# A. Explanation of metrics / Hyperparameters

In this section, we explain the computation of our metrics used to characterize computation and parallelism (max speedup and new ops per task (%) and we detail the hyperparameters used for searching/final fine-tuning.

## A.1. Metrics explanations

In Table 1 and Table 3, we report the *new operations per task*, *total operations*, and *max speedup*. In this subsection, we detail the computation equations of these two metrics.

**New operations per task** We first compute the new operations (FLOPs or Floating-point operations) introduced by the sub-task based on the searched result and weight mask. We normalize the new operations of the task $s_i$ (where $i \in \{1, ..., N\}$) using the total number of operations required for a single $\text{BERT}_{\text{LARGE}}/_{\text{BASE}}$ as Eq. (10):

$$ops_{\%}(s_i) = \frac{ops(s_i)}{ops(\text{BERT}_{\text{LARGE}})} \times 100\% \qquad (10)$$

This percentage $ops_{\%}$ indicates that you only need extra $ops_{\%}$ new operations compared to the operations of an entire transformer (100%) when adding the sub-task to your multi-task system.

New operations per task (%) reported in Table 1 and Table 3 is the average of $ops_{\%}$ for each GLUE sub-task:

$$new\_ops\_per\_task(\%) = \frac{\sum_i^N ops_{\%}(s_i)}{N} \qquad (11)$$

**Total operations (%)** For LeTS, total operations include the extra operations from the nine tasks and the overhead operations from computing pretrained weight and input:

$$Total\_ops(\%) = \frac{\sum_i^N ops(s_i) + overhead}{ops(\text{BERT})} \times 100\% \qquad (12)$$

For example, the total operations of the traditional fine-tuning method will be 9× (nine GLUE tasks) of the operation of $\text{BERT}_{\text{LARGE}}$ (100%×9=900%) and the overhead/ops($\text{BERT}_{\text{LARGE}}$) is 0%. For freezing bottom-12 layers (Sec.4.1), the new operations per task are 50% and the overhead/ops($\text{BERT}_{\text{LARGE}}$) is 50% [1], thus the total normalized operations would be 40%×9 + 50% = 500%.

**Max speedup** When the sub-tasks are independent of each other, the user can leverage the computation reduction to achieve speedup. Yet, in many cases, the sub-tasks are dependent on each other and must be executed in order. In this scenario, LeTS's design space can yield fruitful speedup as we decouple the computation of different attention layers inside each transformer. We first identify the critical

[1]This overhead is to compute the top 12 layers between the pre-trained weights and input.

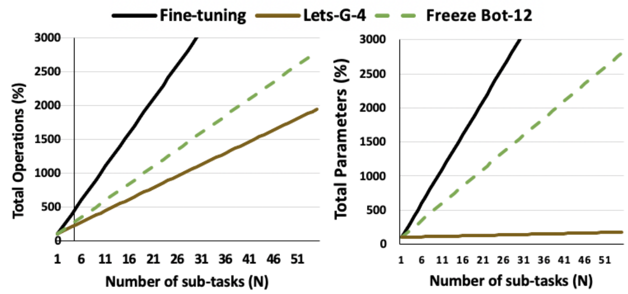


Figure 6: Anticipated *total operation* (%) and *total parameters* (%) v.s. the number of sub-tasks for $\text{BERT}_{\text{LARGE}}$.

path (Hennessy & Patterson, 2011) (example computing of max speedup for LeTS is showed in Figure 2(b) and Sec. 2 ) of evaluating the 9 GLUE tasks. The max speedup in this case would be computed as:

$$max\_speedup = \frac{ops(BERT) \times N}{ops(critical\_path)} \qquad (13)$$

Taking freezing Bot-12 as an example, ops($s_i$)/ops($\text{BERT}_{\text{LARGE}}$) = 50%. Thus, the critical path would be the sum of overhead (50%) and the computation time (50%) for each sub-task (max_speedup = 900%/500% = 1.8×).

Note that both the freezing Bot-12 and original fine-tuning architecture are included in our search space. Yet, the fine-tuning approach is computationally inefficient, and freezing Bot-12 sacrifices the task performance a lot. LeTS pushes the Pareto frontier between task performance and computation reduction when multiple tasks co-exist.

