SuperShaper: Task-Agnostic Super Pre-training of BERT Models with Variable Hidden Dimensions

Anonymous ACL submission

Abstract

Task-agnostic pre-training followed by task-specific fine-tuning is a default approach to train NLU models which need to be deployed on devices with varying resource and accuracy constraints. However, repeating pre-training and fine-tuning across tens of devices is prohibitively expensive. To address this, we propose SuperShaper, a task agnostic pre-training approach wherein we pre-train a single model which subsumes a large number of Transformer models by varying shapes, i.e., by varying the hidden dimensions across layers. This is enabled by a backbone network with linear bottleneck matrices around each Transformer layer which are sliced to generate differently shaped sub-networks. Despite its simple design space and efficient implementation, SuperShaper radically simplifies NAS for language models and discovers networks that effectively trade-off accuracy and model size: Discovered networks are more accurate than a range of hand-crafted and automatically searched networks on GLUE benchmarks. Further, we find two critical advantages of shape as a design variable for Neural Architecture Search (NAS): (a) networks found with these heuristics derived for *good shapes*, match and even improve on carefully searched networks across a range of parameter counts, and (b) the latency of networks across multiple CPUs and GPUs are insensitive to the shape and thus enable device-agnostic search.

1 Introduction

In the past decade, there has been a surge in public and private cloud usage which has centralized compute and storage. However, rising cloud costs, ever powerful client devices, and increased call for privacy favors (distributed) compute on edge (client) devices. Deployment of compute-intensive AI models addressing the distribution-centralization gap requires developers to ensure that their models are deployable on tens of diverse devices spanning CPU and GPU setups on cloud and client devices.

AI models, for NLP and NLU in particular, are typically developed via the pre-train and fine-tune approach (Devlin et al., 2019), where the former is significantly more compute intensive than the latter (Liu et al., 2021). Ideally, this should be done for every point in the product space of multiple tasks and multiple devices with different model variants. However, this is prohibitively expensive and is addressed in one of 3 ways: (a.) Pre-train a single large language model, such as BERT, agnostic of task and device, followed by device and task specific model sizing via knowledge distillation (Tang et al., 2019; Turc et al., 2019a; Sanh et al., 2019a; Jiao et al., 2020), pruning (Michel et al., 2019; Goyal et al., 2020), quantization (Shen et al., 2020), factorization (Ma et al., 2019), etc. (b.) Pre-train a single language model but simultaneously fine-tune many sub-networks of different sizes, in what we call *super fine-tuning*. Then for a chosen task and device, an appropriately sized sub-network can be sampled from the super-network and deployed. Examples of such works are DynaBERT (Hou et al., 2020) and YOCO-BERT (Zhang et al., 2021). (c.) Instead of pre-training one large language model, an entire family of language models is trained, in what we call *super pre-training* which was explored in NAS-BERT (Xu et al., 2021).

Super pre-training is more attractive than the other approaches because the pre-trained model avoids the need for model compression which inherently lossy and reduces generalizability while being aware of model shapes and sizes agnostic of the downstream task. That being said, super pre-training involves searching for pre-training architectures from scratch and existing efforts (Xu et al., 2021; Hou et al., 2020) propose complex methods for reducing the search space by discretizing the network into blocks, heuristic based search space pruning among others. We propose an alternative approach to super-training language models by simplifying this design space, called SuperShaper.

SuperShaper, like NAS-BERT, is task-agnostic but differs from existing methods in two crucial ways: First, it starts out with an existing pretrained BERT model and its search space is defined only by the hidden dimension of each Transformer layer, which we refer to as the *shape* of the network. This is enabled by modifying the BERT backbone with bottleneck matrices at the input and output of each layer, inspired from MobileBERT (Sun et al., 2020). In each batch, differently shaped networks are randomly sampled by slicing the bottleneck matrices and trained. Though a single parameter per layer, the hidden dimension sensitively affects model capacity as the parameter count linearly depends on it. Second, the super pre-training procedure is much simpler with SuperShaper requiring only sliced matrix multiplications on the bottleneck matrices, similar to the earliest techniques proposed for elastic training (Brock et al., 2018; Cai et al., 2019). This is radically simpler than existing NAS techniques which define complex design spaces, architecture modifications, and heuristics for managing the search space. In PyTorch, only 20 lines of additional code are required to add SuperShaper functionality (see Appendix). The SuperShaper model is a proxy for models with various shapes that would otherwise be trained separately. Then, we can use Evolutionary Algorithms (EA) to find optimal sub-networks that are accurate and meet given parameter and device constraints. These sub-networks are fine-tuned for downstream tasks.

Despite the simple design space and efficient implementation, SuperShaper helps identify sub-networks that are competitive on GLUE tasks with BERT-base as well as with many compressed models (both hand-crafted and searched with NAS) at lower parameter counts. In the 60-66M parameter regime, the model found with SuperShaper performs better on GLUE than larger models identified with many successful techniques such as LayerDrop, DistilBERT, Bert-PKD, miniLM, TinyBERT, BERT-of-Theseus, PD-BERT, and YOCO-BERT. Only NAS-BERT, with its much larger search space and knowledge distillation reports a higher accuracy by 1%. Analyses of networks searched via EAs help identify heuristics of good shapes, which suggest a *cigar-like* shape. By applying these heuristics, we hand-craft sub-networks across a range of parameter counts which match and often exceed the performance of networks searched with EAs. Thus, Transformer shapes afford interpretable generalization of model compression across a range of parameter count constraints indicating that NAS can be performed with radically simpler design spaces and implementations focusing only on the hidden sizes, which generalize across tasks, parameter counts, and devices.

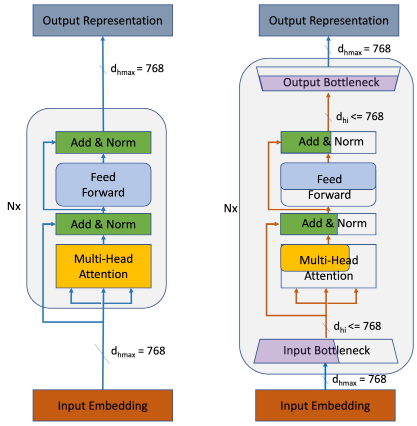

Figure 1: A Transformer layer in (a) BERT, and (b) Backbone in SuperShaper with bottleneck matrices.

2 SuperShaper: The methodology

This section details the SuperShaper methodology focusing on the backbone network, pre-training methods, sub-network search and fine-tuning.

2.1 SuperShaper Backbone

A super pre-training procedure is characterized by a search space of networks. While existing works focus on the number of attention heads, neurons in the FFNs, encoder layers, the use of other operators like separable convolution, etc. for the search space, SuperShaper radically simplifies this by focusing on a single variable - the hidden dimensions for each layer. We focus on SuperShapers based on the Transformer architecture (Vaswani et al., 2017).

