Journal of Machine Learning Research 23 (2022) 1-17 Submitted 1/21; Revised 5/22; Published 9/22

NAS for Transformers

Iris Dania Jimenez IRIS.JIMENEZ@CAMPUS.LMU.DE
Department of Mathematics, Informatics and Statistics
Ludwig-Maximilians-Universität München

Editor: None

Abstract

Transformers have emerged as the leading architecture in deep learning for a wide range of applications, particularly in natural language processing (NLP) and computer vision (CV). Despite their success, designing effective Transformer models remains a complex and resource-intensive task due to their intricate architecture and the substantial computational demands of training and optimization. Neural Architecture Search (NAS) offers a promising solution to these challenges by automating the search for optimal Transformer architectures. In this report, I examine the key concepts related to NAS and Transformers, present notable results achieved in the NAS for Transformers field, and discuss existing limitations as well as potential future directions.

Keywords: NAS, Transformers, BERT

1 Introduction

Transformers have emerged as the leading architecture in deep learning for a wide range of applications, particularly in NLP and CV. The "Attention Is All You Need" (Vaswani et al. (2017)) paper, revolutionized the field of deep learning in 2017 by introducing the Transformer model, which relies solely on self-attention mechanisms, discarding the need for recurrent or convolutional layers traditionally used in NLP models. This innovation dramatically improved the efficiency and scalability of training, allowing for more parallelization and better handling of long-range dependencies in sequences. As a result, the Transformer architecture became the foundation for numerous state-of-the-art models, fundamentally changing the landscape of NLP and paving the way for breakthroughs in various other domains, including computer vision and reinforcement learning. Transformers gained renewed attention with the release of ChatGPT by OpenAI in 2022. This groundbreaking chatbot, built on the Transformer architecture, captured global interest and sparked a new wave of enthusiasm for AI. ChatGPT's impressive capabilities surprised many and generated widespread excitement across the public, media, and virtually every industry sector, highlighting the transformative potential of AI technologies. Despite their success, designing effective Transformer models remains a complex and resource-intensive task due to their intricate architecture and the substantial computational demands of training and optimization. The process demands significant computational power, often necessitating powerful GPUs or TPUs and large-scale distributed computing resources to handle the billions of parameters in state-of-the-art models. Additionally, the extensive time needed for training, sometimes taking weeks or even months, further drives up costs, including substan-

tial energy consumption. To address these challenges, experts and researchers have turned to Neural Architecture Search (NAS) to design optimal Transformer architectures. NAS, a subfield of machine learning, focuses on automating the discovery of high-performing neural network architectures. By leveraging knowledge and techniques from NAS, researchers aim to accelerate the training process and reduce the costs associated with developing efficient and powerful Transformer models. This approach not only reduces the need for extensive manual design but also enhances model performance and efficiency.
The report is organized as follows: Section 2 provides an overview of the key aspects of Transformer models, while Section 3 explores the fundamentals of Neural Architecture Search. Section 4, discusses the most prominent NAS methods applied to Transformers, highlighting their strengths and applications. The report concludes with Section 5, which draws key conclusions, and Section 6, which examines the limitations of NAS and potential future directions in this field.

2 Transformers

The first Transformer was designed for NLP tasks, while later on Transformers were also adapted to handle image data with the release of Vision Transformers(ViTs) (Dosovitskiy et al. (2021)). In this report, to explain the core idea of how Transformers work I will refer to the “original” Transformers as presented in (Vaswani et al. (2017)).
The main idea behind Transformers is the use of self-attention mechanisms to process and understand sequential data, allowing the model to weigh the importance of different words or elements within a sequence relative to each other. Unlike previous models, such as recurrent neural networks (RNNs) (Rumelhart and McClelland (1987)) or long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber (1997)), which process data sequentially and have difficulty managing long-range dependencies, Transformers process entire sequences simultaneously. This is achieved through the self-attention mechanism, which computes a set of attention weights that determine how much focus to place on other parts of the input sequence when encoding a particular word or token.

Transformers consist of an encoder-decoder structure where the encoder reads and processes the input data into a continuous representation, and the decoder uses this representation to generate the output. Within each encoder and decoder, there are multiple layers, each composed of multi-head self-attention and feed-forward neural networks. Multi-head self-attention allows the model to attend to different parts of the sequence from multiple perspectives simultaneously, enhancing the model’s ability to capture intricate relationships within the data. Positional encoding is added to the input embeddings to give the model a sense of the order of the sequence since the self-attention mechanism itself does not inherently capture sequential order. The ability to focus on any part of the input, regardless of its position, enables Transformers to handle longer sequences more effectively and to parallelize computations, making training significantly faster compared to RNNs and LSTMs. Transformers were initially developed as an encoder-decoder architecture, designed to handle both input processing and output generation. However, subsequent models have often specialized by adopting either the encoder or the decoder component exclusively. For instance, Bidirectional Encoder Representations from Transformers (BERT)(Devlin et al. (2019)) by Google is a prominent example of an encoder-only architecture, focusing on understanding and representing input text. In contrast, Generative Pre-trained Transformer (GPT)(Radford and Narasimhan (2018)) is an example of a decoder-only architecture, designed primarily for generating text based on a given input. Figure 1 represent visually how these three Transformers architecture conceptually look like.

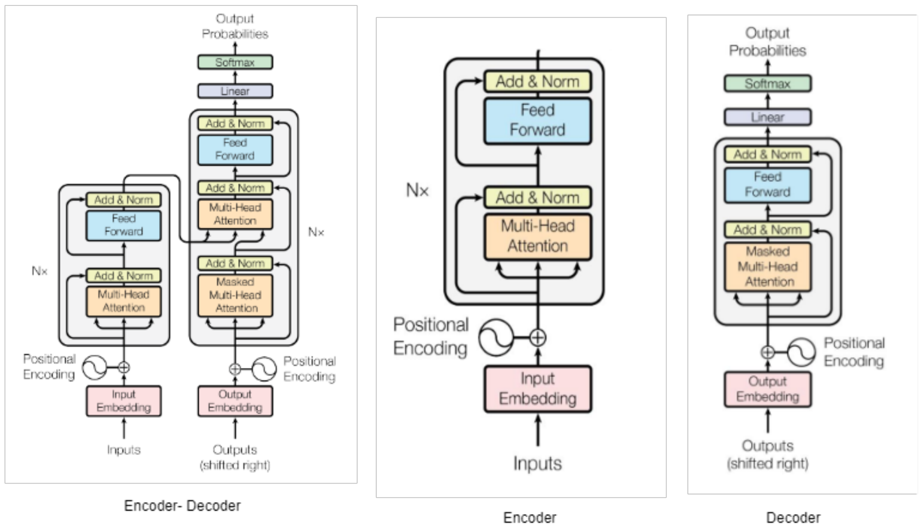


Figure 1: Different Transformers architectures (Vaswani et al. (2017))

The Transformer model's flexibility and scalability have made it the backbone of numerous state-of-the-art NLP models, such as BERT and GPT, which can be fine-tuned for a wide range of tasks, including translation, text generation, and sentiment analysis. The core innovation of the Transformer, relying on self-attention rather than recurrence or convolutions, allows it to model complex dependencies in data while being computationally efficient.

