Operator Selection and Ordering in a Pipeline Approach to Efficiency Optimizations for Transformers

Ji Xin, Raphael Tang, Zhiying Jiang, Yaoliang Yu, and **Jimmy Lin**

David R. Cheriton School of Computer Science
University of Waterloo

{ji.xin,r33tang,zhiying.jiang,yaoliang.yu,jimmylin}@uwaterloo.ca

Abstract

There exists a wide variety of efficiency methods for natural language processing (NLP) tasks, such as pruning, distillation, dynamic inference, quantization, etc. From a different perspective, we can consider an efficiency method as an *operator* applied on a model. Naturally, we may construct a pipeline of operators, i.e., to apply multiple efficiency methods on the model sequentially. In this paper, we study the plausibility of this idea, and more importantly, the *commutativity* and *cumulativeness* of efficiency operators. We make two interesting observations from our experiments: (1) The operators are commutative—the order of efficiency methods within the pipeline has little impact on the final results; (2) The operators are also cumulative—the final results of combining several efficiency methods can be estimated by combining the results of individual methods. These observations deepen our understanding of efficiency operators and provide useful guidelines for building them in real-world applications.

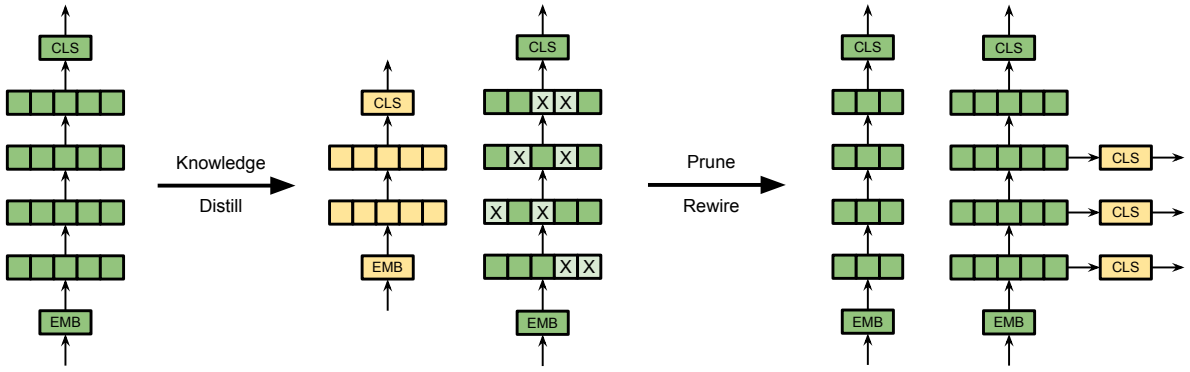
1 Introduction

Natural language processing (NLP) tasks nowadays heavily rely on complex neural models, especially large-scale pre-trained language models based on the transformer architecture (Vaswani et al., 2017), such as BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and GPT (Radford et al., 2019). Despite being more accurate than previous models, transformer-based models are typically slow to execute, making it a non-trivial challenge to apply them in real-world applications. For example, it takes a BERT-base model about 200 ms per query to perform a simple sequence classification task on a commercial CPU, which can be too slow in many scenarios. Therefore, model efficiency has become an increasingly important research direction in the transformer era.

A wide variety of efficiency methods have been individually studied for transformers, like pruning (McCarley et al., 2019), distillation (Sanh et al., 2019), dynamic inference (Xin et al., 2020; Kim and Cho, 2021), and quantization (Shen et al., 2020), just to name a few. There has also been work on applying multiple efficiency methods together as a pipeline (Kim and Hassan, 2020; Lin et al., 2021; Cui et al., 2021), but the construction of such pipelines has not been methodically studied. For a desired accuracy–efficiency tradeoff, it remains unclear how to choose components for the pipeline among numerous possibilities. Furthermore, even with a chosen set of efficiency methods, it is unclear whether we need to exhaustively examine all possible orders to find the best one.

In this paper, we study how to effectively construct a pipeline of efficiency methods, and we do this by exploring their properties. Conceptually, we consider each efficiency method as an *operator* applied on a model. We conduct experiments with the RoBERTa model (Liu et al., 2019) and the following components in our efficiency pipelines: distillation, structured pruning, quantization, early exiting, and dynamic length inference. We empirically study two important properties of efficiency operators: (1) Commutativity: does swapping the order of operators affect the final accuracy–efficiency tradeoff of the model? (2) Cumulativeness: how do the two core metrics of efficiency methods, time savings and accuracy drops, compound across multiple operators? Under the condition of our experiments, we show that, for commutativity, the difference between various orderings of the same set of components is usually small and negligible in practice. For cumulativeness, we show that time

Findings of the Association for Computational Linguistics: ACL 2023, pages 2870–2882