In a standard BERT-like encoder (see Figure 1) the hidden dimension d_h of each layer is a constant, e.g., 768 for BERT-base. But with SuperShaper, we would like to explore sub-networks where layers have different hidden dimensions. The intuition behind this choice is that different layers may perform roles of varying importance. For instance, earlier layers manipulating the input embeddings and the final layers responsible for the output may require larger hidden dimensions. To enable this, we take inspiration from MobileBERT (Sun et al., 2020) which proposed a bottleneck layer to compress the parameter size of BERT. Based on this, we modify the standard Transformer layer as shown in Figure 1 (b). The input and output of each transformer layer are intermediated by bottleneck matrices, which translate between the dimension of a token outside a layer (say 768) and the dimension of a token inside a layer (say 120). To reduce the dimension of a layer to 120, we slice the bottleneck matrix at the input from 768×768 to 768×120. With this change, each layer can have differently sized bottleneck matrices such that the hidden dimension varies across layers and we can generate differently shaped sub-networks for super-pretraining.

2.2 Training with SuperShaper

We denote the SuperShaper backbone as T and any sub-network sliced from the backbone as T_S where S is the *shape* vector that represents the layer-wise hidden dimensions, S_i for layer i. The set of all possible values of S denotes the design space D. The smallest and largest sub-networks in D are denoted as T_{S-} and T_{S+}, respectively, while a random sub-network is denoted as T_{S^r}. To evaluate how well a sub-network T_S has trained, we calculate the validation set perplexity, denoted $P(T_S)$, on the Masked Language Modelling (MLM) task.

From a given design space D, we *sample* n different shapes S and obtain T_S for each by the slicing technique described in the previous subsection. This sampling can be performed in two ways: (a) uniform random sampling from D, and (b) random Sampling with *sandwich rule* (Yu and Huang, 2019), where in addition to (a) we also sample the largest and smallest sub-networks T_{S+} and T_{S-}. Sandwich rule has been shown to perform better for weight-sharing NAS in computer vision (Yu and Huang, 2019; Yu et al., 2020; Wang et al., 2021a). For language modelling, we study both sampling methods and report our findings in Section 3. With the sampled sub-networks, gradient updates are computed and parameters are modified with a standard optimizer. Note that the sub-networks share a large number of their parameters, in particular the earlier rows and columns of the bottleneck matrices. Also parts of matrices inside the layer (such as query, key, and value projection matrices) are shared. This parameter sharing is expected to enable generalization during training across the large space of sub-networks. We evaluate and provide empirical evidence for such generalization in Section 3.

2.3 Fine-tuning T_S from SuperShaper

To fine-tune a sampled sub-network T_S for a specific task, several options exist. First, we can sample T_S and fine-tune it directly on the task - $\overline{G}_{\text{direct}}$. Second, we can further pre-train T_S individually and then fine-tune on the task - $\overline{G}_{\text{partial}}$. Finally, we can randomly initialize the weights of T_S and pre-train from scratch before fine-tuning- $\overline{G}_{\text{scratch}}$. We compare these options by fine-tuning on 8 tasks – MNLI-m, QQP, QNLI, CoLA, SST-2, STS-B, RTE, MRPC – from the GLUE benchmark (Wang et al., 2018) and Squad V1 (Rajpurkar et al., 2016).

2.4 Searching for optimal shapes

Once we have super pre-trained with a design space D, we can sample and deploy T_S for any S, which can then be fine-tuned by methods described in the previous subsection. The design space of all sub-networks can be large: A choice of 7 shapes each for 12 layers can yield 14 billion sub-networks. The search question is to find an optimal shape from S which meets specific constraints on accuracy, parameter count, or latency on devices. We adopt Evolutionary Algorithm (EA) from (Real et al., 2017) as a generic optimization technique, which starts with a population of solutions and over generations create new solutions by applying genetic operations like mutation and crossover and retain the fittest solutions based on defined metrics of interest. For SuperShaper, the genetic representation of sub-networks and genetic operations are natural and simply described by the shape vector S. For the fitness metrics, we use perplexity on language modelling and latency on a device. To amortize the expense of computing these metrics for thousands of solutions, we use fitness predictors that have been studied elsewhere in NAS (Cai et al., 2019; Ganesan et al., 2020).

While EA with fitness predictors can search for sub-networks, the most desirable setting is to find sub-networks by applying a set of heuristics to decide the shape of each layer. We propose a technique to discover such heuristics and then use it to identify sub-networks for varying parameter count constraints. We report results on how these compare against EAs in Section 3.

3 Experimental Setup and Results

We now detail the experimental setup and report a range of findings to evaluate SuperShaper.

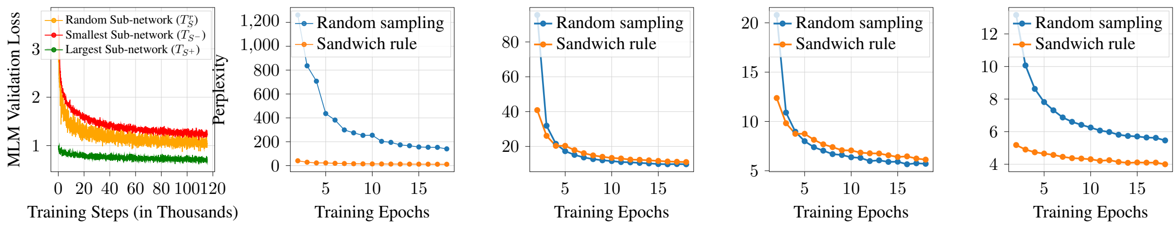

Figure 2: (a) Loss trajectory of T_S^+, T_S^- and T_S networks, (b)-(d) Perplexity trajectory of T_S^-, two randomly sampled T_S^r, and T_S^+ respectively for random-sampling and sandwich rule

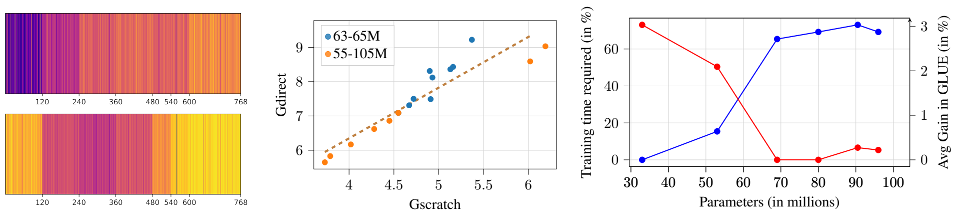

Figure 3: (a) Visualization of input and output bottleneck matrices for the first layer, (b) SuperShaper is a fast and accurate proxy for sub-network perplexity, and (c) $\overline{G}_{\text{partial}}$ inherited sub-networks only require a fraction of pre-training cost (in blue) i.e. 1.3-6.6x reduction to reach optimum. This comes at a higher average gain in GLUE score (in red).