3 Neural Architecture Search

Neural Architecture Search is a subfield of machine learning that focuses on the automated design of neural network architectures. The primary goal of NAS is to discover high-performing models that can achieve optimal performance on a given task without the extensive manual effort typically required in traditional model design. A more precise definition of NAS can be stated as follows:

Given a search space **A**, a dataset **D**, a pipeline **P**, and a time or computation budget t, the goal is to find an architecture $a \in \mathbf{A}$ within budget t which has the highest possible validation accuracy when trained using dataset **D** and training pipeline **P** (White et al. (2023)).

The number of papers published in the NAS field has grown exceptionally in the past few years, as shown in Figure 2, highlighting the increasing research and interest in this area. This surge in interest is partly due to the fact that architectures discovered through NAS have often outperformed or matched human-designed models across a range of tasks (Chen et al. (2018); Du et al. (2020); So et al. (2019); Zoph and Le (2017)).

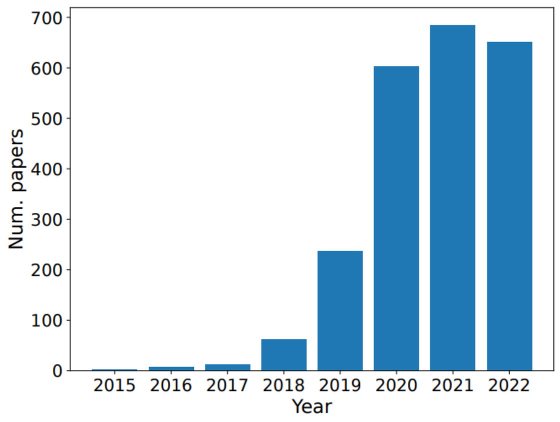


Figure 2: Number of papers published about NAS in the last years (White et al. (2023))

NAS explores a vast space of potential architectures to identify the most effective ones based on specific criteria, such as accuracy. The main components of NAS include the search space, which defines the possible architectures to explore; the search strategy, which guides how the search of a good architecture is conducted, often through methods like reinforcement learning, evolutionary algorithms, or gradient-based optimization; and the performance evaluation, which assesses how well the candidate architectures perform on the given task. Additionally, a notable approach within NAS is the One-Shot model technique, where both the architecture's hyperparameters and its weights are learned simultaneously, resulting in a more efficient and cohesive optimization process. For a visual representation of the NAS components, please refer to Figure 3.

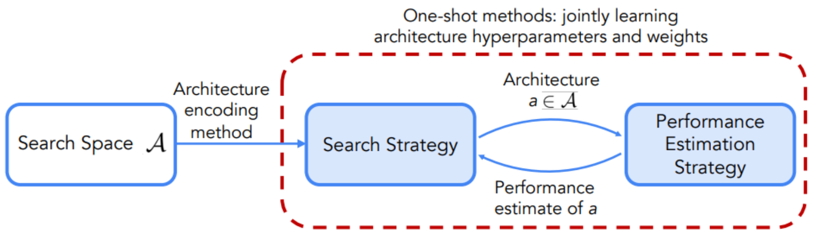

One-shot methods: jointly learning architecture hyperparameters and weights
Search Space $\mathcal{A}$
Architecture encoding method
Search Strategy
Architecture $a \in \mathcal{A}$
Performance Estimation Strategy
Performance estimate of a

Figure 3: NAS components (Elsken et al. (2019); Weng (2020))

The most used search spaces in the case of Transformers are chain-structure search spaces and cell-structure search spaces (White et al. (2023)).

Chain-structured search spaces have a simple architecture topology: a sequential chain of operation layers are used to represent the architecture. The layers in this search space correspond to various operations, such as attention mechanisms or linear transformations. Although these search spaces are straightforward to design and implement, their simplicity can limit the potential for discovering truly innovative architectures. Figure 4 shows the structure of chain-structured search space(on the left).

The cell structure search space focuses on discovering optimal sub-modules or "cells" of a neural network, which are then stacked or repeated to form the entire architecture. In this search space, the objective is to identify the best-performing arrangement of basic operations, such as convolutions, pooling, and activation functions, within a single cell. These cells typically consist of a fixed number of nodes or layers and are connected by directed edges representing the flow of data. By limiting the search to a smaller, more manageable space within a single cell, the cell structure search space allows for efficient exploration and optimization, enabling the discovery of high-performing architectures. Figure 4 shows the structure of cell-structured search space(on the right).

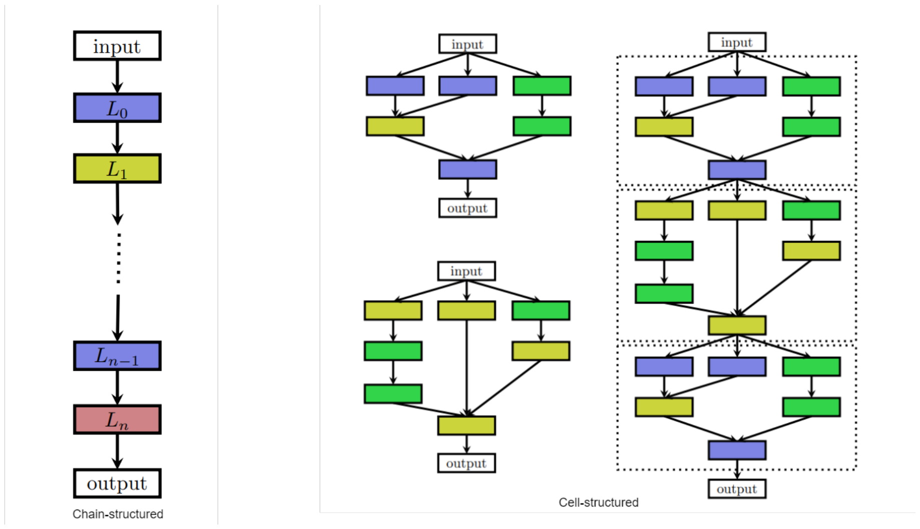


Figure 4: Cell and Chain structured search spaces (Elsken et al. (2019))

One of the most used type of search strategy are evolutionary algorithms (White et al. (2023)). These algorithms are inspired by the principles of natural selection and evolution, where a population of candidate architectures, known as "individuals", undergoes a process of selection, crossover, and mutation to explore the search space. Each individual is evaluated based on a fitness function, typically its performance on a validation dataset, and the best-performing architectures are selected to create the next generation. Over successive iterations, evolutionary algorithms can effectively navigate the search space, discovering novel and high-performing architectures. One significant advantage of using evolutionary algorithms in NAS is their ability to handle a wide range of search spaces and adapt to complex, non-differentiable, and multi-objective optimization problems, making them highly flexible. Additionally, they can maintain a diverse set of solutions, which helps prevent premature convergence to suboptimal architectures. However, a major disadvantage is that these algorithms are very susceptible to their own hyperparameters, which can of course influence the final results. Figure 5 shows the steps of an evolutionary algorithm.

