Model Cascading: Towards Jointly Improving Efficiency and Accuracy of NLP Systems

Neeraj Varshney and Chitta Baral
Arizona State University
`{nvarshn2, cbaral}@asu.edu`

Abstract

Do all instances need inference through the big models for a correct prediction?
Perhaps not; some instances are easy and can be answered correctly by even small capacity models. This provides opportunities for improving the computational efficiency of systems. In this work, we present an explorative study on 'model cascading', a simple technique that utilizes a collection of models of varying capacities to accurately yet efficiently output predictions. Through comprehensive experiments in multiple task settings that differ in the number of models available for cascading (K value), we show that cascading improves both the computational efficiency and the prediction accuracy. For instance, in K=3 setting, cascading saves up to 88.93% computation cost and consistently achieves superior prediction accuracy with an improvement of up to 2.18%. We also study the impact of introducing additional models in the cascade and show that it further increases the efficiency improvements. Finally, we hope that our work will facilitate development of efficient NLP systems making their widespread adoption in real-world applications possible.

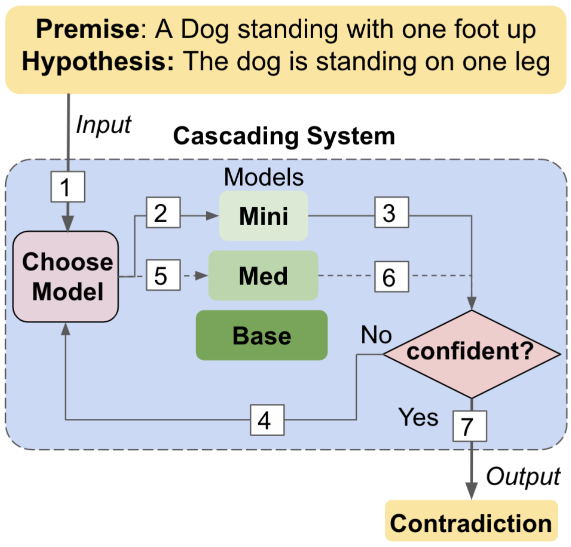

Figure 1: Illustrating a cascading approach with three models (Mini, Med, and Base) arranged in increasing order of capacity. An input is first passed through the smallest model (Mini) which fails to predict with sufficient confidence. Therefore, it is then inferred using a bigger model (Med) that satisfies the confidence constraints and the system outputs its prediction ('contradiction' as dog has four legs). Thus, by avoiding inference through large/expensive models, the system saves computation cost without sacrificing the accuracy.

1 Introduction

Pre-trained language models such as RoBERTa (Liu et al., 2019), ELECTRA (Clark et al., 2020), and T5 (Raffel et al., 2020) have achieved remarkable performance on numerous natural language processing benchmarks (Wang et al., 2018, 2019; Talmor et al., 2019). However, these models have a large number of parameters which makes them slow and computationally expensive; for instance, T5-11B requires $\sim 87 \times 10^{11}$ floating point operations (FLOPs) for an inference. This limits their widespread adoption in real-world applications that prefer computationally efficient systems in order to achieve low response times.

The above concern has recently received considerable attention from the NLP community leading to development of several techniques, such as (1) *network pruning* that progressively removes model weights from a big network (Wang et al., 2020; Guo et al., 2021), (2) *early exiting* that allows multiple exit paths in a model (Xin et al., 2020), (3) *adaptive inference* that adjusts model size by adaptively selecting its width and depth (Goyal et al., 2020; Kim and Cho, 2021), (4) *knowledge distillation* that transfers 'dark-knowledge' from a large teacher model to a shallow student model (Jiao et al., 2020; Li et al., 2022), and (5) *input reduction* that eliminates less contributing tokens from the input text to speed up inference (Modarressi et al., 2022). These methods typically require architectural modifications, network manipulation, saliency quantification, or even complex training procedures. Moreover, computational efficiency in these methods often comes with a compromise on accuracy. In contrast, *model cascading*, a simple technique that utilizes a collection of models of varying capacities to **accurately yet efficiently** output predictions has remained underexplored.