3.1 Experimental Setup

We describe the experimental setups for pre-training and fine-tuning.

Design space We slice the bottleneck matrix to produce Transformer layers of varying hidden dimensions in $\{120, 240, 360, 480, 540, 600, 768\}$, which creates a design space D of 7^{12} or about 14 billion sub-networks.

Super pre-training We initialize our backbone with BERT-base-cased model trained on Wikipedia and BookCorpus with identity bottleneck matrices. We then super pre-train the backbone using Masked language modeling over the C4 RealNews dataset (Raffel et al., 2019) with effective batch size of 2048, max sequence length 128, for 175K steps (or 26 epochs) on 8 A100 GPUs. Other hyperparameters are described in the Appendix.

Fine tuning. Similar to (Xu et al., 2021), we evaluate the effectiveness of SuperShaper by pre-training all our compressed models from scratch and later fine-tune them on 8 GLUE tasks and SQuAD V1. The task details and evaluation metrics are mentioned in the Appendix.

Evolutionary Algorithm (EA). For EA, we adapt the algorithm presented in (Real et al., 2017). We choose a population size of 100, mutation probability of 0.4, and the ratio of parent size to mutation or crossover size as 1. We bound the search algorithm to 300 iterations.

Fitness Predictors. For perplexity predictor, we randomly sample $10,000$ sub-networks and evaluate their perplexity as measured on validation set of C4-RealNews dataset. We use this dataset to build the predictor based on XGBoost model (Chen et al., 2015). For latency predictor, we sample $1,000 - 4,000$ sub-networks and evaluate their latency on the chosen device. We again train a XGBoost model to predict latency from this dataset. We consider 5 devices - 3 GPUs: 1080Ti, 1060Ti and K80, and 2 CPUs: AMD Ryzen CPU and a server class single-core Xeon CPU.

3.2 Pre-training with SuperShaper

Effect of sub-network sampling rule.

In computer vision, sandwich rule is widely used in the context of weight-sharing NAS (Yu and Huang, 2019). We super pre-train the trained backbone network with the sandwich rule. The corresponding loss trajectory for largest, smallest, and randomly sampled sub-networks are shown in Figure 2(a). Clearly, the larger network has a lower perplexity, but the super pre-training ensures that a range of networks are simultaneously trained on the MLM task. Specifically, randomly sampled subnetworks shown as T_{S^r} even though not sampled as frequently as the smallest subnetwork, have a lower perplexity. This provides evidence of generalization during super pre-training.

We now compare the sandwich sampling rule with fully randomised sampling. We plot the perplexity of 4 networks: the largest, smallest, and two other intermediate networks in Figure 2(b)-(e). Sandwich sampling always samples the largest and smallest and thus the perplexity on these networks is significantly lower with sandwich sampling than random sampling. This suggests that sandwich sampling effectively combines good extremum sub-networks with reasonably good intermediate sub-networks. In all subsequent experiments, we use sandwich sampling.

Visualizing bottleneck matrices.

We initialize the bottleneck matrices to identity weights and zero bias. After super pre-training, we visualize these matrices to understand the role of sliced training of sub-networks. We take the softmax of the principal diagonal of the two bottleneck matrices of the first layer, and plot them in Figure 3 (a). We clearly observe that the entries show a banded pattern with boundaries at the shapes in our design space: 120, 240, 360, 480, 540, 600, and 768. This implies that super pre-training learns different linear projections of 768 dimensional input representation to the chosen hidden dimensions. Visualizations for other layers are in the Appendix.

Effectiveness of super pre-training.

We ask two questions towards evaluating the effectiveness of super pre-training: (a) Is the relative performance of sampled sub-networks on the MLM perplexity ($\overline{G}_{\text{direct}}$) correlated with performance of the same sub-networks when pre-trained individually from scratch ($\overline{G}_{\text{scratch}}$)?, and (b) Does the super pre-training afford sub-networks an advantage when being fine-tuned for tasks? For the first question, we sample a set of sub-networks T_S of both varying (33-96M) and similar (63-65M) parameter counts, and plot $\overline{G}_{\text{direct}}$ and $\overline{G}_{\text{scratch}}$ in Figure 3 (b). We notice that $\overline{G}_{\text{direct}}$ and $\overline{G}_{\text{scratch}}$ are highly correlated with a Spearman correlation coefficient of 0.954. This implies that the sub-network's measured MLM perplexity after super pre-training is a good proxy for final performance. We also observe that networks sampled at the similar parameter count (63-65M) have varying performance suggesting the sensitive role of shape in accuracy.

For studying the second question, we pre-train and then fine-tune the varying parameter count sub-networks (33-96M) in two ways (a) by retaining the weights learnt during super pre-training ($\overline{G}_{\text{partial}}$), and (b) starting with random initialization $\overline{G}_{\text{scratch}}$. We plot two quantities in Figure 3 (c): the amount of pre-training time saved with (a) and the additional GLUE score obtained with (a). We observe that models with fewer parameters (30-50M) show significant savings in the pre-training time (up to $6.6\times$) and simultaneously benefit from improved GLUE accuracy (up to 3%). The gains on both axes for larger models are smaller. This suggests that smaller models whose parameters receive more weight updates due to sharing of the earlier rows and columns across sub-networks benefit more from super pre-training. This is encouraging because most effort in deployability is concerned with models of smaller size.

3.3 Comparing sub-networks with other methods

Comparing with BERT-base.

As a first baseline, we search for a SuperShaper-Base model with EA with a constraint of 100M parameters and obtain a model with 96M parameters. This model is comparable against an uncompressed BERT-base model which has 110M parameters. We compare the GLUE and SQuAD V1 performance of SuperShaper-Base ($\overline{G}_{\text{scratch}}$) with two of the top reported results on BERT-Base (Xu et al., 2021; Sanh et al., 2019b). While the task-wise details are in the Appendix, we find that the average GLUE score across the two reported BERT-base baselines (83.7%) is the same as that with SuperShaper-Base (83.7%). For Squad v1, our F1 score of 88.2 is competitive with other baselines - 88.9 and 88.5 Thus, SuperShaper-Base performs competitively with the uncompressed BERT-base with fewer parameters (96M vs 110M).