## A.2. Hyperparameters and Training Overheads

**Final Fine-tuning Hyperparameters.** Table 4 shows the hyperparameters for training our final searched model on GLUE tasks. For final testing, we select the model that achieves the best validation (dev set) result. We use two learning rate schedulers for the bias term in the transformer and all other parameters (including the aggregation linear and Bi-LSTM layers).

Another thing worth mention is that the *max input length* ($l_m$) in our evaluation is set to 128 to match previous baselines. With larger $l_m$ (e.g., 512), the computation overhead of the aggregation layers / pretrained layers and input computation would take less portion to the overall computation cost. The computation reduction of LeTS will also be more explicit. That is because the computation complexity of a transformer is proportional to the input length $l$ ($\mathcal{O}(l^2)$).

**Temperature scheduling and searching setup.** During the search, the initial temperature $T$ in Eq. 14 is set to 4.0 and exponentially annealed by $exp(-0.065) \approx 0.936$ for every $\frac{1}{10}$ epoch. We use an early stop mechanism that terminates the searching phase when the selected model does not change for $\frac{3}{10}$ epochs. Because the model parameters start from pretrained BERT, the searching phase converges much faster than traditional DNAS. For the loss function in Sec.3, $E_{ops}$ is represented in billion operations. We set $a$ to 0.5 and $b$ to 0.5. The learning rate of $a$ is initialized to $5e-4$ which is updated using an Adam optimizer (default optimizer settings in huggingface (Wolf et al., 2020)). The learning rate for $W^t$ is $2e-5$ for all tasks. We do not use another optimizer for the bias parameters in the search phase.

**Training Time.** For all tasks, we set the final fine-tuning to 7-10 epochs, which is larger than the original fine-tuning ($\sim$ 1.5-2.5$\times$ longer fine-tuning time). This is because the additional Bi-LSTM takes more time to converge.

**GPU memory overhead.** LeTS does not explicitly require more memory during final fine-tuning (1.2$\times$ than the traditional fine-tuning) as the pre-trained parameters are frozen (no gradient consumption) and the pruning mask is generated ahead of fine-tuning. For DiffPruning, the pruning mask is searched during fine-tuning. As such, it takes more GPU memory consumption ($\sim 2\times$) than traditional fine-tuning approach.

## B. Gumbel Softmax and Second-order approximation

**Gumbel Softmax.** The architecture parameters discussed in Sec.3.2 will be converted to a probability vector using Gumbel Softmax equations which controlled by a temperature parameter $T$. Specifically, the architecture parameters $\mathbf{a}_{ij}$ associate with the $i$th selector in $j$th layer are computed as Eq. (14).

$$P_{a_{ij}} = Gumbel(a_{ijt}|\mathbf{a}_{ij}) = \frac{exp((a_{ijt}+g_{ijt})/T))}{\sum_t exp((a_{ijt}+g_{ijt})/T))} \quad (14)$$

Here, $g_{ijt} \sim \text{Gumbel}(0,1)$ is a random noise following the Gumbel distribution. The output is a probability vector $P_{a_{ij}}$ $(2\times 1)$.

**Second-order approximation equations.** As discussed in Sec.3.2, we iteratively update weight and architecture parameters of the super network ($a$ and $W^t$). The gradient of weight parameters (denote $W^t$ as $W$ in appendix for simplicity) are updated using traditional gradient descent. And the gradient of architecture parameters ($a$) is computed through a second-order approximation. Specifically, we split the training dataset into two parts (D1 takes 80%, D2 takes 20%). The gradient of the architecture parameters can be approximated as Eq. (15):

$$\nabla_a \mathcal{L}(W_a, a) \approx \nabla_a \mathcal{L}_{D2}(W - x\nabla_W \mathcal{L}_{D1}(W,a), a) \quad (15)$$