In this work, we address the above limitation by first providing mathematical formulation of model cascading and then exploring several approaches to do it. In its problem setup, a collection of models of different capacities (and hence performances) are provided and the system needs to output its prediction by leveraging one or more models. On one extreme, the system can use only the smallest model and on the other extreme, it can use all the available models (ensembling). The former system would be highly efficient but usually poor in performance while the latter system would be fairly accurate but expensive in computation. Model cascading strives to get the best of both worlds by allowing the system to efficiently utilize the available models while achieving high prediction accuracy. This is in line with the 'Efficiency NLP' (Arase and et al., 2021) policy document put up by the ACL community.

Consider the case of CommitmentBank (de Marneffe et al., 2019) dataset on which BERT-medium model having just 41.7M parameters achieves 75% accuracy and a bigger model BERT-base having 110M parameters achieves 82% accuracy. From this, it is clear that the performance of the bigger model can be matched by inferring a large number of instances using the smaller model and only a few instances using the bigger model. Thus, by carefully deciding when to use bigger/more expensive models, the computational efficiency of NLP systems can be improved. *So, how should we decide which model(s) to use for a given test instance?* Figure 1 illustrates an approach to achieve this; it infers an instance sequentially through models (ordered in increasing order of capacity) and uses a threshold over the maximum softmax probability (*MaxProb*) to decide whether to output the prediction or pass it to the next model in sequence. The intuition behind this approach is that MaxProb shows a positive correlation with predictive correctness. Thus, instances that are predicted with high MaxProb get answered at early stages as their predictions are likely to be correct and the remaining ones get passed to the larger models. Hence, by avoiding inference through large and expensive models (primarily for easy instances), cascading makes the system computationally efficient while maintaining high prediction performance.

We describe several such cascading methods in Section 3.2. Furthermore, cascading allows custom computation costs as different number of models can be used for inference. We compute accuracies for a range of costs and plot an accuracy-cost curve. Then, we calculate its area (AUC) to quantify the efficacy of the cascading method. Larger the AUC value, the better the method is as it implies higher accuracy on average across computation costs.

We conduct comprehensive experiments with 10 diverse NLU datasets in multiple task settings that differ in the number of models available for cascading (K value from Section 3). We first demonstrate that cascading achieves considerable improvement in computational efficiency. For example, in case of QQP dataset, cascading system achieves 88.93% computation improvement over the largest model (M_3) in K=3 setting i.e. it requires just 11.07% of the computation cost of model M_3 to attain equal accuracy. Then, we show that cascading also achieves improvement in prediction accuracy. For example, on CB dataset, the cascading system achieves 2.18% accuracy improvement over M_3 in the K=3 setting. Similar improvements are observed in settings with different values of K. Lastly, we show that introducing additional model in the cascade further increases the efficiency benefits.

In summary, our contributions and findings are:

1. **Model Cascading:** We provide mathematical formulation of model cascading, explore several methods, and systematically study its benefits.
2. **Cascading Improves Efficiency**: Using accuracy-cost curves, we show that cascading systems require much lesser computation cost to attain accuracies equal to that of big models.
3. **Cascading Improves Accuracy:** We show that cascading systems consistently achieve superior prediction performance than even the largest model available in the task setting.
4. **Comparison of Cascading Methods:** We compare performance of our proposed cascading methods and find that *DTU* (3.2) outperforms all others by achieving the highest AUC of accuracy-cost curves on average.

We note that model cascading is trivially easy to implement, can be applied to a variety of problems, and can have good practical values.