(a) Distillation (b) Pruning (c) Early exiting

Figure 1: Diagrams for efficiency methods. The model consists of an *embedding layer* (EMB) at the bottom, several *transformer layers* in the middle, and a *classifier* (CLS) at the top. Green blocks represent parameters available from fine-tuning and yellow blocks represent parameters that are initialized and optimized after fine-tuning. (a) Distillation initialized a new student model and distill knowledge from the original teacher model; (b) Pruning removes unimportant parts of the original model and rewires the connection; (c) Early exiting adds extra classifiers for intermediate transformer layers. Dynamic sequence length and quantization are not shown because they do not change the model architecture.

saving and accuracy drop are both cumulative to the extent that we can estimate the performance of a new pipeline by combining the results of individual components. The observation of these properties provides the foundation for us to build new pipelines and estimate their performance without having to carry out time-consuming experiments.

Our main contributions in this paper include: (1) In Section 3, we propose a conceptual framework to treat efficiency methods as operators and efficiency optimization processes as pipelines; (2) In Sections 4 to 5, we demonstrate the properties of operators by experiments. Finally we will conclude the paper and discuss its limitations.

2 Related Work and Background

In this section, we first introduce related work, background, and modeling choices for individual efficiency methods chosen for our experiments. We then discuss related work for applying multiple efficiency methods.

Applying transformers for NLP tasks typically involves three stages: pre-training, fine-tuning, and inference (Radford et al., 2019; Devlin et al., 2019). In this paper, we assume the availability of a pre-trained RoBERTa model and study different ways of fine-tuning it to achieve better tradeoffs between inference accuracy and efficiency. *Training* henceforth refers to fine-tuning in this paper.

2.1 Knowledge Distillation

Knowledge distillation (Hinton et al., 2015) improves efficiency by distilling knowledge from a large and costly *teacher* model to a small and efficient *student* model. The teacher model's output is used as the supervision signal for the student model's training. In the case of transformers, there are two types of distillation, namely task-agnostic and task-specific, depending on whether the student model is trained for a specific task. These two types correspond to the pre-training stage and the fine-tuning stage.

Previously, Tang et al. (2019) perform task-specific distillation from a fine-tuned BERT model into non-transformer architectures such as LSTMs, aligning predicted logits of the teacher and the student. Patient knowledge distillation (Sun et al., 2019) performs task-specific distillation, where the students are transformer models with smaller depth and width; furthermore, they align not only predicted logits but also intermediate states of both models. DistilBERT (Sanh et al., 2019) and TinyBERT (Jiao et al., 2020) perform both task-agnostic and task-specific distillation: first the student model learns from a pre-trained teacher; then it can either be directly fine-tuned like a pre-trained model or learn from another fine-tuned teacher as a student.

In this paper, we focus on *task-specific distillation*, which corresponds to fine-tuning (Figure 1a). We initialize the student model with a DistilRoBERTa$_{\text{BASE}}$ (Sanh et al., 2019) backbone

that comes from task-agnostic distillation. The student model has the same width as the teacher RoBERTa but only half the number of layers. In addition to the most common loss function (teacher supervising student), which is a soft cross-entropy between output logits of the teacher and the student, we introduce two other parts for the loss function: (1) mean squared error (MSE) between the teacher's and the student's embedding layers' outputs; (2) MSE between the teacher's and the student's final transformer layers' outputs. It has been shown in related work that adding objectives to align intermediate states of the teacher and the student helps with distillation (Sun et al., 2019; Sanh et al., 2019). We simply use a ratio of $1:1:1$ for these three parts of the loss function.

2.2 Structured Pruning

Pruning removes unimportant parts of the model and increases the sparsity level of the model. A specific category of pruning, *structured pruning* (Han et al., 2015; Anwar et al., 2017; Gordon et al., 2020), removes high-level units of the model, such as a layer, an attention head, or an entire row/column in the weight matrix of a feed-forward network (FFN). Model sparsity induced by structured pruning can directly translate to faster execution, and therefore we focus on structured pruning in the paper.

Previously, Michel et al. (2019) show that reducing attention heads *after* training/fine-tuning does not significantly degrade the model's effectiveness and argue that in a lot of cases, the number of attention heads can be reduced. MobileBERT (Sun et al., 2020) reduces the intermediate dimension of a transformer layer's FFN by using a funnel-like structure to first shrink the intermediate layer size and then recover it at the end of the layer. McCarley et al. (2019) improves BERT efficiency for question answering by reducing both attention heads and intermediate dimensions.