Different approaches have been explored to evaluate and estimate the performance of architectures, such as proxy models (Qiao et al. (2024)), and ensemble learning methods (Chen et al. (2023)), each with its own strengths and limitations. However, the diverse requirements and characteristics of various tasks and datasets, along with the computational

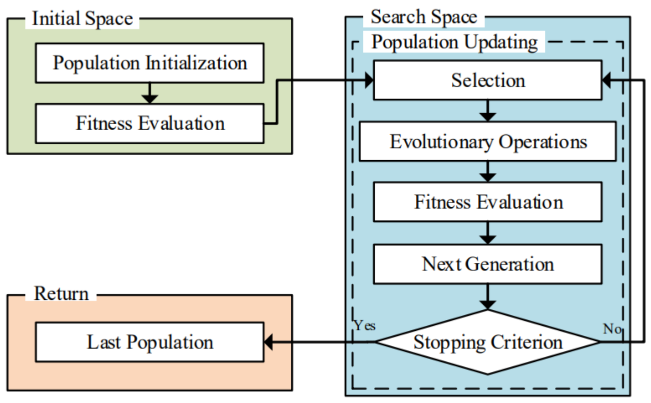


Figure 5: Flowchart of an evolutionary algorithm (Liu et al. (2023))

challenges inherent to Transformer models, have prevented any single estimation strategy from emerging as the most effective or commonly used.

One-shot models are a powerful approach within NAS that aim to streamline the process of discovering optimal neural network architectures by training a single, over-parameterized "Supernet" that encompasses all possible architectures in the search space. The idea is to represent operation like convolution, attention and skipped connections as edges and tensor as nodes, as shown in Figure 6. Instead of training each candidate architecture from scratch, the Supernet is trained once, and different architectures are obtained by selecting different sub-networks from this Supernet. This approach significantly reduces the computational cost and time required for NAS, as it allows for the evaluation of numerous architectures simultaneously by sharing weights among them. However, while One-Shot models provide a more efficient framework for NAS, they may suffer from issues such as weight interference and suboptimal performance due to the approximation involved in using shared weights rather than training each architecture independently.

Hardware-Aware Neural Architecture Search (HW-NAS) is a specialized approach within NAS that focuses on optimizing neural network architectures not only for high accuracy but also for compatibility with specific hardware constraints, such as computational power, memory usage, latency, and energy consumption. This approach is crucial because, in real-world applications, models must run efficiently on various devices, from powerful servers to resource-constrained edge devices like smartphones and IoT gadgets. By incorporating hardware considerations into the NAS process, HW-NAS ensures that the resulting models are both performant and practical for deployment, leading to more efficient use of resources, reduced operational costs, and broader applicability across different platforms and devices.

4 NAS for Transformers

In this section, I will present some relevant work in the field of NAS for Transformers, highlighting key advancements and innovations. Specifically, I will discuss two notable examples: The Evolved Transformer (So et al. (2019)) and NAS-BERT (Xu et al. (2021)).

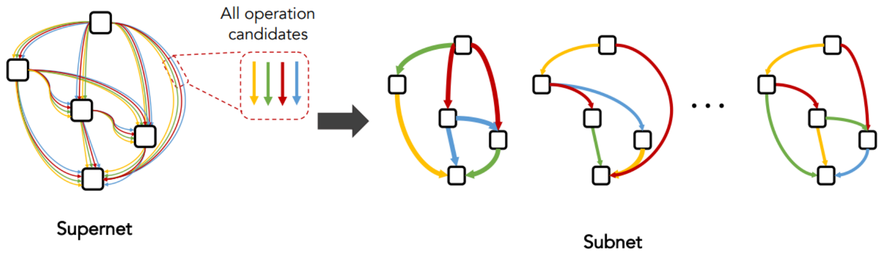


Figure 6: A supernet encompass all possible networks as subnetowrks (White et al. (2023))

4.1 The Evolved Transformer

The Evolved Transformer represents a variation of the original Transformer architecture introduced in (Vaswani et al. (2017)). This model introduces several innovative modifications to the classic Transformer structure, which have been shown to boost performance on various natural language processing tasks.
The Evolved Transformer utilizes a cell-based search space, where the architecture is designed by creating two distinct cells: one for the encoder and one for the decoder. These cells are constructed from multiple blocks, six for the encoder and eight for the decoder. The final Transformer architecture is formed by stacking these cells multiple times as shown in Figure 7.

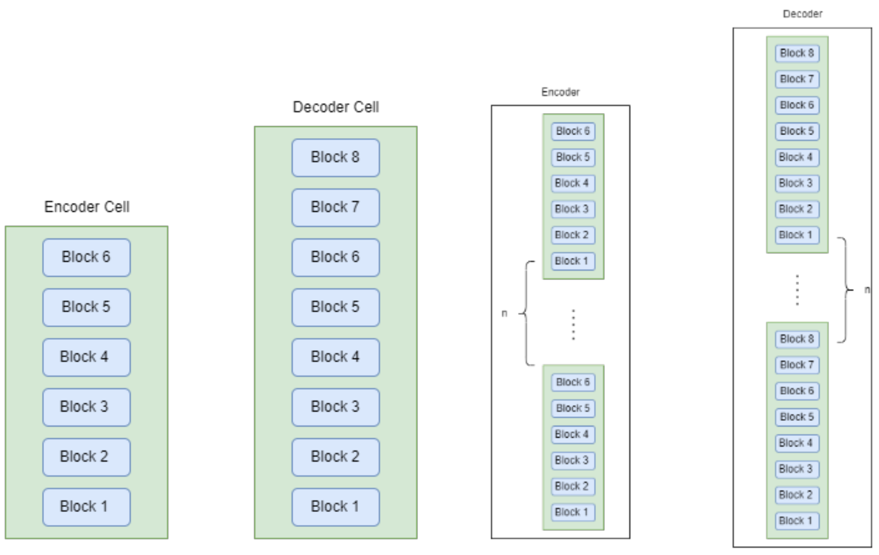


Figure 7: The final architecture is created by stacking the encoder and decoder cells

At the heart of the search space is the design of the individual blocks. Each block is divided into two branches (left and right), with each branch receiving different input data, performing a unique transformation, and then combining their outputs into a single block output and creating a new hidden state. The operations available for selection by the NAS algorithm were manually chosen by experts and are detailed in Appendix A. Figure 8 shows a visual representation of the block, the branches and how it relates to the cell structure of the encoder and the decoder.