2 Related Work

In recent times, several techniques have been developed to improve the efficiency of NLP systems, such as *network pruning* (Wang et al., 2020; Guo et al., 2021; Chen et al., 2020), *quantization* (Shen et al., 2020; Zhang et al., 2020; Tao et al., 2022), *knowledge distillation* (Clark et al., 2019; Jiao et al., 2020; Li et al., 2022; Mirzadeh et al., 2020), and *input reduction* (Modarressi et al., 2022). Our work is more closely related to dynamic inference (Xin et al., 2020) and adaptive model size (Goyal et al., 2020; Kim and Cho, 2021; Hou et al., 2020; Soldaini and Moschitti, 2020).

Xin et al. (2020) proposed Dynamic early exiting for BERT (DeeBERT) that speeds up BERT inference by inserting extra classification layers between each transformer layer. It allows an instance to choose conditional exit from multiple exit paths. All the weights (including newly introduced classification layers) are jointly learnt during training.

Goyal et al. (2020) proposed Progressive Word-vector Elimination (PoWER-BERT) that reduces intermediate vectors computed along the encoder pipeline. They eliminate vectors based on significance computed using self-attention mechanism. Kim and Cho (2021) extended PoWER-BERT to Length-Adaptive Transformer which adaptively determines the sequence length at each layer. Hou et al. (2020) proposed a dynamic BERT model (DynaBERT) that adjusts the size of the model by selecting adaptive width and depth. They first train a width-adaptive BERT and then distill knowledge from full-size models to small sub-models.

Lastly, cascading has been studied in machine learning and vision with approaches such as Haar-cascade (Soo, 2014) but is underexplored in NLP (Li et al., 2021). We further note that cascading is non-trivially different from 'ensembling' as ensembling always uses all the available models instead of carefully selecting one or more models for inference.

Our work is different from existing methods in the following aspects: (1) Existing methods typically require architectural changes, network manipulation, saliency quantification, knowledge distillation, or complex training procedures. In contrast, cascading is a simple technique that is easy to implement and does not require such modifications, (2) The computational efficiency in existing methods often comes with a compromise on accuracy. Contrary to this, we show that model cascading surpasses the accuracy of even the largest models, (3) Existing methods typically require training a separate model for each computation budget; on the other hand, a single cascading system can be adjusted to meet all the computation constraints. (4) Finally, cascading does not require an instance to be passed sequentially through the model layers; approaches such as *routing* (section 3) allow passing it directly to a suitable model.

3 Model Cascading

We define model cascading as follows:

Given a collection of models of varying capacities, the system needs to leverage one or more models in a computationally efficient way to output accurate predictions.

As previously mentioned, a system using only the smallest model would be highly efficient but poor in accuracy and a system using all the available models would be fairly accurate but expensive in computation. The goal of cascading is to achieve high prediction accuracy while efficiently leveraging the available models. The remainder of this section is organized as follows: we provide mathematical formulation of cascading in 3.1 and describe its various approaches in 3.2.

3.1 Formulation

Consider a collection of K trained models $(M_1, ..., M_K)$ ordered in increasing order of their computation cost i.e. for an instance x, $c_j^x < c_k^x$ ($\forall j < k$) where c corresponds to the cost of inference. The system needs to output a prediction for each instance of the evaluation dataset D leveraging one or more models. Let M_j^x be a function that indicates whether model M_j is used by the system to make inference for the instance x i.e.

$$M_j^x = \begin{cases} 1, & \text{if model } M_j \text{ is used for instance } x \\ 0, & \text{otherwise} \end{cases}$$

Thus, the average cost of the system for the entire evaluation dataset D is calculated as:

$$Cost_D = \frac{\sum_{x_i \in D} \sum_{j=1}^{K} M_j^{x_i} \times c_j^{x_i}}{|D|}$$

In addition to this cost, we also measure accuracy i.e. the percentage of correct predictions by the system. The goal is to achieve high prediction accuracy while being computationally efficient.