In this paper, we follow the work by McCarley et al. (2019); Kim and Hassan (2020) and choose two aspects of the model and prune them separately: the number of attention heads and the intermediate dimension of the fully connected layer within a transformer layer (Figure 1b). We calculate the *importance* of attention heads and intermediate dimensions with a first-order method: run inference for the entire dev set and accumulate the first-order gradients for each attention head and intermediate dimension. We then remove the least important attention heads and intermediate dimensions, according to the desired sparsity level, and then rewire the model connections so it becomes a smaller but complete model. After pruning, we perform another round of knowledge distillation from the original model to the pruned model as described in the previous subsection, which further improves the pruned model's accuracy without sacrificing efficiency.

2.3 Dynamic Inference

Dynamic inference (Teerapittayanon et al., 2016; Graves, 2016; Dehghani et al., 2019) accelerates inference by reducing the amount of computation adaptively, depending on the nature of the input example. We discuss two types of dynamic inference in this section.

2.3.1 Dynamic Depth: Early Exiting

For dynamic depth, early exiting (Xin et al., 2020; Liu et al., 2020) converts the original fine-tuned model into a multi-output one, and dynamically chooses the number of layers used for the inference of each example, based on model confidence (Schwartz et al., 2020), model patience (Zhou et al., 2020), or the prediction of an external controlling module (Xin et al., 2021).

Early exiting training We first modify a fine-tuned model by adding extra classifiers to intermediate transformer layers (Figure 1c). In order to use these extra classifiers, we further train the model before inference. The additional training is done by minimizing the sum of loss functions of all classifiers, and the loss function has the same form for each classifier: the cross entropy between ground truth labels and the classifier's prediction logits. A special case to notice here is that the training of distillation and pruning needs to be adjusted *after* adding early exiting.

- Distillation after early exiting. When we initialize the student model (e.g., from TinyBERT), we also add early exiting classifiers to it. For training, the i^{th} layer of the student model uses the prediction from the $2i^{\text{th}}$ layer of the teacher model as supervision.
- Pruning after early exiting. When we prune the transformer layers, we do not change the classifiers. For the additional round of distillation, each layer of the student model uses the prediction from its corresponding layer of the teacher model as supervision.

Early exiting inference The early exiting model produces an output probability distribution at each layer's classifier. If the confidence of a certain layer's output exceeds a preset *early exiting threshold*, the model immediately returns the current output; otherwise, inference continues at the next layer, and so forth until the final layer. In this way, when the model is confident enough at an early layer, we no longer need to execute the remaining layers, thereby saving inference computation.

2.3.2 Dynamic Sequence Length

Pre-trained language models come with a fixed input sequence length (e.g., 512 for RoBERTa) that aligns with the design of positional embeddings (Devlin et al., 2019). Inputs longer than the fixed length are truncated and shorter inputs are padded with zero vectors. This fixed length, while being useful for tasks with long inputs, is often unnecessarily large for most downstream applications and leads to a waste of computation.

Previously, PoWER-BERT (Goyal et al., 2020) shrinks the sequence length gradually as inference progresses into deep layers, eventually reducing the sequence length to 1 at the final layer for sequence-level prediction. Length-Adaptive Transformer (Kim and Cho, 2021) extends the idea to token-level prediction by first reducing the sequence length and then recovering missing tokens' outputs.

In this paper, we use a simple method for length reduction: for each batch, we dynamically set the input sequence length to the maximum length of inputs within the batch. This reduces the number of zero paddings in input sequences and reduces unnecessary computation. Different from previous methods, dynamic sequence length does not affect the model's accuracy.

2.4 Quantization

Quantization (Lin et al., 2016; Shen et al., 2020) improves model efficiency by using fewer bits to store and process data. The idea itself is straightforward, but implementation can be highly hardware dependent. Since we run inference on CPUs, we first export the trained model to ONNX[1] and then run it with 8-bit quantization, following Fastformers (Kim and Hassan, 2020).

[1] https://onnx.ai/.

2.5 Applying Multiple Efficiency Methods

With all the individual efficiency methods available, there has been work on applying multiple ones together. For example, Cui et al. (2021); Aghli and Ribeiro (2021); Park and No (2022) combine pruning and distillation for model compression and acceleration. Phuong and Lampert (2019) explore using distillation to improve the training of early exiting models. Lin et al. (2021) propose a bag of tricks to accelerate the inference stage of neural machine translation models. Fastformers (Kim and Hassan, 2020) propose a pipeline consisting of several components which together provide more than $100\times$ acceleration. Despite the success of combining efficiency methods, it remains underexplored how to build an efficiency pipeline in order to achieve the best accuracy–efficiency tradeoffs. We aim to tackle this problem in our paper.

3 Experimental Design

In this section, we introduce the detailed design and setups for our experiments. Since the experiments are exploratory rather than SOTA-chasing, we focus on providing a fair comparison.

3.1 Conceptual Framework

In our experiments, we work with *pipelines* consisting of multiple efficiency *operators* that are applied to the model sequentially. We represent a pipeline with a string of bold capital letters, where each letter represents an efficiency operator and the order of these letters represents their order.

