillation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1989–1998.

[161] J. Gao, H. Xu, H. Shi, X. Ren, L. Philip, X. Liang, X. Jiang, and Z. Li, "AutoBERT-Zero: Evolving BERT backbone from scratch," in *Proc. AAAI Conf. Artif. Intell.*, 2022, vol. 36, no. 10, pp. 10663–10671.

[162] J. Xu, X. Tan, K. Song, R. Luo, Y. Leng, T. Qin, T.-Y. Liu, and J. Li, "Analyzing and mitigating interference in neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 24646–24662.

[163] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 544–560.

[164] R. Wang, Q. Bai, J. Ao, L. Zhou, Z. Xiong, Z. Wei, Y. Zhang, T. Ko, and H. Li, "LightHuBERT: Lightweight and configurable speech representation learning with once-for-all hidden-unit BERT," 2022, *arXiv:2203.15610*.

[165] W.-N. Hsu, B. Bolte, Y.-H.-H. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed, "HuBERT: Self-supervised speech representation learning by masked prediction of hidden units," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 29, pp. 3451–3460, 2021.

[166] H. Tsai, J. Ooi, C.-S. Ferng, H. W. Chung, and J. Riesa, "Finding fast transformers: One-shot neural architecture search by component composition," 2020, *arXiv:2008.06808*.

[167] H. Tsai, J. Riesa, M. Johnson, N. Arivazhagan, X. Li, and A. Archer, "Small and practical BERT models for sequence labeling," 2019, *arXiv:1909.00100*.

[168] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-BERT: Hessian based ultra low precision quantization of BERT," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 5, pp. 8815–8821.

[169] W. Zhu, X. Qiu, Y. Ni, and G. Xie, "AutoRC: Improving BERT based relation classification models via architecture search," 2020, *arXiv:2009.10680*.

[170] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight NAS," 2021, *arXiv:2101.08134*.