Performance Evaluation: With the increase in the computation cost, the accuracy usually also increases as the system leverages large models (that are often more accurate) for more number of instances. To quantify the performance of a cascading method, we first plot its accuracy-cost curve by varying the computation costs and then calculate the area under this curve (AUC). **Larger the AUC value, the better the cascading method is** as it implies higher accuracy on average across all computation costs. We note that the computation cost of the cascading system can be varied by adjusting the confidence thresholds of models in the cascade (described in the next subsection).

Along with the AUC metric, we evaluate efficacy of cascading on two additional parameters:

1. **Comparing computation cost of the cascading system at accuracies achieved by each individual model of cascade:** Consider a setting in which the model M_2 achieves accuracy a_2 at computation cost c_2; from the accuracy-cost curve of the cascading system, we compare c_2 with the cost of the cascading system when its accuracy is a_2.
2. **Comparing the maximum accuracy of the cascading system with that of the largest model of collection:** We compare accuracy of the largest individual model with the maximum accuracy achieved by the cascading system.

Note that the first parameter corresponds to the point of intersection obtained by drawing a horizontal line from accuracy-cost point of each individual model on the accuracy-cost curve. Refer to the red dashed lines in Figure 2 and 4 for illustration. For a cascading system to perform better than the individual models in the cascade, it should have a lower computation cost (in parameter one) and a higher accuracy (in parameter two).

3.2 Approaches

We explore the following approaches of selecting which model(s) to use for inference.

Maximum Softmax Probability (MaxProb): Usually, the last layer of a model has a softmax activation function that distributes its prediction probability $P(y)$ over all possible answer candidates Y. MaxProb corresponds to the maximum softmax probability assigned by the model i.e.

$$MaxProb = \max_{y \in Y} P(y)$$

MaxProb (often termed as prediction confidence) has been shown to be positively correlated with predictive correctness (Hendrycks and Gimpel, 2017; Hendrycks et al., 2020; Varshney et al., 2022c) i.e. a high MaxProb value implies a high likelihood for the model's prediction to be correct. We leverage this characteristic of MaxProb in our first cascading approach. Specifically, we infer the given input instance sequentially through the models starting with M_1 and use a confidence threshold over MaxProb value to decide whether to output the prediction or pass the instance to the next model in sequence.

Consider an instance x for which the models till M_{z-1} fail to surpass their confidence thresholds and M_z exceeds its threshold then:

$$M_j^x = \begin{cases} 1, & \text{if } j \leq z \\ 0, & \text{if } j > z \end{cases}$$

The confidence thresholds could be different at different stages. Figure 1 illustrates this approach.

It provides efficiency benefits as it avoids passing easy instances (that can be potentially answered correctly by low-compute models) to the computationally expensive models. Furthermore, it does not sacrifice the accuracy of system because the difficult instances would often end up being answered by the large (and more accurate) models. We note that this approach requires additional computation for comparing MaxProb values with thresholds but its cost is negligible in comparison to the cost of model inferences and hence ignored in the overall cost calculation.

Distance To Uniform Distribution (DTU): In this approach, we use the distance between the model's softmax probability distribution and the uniform probability distribution as the confidence estimate (in place of MaxProb) to decide whether to output the prediction or pass the instance to the next model in sequence i.e.

$$DTU = ||P(Y) - U(Y)||_2$$

where $U(Y)$ corresponds to the uniform output distribution. For example, in case of a task with 4 classification labels, $U(Y) = [0.25, 0.25, 0.25, 0.25]$. The intuition behind this approach is to leverage the entire shape of the output probability distribution and not just the highest probability as in MaxProb.

Random: In this approach, instead of using a metric such as MaxProb or DTU to decide which instances to pass to the next model in sequence, we do this instance selection process at random. This serves as a baseline cascading method.

Heuristic: Here, we use a heuristic derived from the input text to decide which instances to pass to the next model in sequence. Specifically, we use length of the input text as the heuristic.