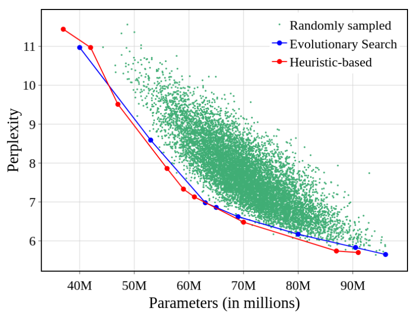

Figure 4: Evolutionary search finds optimal models while simple heuristics yield competitive models.

Comparing with compressed models

We now compare against state-of-the-art compressed models either hand-crafted or found by NAS algorithms (see Table 1). Since several of these models are in the range of 60-67M, we search for a sub-network from SuperShaper with a parameter constraint of 66M. The task-wise performance of the obtained sub-network is reported in Table 1. On GLUE benchmark, SuperShaper outperforms many prominent hand-crafted or compressed networks proposed over the last two years by a significant margin. Across NAS-based methods, SuperShaper performs competitively despite a much simpler design space. On SQuAD, we outperform Bert-PKD, ELM and have competitive results compared to DistillBert while having lesser parameters (63M vs 66-67M). Only NAS-BERT reports a higher average GLUE and better EM/F1 scores, which may be attributed to the use of novel operators such as separable convolution in the design space. Also, NAS-BERT and DynaBERT use knowledge distillation and data augmentation. These methods are orthogonal to shaping and can be combined with our approach. In summary, we establish that SuperShaper with a simple design space and efficient super pre-training implementation performs competitively in compressing models to a given parameter count.

We now apply EA to search for sub-networks at varying parameter count, ranging from 40 to 110M. To understand the effectiveness of EA search, we sample 10,000 random sub-networks and compute their perplexity. We then plot these points along with the networks searched by EA in Figure 4. First, we observe that sub-network's shape critically affects language modeling perplexity. Second, EA effectively searches for accurate networks across the parameter range(33M-100M). We report GLUE scores for these networks in the Appendix.

3.4 Shape analysis of Super-Networks.

In contrast to other NAS techniques, the design space of SuperShaper is interpretable - the network shape. We can thus ask the question: Are there *good shapes* for different model sizes?

Models with templated shapes.

We evaluate the performance of the following templated shapes in the 63-65M parameter range: hidden sizes increase from lower layers to the higher layers, upper triangle, rectangle (all layers have similar hidden sizes), diamond, inverted diamond, bottle, and inverted bottle. Details of the hidden dimensions and sub-network perplexity for each network are in the Appendix. We observe that lower triangle has the lowest perplexity (7.31) while inverted bottle (9.22) has the highest. This wide range reiterates that shape sensitively affects performance. Further, we observe that more parameters in deeper layers benefits model performance.

Feature importance from optimal sub-networks. From the analysis of sub-networks searched by EA and the templated shapes, we find that accurate networks have more parameters in later layers. We analyse this using the perplexity predictor trained to estimate $\overline{G}_{\text{partial}}$ given the shape. For this predictor, we compute the feature importance (plot in the Appendix) of each layer's shape and find it to be highest for the last few layers and the first layer. Based on these observations, we derive a set of *heuristics* indicating good shapes: (a) a large dimension in the last layer, (b) moderately large dimension in the first layer, (c) low dimensions in early middle layers (2-5), and (d) moderate dimensions in later middle layers (6-11). We characterize this as a *cigar-like shape*.

Heuristically shaped models. Based on the above heuristics, we hand-shape sub-networks with the following algorithm: (a) construct a reference model T_{S^*} following the heuristics at a given parameter range (say 60-65M), (b) for a target parameter count, scale the shape S_i of every layer linearly, (c) for early middle layers, round down the scaled S_i (as they have lesser importance) and for remaining layers round up S_i to the nearest configuration in D. Based on this algorithm, we identify sub-networks across the parameter count with cigar-like shapes as shown in Figure 6. We evaluate these hand-crafted sub-networks on perplexity $\overline{G}_{\text{direct}}$ and find that they are competitive and even outperform sub-networks searched with EAs (see Figure 4). We also pre-train and evaluate one of the heuristic models with a parameter count of 61M on the Glue tasks (see Table 1). We observe that, similar to our evolutionary-search sub-network (63M), the heuristic model outperforms prominent hand-crafted or compressed networks This strongly demonstrates the generalization of the derived heuristics across model size. To the best or our knowledge, this is the first such generalization demonstrated for NAS.

3.5 Device-specific efficient models.

We now discuss searching for sub-networks based on latency on a device. We consider 5 devices - 3 GPUs 1080Ti, 1060Ti and K80, 2 CPUs - AMD Ryzen CPU and a server class single-core Xeon

Model	Params	MNLI-m	QQP	QNLI	CoLA	SST-2	STS-B	RTE	MRPC	Avg. GLUE	SQuAD V1
LayerDrop (Fan et al., 2019)	66M	80.7	88.3	88.4	45.4	90.7	-	65.2	85.9	-	-
DistilBERT (Sanh et al., 2019b)	66M	82.2	88.5	89.2	51.3	91.3	86.9	59.9	87.5	79.6	79.1 / 86.9
Bert-PKD (Sun et al., 2019a)	66M	81.5	70.7	89.0	-	92.0	-	65.5	85.0	-	77.1 / 85.3
MiniLM (Wang et al., 2020b)	66M	84.0	91.0	91.0	49.2	92.0	-	71.5	88.4	-	-
Ta-TinyBert (Jiao et al., 2020)	67M	83.5	90.6	90.5	42.8	91.6	86.5	72.2	88.4	80.8	-
Tiny-BERT (Jiao et al., 2020)	66M	84.6	89.1	90.4	51.1	93.1	83.7	70.0	82.6	80.6	79.7 / 87.5
BERT-of-Theseus (Xu et al., 2020)	66M	82.3	89.6	89.5	51.1	91.5	88.7	68.2	-	-	-
PD-BERT (Turc et al., 2019b)	66M	82.5	90.7	89.4	-	91.1	-	66.7	84.9	-	-
ELM (Jiao et al., 2021)	67M	84.2	91.1	90.8	54.2	92.7	88.9	72.2	89.0	82.9	77.2 / 85.7
NAS-BERT∗ (Xu et al., 2021)	60M	83.3	90.9	91.3	55.6	92.0	88.6	78.5	87.5	83.5	80.5 / 88.0
DynaBERT† (Hou et al., 2020)	60M	84.2	91.2	91.5	56.8	92.7	89.2	72.2	84.1	82.8	-
YOCO-bert (Zhang et al., 2021)	59-67M	82.6	90.5	87.2	59.8	92.8	-	72.9	90.3	-	-
SuperShaper (ours)	63M	82.2	90.2	88.1	53.0	91.9	87.6	79.1	89.5	82.7	78.25 / 86.01
SuperShaper heuristic-shaped (ours)	61M	82.0	90.3	88.4	52.6	91.6	87.8	77.6	86.5	82.1	77.86 / 85.83