The operators include: **D**istillation, Structured **P**runing, **E**arly Exiting, Dynamic **L**ength, and **Q**uantization. For example, the string "**DEPLQ**" represents a pipeline of sequentially applying the following operators to a fine-tuned model: (1) distill it into a student model; (2) add early exiting classifiers to it and train; (3) apply structured pruning to make each layer "thinner" and distill from the unpruned model; and (4) use dynamic length and quantization for the final inference. Additionally, we use **O** to represent an "empty" pipeline, i.e., directly applying the **O**riginal fine-tuned model.

Not all combinations of operators constitute a meaningful pipeline. Among the operators discussed in this paper, **D**, **P**, and **E** require additional training steps, while **Q** and **L** are directly applicable right before inference. Therefore, **D**, **P**, and **E** (Group I) should always appear before **Q** and **L** (Group II) in the pipeline. Moreover, applying

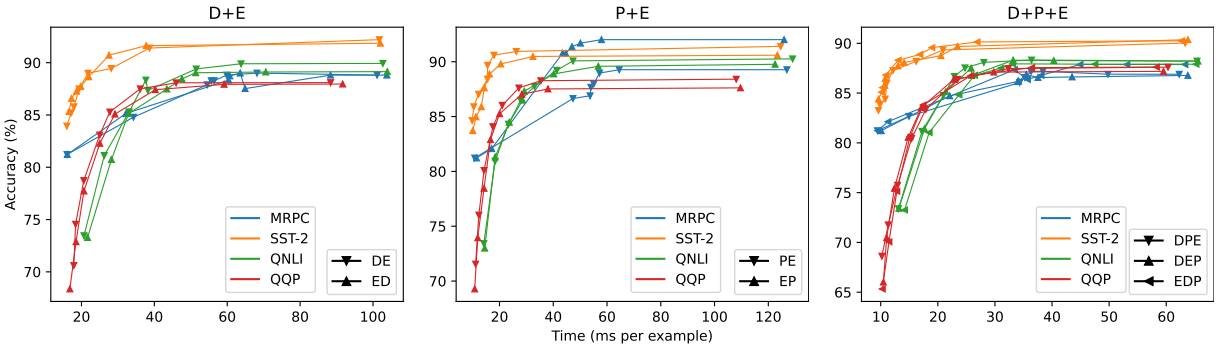


Figure 2: Different orderings of the same set of operators have similar tradeoff curves. The title of each subfigure shows the set of operators; each color represents a dataset; each marker shape represents a component ordering.

D after **P** does not make sense, since **D** initializes a small student, and the efficiency brought by the pruning step cannot be passed over to the student. With these constraints, the number of meaningful pipelines is significantly reduced.

3.2 Datasets and Implementation

We conduct experiments with the RoBERTa-base model (Liu et al., 2019) on four sequence classification tasks: MRPC (Dolan and Brockett, 2005), SST-2 (Socher et al., 2013), QNLI (Rajpurkar et al., 2016; Wang et al., 2018), and QQP (Sharma et al., 2019). Our implementation of efficiency methods are adopted from Transformers (Wolf et al., 2020), Fastformers (Kim and Hassan, 2020), and DeeBERT (Xin et al., 2020). We train all the models with an NVIDIA Tesla T4 GPU. We evaluate them with an AMD Ryzen 5800X CPU, which provides more stable measurements for inference latency. Wall-clock runtime is used as the efficiency metric.

3.3 Settings for Pipelines and Operators

Before experimenting with pipelines, we explore the optimal setting (e.g., learning rate, batch size) for each individual operator and use the same setting in the pipelines. This is a realistic approach since it is impractical to search for the optimal setting for every component in every new pipeline.

For training the RoBERTa model, including original fine-tuning, distillation, and training with early exiting, we use the same hyperparameters as in the Transformers library (Wolf et al., 2020): learning rate is set to 10^{-5}; batch size is set to 8; all training procedures consist of 10 epochs with no early stopping. For pruning, we prune the number of attention heads from 12 to 8 and the intermediate dimension from 3072 to 1536. The reason for this is that in our preliminary experiments, the above combination of hyperparameters is a sweet spot on the Pareto frontier.

4 Operator Commutativity and Order

Given a set of operators, we naturally wonder about the best order to apply them. Although this question seems formidable due to the exponentially large number of possible orderings, we show that the question is actually simpler than expected: on the one hand, we have eliminated a number of invalid orderings as described in Section 3; on the other, we show that operators are commutative in the remaining ordering candidates.

4.1 Commutative Properties of Operators

In this subsection, we discuss operator commutativity separately for the two groups.