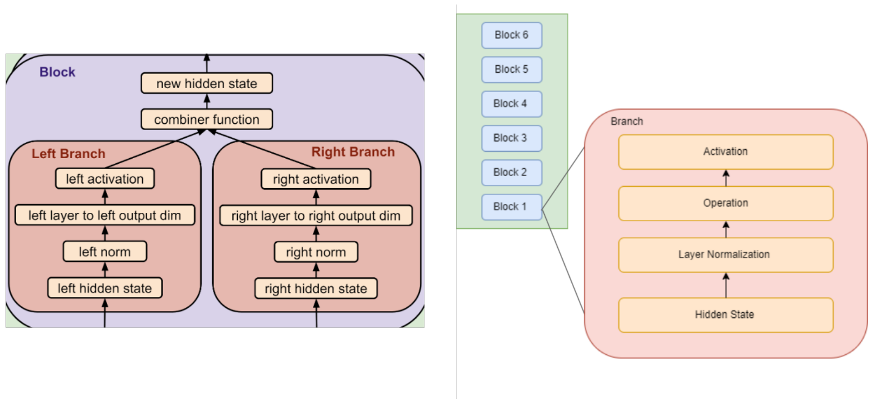


Figure 8: Each block is divided in left and right branches. Left image from (So et al. (2019))

The authors used evolutionary algorithms as a search strategy but given the complexity of the task, they had to adopt new solutions to speed up the training and they introduced a novel evolutionary algorithm called Progressive Dynamic Hurdles (PDH) which dynamically allocate resources to more promising architectures according to their fitness. The process starts with a standard tournament selection, where a random group of candidate architectures is evaluated, and the ones with the highest performance are selected as parents of the generation. To these architectures are then applied mutation and trained for a short time and then evaluated according to their fitness. After the initial evaluation, a hurdle is established, defined as the average performance of the architectures in that generation. The process then repeats: new architectures are selected as parents, undergo mutation, and are trained for a short period. In this round, the performance of the current generation's architectures is compared against the previous hurdle. If an architecture's performance exceeds this hurdle, it is granted additional training time. For the next parent selection, architectures will be chosen from those that have received more training time, ensuring that the most promising architectures continue to evolve and improve. In essence, PDH acts like a series of fitness tests with progressively higher bars. Only the architecture that consistently perform well enough to clear these hurdles get more training time, allowing for a more efficient search for the best architecture. Figure 9 represent a flowchart of PDH.

The Evolved Transformer outperforms the original Transformer both in terms of perplexity and BLUE scores in machine translation and language modeling tasks for different parameter sizes, as shown in Figure 10. This is evidenced by its lower perplexity scores (ET Perp) and higher BLEU scores (ET BLEU) across various language pairs and model configurations. The perplexity metric, which quantifies the model's uncertainty when predicting the next word in a sequence, is consistently lower for the Evolved Transformer than for the original Transformer, indicating a better performance for the new architecture. Similarly, the BLEU scores, which measure the quality of machine-generated translations against reference translations, are higher for the Evolved Transformer, also indicating a better performance for the Evolved Transformer.

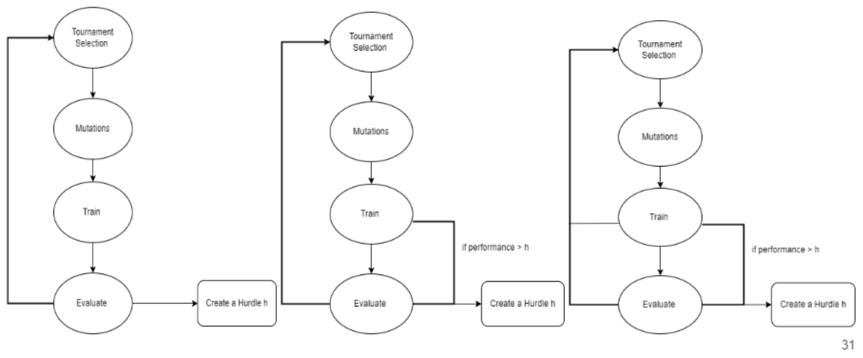
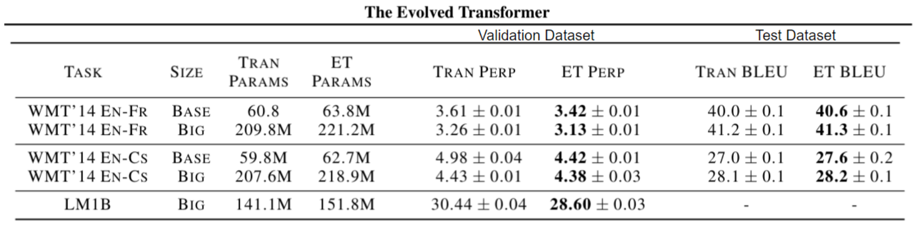


Figure 9: Progressive Dynamic Hurdle flowchart



The Evolved Transformer

				Validation Dataset		Test Dataset	
Task	Size	Tran Params	ET Params	Tran Perp	ET Perp	Tran BLEU	ET BLEU
WMT'14 En-Fr	Base	60.8	63.8M	3.61 ± 0.01	$\mathbf{3.42} \pm 0.01$	40.0 ± 0.1	$\mathbf{40.6} \pm 0.1$
WMT'14 En-Fr	Big	209.8M	221.2M	3.26 ± 0.01	$\mathbf{3.13} \pm 0.01$	41.2 ± 0.1	$\mathbf{41.3} \pm 0.1$
WMT'14 En-Cs	Base	59.8M	62.7M	4.98 ± 0.04	$\mathbf{4.42} \pm 0.01$	27.0 ± 0.1	$\mathbf{27.6} \pm 0.2$
WMT'14 En-Cs	Big	207.6M	218.9M	4.43 ± 0.01	$\mathbf{4.38} \pm 0.03$	28.1 ± 0.1	$\mathbf{28.2} \pm 0.1$
LM1B	Big	141.1M	151.8M	30.44 ± 0.04	$\mathbf{28.60} \pm 0.03$	-	-

Figure 10: Performance of The Evolved Transformers(ET) and the original Transformer in terms BLUE and Perplexity scores (So et al. (2019))

The Evolved Transformer introduced a novel manually designed search space and a new variation of evolutionary algorithms, the Progressive Dynamic Hurdle. These methods lead to the development of a new variation of the original Transformer, which achieves a better performance on different NLP tasks.