Table 1: Comparison of SuperShaper with 60-67M parameter constraint models on development set of GLUE. † indicates models trained with data augmentation, ∗ indicates model trained without knowledge distillation in the fine-tuning stage

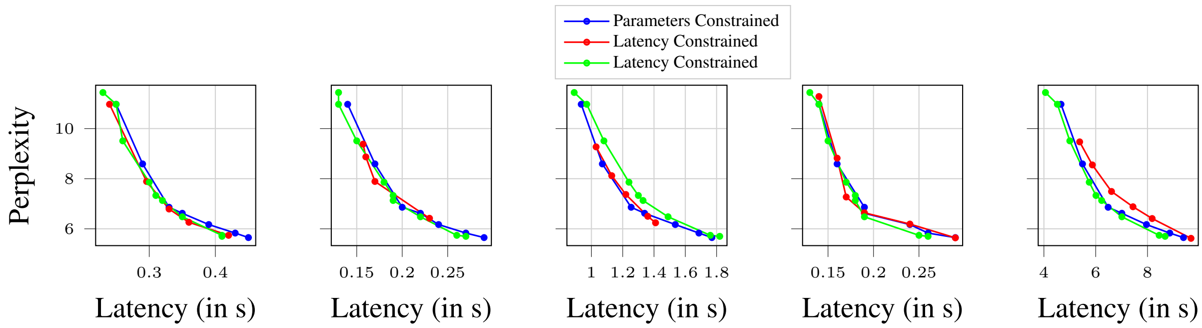

Figure 5: Perplexity vs Latency for optimal models searched using EA with parameter and latency constrained and for heuristically shaped models across: (a) 1080Ti GPU, (b) Xeon CPU, (c) K80 GPU, (d) 1060Ti GPU, and (e) AMD-Ryzen CPU

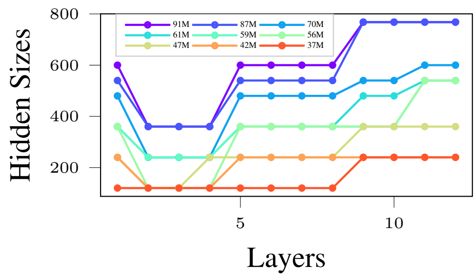

Figure 6: Heuristically shaped models have a cigar-like shape

CPU (for quality of fitness predictors for these devices see Appendix). The feature importance of the latency predictors for these devices strongly favours total parameters and only very weakly depends on layer dimensions (see Appendix). This is a crucial insight: the shape of the network for a given parameter count is a free variable that can be optimized for accuracy. Thus for deployment on a device, we need to identify the right parameter count that meets the latency constraint while the shape can be identified with EA or the heuristics we have laid out.

We run EA under two settings - parameter constraints and latency constraints for all devices. We also evaluate the hand-crafted models. The latency and perplexity of these models are shown in Figure 5. As can be seen, all three techniques result in similar performance. This corroborates that latency is insensitive to shape and that the heuristics identify competitive networks.

In summary, we showed that SuperShaper effectively generalizes training across sub-networks, and finds competitive networks at various sizes. This training on language models enables generalization across tasks. Further we derived a set of simple rules to shape a network which is competitive with EA search, thereby easily generalizing the search across model size. And finally we established that latency on devices is insensitive to shapes and thus EA search on parameter count or hand-crafted networks generalize across devices. Thus, with a simple and effective super pre-training procedure we identify sub-networks that generalize across tasks, model sizes, and devices.

4 Related Work

Over the years, a number of solutions have been proposed for efficient deployment of language models. These can be broadly grouped into the following categories.

4.1 Model Compression

In the context of language models, model compression has been widely applied to reduce computational complexity. Prominent efforts include low-rank approximation of weight matrices (Wang et al., 2019; Ma et al., 2019), pruning attention heads (Michel et al., 2019), tokens (Goyal et al., 2020; Wang et al., 2021b; Kim et al., 2021) or layers (Fan et al., 2019; Sajjad et al., 2020), applying lottery-ticket hypothesis (Frankle and Carbin, 2018) to BERT models (Prasanna et al., 2020; Chen et al., 2020c,d; Yu et al., 2019), and using quantization of weights to lower precisions (Shen et al., 2020; Zafrir et al., 2019).

4.2 Knowledge Distillation

Knowledge distillation (KD) (Hinton et al., 2015) aims to compress the knowledge from a large teacher model to a compact and fast student model. Traditionally, the student models are trained by minimizing the error relative to the soft-targets obtained from the teacher model from the final prediction layer, embedding layer outputs (Sanh et al., 2019a; Jiao et al., 2020), hidden states (Jiao et al., 2020; Sun et al., 2020) or even self-attention outputs (Wang et al., 2020b; Jiao et al., 2020).

KD can either be task-specific (Tang et al., 2019; Turc et al., 2019a; Sun et al., 2019b; Chen et al., 2020a) or task-agnostic (Sanh et al., 2019a; Jiao et al., 2020; Sun et al., 2020) depending on whether the teacher model is fine-tuned on all downstream tasks before distillation.

4.3 Neural Architecture Search

Neural Architecture Search (Zoph and Le, 2017) automates the design of DNNs by searching through a large space of network topologies. Weight-sharing based NAS defines current state-of-the-art (Cai et al., 2019; Yu et al., 2020; Wang et al., 2021a), where model training and sub-network search are decoupled by the use of a super-network subsuming many sub-networks. This process is challenging for language modeling that involves task-agnostic pre-training and task-specific finetuning.

In NLP, many efforts apply NAS to the task-specific fine-tuning stage for optimal NLU models (Gao et al., 2021; Chen et al., 2020b). Recent contemporary efforts focus on the challenging search for task-agnostic models using techniques such as block-wise search, progressive shrinking and stochastic gradient optimization (Xu et al., 2021; Zhang et al., 2021).

In contrast, SuperShaper is a super-pretraining methodology to train a large number of task-agnostic and device-insensitive models in one-shot, thereby simplifying NAS. Many of the model-compression and knowledge-distillation efforts described here are complementary to SuperShaper and can be applied together for more gains. Most importantly, SuperShaper uses a simple design space to effectively train models unlike other contemporary efforts (Xu et al., 2021; Zhang et al., 2021).

5 Conclusions and Future Work

To address the problem of deploying NLU models across a range of devices, we propose SuperShaper, a NAS technique to pre-train language models by shaping Transformer layers. SuperShaper identifies networks that outperform state-of-the-art model compression techniques on GLUE benchmarks. We discovered that cigar-like shapes of networks generalize across parameter counts and device latency is insensitive to shape. Consequently, we demonstrate that NAS can be performed with radically simple design space and implementation, while deriving generalization across tasks, model sizes, and devices. This work can be extended (a) to other tasks such as NLG, and (b) to generate smaller models in combination with other compression techniques.