[171] Q. Zhou, K. Sheng, X. Zheng, K. Li, X. Sun, Y. Tian, J. Chen, and R. Ji, "Training-free transformer architecture search," 2022, *arXiv:2203.12217*.
[172] W. Chen, W. Huang, X. Du, X. Song, Z. Wang, and D. Zhou, "Auto-scaling vision transformers without training," 2022, *arXiv:2202.11921*.
[173] M. Chen, K. Wu, B. Ni, H. Peng, B. Liu, J. Fu, H. Chao, and H. Ling, "Searching the search space of vision transformer," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 8714–8726.
[174] B. Ni, G. Meng, S. Xiang, and C. Pan, "NASformer: Neural architecture search for vision transformer," in *Proc. Asian Conf. Pattern Recognit.* Cham, Switzerland: Springer, 2022, pp. 47–61.
[175] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015, *arXiv:1511.07122*.
[176] M. Ding, X. Lian, L. Yang, P. Wang, X. Jin, Z. Lu, and P. Luo, "HR-NAS: Searching efficient high-resolution neural architectures with lightweight transformers," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 2982–2992.
[177] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, and W. Liu, "Deep high-resolution representation learning for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 10, pp. 3349–3364, Oct. 2021.
[178] M. Tan and Q. V. Le, "MixConv: Mixed depthwise convolutional kernels," 2019, *arXiv:1907.09595*.
[179] C. Li, T. Tang, G. Wang, J. Peng, B. Wang, X. Liang, and X. Chang, "BossNAS: Exploring hybrid CNN-transformers with block-wisely self-supervised neural architecture search," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Oct. 2021, pp. 12281–12291.
[180] C. Liu, P. Dollár, K. He, R. Girshick, A. Yuille, and S. Xie, "Are labels necessary for neural architecture search?" in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 798–813.
[181] A. Srinivas, T.-Y. Lin, N. Parmar, J. Shlens, P. Abbeel, and A. Vaswani, "Bottleneck transformers for visual recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 16519–16529.
[182] L. Yang, Y. Hu, S. Lu, Z. Sun, J. Mei, Y. Han, and X. Li, "Searching for BurgerFormer with micro-meso-macro space design," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 25055–25069.
[183] J. Yu, P. Jin, H. Liu, G. Bender, P. J. Kindermans, M. Tan, T. Huang, X. Song, R. Pang, and Q. Le, "BigNAS: Scaling up neural architecture search with big single-stage models," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 702–717.
[184] J. Liu, H. Li, G. Song, X. Huang, and Y. Liu, "UniNet: Unified architecture search with convolution, transformer, and MLP," 2021, *arXiv:2110.04035*.
[185] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, and M. Lucic, "MLP-Mixer: An all-MLP architecture for vision," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 24261–24272.
[186] H. He, J. Liu, Z. Pan, J. Cai, J. Zhang, D. Tao, and B. Zhuang, "Pruning self-attentions into convolutional layers in single path," 2021, *arXiv:2111.11802*.
[187] Y. Zhou, H. Wang, S. Huo, and B. Wang, "Full-attention based neural architecture search using context auto-regression," 2021, *arXiv:2111.07139*.
[188] F. Long, Z. Qiu, Y. Pan, T. Yao, J. Luo, and T. Mei, "Stand-alone inter-frame attention in video models," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 3192–3201.
[189] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7794–7803.
[190] H. Zhang, K. Hao, W. Pedrycz, L. Gao, X. Tang, and B. Wei, "Vision transformer with convolutions architecture search," 2022, *arXiv:2203.10435*.
[191] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive DARTS: Bridging the optimization gap for NAS in the wild," *Int. J. Comput. Vis.*, vol. 129, no. 3, pp. 638–655, Mar. 2021.
[192] C. Jin, P. M. Phothilimthana, and S. Roy, "αNAS: Neural architecture search using property guided synthesis," 2022, *arXiv:2205.03960*.
[193] B. Wu, C. Li, H. Zhang, X. Dai, P. Zhang, M. Yu, J. Wang, Y. Lin, and P. Vajda, "FBNetV5: Neural architecture search for multiple tasks in one run," 2021, *arXiv:2111.10007*.
[194] B. Cheng, A. Schwing, and A. Kirillov, "Per-pixel classification is not all you need for semantic segmentation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 17864–17875.
[195] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ADE20K dataset," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 633–641.
[196] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 1–7.
[197] Y. Jiang, S. Chang, and Z. Wang, "TransGAN: Two pure transformers can make one strong GAN, and that can scale up," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 14745–14758.
[198] W. Li, S. Wen, K. Shi, Y. Yang, and T. Huang, "Neural architecture search with a lightweight transformer for text-to-image synthesis," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 3, pp. 1567–1576, May 2022.
[199] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-UCSD birds-200–2011 dataset," Tech. Rep., 2011.
[200] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Proc. 6th Indian Conf. Comput. Vis., Graph. Image Process.*, Dec. 2008, pp. 722–729.
[201] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2014, pp. 740–755.
[202] S. Lu, J. Li, J. Tan, S. Yang, and J. Liu, "TNASP: A transformer-based NAS predictor with a self-evolution framework," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 15125–15137.
[203] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," 2020, *arXiv:2001.00326*.
[204] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, and C. Xu, "AdderNet: Do we really need multiplications in deep learning?" in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1468–1477.
[205] H. You, X. Chen, Y. Zhang, C. Li, S. Li, Z. Liu, Z. Wang, and Y. Lin, "ShiftAddNet: A hardware-inspired deep network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 2771–2783.
[206] H. You, B. Li, H. Shi, Y. Fu, and Y. Lin, "ShiftAddNAS: Hardware-inspired search for more accurate and efficient neural networks," 2022, *arXiv:2205.08119*.
[207] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2016.
[208] D. Cummings, A. Sarah, S. N. Sridhar, M. Szankin, J. P. Munoz, and S. Sundaresan, "A hardware-aware framework for accelerating neural architecture search across modalities," 2022, *arXiv:2205.10358*.
[209] R. Luo, X. Tan, R. Wang, T. Qin, J. Li, S. Zhao, E. Chen, and T.-Y. Liu, "LightSpeech: Lightweight and fast text to speech with neural architecture search," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 5699–5703.
[210] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "FastSpeech: Fast, robust and controllable text to speech," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–10.
[211] L. Kaiser, A. N. Gomez, and F. Chollet, "Depthwise separable convolutions for neural machine translation," 2017, *arXiv:1706.03059*.
[212] R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture search with GBDT," Tech. Rep., 2020.
[213] M. Javaheripi, S. Shah, S. Mukherjee, T. L. Religa, C. C. T. Mendes, G. H. de Rosa, S. Bubeck, F. Koushanfar, and D. Dey, "LiteTransformerSearch: Training-free on-device search for efficient autoregressive language models," 2022, *arXiv:2203.02094*.
[214] V. Ganesan, G. Ramesh, and P. Kumar, "SuperShaper: Task-agnostic super pre-training of BERT models with variable hidden dimensions," 2021, *arXiv:2110.04711*.
[215] W. Niu, Z. Kong, G. Yuan, W. Jiang, J. Guan, C. Ding, P. Zhao, S. Liu, B. Ren, and Y. Wang, "Real-time execution of large-scale language models on mobile," 2020, *arXiv:2009.06823*.
[216] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and M. Kudlur, "TensorFlow: A system for Large-Scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.
[217] Y. Yin, C. Chen, L. Shang, X. Jiang, X. Chen, and Q. Liu, "AutoTinyBERT: Automatic hyper-parameter optimization for efficient pre-trained language models," 2021, *arXiv:2107.13686*.