4.2 NAS-BERT

BERT (Devlin et al. (2019)), developed by Google, is a powerful pre-trained Transformer model widely used for natural language understanding tasks such as question answering, text classification, and sentiment analysis. While BERT has set new benchmarks across various NLP tasks (Devlin et al. (2019)), its architecture is manually designed and can be computationally intensive and resource-demanding. NAS-BERT (Xu et al. (2021)) seeks to address these challenges by automatically discovering more efficient and high-performing versions of the BERT architecture using HW-NAS methods. The authors used a chain-structured search space with a pre-defined set of operations as shown in Appendix B. NAS-BERT employs a One-Shot approach to optimize its architecture. This method involves dividing a Supernet into 24 layers, where each layer contains all possible operations that can be selected during the search process. The optimal architecture is determined by connecting selected operations across these layers. To accelerate training, the authors implemented a block-wise search strategy, partitioning the Supernet into four blocks, each containing six layers. They further enhanced the training process using knowledge distillation, leveraging

BERT Base pre-trained on the Wikipedia corpus as a teacher model and training the candidate architectures to replicate its performance, block by block. Additionally, latency was considered a hardware constraint within each block, enabling the algorithm to tailor the architecture to different hardware specifications.

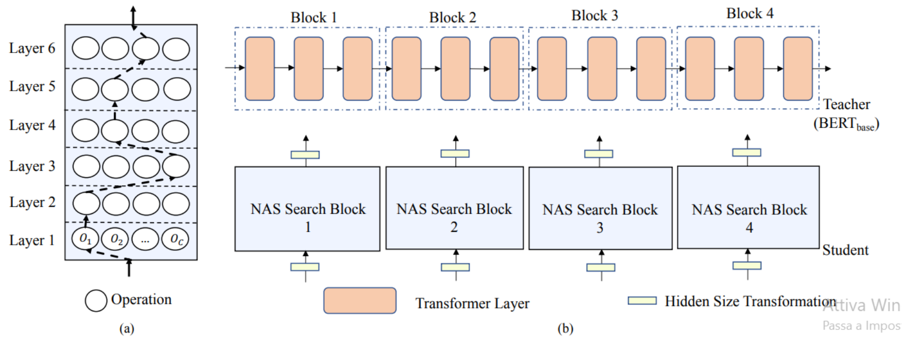
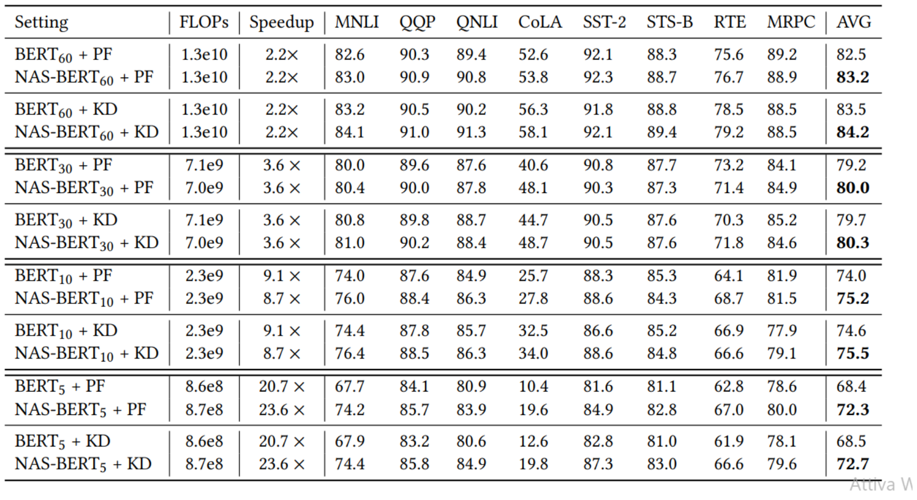


Figure 11: (a) Block of the Supernet (b) NAS-BERT training with knowledge distillation (Xu et al. (2021))

As a result, various architectures were discovered, each optimized for different parameter sizes and latency constraints, as detailed in Appendix B. The final architectures were retrained with knowledge distillation on the English Wikipedia and Book Corpus, and subsequently fine-tuned on benchmark datasets such as the General Language Evaluation Benchmark (GLUE) (Wang et al. (2019)) and the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al. (2018)). The results, illustrated in Figure 12, demonstrate the efficacy of NAS-BERT.



Setting	FLOPs	Speedup	MNLI	QQP	QNLI	CoLA	SST-2	STS-B	RTE	MRPC	AVG
BERT$_{60}$ + PF	1.3e10	2.2×	82.6	90.3	89.4	52.6	92.1	88.3	75.6	89.2	82.5
NAS-BERT$_{60}$ + PF	1.3e10	2.2×	83.0	90.9	90.8	53.8	92.3	88.7	76.7	88.9	**83.2**
BERT$_{60}$ + KD	1.3e10	2.2×	83.2	90.5	90.2	56.3	91.8	88.8	78.5	88.5	83.5
NAS-BERT$_{60}$ + KD	1.3e10	2.2×	84.1	91.0	91.3	58.1	92.1	89.4	79.2	88.5	**84.2**
BERT$_{30}$ + PF	7.1e9	3.6 ×	80.0	89.6	87.6	40.6	90.8	87.7	73.2	84.1	79.2
NAS-BERT$_{30}$ + PF	7.0e9	3.6 ×	80.4	90.0	87.8	48.1	90.3	87.3	71.4	84.9	**80.0**
BERT$_{30}$ + KD	7.1e9	3.6 ×	80.8	89.8	88.7	44.7	90.5	87.6	70.3	85.2	79.7
NAS-BERT$_{30}$ + KD	7.0e9	3.6 ×	81.0	90.2	88.4	48.7	90.5	87.6	71.8	84.6	**80.3**
BERT$_{10}$ + PF	2.3e9	9.1 ×	74.0	87.6	84.9	25.7	88.3	85.3	64.1	81.9	74.0
NAS-BERT$_{10}$ + PF	2.3e9	8.7 ×	76.0	88.4	86.3	27.8	88.6	84.3	68.7	81.5	**75.2**
BERT$_{10}$ + KD	2.3e9	9.1 ×	74.4	87.8	85.7	32.5	86.6	85.2	66.9	77.9	74.6
NAS-BERT$_{10}$ + KD	2.3e9	8.7 ×	76.4	88.5	86.3	34.0	88.6	84.8	66.6	79.1	**75.5**
BERT$_5$ + PF	8.6e8	20.7 ×	67.7	84.1	80.9	10.4	81.6	81.1	62.8	78.6	68.4
NAS-BERT$_5$ + PF	8.7e8	23.6 ×	74.2	85.7	83.9	19.6	84.9	82.8	67.0	80.0	**72.3**
BERT$_5$ + KD	8.6e8	20.7 ×	67.9	83.2	80.6	12.6	82.8	81.0	61.9	78.1	68.5
NAS-BERT$_5$ + KD	8.7e8	23.6 ×	74.4	85.8	84.9	19.8	87.3	83.0	66.6	79.6	**72.7**