References

Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer.

Andrew Brock, Theo Lim, JM Ritchie, and Nick Weston. 2018. Smash: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020a. Adabert: Task-adaptive bert compression with differentiable neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2463–2469. International Joint Conferences on Artificial Intelligence Organization. Main track.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020b. Adabert: Task-adaptive bert compression with differentiable neural architecture search. *Cell*, 2(3):4.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020c. The lottery ticket hypothesis for pre-trained bert networks. *arXiv preprint arXiv:2007.12223*.

Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, et al. 2015. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4.

Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. 2020d. Early-bert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Vinod Ganesan, Surya Selvam, Sanchari Sen, Pratyush Kumar, and Anand Raghunathan. 2020. A case for generalizable dnn cost models for mobile devices. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 169–180. IEEE.

Jiahui Gao, Hang Xu, Xiaozhe Ren, Philip LH Yu, Xiaodan Liang, Xin Jiang, Zhenguo Li, et al. 2021. Autobert-zero: Evolving bert backbone from scratch. *arXiv preprint arXiv:2107.07445*.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR.

Geoffrey E. Hinton, Oriol Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. In *Advances in Neural Information Processing Systems*, volume 33, pages 9782–9793. Curran Associates, Inc.

Xiaoqi Jiao, Huating Chang, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2021. Improving task-agnostic bert distillation with layer mapping search. *Neurocomputing*, 461:194–203.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Joseph Hassoun, and Kurt Keutzer. 2021. Learned token pruning for transformers. *arXiv preprint arXiv:2107.00910*.

Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. 2021. Pay attention to mlps. *ArXiv*, abs/2105.08050.

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

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. 2019. A tensorized transformer for language modeling. *Advances in Neural Information Processing Systems*, 32:2232–2242.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*.

Leann Myers and Maria J Sirois. 2004. Spearman correlation coefficients, differences between. *Encyclopedia of statistical sciences*, 12.

Sai Prasanna, Anna Rogers, and Anna Rumshisky. 2020. When bert plays the lottery, all tickets are winning. *arXiv preprint arXiv:2005.00561*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. Poor man's bert: Smaller and faster transformer models. *arXiv e-prints*, pages arXiv–2004.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019a. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019b. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019a. Patient knowledge distillation for bert model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019b. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online. Association for Computational Linguistics.

Raphael Tang, Yao Lu, L. Liu, Lili Mou, Olga Vechtomova, and Jimmy J. Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. *ArXiv*, abs/1903.12136.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019a. Well-read students learn better: The impact of student initialization on knowledge distillation. *ArXiv*, abs/1908.08962.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019b. Well-read students learn better: The impact of student initialization on knowledge distillation. *arXiv preprint arXiv:1908.08962*, 13.

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

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355.

Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. 2021a. Attentivenas: Improving neural architecture search via attentive sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6418–6427.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020a. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*.

Hanrui Wang, Zhekai Zhang, and Song Han. 2021b. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020b. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. BERT-of-Theseus: Compressing BERT by Progressive Module Replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869.

Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. 2021. Nas-bert: Task-agnostic and adaptive-size bert compression with neural architecture search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. 2019. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. In *International Conference on Learning Representations*.

Jiahui Yu and Thomas S Huang. 2019. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1803–1811.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. 2020. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*.

Shaokun Zhang, Xiawu Zheng, Chenyi Yang, Yuchao Li, Yan Wang, Fei Chao, Mengdi Wang, Shen Li, Jun Yang, and Rongrong Ji. 2021. You only compress once: Towards effective and elastic bert compression via exploit-explore stochastic nature gradient. *arXiv preprint arXiv:2106.02435*.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

A Fine tuning tasks and Evaluation metrics

We report performance metrics on the `dev` version of the benchmark. For RTE, MRPC and STS-B, we start with a model fine-tuned on MNLI similar to (Liu et al., 2019; Xu et al., 2021). For metrics, we report Matthews correlation for CoLA (Wang et al., 2018), Spearman correlation for STS-B (Wang et al., 2018) and accuracy for all other tasks. For MNLI-m (Wang et al., 2018), we report accuracy on the matched set. For Squad, we report exact match and F1 score. Following (Devlin et al., 2019; Xu et al., 2021; Zhang et al., 2021; Hou et al., 2020), we also exclude the problematic WNLI dataset. For all the datasets in GLUE, we use the official train and dev splits and download the datasets from HuggingFace datasets[1].

B Hyperparameters used in SuperShaper

The hyperparameters we used for MLM pretraining and finetuning tasks are detailed in Table 2 and Table 3

[1]https://huggingface.co/datasets/glue

Data	C4/RealNews
Max sequence length	128
Batch size	2048
Peak learning rate	2e-5
Number of steps	175K
Warmup steps	10K
Hidden dropout	0
GeLU dropout	0
Attention dropout	0
Learning rate decay	Linear
Optimizer	AdamW
Adam ϵ	1e-6
Adam (β_1, β_2)	(0.9, 0.999)
Weight decay	0.01
Gradient clipping	0

Table 2: Hyperparameters for MLM super pre-training on C4 RealNews. Super pre-training was done on 8 A100 GPUs

	CoLA	Other GLUE tasks	Squad V1
Batch size	{16, 32}	32	{8, 16, 32}
Weight decay	{0, 0.1}	0	{0, 0.1}
Warmup steps	{0, 400}	0	{0, 1000}
Max sequence length	128	128	512
Peak learning rate	5e-5	5e-5	1e-5
Number of epochs	10		
Hidden dropout	0		
GeLU dropout	0		
Attention dropout	0		
Learning rate decay	Linear		
Optimizer	AdamW		
Adam ϵ	1e-6		
Adam (β_1, β_2)	(0.9, 0.999)		
Gradient clipping	0		

Table 3: Hyperparameters for fine-tuning on GLUE and SQuAD V1

C Efficient Deployment of SuperShaper sub-networks

Once the sub-networks are identified through evolutionary-search or proposed heuristics, we combine the output bottleneck matrices of layer i with the input bottleneck matrices of layer $i+1$ for further parameter-efficiency while retaining the functionality.

D Bottleneck Visualization

The visualization of principal diagonals for input and output bottleneck matrices clearly show a banded pattern across all the 12 layers (see Figure 7), strongly corroborating the insight that super pre-training learns different linear projections of 768 dimensional input representation to the chosen hidden dimensions.

