Structural Pruning of Large Language Models via Neural Architecture Search

Aaron Klein[1] **Jacek Golebiowski**[1] **Xingchen Ma**[1] **Valerio Perrone**[1] **Cedric Archambeau**[1]

[1]AWS AI Labs

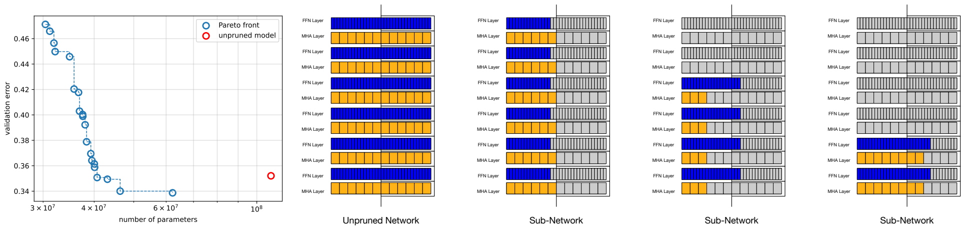
Abstract Large Language Models (LLM) achieved considerable results on natural language understanding tasks. However, their sheer size causes a large memory consumption or high latency at inference time, which renders deployment on hardware-constrained applications challenging. Neural architecture search (NAS) demonstrated to be a promising framework to automatically design efficient neural network architectures. In this work, we discuss the relationship between NAS and structural pruning and apply multi-objective NAS to compress LLMs while optimizing their performance when fine-tuned on a downstream task. We provide an in-depth analysis of existing NAS strategies and compare different search space definitions. On sentence classification tasks, we can prune the popular BERT model by up to 50% without deteriorating performance.

1 Introduction

Large language models (LLM) are now the de-facto standard for natural language understanding (NLU) tasks (Devlin et al., 2019). While LLM started to become ubiquitous, deploying them for inference can be challenging due to their considerable size. Contemporary LLMs require large GPU memory when deployed and tend to have large inference latency that is infeasible for real-world applications within a web service or an embedded system.

A recent body of work (Blalock et al., 2020; Kwon et al., 2022; Michel et al., 2019; Sajjad et al., 2022) showed that often only a subset of the pre-trained model actually contributes to the downstream task performance. Unstructured pruning (Blalock et al., 2020) computes a score for each weight in the network, for example the weight's magnitude, and prunes all weights with a score lower than a pre-defined threshold. It often achieves high pruning rates with small deterioration in performance, but also causes sparse weight matrices, which are hardly supported by commonly used machine learning frameworks. Structured pruning considers larger components of the networks, such as layers or heads. Even though it usually does not achieve the same pruning rates as unstructured pruning, it only prunes full columns/rows of the weight matrix, making it amenable for popular deep learning frameworks and hardware.

Neural Architecture Search (NAS) (see Elsken et al. (2018) for an overview) automates the design of neural network architectures to maximize generalization performance and efficiency, for example, in terms of latency, model size or memory consumption. The limiting factor of NAS is the computational burden of the search, which consists of multiple rounds of training and validating neural network architectures (Zoph and Le, 2017; Real et al., 2017). To reduce the overall cost, weight-sharing NAS (Pham et al., 2018; Liu et al., 2019) trains a single super-network consisting of all architectures in the search space. Initially, Liu et al. (2019) framed this as a bi-level optimization problem, where the inner objective represents the optimization of the network weights, and the outer objective the selection of the architecture. After training the super-network, the best architecture is selected based on the shared weights and then re-trained from scratch. However, several papers (Li and Talwalkar, 2020; Yang et al., 2020) reported that this formulation heavily depends on the search space and does not yield better results than just randomly sampling an architecture. Yu et al. (2020) proposed to update this strategy with a two-stage NAS process. First,



(a) Pareto Front (b) Sub-Networks

Figure 1: a) Pareto front of sub-networks on the RTE dataset. We can find sub-networks that are roughly 50% smaller without loss in performance compared to the un-pruned network. b) Illustration of our approach. We treat the entire pre-trained architecture as super-network with multi-head (yellow) and fully-connected layers (blue) and select sub-networks (coloured) by placing a mask over head and units (grayed out).

the super-network is trained by updating individual sub-networks in each iteration (as opposed to the whole super-network). After training, the final model is selected by running gradient-free optimization based on the shared-weights of the super-network without any further training. Cai et al. (2020) adapted this to the multi-objective setting to train a single super-network and then search for sub-networks to minimize latency on some target device.