Routing: In this approach, instead of sequentially passing an instance to bigger and bigger models, we skip intermediate models and pass the instance directly to a suitable model based on its maxProb value. For example, in $K = 3$ setting, we first infer using M_1 and if its maxProb is very low then we skip M_2 and directly pass it to M_3. On the other hand, if its maxProb is sufficiently high (but below M_1's output threshold) then we pass it to M_2. The intuition behind this approach is that the system might save inference cost of intermediate models by directly using a suitable model that is likely to answer it correctly. This approach is not applicable for $K = 2$ as there is only one option to route after inference through the model M_1.

4 Experiments

4.1 Experimental Details

Datasets: We experiment with a diverse set of NLU classification datasets: SNLI (Bowman et al., 2015), Multi-NLI (Williams et al., 2018), Dialogue-NLI (Welleck et al., 2019), Question-NLI (Wang et al., 2018), QQP (Iyer et al., 2017), MRPC (Dolan and Brockett, 2005), PAWS (Zhang et al., 2019), SST-2 (Socher et al., 2013), COLA (Warstadt et al., 2019), and CommitmentBank (de Marneffe et al., 2019).

Models: We use the following variants of BERT (Devlin et al., 2019): BERT-mini (11.3M parameters), BERT-medium (41.7M parameters), BERT-base (110M parameters), and BERT-large (340M parameters) for our experiments. Table 1 shows the computation cost (in FLOPs) of these models for different input text sequence lengths. We use sequence length of 50 for COLA, 80 for SST2, 100 for QQP, 120 for MNLI, DNLI, SNLI, 150 for QNLI, MRPC, PAWS, and 275 for Commitment-Bank datasets following the standard experimental practice. We run all our experiments on Nvidia V100 GPUs with a batch size of 32 and learning rate ranging in $\{1{-}5\}e{-}5$.

Length	**Mini** (128M)	**Medium** (474M)	**Base** (1.3G)	**Large** (3.8G)
50	0.16	1.26	4.25	5.10
80	0.25	2.01	6.80	24.16
100	0.31	2.52	8.49	30.20
120	0.38	3.02	10.19	36.24
150	0.47	3.78	12.74	45.30
220	0.69	5.54	18.69	66.44
275	0.87	6.92	23.36	83.05

Table 1: Inference cost (in 10^9 FLOPs) of BERT variants for different input text sequence lengths. We also specify the storage size of the models in this table.

In the following subsections, we study the effect of cascading in multiple settings that differ in the number of models in the cascade i.e. K value in the task formulation.

4.2 Cascading with Two Models (K=2)

4.2.1 Problem Setup

In this setting, we consider two trained models BERT-medium (41.7M parameters) as M_1 and BERT-base (110M parameters) as M_2. We analyze results for other model combinations (such as medium, large and mini, large) in Appendix C.

4.2.2 Results

Recall that the computation cost of a cascading system can be controlled by changing the M_j values. For example, in case of MaxProb, changing the confidence threshold value would result in different M_j values and hence different cost and accuracy values. Figure 2 shows accuracy-cost curves for two cascading approaches: MaxProb (in blue) and Random Baseline (in black). In the same figure, we also show accuracy-cost points for the individual models M_1 and M_2. To avoid cluttering these figures, we plot accuracy-cost curves for other approaches in separate figures and present them in Appendix C. However, to compare the performance of these methods, we provide their AUC values (of their respective accuracy-cost curves) in Table 2.

Efficiency Improvement: The accuracy-cost curves show that the cascading system matches the accuracy of the larger model M_2 at considerably lesser computation cost. This cost value corresponds to the point of intersection on the curve with a straight horizontal line drawn from M_2 (red dashed line). For example, in case of QQP, model M_2 achieves 89.99% accuracy at average computation cost of 8.49×10^9 FLOPs while the cascading system achieves the same accuracy at only 2.82