Group I We show the results of swapping the order of operators from Group I in Figure 2. Since early exiting is involved, which means the model can achieve different tradeoffs between accuracy and inference time, we present each ordering as a *tradeoff curve*, where points are drawn by varying the early exiting threshold of confidence. We can see that when we use the same set of operators (same color), different orderings have similar tradeoff curves, in most cases.

Exceptions exist, however, in the **E**+**P** combination on the MRPC dataset. We hypothesize that this is due to training randomness, since MRPC has smallest size of all. In order to study randomness, we repeat the experiment with additional random seeds and show in Figure 3 the results on MRPC. We can see that (1) the gap between the *mean* curves is smaller than the gap between curves corresponding to using a single seed; (2) the mean curve of each ordering lies within the 95% confidence interval (95% CI) of other orderings. This

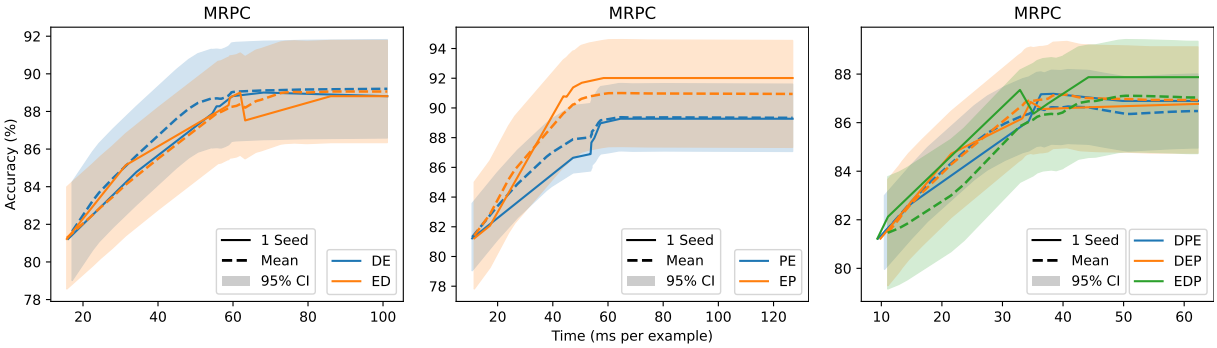


Figure 3: Comparing the results of a single run (solid lines; same as the ones from Figure 2) and the results from multiple runs (dashed lines for the mean and shaded areas for 95% confidence intervals).

shows that the differences between tradeoff curves of different orderings can, at least partly, be attributed to training randomness.

To further quantify the degree of dissimilarity between different orderings, we define and calculate the *distance* between tradeoff curves. The distance between two tradeoff curves is defined as the maximum accuracy (y-axis) difference at the same inference time (x-axis) point. We compare distances between tradeoff curves (1) generated by the same operator order but with different random seeds; and (2) generated by different operator orders. We show the results in Table 1. We can see that while tradeoff curves generated by the same operator order tend to have a smaller average distance, the difference between same/different orders is typically small and the one-standard-deviation (1-SD) intervals of both sides always overlap. Although we are unable to find a suitable significance test since the distances are not independent, the above analysis shows that the difference of distances between curves from same/different orders is likely not significant. More importantly, as shown in Figure 3, the gap between mean curves of different orderings is smaller than the deviation caused by different random seeds. Therefore, in practice, we can regard the operators as commutative.

Group II The two operators, **Q** and **L**, are independent of each other, and therefore their order can be arbitrarily swapped (i.e., they are strictly commutative by definition). We show the results of applying **Q** and/or **L** at the end of different pipelines in Table 2. We do not report the accuracy of +**L** since using dynamic length does not change the model's accuracy.

Based on the above discussion, when we have a set of components to apply, it suffices to simply pick a reasonable order from the candidate space, rather than extensively searching for the optimal setting.

Dataset	Order	D+E	P+E	D+P+E
MRPC	Same	1.57 ± 0.69	2.31 ± 0.85	1.53 ± 0.62
	Diff.	1.74 ± 0.40	4.12 ± 1.10	2.59 ± 0.97
SST-2	Same	1.30 ± 0.39	1.64 ± 0.46	1.49 ± 0.53
	Diff.	1.48 ± 0.46	1.84 ± 0.62	1.98 ± 0.82
QNLI	Same	2.24 ± 1.20	4.40 ± 2.49	3.41 ± 2.51
	Diff.	3.93 ± 0.82	4.58 ± 2.39	4.91 ± 2.44
QQP	Same	2.38 ± 1.36	2.11 ± 0.86	2.30 ± 1.05
	Diff.	3.64 ± 1.14	3.30 ± 1.27	4.56 ± 1.71

Table 1: The mean and the standard deviation (SD) of *distances* between tradeoff curves belonging to same/different orders (the same ordering is run with multiple random seeds). For all entries, the 1-SD intervals of same/different orders overlap.