Weight-sharing NAS can be seen as structural pruning, where we prune the super-network to select a sub-network. However, compared to other pruning strategies, NAS allows for a multi-objective approach to find the Pareto optimal set of sub-networks that balance between performance and model size instead of just returning a single solution based on a pre-defined threshold on the number of parameters. Especially in settings without a hard constraint on the model size, it might be difficult to define such fixed threshold a-priori. Furthermore, a multi-objective approach allows us to model the often non-linear relationship (see Figure 1) between model size and performance such that we can select post-hoc the best model that fits our requirements. In this paper we explore this property and provide a NAS approach to prune pre-trained LLMs when fine-tuned on a downstream task. Our contributions are:

- We propose to treat structural pruning of LLMs as a multi-objective NAS problem. Based on recent work on weight-sharing based NAS, we view the pre-trained network as a super-network and search for the set of sub-networks that optimally balance between downstream task performance and parameter count. As shown in Figure 1, our NAS strategy is able to prune a BERT model by up to 50% without loss in performance.
- We present a carefully designed ablation study of weight-sharing based NAS to analyse the effect of different search space definitions and NAS techniques, such as super-network training and sub-network search. We show that less expressive search spaces are superior for a fixed search budget. We also present a simple local search approach that outperforms more sophisticated state-of-the-art methods.

2 Structural Pruning via Neural Architecture Search

Given a pre-trained transformer model with parameters $\mathbf{w} \in \mathbb{R}^n$, where n is the total number of trainable parameters, denote its number of heads $h \in \mathbb{N}$ for each multi-head attention layer, the number of units $u \in \mathbb{N}$ in the intermediate fully-connected layer and the number of blocks $l \in \mathbb{N}$ consisting of one multi-head attention and one fully-connected layer. We first define a search space Θ that contains a large, but finite set of possible sub-networks with varying number of heads $\leq h$, units $\leq u$ and layers $\leq l$ (see Figure 1(b) for an example). To select a sub-network, we place a binary mask over all heads for each multi-head attention layer and every unit for each fully-connected layer (see Appendix B for more details). Based on Θ, we jointly optimize the validation error and parameter count of the sub-networks. In the multi-objective setting, there is no single sub-network $\boldsymbol{\theta} \in \Theta$ that jointly optimizes all objectives. Instead we search for the Pareto set of sub-networks that dominate all other sub-networks in at least one objective.

Next, we describe our super-network training strategy and how to tackle this multi-objective problem. Afterwards, we discuss several example to define a search space Θ.

2.1 Weight-sharing based NAS

Super-network training. We treat the pre-trained network as super-network with shared weights that contains all possible sub-networks $\forall \boldsymbol{\theta} \in \Theta$. To avoid that sub-networks co-adapt and still work outside the super-network, previous work (Yu et al., 2020; Wang et al., 2021) suggested to update only a subset of sub-networks in each update step, instead of the full super-network. We adapt this strategy and in each update step, sample sub-networks according to the sandwich rule (Yu et al., 2020), which always updates the smallest, the largest and k random sub-networks. The sandwich rule has shown promising results in NAS (Yu et al., 2020; Wang et al., 2021) to joinlty train independent sub-networks. The smallest and largest sub-network correspond to the lower and upper bound of Θ, respectively. Here the upper bound is equal to full network architecture, i.e, the super-network and the lower bound removes all layers except the embedding and classification layer. Additionally, we use in-place knowledge distillation (Yu et al., 2019) which has been shown to stabilize the training in NAS. The idea is to distill the knowledge of the super-network into the sub-networks by minimizing the Kullback-Leiber divergence between the logits of the super-network - which we obtain for free with the sandwich rule - and sub-networks.

Sub-networks selection. After training the super-network, we run multi-objective search to jointly optimize validation performance and parameter count of sub-networks. Each function evaluation only requires a full pass over the validation without any further training.