[218] B. Akin, S. Gupta, Y. Long, A. Spiridonov, Z. Wang, M. White, H. Xu, P. Zhou, and Y. Zhou, "Searching for efficient neural architectures for on-device ML on edge TPUs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2022, pp. 2667–2676.

[219] G. Bender, H. Liu, B. Chen, G. Chu, S. Cheng, P.-J. Kindermans, and Q. V. Le, "Can weight sharing outperform random architecture search? An investigation with TuNAS," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 14323–14332.

[220] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "MobileBERT: A compact task-agnostic BERT for resource-limited devices," 2020, *arXiv:2004.02984*.

[221] S. Huang, N. Liu, Y. Liang, H. Peng, H. Li, D. Xu, M. Xie, and C. Ding, "An automatic and efficient BERT pruning for edge AI systems," in *Proc. 23rd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2022, pp. 1–6.

[222] X. Zhang, Z. Zhou, D. Chen, and Y. E. Wang, "AutoDistill: An end-to-end framework to explore and distill hardware-efficient language models," 2022, *arXiv:2201.08539*.

[223] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 1487–1495.

[224] H. Benmeziane, H. Ouarnoughi, K. E. Maghraoui, and S. Niar, "Real-time style transfer with efficient vision transformers," in *Proc. 5th Int. Workshop Edge Syst., Anal. Netw.*, Apr. 2022, pp. 31–36.

[225] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural style transfer: A review," *IEEE Trans. Vis. Comput. Graphics*, vol. 26, no. 11, pp. 3365–3385, Nov. 2020.

[226] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," 2015, *arXiv:1508.06576*.

[227] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1501–1510.

[228] Y. Deng, F. Tang, X. Pan, W. Dong, C. Ma, and C. Xu, "StyTr^2: Unbiased image style transfer with transformers," 2021, *arXiv:2105.14576*.

[229] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian, and M. Wang, "Swin-UNet: UNet-like pure transformer for medical image segmentation," 2021, *arXiv:2105.05537*.

[230] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," 2020, *arXiv:2012.09699*.

[231] X. Dong, C. Long, W. Xu, and C. Xiao, "Dual graph convolutional networks with transformer and curriculum learning for image captioning," in *Proc. 29th ACM Int. Conf. Multimedia*, Oct. 2021, pp. 2615–2624.

[232] C. Cao, Y. Zhang, Y. Wu, H. Lu, and J. Cheng, "Egocentric gesture recognition using recurrent 3D convolutional neural networks with spatiotemporal transformer modules," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 3763–3771.

[233] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "PCT: Point cloud transformer," *Comput. Vis. Media*, vol. 7, no. 2, pp. 187–199, Jun. 2021.

[234] E. Mueller, V. Studenyak, D. Auge, and A. Knoll, "Spiking transformer networks: A rate coded approach for processing sequential data," in *Proc. 7th Int. Conf. Syst. Informat. (ICSAI)*, Nov. 2021, pp. 1–5.

[235] S. Gupta and B. Akin, "Accelerator-aware neural network design using AutoML," 2020, *arXiv:2003.02838*.

[236] K. T. Chitty-Venkata and A. K. Somani, "Array-aware neural architecture search," in *Proc. IEEE 32nd Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2021, pp. 125–132.

[237] W. Jiang, X. Zhang, E. H.-M. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. Efficiency: Achieving both through FPGA-implementation aware neural architecture search," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[238] Z. Yuan, J. Liu, X. Li, L. Yan, H. Chen, B. Wu, Y. Yang, and G. Sun, "NAS4RRAM: Neural network architecture search for inference on RRAM-based accelerators," *Sci. China Inf. Sci.*, vol. 64, no. 6, pp. 1–11, Jun. 2021.

[239] L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, "Fast hardware-aware neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 692–693.

[240] C. Donegan, H. Yous, S. Sinha, and J. Byrne, "VPU specific CNNs through neural architecture search," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, Jan. 2021, pp. 9772–9779.

[241] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 tensor core GPU: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, Mar. 2021.

[242] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 764–775.

[243] Y. Zhang, Y. Fu, W. Jiang, C. Li, H. You, M. Li, V. Chandra, and Y. Lin, "DIAN: Differentiable accelerator-network co-search towards maximal DNN efficiency," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2021, pp. 1–6.