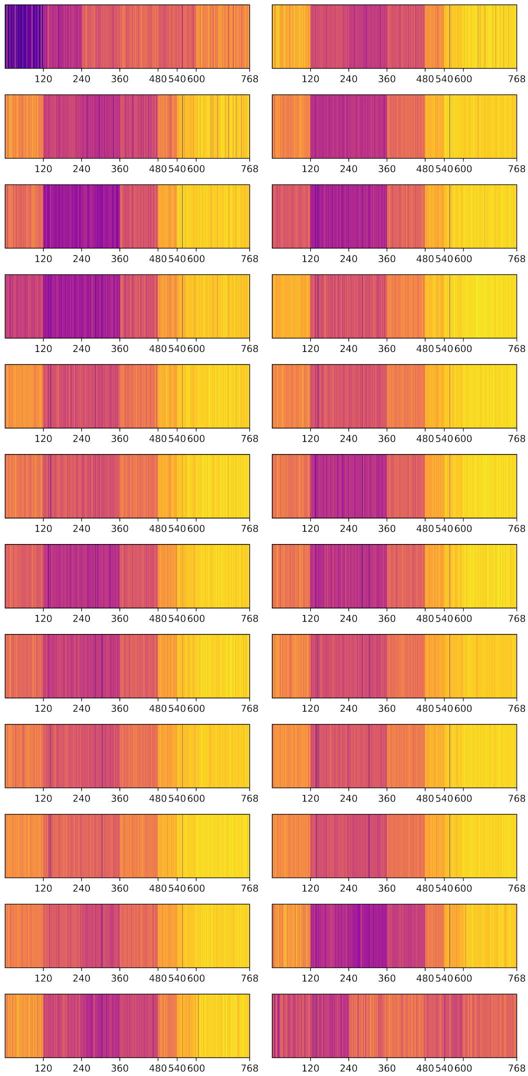

Figure 7: Visualization of input and output Bottleneck matrix diagonals for all the 12 layers.

E Feature importances for optimal sub-networks.

E.1 Perplexity Predictor importances.

Figure 8 shows the importance scores from the perplexity predictor. The patterns used to derive the heuristically-shaped networks are very clear.

E.2 Latency Predictor Importances.

Figure 8 shows the importance scores from latency predictor for 1080Ti, K80 GPUs and Xeon CPUs respectively. Evidently, the importances are favored largely towards the parameters suggesting the insensitivity of device latencies to shape.

F Efficient Pytorch implementation

Pytorch code addition for slicing

```
class CustomLinear(nn.Linear):
    def __init__(
        self, super_in_dim,
            super_out_dim, bias=True,
             uniform_=None,
            non_linear="linear"
    ):
    self.samples = {}
    ...
    def set_sample_config(self,
        sample_in_dim, sample_out_dim
        ):
        sample_weight = weight[:, :
            sample_in_dim]
        sample_weight = sample_weight
            [:sample_out_dim, :]
        self.samples["weight"] =
            sample_weight
        self.samples["bias"] = self.
            bias[..., : self.
            sample_out_dim]

    def forward(self, x):
        #override the Forward pass to
            use the sampled weights
            and bias
        return F.linear(x, self.
            samples["weight"], self.
            samples["bias"])
```

The above code shows the additional lines added to PyTorch linear layer to support slicing for super pre-training. We add this to all the fundamental layers - *embedding layer*, *Linear layer* and *Layernorm* which adds up to 20 additional lines. This implementation is inspired from HAT[2]

G Latency Predictor Performance

Figure 9 illustrates the actual-vs-predicted latency for all network pairs in the test set for the 2 GPUs and 1 CPU devices (30% of the dataset). The points are closer to y=x line denoting high accuracy. Quantitatively, the R^2 values of these predictors are high proving the efficacy of these models to be reliable performance indicators.

[2] https://github.com/mit-han-lab/hardware-aware-transformers

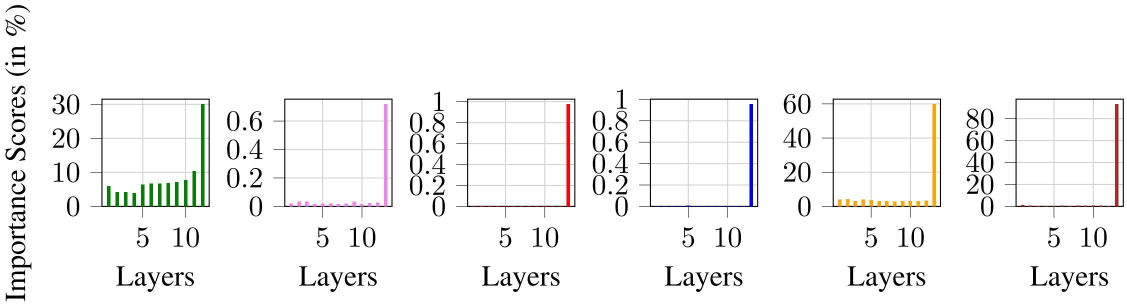

Figure 8: Importance scores for (a) Perplexity Predictor, and (b)-(f) Latency predictor for 1080Ti, K80 GPU, Xeon CPU, 1060Ti GPU, and AMD Ryzen CPU respectively. The features for (a) is the shape S, i.e., the dimensions across the 12 layers, while the latency predictor uses parameter count as a feature in addition to the shape S.

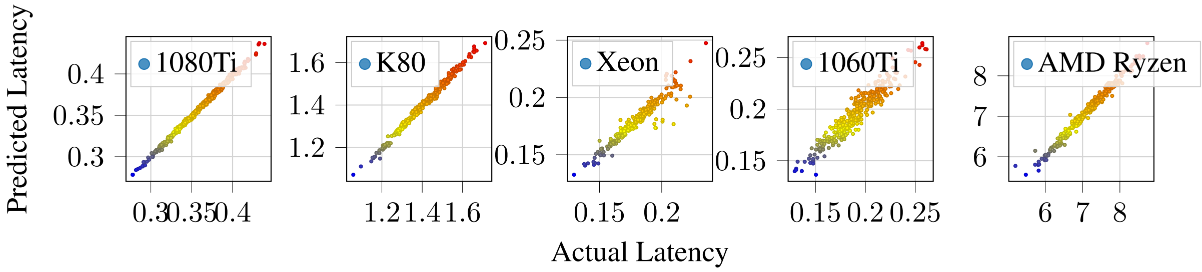

Figure 9: The latency predictors are very accurate with R^2 scores of 0.993, 0.988, 0.892, 0.87, and 0.97 respectively.