Previous work (White et al., 2021) showed that simple local search often performs competitively to more advanced NAS method. We propose a simple multi-objective local search approach. Given the current Pareto front P, we first sample a random element $\boldsymbol{\theta}_\star \sim P$ and then sample a random neighbour point by permuting a single random entry of $\boldsymbol{\theta}_\star$. We provide pseudo code for our local search in Appendix D.

2.2 Search Space

The search space Θ defines structural components of the pre-trained network architecture to be pruned. An expressive Θ allows for fine-grained pruning but might also become infeasible to explore. We identify the following search spaces that exhibit different levels of complexity:

- **Large**: For each head and neuron in the fully-connected layer we define a single binary $\Theta = \{0, 1\}^{l*(h+u)}$. This is the most expressive search space, but also grows exponentially with the model size. This search space is also commonly used by other structural pruning approaches (Kwon et al., 2022). It might not be very useful in practice, because we cannot easily remove single entries of weight matrices with most transformer implementations and hence it will not necessarily reduce the inference latency . However, it provides us a reference in terms of predictive performances that can be retained under a certain pruning ratios.
- **Layer**: Instead of single heads and neurons, we prune individual attention and fully-connected layers. The search space $\Theta = \{0, 1\}^{2*l}$ contains one binary for each multi-head attention layer and fully-connected layer, respectively.
- **Small**: We define the number of heads $\mathcal{H} = [0, H]$, the number of units $\mathcal{U} = [0, U]$ and the total number of layers $\mathcal{L} = [0, L]$, such that $\Theta = \mathcal{H} \times \mathcal{U} \times \mathcal{L}$. Compared to the other search spaces, the

dimensionality of this search space remains constant when we scale-up the number of layers, and only its upper bound increases. For each layer, we always keep the first $h \in \mathcal{H}$ heads and $u \in \mathcal{U}$ units, respectively.

- **Medium**: Based on the previous search space, we allow for a flexible number of heads / units per layer. For each layer $l \in [0, L]$, we define $\mathcal{H}_l = [0, H]$ and $\mathcal{U}_l = [0, U]$, such that the final search space is $\Theta = \mathcal{H}_0 \times \mathcal{U}_0 \dots \mathcal{H}_L \times \mathcal{U}_L$.

Each search space induces a different pattern for the head / unit masks that we place over the super-network to select sub-networks (see Appendix B for some examples). We also show in Appendix C the variations of the distribution of number of parameters for randomly sampled $\boldsymbol{\theta}$ from the different search spaces.

3 Experiments

To evaluate our approach, we use four datasets from the GLUE (Wang et al., 2019) benchmark suite: RTE, MRPC, COLA and STSB. GLUE datasets come with a predefined training and evaluation set with labels and a hold-out test set without labels. We split the training set into a training and validation set (70%/30% split). For all multi-objective search methods, we use Syne Tune (Salinas et al., 2022) on a single GPU instance. We use BERT-base (Devlin et al., 2019) (cased) as pre-trained network, which consists of $l = 12$ layers, $u = 3072$ units and $h = 12$ heads. Weights are quantized as float16 during the fine-tuning of the super-network.

To evaluate the performance of a Pareto set, we compute the Hypervolume (Zitzler et al., 2003) based on a fixed reference point. We first normalize each objective independently based on all observed values across all methods and repetitions via Quantile normalization. This results in a uniform distribution between $[0, 1]$, and we use $(2, 2)$ as reference point. We train each super-network five times with a different random seed. For each model checkpoint, i.e super-network, we run multi-objective NAS five times also with different random seeds. This leads to 25 different Pareto sets and we report mean and standard deviation of the corresponding hypervolume. We report here results averaged across all datasets in Figure 2 and show results per dataset in Appendix A.

3.1 Search Space

First, we compare the search spaces definitions from Section 2.2. We fine-tune the super-network as described in Section 2.1 and sample 100 sub-networks uniformly at random to compute the hypervolume. Within this budget (see Figure 2 left), the *Small* search space achieves the best performance. Interestingly, even though the *Medium* search space allows for a more fine-grained per layer pruning, it leads to worse results. We attribute this to the non-uniform distribution of parameter count (see Appendix C). The *Large* search space, which is a superset of the other search spaces, seems infeasible to explore with random sampling over so few observations.