Figure 12: NAS-BERT vs BERT (Xu et al. (2021))

In this context, “PF” denotes models that have been both pre-trained and fine-tuned, while “KD” refers to those trained using the knowledge distillation approach described above. Floating Point Operations Per Second (FLOPs) serve as a metric for assessing the

computational efficiency of different algorithms, with higher FLOPs indicating a greater capacity for handling complex calculations quickly. In the study, speedup was evaluated against baseline BERT models (models with the same parameter count trained without NAS techniques), although it remains unclear whether this speedup pertains to inference or training time. Performance metrics were reported across the GLUE datasets, with accuracy used for all datasets except STS-B (Pearson correlation) and CoLA (Matthews correlation). The final columns summarize the average performance of each model. Across all parameter sizes and training methods, NAS-BERT models consistently achieve superior performance compared to their BERT counterparts, with the performance advantage being more pronounced in models with fewer parameters, which also exhibit greater speedup.
By utilizing a One-Shot approach within a chain-structured search space, NAS-BERT efficiently explores a vast array of possible architectures, optimizing not only for performance but also for hardware constraints and varying parameter sizes. The results show that NAS-BERT consistently outperforms baseline BERT models across multiple benchmarks, achieving higher accuracy and greater computational efficiency.

5 Conclusion

The Evolved Transformer and NAS-BERT both employ NAS methods to discover high-performing Transformer architectures, but they approach the task in fundamentally different ways. The Evolved Transformer uses a cell-structured search space, whereas NAS-BERT operates within a chain-structured search space. Additionally, the Evolved Transformer leverages a novel variation of an evolutionary algorithm known as the Progressive Dynamic Hurdle, while NAS-BERT employs a One-Shot approach. NAS-BERT also considers different hardware constraints and optimizes for varying numbers of parameters, unlike the Evolved Transformer, which focuses on a single architecture that is subsequently scaled to different parameter sizes. Despite these differences, the two methods share some similarities: both use a manually designed search space and incorporate techniques to accelerate the training process.

In conclusion, this report has explored the application of Neural Architecture Search to optimize Transformer architectures, demonstrating its potential to discover high-performing models with limited manual intervention. By examining the fundamental components of Transformers and the key elements of NAS, including the search space, search strategies, performance estimation strategies, and the One-Shot approach, we have highlighted how NAS can effectively automate the design process. The focus on chain-structured and cell-structured search spaces, along with evolutionary algorithms, illustrates the diverse methodologies employed to enhance model performance and efficiency. The case studies of the Evolved Transformer and NAS-BERT further exemplify the success of NAS in generating optimized architectures tailored to specific tasks and hardware constraints. Overall, the report underscores the significant impact of NAS in advancing Transformer design, paving the way for more innovative and resource-efficient models in the future.

6 Limitations and Future Directions

The field of NAS for Transformers, including approaches like the Evolved Transformer and NAS-BERT, has made significant progress in automating the design of high-performing neural architectures. However, there are several limitations that currently restrict the full potential of NAS for Transformers and present opportunities for future research. One key limitation is the reliance on manually designed search spaces. This approach can undermine the core objective of NAS by introducing human biases, restricting the exploration to familiar architectures, and preventing the discovery of genuinely novel structures. By relying on predefined configurations, the process risks overlooking innovative architectures that might arise from a more automated and unbiased search.

While methods like NAS-BERT and the Evolved Transformer have introduced certain speed-up techniques, such as block-wise search and knowledge distillation, to enhance the efficiency of the NAS process, there remains considerable room for further advancements. In the future, developing more HW-NAS methods or adapting existing speed-up techniques, could provide significant benefits to the research community. Additionally, recent advancements in AI, such as OpenAI's GPT and Google's BERT, have predominantly been driven by private companies with strong financial incentives to optimize the discovery and training of these models. This underscores the importance of HW-NAS, as it not only addresses computational efficiency but also potentially reduces costs, making it increasingly relevant for both academic research and industry applications moving forward.

Addressing these limitations could be beneficial for advancing the field of NAS for Transformers. By overcoming these challenges, future research can fully harness the potential of NAS to discover truly innovative and highly efficient Transformer architectures across diverse tasks and domains.

References

Kunlong Chen, Liu Yang, Yitian Chen, Kunjin Chen, Yidan Xu, and Lujun Li. Gp-nas-ensemble: a model for nas performance prediction, 2023. URL `https://arxiv.org/abs/2301.09231`.

Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction, 2018. URL `https://arxiv.org/abs/1809.04184`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL `https://arxiv.org/abs/1810.04805`.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL `https://arxiv.org/abs/2010.11929`.

Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V. Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for recognition and localization, 2020. URL `https://arxiv.org/abs/1912.05027`.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey, 2019. URL `https://arxiv.org/abs/1808.05377`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 34(2):550–570, February 2023. ISSN 2162-2388. doi: 10.1109/tnnls.2021.3100554. URL `http://dx.doi.org/10.1109/TNNLS.2021.3100554`.

Ye Qiao, Haocheng Xu, and Sitao Huang. Tg-nas: Leveraging zero-cost proxies with transformer and graph convolution networks for efficient neural architecture search, 2024. URL `https://arxiv.org/abs/2404.00271`.

Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018. URL `https://api.semanticscholar.org/CorpusID:49313245`.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018. URL `https://arxiv.org/abs/1806.03822`.

David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

David R. So, Chen Liang, and Quoc V. Le. The evolved transformer, 2019. URL `https://arxiv.org/abs/1901.11117`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL `https://arxiv.org/abs/1706.03762`.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019. URL `https://arxiv.org/abs/1804.07461`.

Lilian Weng. Neural architecture search. *lilianweng.github.io*, Aug 2020. URL `https://lilianweng.github.io/posts/2020-08-06-nas/`.

Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers, 2023. URL `https://arxiv.org/abs/2301.08727`.

Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. Nasbert: Task-agnostic and adaptive-size bert compression with neural architecture search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery amp; Data Mining*, KDD '21. ACM, August 2021. doi: 10.1145/3447548.3467262. URL `http://dx.doi.org/10.1145/3447548.3467262`.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017. URL `https://arxiv.org/abs/1611.01578`.

Appendix A.