5 Operator Cumulativeness and Predictability of Pipelines

In order to choose components for an efficiency pipeline, an important question is whether time savings and accuracy drops of individual operators are cumulative. In this subsection, we show that they are indeed cumulative to the degree that accuracy–efficiency tradeoffs of a new pipeline can be estimated, simply by combining the results of individual operators.

We first discuss operators from Group I. In Figure 4, we show how we can estimate the tradeoff curve of a new pipeline based on the results of its constituents, using the two larger and more stable datasets, QQP and QNLI. For example, in the top-right subfigure, we show the estimation for the tradeoff curves of pipelines comprising **E**, **D**, and **P**, based on the results of individually applying each of these operators.

Dataset	Pipeline	Accuracy (%)		Time (ms per example)				
		Raw	**+Q** (relative diff.)	Raw	**+Q**	**+L**	**+QL**	**+QL** (est.)
MRPC	**O**	92.7	92.5 (−0.2%)	170.7	−50%	−83%	−94%	−92%
	D	89.2	88.8 (−0.4%)	85.5	−49%	−82%	−94%	−91%
	P	91.0	89.0 (−2.2%)	122.4	−64%	−86%	−94%	−95%
	DP	88.9	87.9 (−1.1%)	59.3	−62%	−84%	−94%	−94%
SST-2	**O**	93.7	93.5 (−0.2%)	170.8	−50%	−86%	−97%	−93%
	D	92.3	92.3 (−0.0%)	85.5	−49%	−86%	−97%	−93%
	P	92.4	91.7 (−0.8%)	126.7	−66%	−89%	−97%	−96%
	DP	92.0	90.9 (−1.2%)	62.9	−65%	−88%	−97%	−96%
QNLI	**O**	92.3	92.1 (−0.2%)	174.2	−51%	−83%	−95%	−92%
	D	91.3	90.7 (−0.7%)	86.9	−50%	−82%	−95%	−91%
	P	91.5	91.4 (−0.1%)	121.5	−64%	−86%	−95%	−95%
	DP	89.8	89.6 (−0.2%)	62.6	−65%	−85%	−95%	−95%
QQP	**O**	88.6	88.3 (−0.3%)	172.3	−51%	−86%	−96%	−93%
	D	87.9	87.7 (−0.2%)	88.2	−51%	−85%	−97%	−93%
	P	88.5	88.5 (−0.0%)	118.3	−63%	−87%	−97%	−95%
	DP	87.6	87.6 (−0.0%)	58.8	−62%	−86%	−97%	−95%

Table 2: Accuracy drops and time savings provided by quantization (**Q**) and dynamic length inference (**L**) applied at the end of pipelines. The accuracy drops and time savings of most operators are cumulative.

The idea for estimating accuracy drops is based on the following *cumulativeness assumption*. Suppose **R** is a pipeline and A_* is the accuracy for a pipeline *, the assumption is:

$$A_{\mathbf{R+D}} = \frac{A_{\mathbf{D}}}{A_{\mathbf{O}}} \times A_{\mathbf{R}}, \quad (1)$$

$$A_{\mathbf{R+P}} = \frac{A_{\mathbf{P}}}{A_{\mathbf{O}}} \times A_{\mathbf{R}}. \quad (2)$$

In other words, our assumption is that adding **D** or **P** to *any* pipeline should result in similar relative accuracy drops. We can therefore estimate the accuracy of **ED**, **EP**, and **EDP** (and other orders of the same set of operators) as follows: (1) calculate accuracy drops of **D** and **P** relative to **O**; (2) multiply the relative accuracy drops to points on **E**'s tradeoff curve.

The idea for estimating time savings is also similar, but additional modifications are necessary:

- When we add **P** to **E**, since they work on reducing different dimensions of the model (width and depth), the time savings are independent and directly cumulative:

  $$T_{\mathbf{E+P}} = \frac{T_{\mathbf{P}}}{T_{\mathbf{O}}} \times T_{\mathbf{E}}, \quad (3)$$

  where similarly, T_* is the inference time for a pipeline *.

- When we add **D** to **E**, we need to consider the fact that both **D** and **E** reduce the number of layers. Therefore, our estimation is based on interpolating the following two extreme cases. When the early exiting threshold is extremely large and the model uses all layers for inference, the relative time saving will be close to $T_{\mathbf{D}}/T_{\mathbf{O}}$; when the early exiting threshold is extremely small and the model exits after the first layer, adding **D** provides no extra time saving. The final time saving estimation for **E+D** is therefore the following interpolation:

  $$T_{\mathbf{E+D}} = t_{\mathbf{E}} + (T_{\mathbf{E}} - t_{\mathbf{E}}) \times \frac{T_{\mathbf{D}}}{T_{\mathbf{O}}}, \quad (4)$$

  where $t_{\mathbf{E}}$ is the minimum value of time in the tradeoff curve of **E** (i.e., the point where we early exit after only one layer).