H Performance of SuperShaper

H.1 Fine-tuning T_S from SuperShaper

Table 6 compares the different methods of fine-tuning T_S, i.e. $\overline{G}_{\text{direct}}$, $\overline{G}_{\text{scratch}}$, and $\overline{G}_{\text{partial}}$ respectively for a 63M network configuration obtained through evolutionary search. From the table, it is clear that $\overline{G}_{\text{scratch}}$ and $\overline{G}_{\text{partial}}$ have better average GLUE performance. It is noteworthy, however, that SuperShaper is able to already provide good models that perform close to the best performance. When it comes to $\overline{G}_{\text{partial}}$ and $\overline{G}_{\text{scratch}}$, a more rigorous analysis has been done in the main paper across parameters and we refer the readers to that.

H.2 Comparing with BERT models

Table 4 shows the performance of a base model for SuperShaper, searched for 100M constraint compared against BERT-Base. As discussed in the main paper, SuperShaper provides models that match the performance of BERT-Base models for a significantly fewer parameters.

H.3 Shape difference vs performance.

To further study the effect of shape on performance, we test if the shape difference between random subnetworks and an optimal subnetwork (determined by evolutionary search) in the same parameter range, correlates with their differences in performance. The shape difference between two subnetworks with shapes S_1 and S_2 and their respective difference in performance ($\overline{G}_{\text{direct}}$) is characterised by : $Diff = \|S_1 - S_2\|$

We choose points across different parameter ranges (50-100M) from the 10,000 random sampled subnetworks from section Section 3 and compute their shape and performance differences with the optimal evolutionary-search model. The Spearman and Pearson correlation coefficient (Myers and Sirois, 2004; Benesty et al., 2009) across the shape and performance L2 norms are detailed in Table Table 7. Clearly, we see a positive correlation between shapes and performance further reinstating the sensitivity of shape in determining optimal performance of a model.

H.4 Average GLUE performance of best models from Evolutionary Search

Table 5 shows the average GLUE performance for all the best models found through evolutionary search for reference.

Model	Params	MNLI-m	QQP	QNLI	CoLA	SST-2	STS-B	RTE	MRPC	Avg. GLUE	Squad V1
BERT-Base (from NAS-BERT)	110M	85.2	91	91.3	61	92.9	90.3	76	87.7	84.4	81.8 / 88.9
BERT-Base (from DistilBERT)	110M	86.7	89.6	91.8	56.3	92.7	89	69.3	88.6	83	81.2 / 88.5
SuperShaper (ours)	96M	83.9	90.86	90.92	56.58	92.89	88.3	77.98	88.48	83.7	80.19 / 88.2

Table 4: Comparing SuperShaper with BERT-Base models.

Params (M)	$\overline{G}_{\text{partial}}$	$\overline{G}_{\text{scratch}}$	MNLI-m	QQP	QNLI	CoLA	SST-2	STS-B	RTE	MRPC	Average GLUE
33	10.82	12.44	73.45	84.71	80.52	10.27	85.32	82.65	65.70	82.11	70.59
53	8.59	6.02	79.40	89.51	86.38	33.85	89.11	86.66	68.23	84.56	77.21
63	7.09	4.55	82.23	90.18	88.05	53.00	91.86	87.63	79.06	89.46	82.68
69	6.62	4.28	82.74	90.45	89.54	54.98	91.28	88.42	77.98	87.75	82.89
80	6.17	4.02	83.05	90.56	89.22	54.87	93.10	88.46	80.14	87.75	83.39
90.5	5.83	3.79	83.06	90.51	88.72	58.87	91.51	88.47	77.26	88.97	83.42
96	5.65	3.73	83.90	90.86	90.92	56.58	92.89	88.30	77.98	88.48	83.74

Table 5: Performance of best models from parameter-constrained evolutionary search

Shapes	Params (M)	G_direct	G_scratch	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12
EvoSearch 1	65	6.86	4.45	480	360	360	240	240	360	480	480	360	480	540	540
Evo Search 2	63	7.09	4.55	480	240	360	240	540	480	360	360	360	360	540	480
Lower Triangle	64	7.31	4.67	120	120	240	240	360	360	360	480	540	540	600	768
Random	64	7.49	4.91	480	360	360	540	480	540	360	480	540	120	360	540
Rectangle	58	7.5	4.72	360	360	360	360	360	360	360	360	360	360	360	360
Inverted Diamond	65	8.12	4.93	768	600	360	240	240	120	120	240	240	360	600	768
Bottle	64	8.31	4.9	120	120	120	120	120	120	600	600	600	600	600	768
Diamond	64	8.36	5.13	120	240	360	480	480	540	768	540	480	360	240	120
Upper Triangle	64	8.43	5.16	768	600	540	540	480	360	360	360	240	240	120	120
Inverted Bottle	64	9.22	5.37	768	600	600	600	600	600	120	120	120	120	120	120

Table 6: Hidden dimensions of templatized shapes and their corresponding perplexities for $\overline{G}_{\text{scratch}}$ and $\overline{G}_{\text{direct}}$.

Evo-search parameters	$\overline{G}_{\text{direct}}$	Parameter range	Number of networks	Spearman Correlation	Pearson Correlation
53	8.59	52-54	54	71.03	67.95
63	7.09	62-64	704	80.34	82.52
65	6.86	63-65	862	69.47	71.34
69	6.62	68-70	1065	47.22	52.06
80	6.17	79-81	486	72.32	67.34
90.5	5.83	89-91	47	56.8	54.67
96	5.65	95-97	6	65.71	69.65

Table 7: Shape difference positively correlates with $\overline{G}_{\text{direct}}$difference across a wide parameter range

H.5 Comparison with HAT(Wang et al., 2020a) and OFA(Cai et al., 2019)

HAT compresses encoder-decoder models by elasticizing number of layers, hidden size and number of attention heads for machine translation task while OFA proposed a compression for CNN based models on image classification task. To compare our approach with these techniques, We use the results from (Zhang et al., 2021) who reimplement these approaches and report on three Glue tasks - MRPC, SST2 and RTE for 2 compression ratios - 0.75x and 0.5x. We compare these results against our pareto evolutionary search models that have compression ratios of 0.78x (90.5M) and 0.54x (63M). The re-

Model	MRPC		SST2		RTE		AVG
	Compression Ratio						
	0.75x	**0.5x**	**0.75x**	**0.5x**	**0.75x**	**0.5x**	
HAT-BERT	82.2	82.6	88.6	88.6	65.0	64.6	78.6
OFA-BERT	87.6	85.2	89.3	89.8	62.8	65.3	80.0
YOCO-BERT	90.4	87.6	92.9	91.9	75.1	69.3	84.5
SuperShaper(ours)$*$	88.97	89.46	91.51	91.86	77.26	79.06	**86.4**

Table 8: Comparison with HAT, OFA and YocoBert with SuperShaper. $*$ We use models with compression ratios of 0.78x (90.5M) and 0.54x (63M)

sults are reported in table table 8 and we see that SuperShaper outperforms both these approaches with a significant margin.