Here, $W_a$ is the final pre-trained model given architecture parameter $a$. The l.h.s of Eq. (15) means we need to fine-tuning the entire model before training $a$ for only one step. To reduce the search cost, the idea of differentiable NAS (DNAS) is to approximate the final $W_a$ by adapting $W$ using only a single training step (Liu et al., 2019). To prevent the searched model from over-fitting to the training dataset, we split the training datasets (D1, D2) to update weight and architecture parameters, respectively. The r.h.s of Eq. (15) can be expanded into Eq. (16) as:

$$\nabla_a \mathcal{L}_{D2}(W', a) - x\nabla^2_{a,W}\mathcal{L}_{D1}(W,a)\nabla_{W'}\mathcal{L}_{D2}(W',a) \quad (16)$$

where $W' = W - x\nabla_W \mathcal{L}_{D1}(W,a)$. The second term can be computed through a finite difference approximation. Assume $e$ is a small scalar and $W^{\pm} = W \pm e\nabla_{W'}\mathcal{L}_{D2}(W',a)$. Then:

$$\nabla^2_{a,W}\mathcal{L}_{D1}(W,a)\nabla_{W'}\mathcal{L}_{D2}(W',a) \approx \frac{\nabla_a \mathcal{L}_{D1}(W^+,a) - \nabla_a \mathcal{L}_{D1}(W^-,a)}{2e} \quad (17)$$

In summary, during the architecture parameter update, we first compute the $\nabla_a \mathcal{L}_{D1}(w^+, a)/\nabla_a \mathcal{L}_{D1}(w^-, a)$ through two forward and backward passes and approximate the second term in Eq. (16) using Eq. (17). Due to the small size of $\mathbf{a}$ in the super network, the second-order approximation is feasible and is more accurate than gradient descent (i.e., when $x = 0$).

## C. Ablation study of gradient-accumulation initialization

**Task-specific initialization and sparsity.** For the initialization of $W^d$ in Delta-Pruning (Sec.3.1), we show that using an accumulate gradients method to initialize $\hat{W}^d$ can represent the final fine-tuned $W^d$. We visualize the weight mask $c$ generated from final $W^d$ (computed from fully fine-tuned $W^f$ and $W^p$ by $W^d = W^f - W^p$) and gradient accumulation $\hat{W}^d$ ($N_{steps}$=100) for selected tasks with the sparsity ratio $k$=0.5% as shown in Figure 8.

We also conduct an ablation study by replacing the gradient accumulation initialization with random initialization to generated mask $c$ (visualized in Figure 8). The mask $c$ generated from randomly initialized $\hat{W}^d$ is very distinct from the mask generated from final $W^d$.

## D. Detailed Delta-pruning algorithm.

Due to the space limitation of the paper, we put the summary of Delta-pruning algorithm (Sec. 3.1) here as shown in Algorithm 2.

## References

Hennessy, J. L. and Patterson, D. A. *Computer architecture: a quantitative approach*. Elsevier, 2011.

Table 4: Hyperparameters for final fine-tuning. We use the default learning rate scheduler in transformer ((Vaswani et al., 2017)) on the two learning rate.

| | QNLI | SST-2 | MNLI | CoLA | MRPC | STS-B | RTE | QQP |
|---|---|---|---|---|---|---|---|---|
| Epochs | 7 | 7 | 7 | 10 | 10 | 10 | 10 | 7 |
| Batch size | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| Learning Rate (bias terms) | 5e-4 | 5e-4 | 5e-4 | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 5e-4 |
| Learning Rate (other parameters) | 2e-5 | 2e-5 | 2e-5 | 2e-5 | 2e-5 | 2e-5 | 2e-5 | 2e-5 |
| Warm-up steps | 1986 | 1256 | 7432 | 420 | 350 | 720 | 350 | 28318 |

**Algorithm 2** Delta-pruning

**input** Pre-trained parameter $W^p$, offset parameter $W^d$, the desired $W^d$ sparsity constraint $k$, training dataset $D$
**output** Mask **c**

Warn up fine-tuning $W^p$ for 100 epochs and get $\mathbb{W}^f$
$\mathbb{W}^d \leftarrow \mathbb{W}^f - W^p$, $c \leftarrow 1^d$
Set trainable mask and perform one mini-batch training to get $\Delta L(\mathbb{W}^f; \mathcal{D})$ (Eq. (3) in the paper).
**for** i in $\{1...d\}$ **do**
  score $s_e = \frac{|g_e(\mathbb{W}^f;\mathcal{D})|}{\sum_{k=1}^{d} |g_k(\mathbb{W}^f;\mathcal{D})|}$
**end for**
Descending sort all the score $s$.
**for** i in $\{1...d\}$ **do**
  $c_i \leftarrow \mathbb{1}[s_i - \hat{s}_k \leq 0]$
**end for**


Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Huggingface's transformers: State-of-the-art natural language processing, 2020.