- When we add both **P** and **D** to **E**, we combine the above two estimations:

  $$T_{\mathbf{E+DP}} = \left(t_{\mathbf{E}} + (T_{\mathbf{E}} - t_{\mathbf{E}}) \times \frac{T_{\mathbf{D}}}{T_{\mathbf{O}}}\right) \times \frac{T_{\mathbf{P}}}{T_{\mathbf{O}}}. \quad (5)$$

We use the above ideas to estimate tradeoff curves of new pipelines and show the results in Figure 4. From the figure, we can see that the estimation curves (orange) align well with the measured

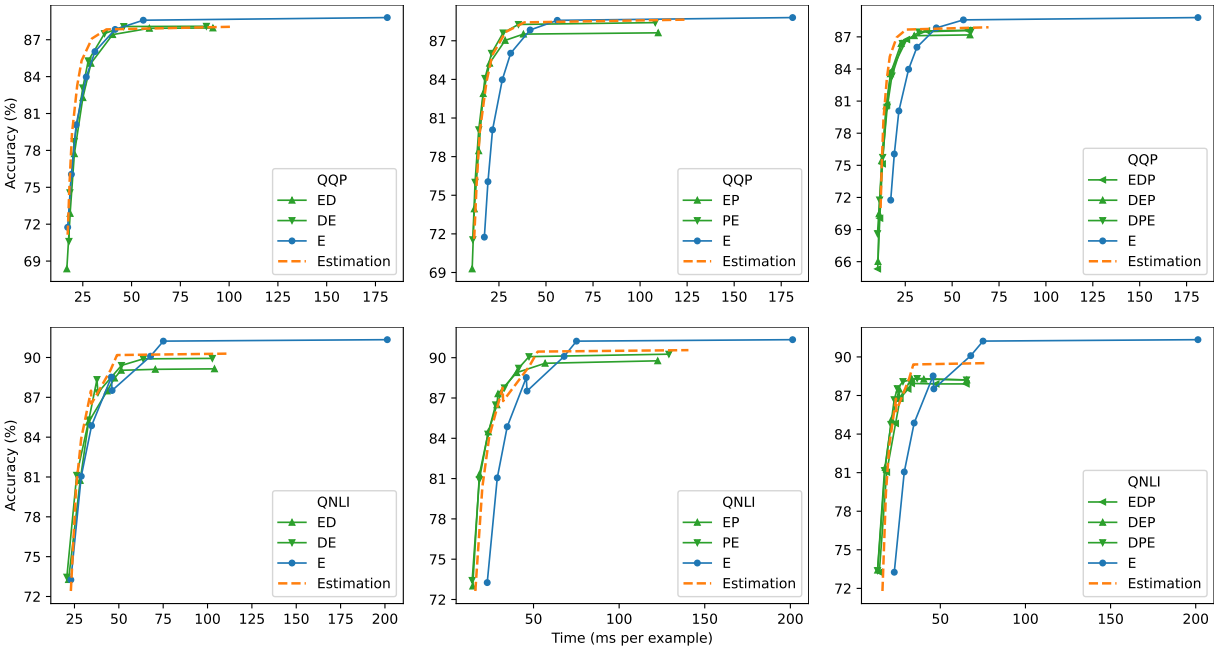


Figure 4: Estimating the tradeoff curves of target pipelines based on the results of individually applying operators. Green curves: measured tradeoff curves of target pipelines; blue curves: measured tradeoff curves of individually applying the operator **E**; orange curves: estimated tradeoff curves for the target pipelines.

curves (green), across different datasets and operator sets. This shows that individual components from Group I are cumulative with each other under these settings.

For operators from Group II, we refer to Table 2. We see that on the same dataset, **Q** leads to similar accuracy drops when added to any pipeline, especially on the larger and more stable datasets, QNLI and QQP. Time savings, on the other hand, are trickier:

- **L** provides consistent time savings for all pipelines, showing that it is cumulative with any operator from Group I.
- **L** and **Q** are also cumulative with each other, as evidenced by the fact that the measured time savings of **+QL** align well with the estimation of **+QL**, which is simply multiplying the respective savings of **Q** and **L**.
- **Q**, however, is cumulative only with **D** and **E**, but not **P**—it saves more time for pipelines with **P**. This is because quantization's acceleration is different for different types of operations, and pruning changes the proportion of each type of operations within a transformer layer, while distillation or early exiting does not. When we estimate the tradeoff of a pipeline containing both **Q** and **P**, **PQ** needs to be treated as a compound operator, and it is cumulative with others. This also applies to other operators that change the connection within a transformer layer.