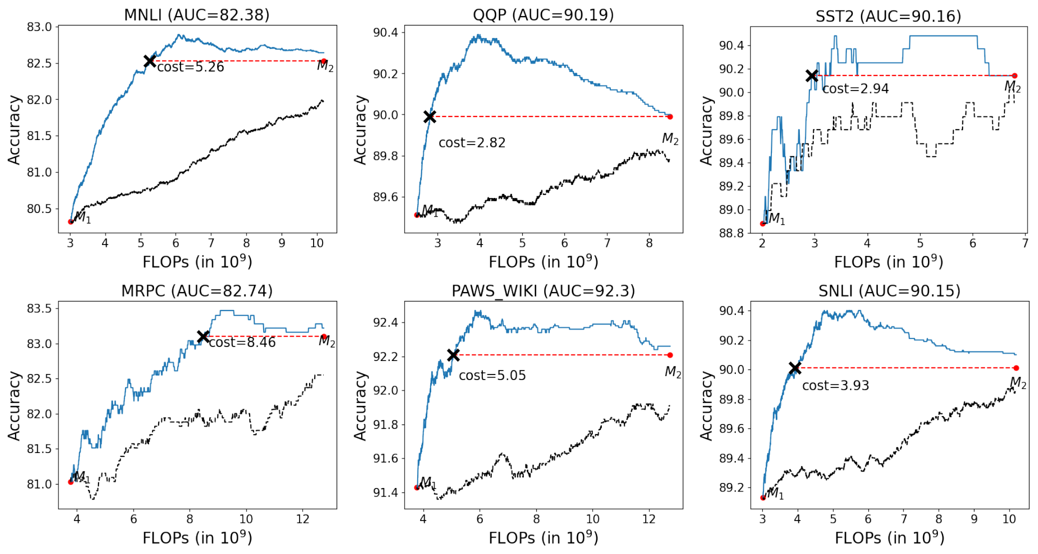

Figure 2: Accuracy-computation cost curves for cascading with *MaxProb* (in blue) and *Random* baseline (in black) methods in K=2 setting. Red points correspond to the accuracy-cost values of individual models M_1 and M_2. Points of intersection of red dashed lines drawn from M_2 on the blue curve correspond to the evaluation parameters described in Section 3. *MaxProb* outperforms *Random* baseline as it achieves considerably higher AUC.

Method	**MNLI**	**QNLI**	**QQP**	**SST2**	**COLA**	**CB**	**DNLI**	**MRPC**	**PAWS**	**SNLI**
Random	81.16	84.24	89.64	89.64	77.97	77.27	85.35	81.76	91.63	89.50
Heuristic	81.13	84.15	89.69	89.06	79.45	76.79	85.30	81.54	91.65	89.45
MaxProb	82.38	85.73	90.19	90.16	80.25	80.40	85.27	82.74	92.30	90.15
DTU	82.38	85.73	90.18	90.16	80.25	80.70	85.24	82.74	92.30	90.16

Table 2: Comparing AUC values of different cascading methods in K=2 setting. *Random* and *Heuristic* correspond to the cascading baselines. *MaxProb* and *DTU* outperform both the baselines.

$\times$ 10^9 FLOPs. Similarly, in case of MNLI, M_2 achieves 82.53% accuracy at cost of 10.19×10^9 FLOPs while the cascading system achieves the same accuracy at only 5.26×10^9 FLOPs. Such improvements are observed for all datasets. This efficiency benefit comes from using the smaller models for a large number of instances and passing only a few instances to the larger models.

Accuracy Improvement: From the accuracy-cost curves, it can be observed that beyond the cost value identified in the previous paragraph (where the red dashed line intersects the accuracy-cost curve), the cascading system outperforms model M_2 in terms of accuracy. For example, in case of QQP, cascading with MaxProb achieves accuracy of up to 90.39% that is higher than the accuracy of M_2 (89.99%). Similar improvements are observed for all other datasets. We note that the accuracy improvement is a by-product of cascading, its primary benefit remains to be the improvement in computational efficiency.