3.2 Super-network Training

Next, we compare the following super-network training strategies:

- *standard*: Which trains all weights of super-network in the standard fine-tuning setting
- *random*: Samples a single random sub-network in each update steps
- *random-linear*: We either sample a random sub-network with probability p or the full-network with probability of $1 - p$ in each update step. Thereby, p is linearly increased from 0 to 1 after each step.
- *sandwich*: The super-network is updated according to the sandwich rule described in Section 2.1. We set the number of random sub-networks in each update step to $k = 2$.

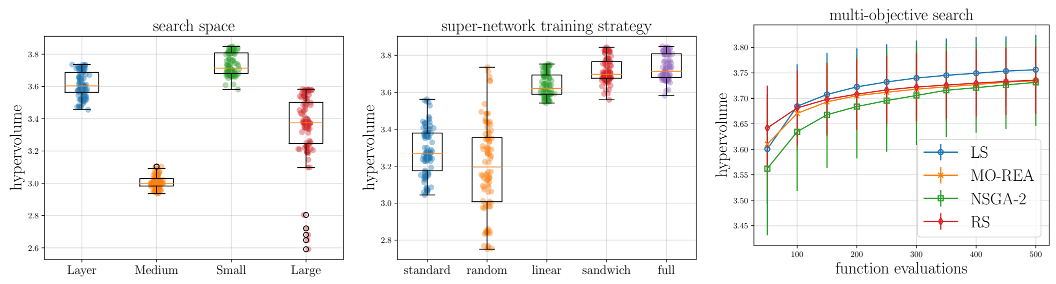


Figure 2: Results of the ablation study averaged across repetitions and datasets.

- *full*: Implements the training protocol described in Section 2.1, i.e it combines the sandwich rule with in-place knowledge distillation to update sub-networks.

Based on the previous results, we use the *Small* search space for each training strategy. Figure 2 middle shows the hypervolume based on the same set of 100 random configurations. Standard fine-tuning without accounting for sub-networks leads to significantly worse results. Linearly increasing the probability of sampling a random sub-networks leads to better results than always sampling a random sub-network. Better results are achieved by using the sandwich rule. Thereby, combining it with knowledge distillation slightly improves results further.

3.3 Multi-objective Search

Lastly, we compare (see Figure 2 right) the following multi-objective search methods against our local search (LS): Random search (RS) (Bergstra and Bengio, 2012) samples architectures uniformly at random from the search space. MO-REA is a multi-objective version of the popular NAS method regularized evolution (Real et al., 2019) where elements in the population are sorted via non-dominated sorting. NSGA-2 is a frequently used genetic algorithm from the multi-objective literature.

NSGA-2 performs slightly slower in the beginning than other methods, which we attribute to the initialization of the population. However, with larger budget it catches up and on RTE dataset, where it slightly improves upon RS and MO-REA. LS starts to outperform all other baseline after roughly 100 iterations.

4 Conclusions

We propose weight-sharing based multi-objective NAS for structural pruning of LLMs to approximate the Pareto set of sub-networks that optimize model size and performance. In the future, we will extend this study to larger auto-regressive models. We will also explore the relationship to quantization. Apart from this, we think there are several direction to improve the local search method, such as multi-fidelity optimization or transfer learning.

5 Limitation

The main limitation is the fine-tuning of the super-network on each downstream task independently, which becomes infeasible for very large models. Future work could explore the compression of large auto-regressive models that can directly be applied to task without any parameter updates.

6 Broader Impact Statement

While still in early stage, we hope that this line of work will eventually lead to more efficient LLMs, to reduce the demand for large compute resources which mitigates costs and carbon footprint. In the long run, we hope our work will help to make LLMs more accessible and democratizes research in this timely area.

References

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research.*

Blalock, D., Ortiz, J. J. G., Frankle, J., and Guttag, J. (2020). What is the state of neural network pruning? *arXiv:2003.03033 [cs.LG].*

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. (2020). Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR'20).*

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.*

Elsken, T., Metzen, J. H., and Hutter, F. (2018). Neural architecture search: A survey. *arXiv:1808.05377 [stat.ML].*

Kwon, W., Kim, S., Mahoney, M. W., Hassoun, J., Keutzer, K., and Gholami, A. (2022). A fast post-training pruning framework for transformers. *arXiv:2204.09656 [cs.CL].*

Li, L. and Talwalkar, A. (2020). Random search and reproducibility for neural architecture search. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference.*