Empirically, the observation that operators are cumulative facilitates future experiments on efficiency pipelines: for pipelines that are computationally expensive to train and evaluate, simply measuring the performance of their components can provide us with a reliable estimation of the pipeline's behavior. Therefore, choosing efficiency methods for a pipeline according to desired accuracy–efficiency tradeoffs becomes easy calculation once the measurement of individual operators is finished.

On the theoretical side, the cumulativeness observation also makes it easier to analyze the contribution of each component, i.e., how much time each operator saves and how much accuracy each sacrifices. The Shapley value (Shapley, 1997) of each component, for instance, can be approximated by simply using the standalone estimation (Fréchette et al., 2016).

6 Conclusion

In this paper, we propose a conceptual framework to consider efficiency methods as operators applied on transformer models and study the properties of these operators when used as pipelines. We ob-

serve that, under the condition of our experiments, (1) efficiency operators are commutative: changing their order has little practical impact on the final efficiency–accuracy tradeoff; (2) efficiency operators are cumulative: a new pipeline's performance can be estimated by aggregating time savings and accuracy drops of each component. These observations facilitate future construction, evaluation, and application of efficiency pipelines, and also provide an interesting direction to better understand efficiency pipelines.

Limitations

There exist so many different transformer models and efficiency methods that it is extremely difficult to conduct exhaustive experiments for all of them. Although our experiments demonstrate nice properties for efficiency operators, the observations are restricted to our experimental setup. Considering the huge space of all combinations of transformer models, efficiency methods, and datasets, our experiments provide understanding for an important but small subspace, and it is possible that the conclusions no longer hold when we explore further. We hope that our discoveries can inspire more future research, both empirical and theoretical, to push further the frontier of our understanding of the space.

Acknowledgements

We thank anonymous reviewers for their constructive suggestions. This research is supported in part by the Canada First Research Excellence Fund and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

Nima Aghli and Eraldo Ribeiro. 2021. Combining weight pruning and knowledge distillation for cnn compression. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3185–3192.

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18.

Baiyun Cui, Yingming Li, and Zhongfei Zhang. 2021. Joint structured pruning and dense knowledge distillation for efficient transformer model compression. *Neurocomputing*, 458:56–69.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Alexandre Fréchette, Lars Kotthoff, Tomasz Michalak, Talal Rahwan, Holger Hoos, and Kevin Leyton-Brown. 2016. Using the Shapley value to analyze algorithm portfolios. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Mitchell Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 143–155, Online. Association for Computational Linguistics.

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 *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3690–3699. PMLR.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

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 *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Gyuwan Kim and Kyunghyun Cho. 2021. Length-adaptive transformer: Train once with length drop, use anytime with search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6501–6511, Online. Association for Computational Linguistics.

Young Jin Kim and Hany Hassan. 2020. FastFormers: Highly efficient transformer models for natural language understanding. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 149–158, Online. Association for Computational Linguistics.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2849–2858, New York, New York, USA. PMLR.

Ye Lin, Yanyang Li, Tong Xiao, and Jingbo Zhu. 2021. Bag of tricks for optimizing transformer efficiency. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4227–4233, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. FastBERT: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6035–6044, Online. Association for Computational Linguistics.

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*.

JS McCarley, Rishav Chakravarti, and Avirup Sil. 2019. Structured pruning of a BERT-based question answering model. *arXiv preprint arXiv:1910.06360*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Jinhyuk Park and Albert No. 2022. Prune your model before distill it. In *European Conference on Computer Vision*, pages 120–136. Springer.

Mary Phuong and Christoph Lampert. 2019. Distillation-based training for multi-exit architectures. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1355–1364.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*.

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.

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

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6640–6651, Online. Association for Computational Linguistics.

Lloyd S. Shapley. 1997. A value for n-person games. *Classics in game theory*, 69.

Lakshay Sharma, Laura Graesser, Nikita Nangia, and Utku Evci. 2019. Natural language understanding with the Quora Question Pairs dataset. *arXiv preprint arXiv:1907.01041*.

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.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. 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, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE.

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*, volume 30. Curran Associates, Inc.

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, Brussels, Belgium. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021. BERxiT: Early exiting for BERT with better fine-tuning and extension to regression. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 91–104, Online. Association for Computational Linguistics.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. BERT loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*, volume 33, pages 18330–18341. Curran Associates, Inc.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
The Limitations section on page 9.

☑ A2. Did you discuss any potential risks of your work?
The Limitations section on page 9.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract and Introduction (Section 1)

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 3.2

☑ B1. Did you cite the creators of artifacts you used?
Section 3.2

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
I've never seen such things in previous papers.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Not applicable. Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Not applicable. Left blank.

C ☑ Did you run computational experiments?

Section 3.2

☒ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
number of parameters: everyone knows the total computational budget (e.g., GPU hours), and computing infrastructure used: nobody cares

---

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 3.2

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Section 4

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 3.2

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.