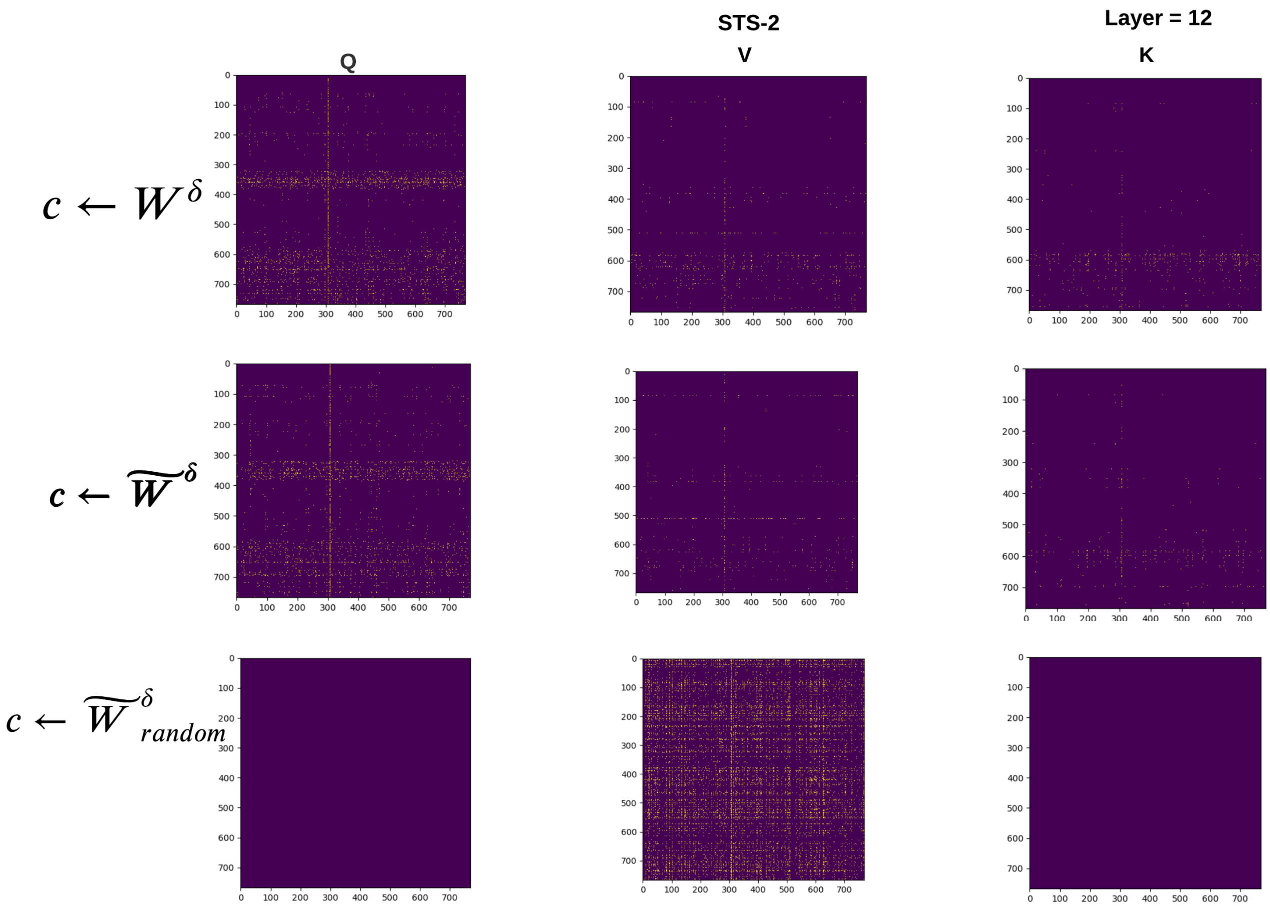


Figure 7: Visualization of weight mask generated using (1) final $W^d$ computed from $W^f - W^p$ and (2) $\widetilde{W}^d$ initialized using the gradient accumulation for 100 steps (the method used in Delta-Pruning) (3) $\widetilde{W}^d$ initialized randomly. We show the weight mask $c$ of $W^q$/$W^v$/$W^k$ in the middle ($12th$) layer of $\text{BERT}_{\text{LARGE}}$ on SST-2. Yellow pixels indicate the unmasked parameters.