Liu, H., Simonyan, K., and Yang, Y. (2019). DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR'19).*

Michel, P., Levy, O., and Neubig, G. (2019). Are sixteen heads really better than one? In *Proceedings of the 32th International Conference on Advances in Neural Information Processing Systems (NIPS'19).*

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18).*

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized Evolution for Image Classifier Architecture Search. In *Proceedings of the Conference on Artificial Intelligence (AAAI'19).*

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17).*

Sajjad, H., Dalvi, F., Durrani, N., and Nakov, P. (2022). On the effect of dropping layers of pre-trained transformer models. *arXiv:2004.03844 [cs.CL].*

Salinas, D., Seeger, M., Klein, A., Perrone, V., Wistuba, M., and Archambeau, C. (2022). Syne tune: A library for large scale hyperparameter tuning and reproducible research. In *First Conference on Automated Machine Learning (Main Track).*

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR'19).*

Wang, D., Li, M., Gong, C., and Chandra, V. (2021). AttentiveNAS: Improving Neural Architecture Search via Attentive Sampling. *arXiv:2011.09011 [cs.CV].*

White, C., Nolen, S., and Savani, Y. (2021). Exploring the loss landscape in neural architecture search. In *Proceedings of the 37th conference on Uncertainty in Artificial Intelligence (UAI'21)*.

Yang, A., Esperança, P. M., and Carlucci, F. M. (2020). NAS valuation is frustratingly hard. In *International Conference on Learning Representations (ICLR'20)*.

Yu, J., Jin, P., Liu, H., Bender, G., Kindermans, P. J., Tan, M., Huang, T., Song, X., Pang, R., and Le, Q. (2020). BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models. In *The European Conference on Computer Vision (ECCV'20)*.

Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. (2019). Slimmable neural networks. In *International Conference on Learning Representations (ICLR'19)*.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and Fonseca, V. G. D. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation*.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR'17)*.

A Detailed Results

Figure 3 shows the comparison of different search spaces across all datasets. Figure 5 and Figure 4 the comparison of super-network training strategies and multi-objective search methods, respectively. See main text for a more detailed analysis of the results.

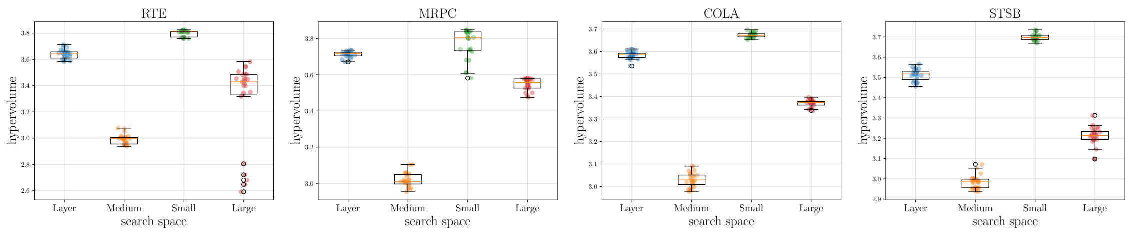


Figure 3: Comparison of different search spaces to define sub-networks. Even though larger search spaces are more expressive, they under-perform within the select budget.

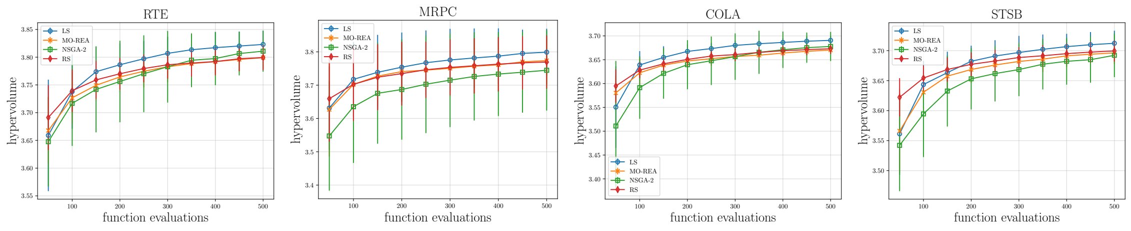


Figure 4: Comparison of different multi-objective search strategies. Simple local search improves upon more sophisticated genetic or evolutionary algorithms.