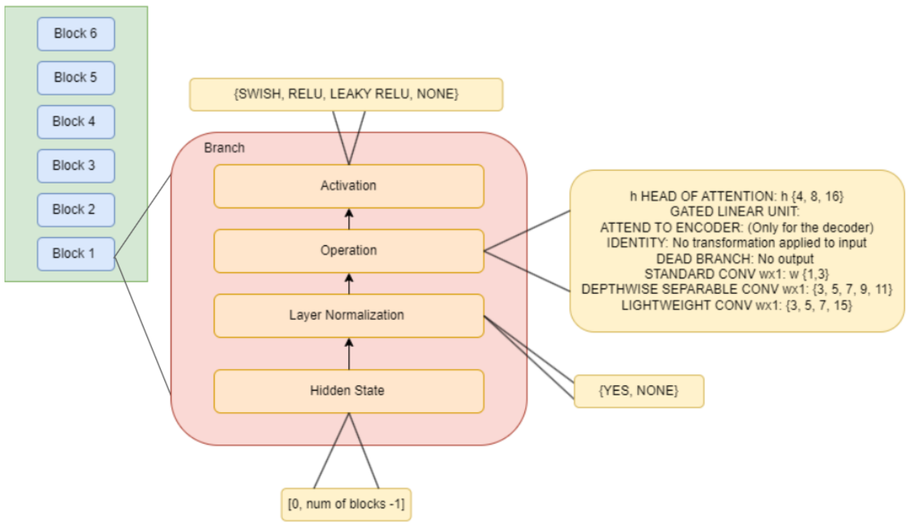

Block 6
Block 5
Block 4
Block 3
Block 2
Block 1
{SWISH, RELU, LEAKY RELU, NONE}
Branch
Activation
Operation
Layer Normalization
Hidden State
h HEAD OF ATTENTION: h {4, 8, 16}
GATED LINEAR UNIT:
ATTEND TO ENCODER: (Only for the decoder)
IDENTITY: No transformation applied to input
DEAD BRANCH: No output
STANDARD CONV wx1: w {1,3}
DEPTHWISE SEPARABLE CONV wx1: {3, 5, 7, 9, 11}
LIGHTWEIGHT CONV wx1: {3, 5, 7, 15}
{YES, NONE}
[0, num of blocks -1]

Figure 13: For each branch there are four operations: Hidden State, Layer Normalization, Operation and Activation. For each of these operations are listed the possible options.

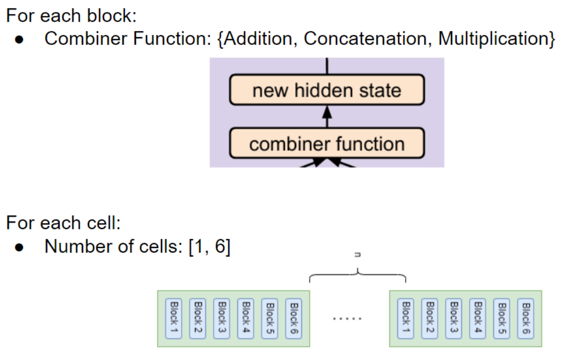


Figure 14: For each block two options are available for the combiner function: Concatenation and Multiplication. After applying the combiner function a new hidden state is created. For each cell the final number of cell can be decided.

The authors specifically stated the final encoder cell has six blocks and the final decoder cell has eight blocks, but in Figure 15 we can count five encoder blocks and six decoder blocks. The matter is also discussed here.

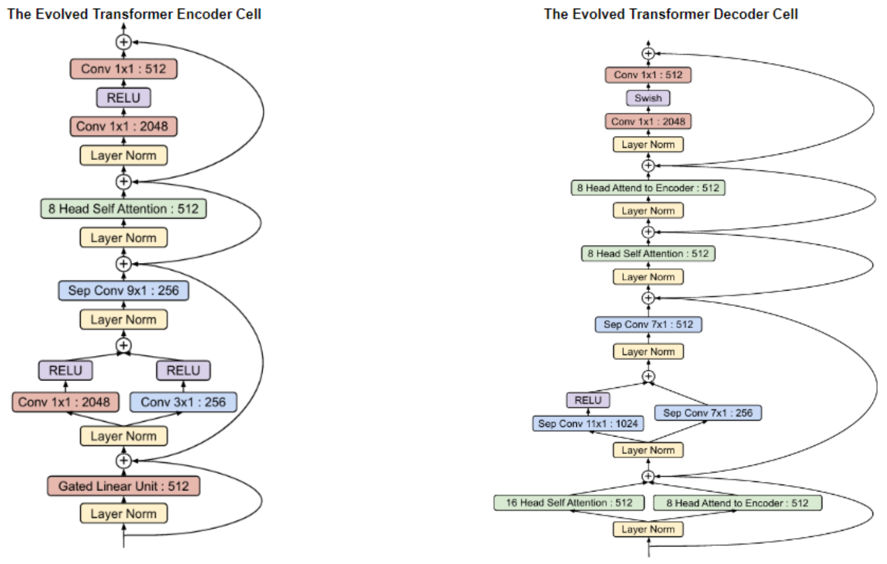


Figure 15: Final ET architecture (So et al. (2019))

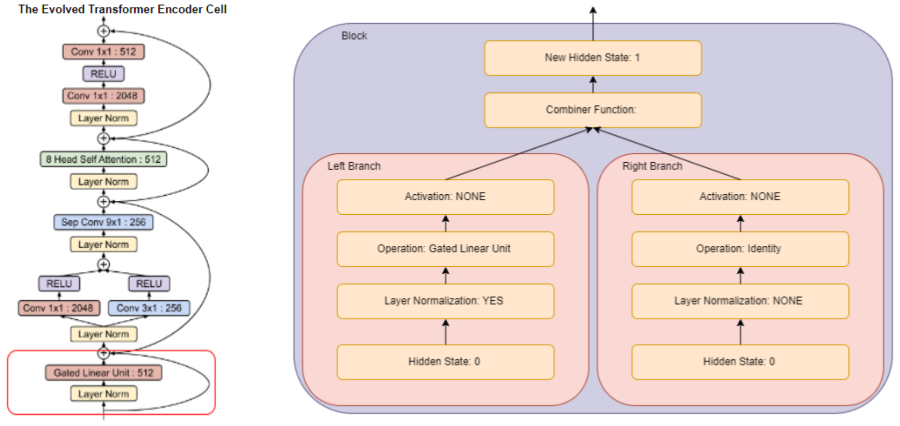
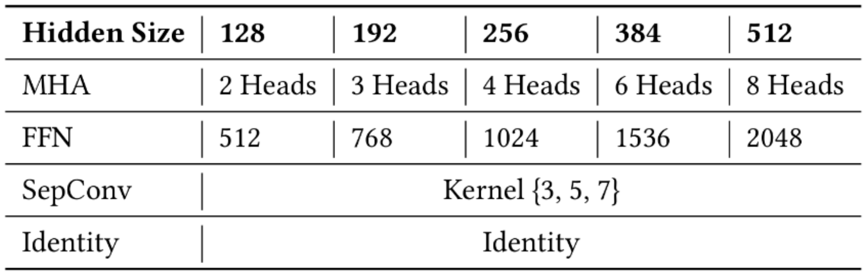