[244] Z. Lu, J. Li, H. Liu, C. Huang, L. Zhang, and T. Zeng, "Transformer for single image super-resolution," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2022, pp. 457–466.

[245] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "SegFormer: Simple and efficient design for semantic segmentation with transformers," in *Proc. Adv. Neural Inf. Process. Sys. (NIPS)*, vol. 34, Dec. 2021, pp. 12077–12090.

[246] J. Chen, Y. Lu, Q. Yu, X. Luo, E. Adeli, Y. Wang, L. Lu, A. L. Yuille, and Y. Zhou, "TransUNet: Transformers make strong encoders for medical image segmentation," 2021, *arXiv:2102.04306*.

[247] J. Fernandez-Marques, P. Whatmough, A. Mundy, and M. Mattina, "Searching for Winograd-aware quantized networks," *Proc. Mach. Learn. Syst.*, vol. 2, pp. 14–29, Mar. 2020.

[248] W. Li, X. Ning, G. Ge, X. Chen, Y. Wang, and H. Yang, "FTT-NAS: Discovering fault-tolerant neural architecture," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 211–216.

[249] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, "BRP-NAS: Prediction-based NAS using GCNs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 10480–10490.

[250] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, and Y. Lin, "HW-NAS-Bench: Hardware-aware neural architecture search benchmark," 2021, *arXiv:2103.10584*.

[251] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, and E. Burnaev, "NAS-Bench-NLP: Neural architecture search benchmark for natural language processing," 2020, *arXiv:2006.07116*.

[252] H. Jin, Q. Song, and X. Hu, "Auto-Keras: An efficient neural architecture search system," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1946–1956.

[253] Microsoft. (2022). *Archai*. [Online]. Available: https://github.com/microsoft/archai

KRISHNA TEJA CHITTY-VENKATA received the Bachelor of Engineering degree in electronics and communication from the University College of Engineering, Osmania University, Hyderabad, India, in 2017. He is currently pursuing the Ph.D. degree with Iowa State University, Ames, IA, USA. He interned at the Argonne National Laboratory, Intel Corporation and AMD, where he worked on various problems on efficient deep learning. His research interests include designing network design algorithms and compression methods for neural network processing, such as pruning, quantization and neural architecture search, and for general and special purpose hardware platforms.

MURALI EMANI received the Ph.D. degree from the School of Informatics, The University of Edinburgh, U.K., in 2015. He is currently an Assistant Computer Scientist with the Argonne National Laboratory, Data Science Group, Argonne Leadership Computing Facility (ALCF). Prior to this, he was a Postdoctoral Research Staff Member at the Lawrence Livermore National Laboratory. His research interests include scalable machine learning, emerging HPC and AI architectures, and AI for science. He serves as the Co-Chair for MLPerf HPC Group at MLCommons to benchmark large scale ML on HPC systems. He also co-leads the AI Testbed at ALCF to evaluate novel AI accelerators for scientific machine learning applications. He has organized workshops and participated in tutorials that include benchmarking deep learning workloads on emerging hardware, MLPerf-Bench at MLSys'20, MLSys'21, ISPASS'20, ISPASS'21, and ASPLOS'21. He has also co-chaired the MLPerf birds-of-a-feather sessions at SC'19, SC'20, and SC'21.

VENKATRAM VISHWANATH received the Ph.D. degree in computer science from the University of Illinois at Chicago, in 2009. He is currently a Computer Scientist at the Argonne National Laboratory. He is also the Data Science Team Lead at the Argonne Leadership Computing Facility (ALCF). His current focus is on algorithms, system software, and workflows to facilitate data-centric applications on supercomputing systems. His research interests include scientific applications, supercomputing architectures, parallel algorithms and runtimes, scalable analytics, and collaborative workspaces. He has received best papers awards at venues, including HPDC and LDAV, and a Gordon Bell Finalist.

ARUN K. SOMANI (Life Fellow, IEEE) received the M.S. and Ph.D. degrees in electrical engineering from McGill University, Montreal, in 1983 and 1985, respectively. He worked as a Scientific Officer with the Government of India, New Delhi, and a Faculty Member with the University of Washington, Seattle. He is currently an Anson Marston Distinguished Professor in electrical and computer engineering with Iowa State University. His research interests include the areas of computer system design, architecture, fault tolerant computing, computer interconnection networks, optical networking, and reconfigurable and parallel computer systems. He served as an IEEE Distinguished Visitor, an IEEE Distinguished Tutorial Speaker, and an IEEE Communication Society Distinguished Visitor. He has delivered several keynote speeches, and distinguished and invited talks all over the world. He is a Distinguished Engineer of ACM, an Eminent Engineer of Tau Beta Pi, and a fellow of AAAS.