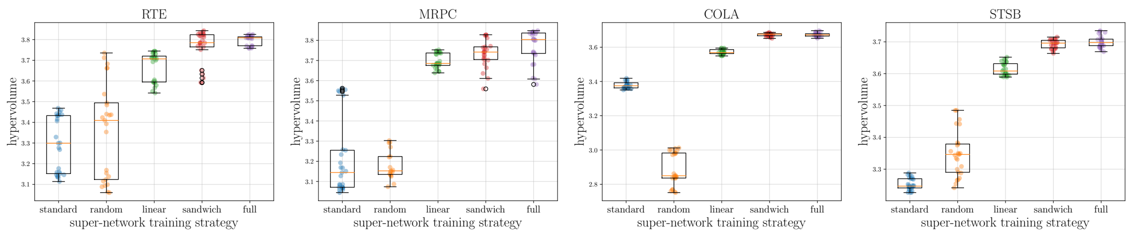


Figure 5: Comparison of super-network training strategies. More advances training strategies that sample a set of sub-network outperform standard fine-tuning as well as just sampling a single random sub-network. Full corresponds to our approach described in Section 2.1 which combines the sandwich rule with in-place knowledge distillation.

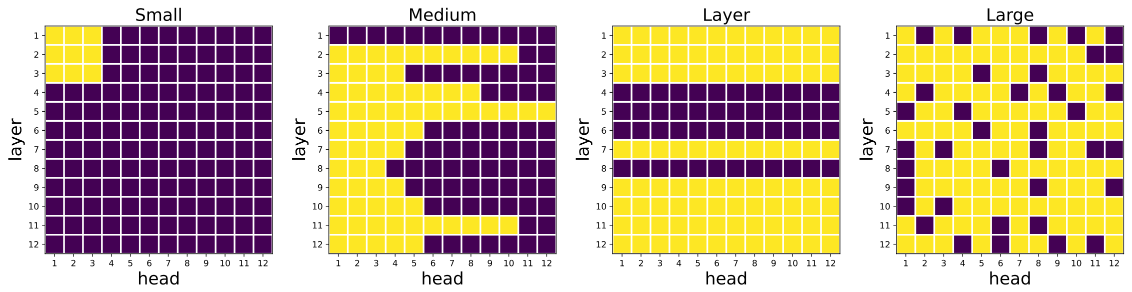


Figure 6: Example masks for heads by different search spaces.

B Masking

We define a binary mask $m_{head} \in \{0,1\}^{L \times H}$ for each head in the multi-head attention layer and $m_{neuron} \in \{0,1\}^{L \times U}$ for each neuron in the fully-connected layers. We query a sub-network $W_{subnetwork} \subset W$ by selecting only weights that correspond to the active heads / neurons in the corresponding mask. Given a search space Θ described in Section 2.2 that contains a finite set of possible sub-networks, we use an auxiliary function $m_{head}, m_{neuron} = CREATEMASK(\boldsymbol{\theta})$ that maps from configurations $\boldsymbol{\theta} \in \Theta$ in the search space Θ to a binary mask for head and neurons, respectively (see Figure 1 right for an illustration). Let's denote the function $f_0 : \Theta \longrightarrow \mathbb{R}$ as the validation error based on $W_{subnetwork}$ on some downstream task and $f_1 : \Theta \longrightarrow \{0, ..., N\}$ the total number of trainable parameters, i.e $|W_{subnetwork}|$, we minimize:

$$min(f_0(\boldsymbol{\theta}), f_1(\boldsymbol{\theta})). \tag{1}$$

Figure 6 shows the head mask based on random $\theta \in \Theta$ for each of the search spaces. The different search space induce different patterns in these head masks.

C Sampling Distributions of Search Spaces

For each Θ described in Section 2.2 we sample $N = 500$ random configurations $\{\theta_0, ..., \theta_N\}$ and compute the number of traininable parameters $\{f_1(\theta_0), ..., f_1(\theta_N\}$. Figure 7 shows the distribution over the number of parameters for each search space. Interestingly, the small search space is somewhat bias towards smaller networks. The medium search, even though it's more expressive, is focused on a medium sized networks. For the Layer and Large search space, we can achieve a uniform distribution over the number of parameters, by first sampling an integer $k \sim U(0, K)$, where $k = 2 * L$ for the Layer search space, and $k = L * (U + H)$ for the Large search space. Afterwards, we randomly select k entries of the binary vector $\boldsymbol{\theta}$ and set them to 1.