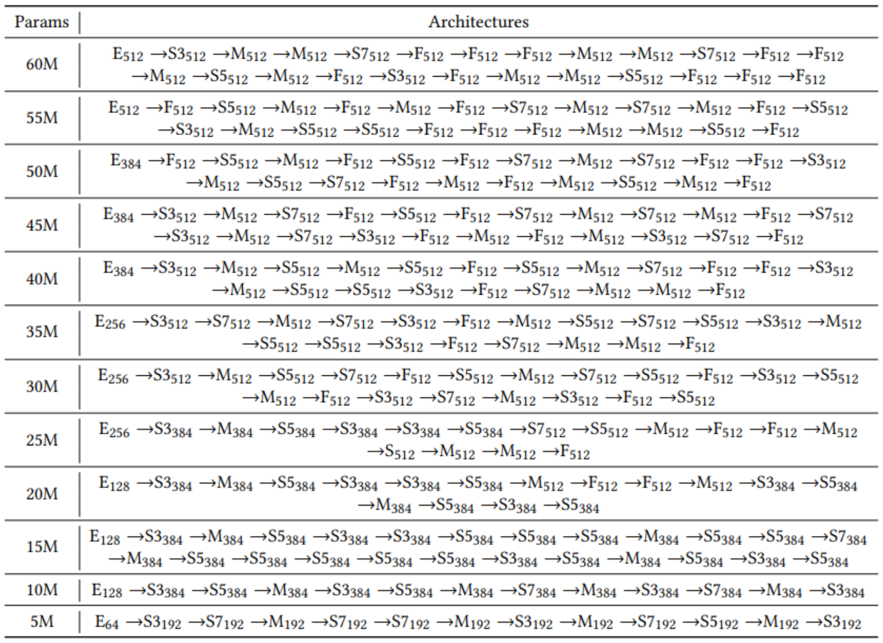
Figure 16: How the final ET architecture relates to the cell search space.

Appendix B.



Hidden Size	**128**	**192**	**256**	**384**	**512**
MHA	2 Heads	3 Heads	4 Heads	6 Heads	8 Heads
FFN	512	768	1024	1536	2048
SepConv	Kernel {3, 5, 7}				
Identity	Identity				

Figure 17: Operations for the NAS-BERT search space (Xu et al. (2021))



Params	Architectures
60M	$E_{512} \rightarrow S3_{512} \rightarrow M_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow S3_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow F_{512}$
55M	$E_{512} \rightarrow F_{512} \rightarrow S5_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow S7_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow S5_{512} \rightarrow S3_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow S5_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow F_{512}$
50M	$E_{384} \rightarrow F_{512} \rightarrow S5_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow S5_{512} \rightarrow F_{512} \rightarrow S7_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow S3_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow S7_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow M_{512} \rightarrow F_{512}$
45M	$E_{384} \rightarrow S3_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow F_{512} \rightarrow S5_{512} \rightarrow F_{512} \rightarrow S7_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow S7_{512} \rightarrow S3_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow S3_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow S3_{512} \rightarrow S7_{512} \rightarrow F_{512}$
40M	$E_{384} \rightarrow S3_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow F_{512} \rightarrow S5_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow S3_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow S5_{512} \rightarrow S3_{512} \rightarrow F_{512} \rightarrow S7_{512} \rightarrow M_{512} \rightarrow M_{512} \rightarrow F_{512}$
35M	$E_{256} \rightarrow S3_{512} \rightarrow S7_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow S3_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow S7_{512} \rightarrow S5_{512} \rightarrow S3_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow S5_{512} \rightarrow S3_{512} \rightarrow F_{512} \rightarrow S7_{512} \rightarrow M_{512} \rightarrow M_{512} \rightarrow F_{512}$
30M	$E_{256} \rightarrow S3_{512} \rightarrow M_{512} \rightarrow S5_{512} \rightarrow S7_{512} \rightarrow F_{512} \rightarrow S5_{512} \rightarrow M_{512} \rightarrow S7_{512} \rightarrow S5_{512} \rightarrow F_{512} \rightarrow S3_{512} \rightarrow S5_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow S3_{512} \rightarrow S7_{512} \rightarrow M_{512} \rightarrow S3_{512} \rightarrow F_{512} \rightarrow S5_{512}$
25M	$E_{256} \rightarrow S3_{384} \rightarrow M_{384} \rightarrow S5_{384} \rightarrow S3_{384} \rightarrow S3_{384} \rightarrow S5_{384} \rightarrow S7_{512} \rightarrow S5_{512} \rightarrow M_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow S_{512} \rightarrow M_{512} \rightarrow M_{512} \rightarrow F_{512}$
20M	$E_{128} \rightarrow S3_{384} \rightarrow M_{384} \rightarrow S5_{384} \rightarrow S3_{384} \rightarrow S3_{384} \rightarrow S5_{384} \rightarrow M_{512} \rightarrow F_{512} \rightarrow F_{512} \rightarrow M_{512} \rightarrow S3_{384} \rightarrow S5_{384} \rightarrow M_{384} \rightarrow S5_{384} \rightarrow S3_{384} \rightarrow S5_{384}$
15M	$E_{128} \rightarrow S3_{384} \rightarrow M_{384} \rightarrow S5_{384} \rightarrow S3_{384} \rightarrow S3_{384} \rightarrow S5_{384} \rightarrow S5_{384} \rightarrow S5_{384} \rightarrow M_{384} \rightarrow S5_{384} \rightarrow S5_{384} \rightarrow S7_{384} \rightarrow M_{384} \rightarrow S5_{384} \rightarrow S5_{384} \rightarrow S5_{384} \rightarrow S5_{384} \rightarrow S5_{384} \rightarrow S3_{384} \rightarrow S5_{384} \rightarrow M_{384} \rightarrow S5_{384} \rightarrow S3_{384} \rightarrow S5_{384}$
10M	$E_{128} \rightarrow S3_{384} \rightarrow S5_{384} \rightarrow M_{384} \rightarrow S3_{384} \rightarrow S5_{384} \rightarrow M_{384} \rightarrow S7_{384} \rightarrow M_{384} \rightarrow S3_{384} \rightarrow S7_{384} \rightarrow M_{384} \rightarrow S3_{384}$
5M	$E_{64} \rightarrow S3_{192} \rightarrow S7_{192} \rightarrow M_{192} \rightarrow S7_{192} \rightarrow S7_{192} \rightarrow M_{192} \rightarrow S3_{192} \rightarrow M_{192} \rightarrow S7_{192} \rightarrow S5_{192} \rightarrow M_{192} \rightarrow S3_{192}$

Figure 18: Final architectures found by NAS-BERT. "E", "M", "F" and "S" refer to Embedding, MHA, FFN and SepConv. "S3" refers to SepConv with the kernel size 3. The number in the subscript refers to the hidden size (Xu et al. (2021))