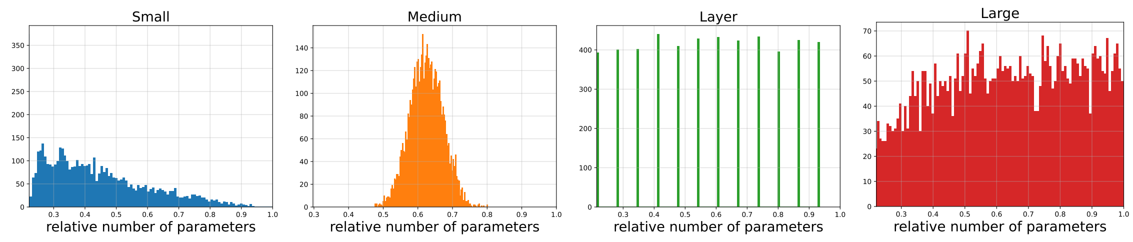


Figure 7: Sampling distribution of the parameter count for uniformly sampled $\boldsymbol{\theta}$ from the four search spaces described in Section 2.2.

D Multi-objective Local Search

We start with evaluating a starting point $\boldsymbol{\theta}_{start}$, which we set as the upper bound of our search space. The initial Pareto front P_0 is initialized with the start point $P_0 \leftarrow \{\boldsymbol{\theta}_{start}\}$. Afterwards, in each step our local search samples a random neighbour of a randomly selected points of the current Pareto front until we reach a fix number of iteration. We consider 1-step neighbourhood, which randomly permutes the given point only in a single dimension.

```
input  : Search space $\Theta$, number of iteration $T$, starting point $\boldsymbol{\theta}_{start}$
output : Pareto front $P$
/* evaluate starting point */
$P_0 \leftarrow \{\boldsymbol{\theta}_{start}\}$;
$y_{start} = [f_0(\boldsymbol{\theta}_{start}), f_1(\boldsymbol{\theta}_{start}]$;
$Y \leftarrow \{y_{start}\}$;
/* main loop */
for $t = 1, \dots, T$ do
    /* sample random element from the population */
    $\boldsymbol{\theta}_t \sim \mathcal{U}(P_{t-1})$;
    /* mutate */
    $d \sim \mathcal{U}(0, |\boldsymbol{\theta}_t|)$;   // sample random dimension
    $\hat{\boldsymbol{\theta}} \leftarrow copy(\boldsymbol{\theta}_t)$;
    $\hat{\boldsymbol{\theta}}[d] \leftarrow \mathcal{U}(\Theta_d)$;   // sample a new value from the search space
    /* evaluate */
    $y_t = [f_0(\hat{\boldsymbol{\theta}}), f_1(\hat{\boldsymbol{\theta}})]$;
    $Y \leftarrow Y \cup y_t$
    /* update population */
    $S(Y) = \{y' \in Y : \{y'' \in Y : y'' \succ y', y' \neq y''\} = \emptyset\}$;   // Pareto front
    $P_t \leftarrow \{\boldsymbol{\theta} : y(\boldsymbol{\theta}) \in S(Y)\}$;
end
```

Algorithm 1: Local Search

E Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Experiment Section

   (b) Did you describe the limitations of your work? [Yes] See Limitations

   (c) Did you discuss any potential negative societal impacts of your work? [No] We do not see any negative societal impacts of our work

   (d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [No] Code will be shared with the full paper submission

   (b) Did you include the raw results of running the given instructions on the given code and data? [N/A]

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [N/A]

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [N/A]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [N/A]

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] More details can be found in the Experiment section of our paper

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] See Experiment Section

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes]

   (i) Did you compare performance over time? [No] All search strategies have a negligible overhead and function evaluations consume the same time, hence the analysis would not change

   (j) Did you perform multiple runs of your experiments and report random seeds? [Yes]

   (k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

   (l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [N/A]

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] One contribution of the paper is to provide a detailed ablation study.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [No] Licenses can be found in the original publication which is referenced in the paper.

   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]