Higher accuracy achieved by the cascading system (that uses M_1 for some instances and conditionally also uses M_2 for others) than the larger model M_2 implies that M_1, despite being smaller in size is more accurate than M_2 on at least a few instances. Though, on average across all instances, M_2 has higher accuracy than M_1. The cascading system uses M_1 for instances on which it is sufficiently confident and thus more likely to be correct. Only the instances on which it is not sufficiently confident get passed to the bigger model. This supports the findings of recent works such as (Zhong et al., 2021; Varshney et al., 2022b) that conduct instance-level analysis of models' predictions. We further analyze these results in the next paragraphs.

Comparing Cascading Approaches: Figure 2 demonstrates that MaxProb cascading approach clearly outperforms the 'Random' cascading baseline. In Table 2, we compare AUC of respective accuracy-cost curves of various cascading approaches. Both MaxProb and DTU outperform

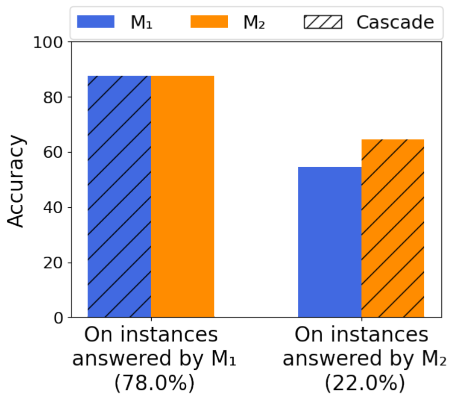

Figure 3: Comparing accuracy of individual models M_1 and M_2 on the instances answered by each model when used as cascade for MNLI dataset in K=2 setting.

both the baseline methods (Random and Heuristic). In $K = 2$ setting, both MaxProb and DTU achieve roughly the same performance on average across all datasets. The gap between MaxProb and DTU becomes more significant in $K = 3$ setting (4.3).

Contribution of M_1 and M_2 in the Cascade: To further analyze the performance of the cascading system, we study the contribution of individual models M_1 and M_2 in the cascade. Figure 3 shows the contribution of M_1 and M_2 for MNLI dataset when the cost is 5.26×10^9 FLOPs i.e. the point at which the accuracy of the cascading system is equal to that of the bigger model M_2 (intersection point of the horizontal red dashed line with the accuracy-cost curve of the cascading system in Fig 2). At this point, the cascade system uses M_1 for 78% instances and M_2 for the remaining 22% instances. The accuracy of M_1 on its 78% instances (87.6%) would be equal to that of M_2 on those 78% instances as the overall accuracy of system on complete dataset (100% instances) is equal to that of M_2. However, this does not imply that the instance-level predictions of the two models on those 78% would be exactly the same. Though, their predictions overlap in majority of the cases.

Figure 3 also shows that the accuracy of model M_1 on the instances that got passed to M_2 in the cascade system is significantly lesser (by 33.12%) than on the instances that M_1 answered (blue bars). M_2 achieves 10.12% higher accuracy on those instances than M_1. Therefore, the cascading system utilizes the models efficiently by using the smaller model M_1 for the easy instances and the larger model M_2 for the difficult ones. We analyze these results for other datasets in Appendix C.2.1.

4.3 Cascading with Three Models (K=3)

4.3.1 Problem Setup

Now, we study the effect of introducing another model in the problem setup of K=2 setting. Specifically, we consider three models: BERT-mini (11.3M parameters) as M_1, BERT-medium (41.7M parameters) as M_2, and BERT-base (110M parameters) as M_3 in this setting. **Note that BERT-medium is referred to as M_2 in this setting as it is the second model in cascading setup unlike the $K = 2$ setting (4.2) in which it was M_1.**

4.3.2 Results and Analysis

Figure 4 shows the accuracy-cost curves of two cascading approaches: MaxProb (in blue) and Random Baseline (in black) and Table 3 compares AUC values achieved by various cascading approaches. In general, cascading achieves larger improvement (in magnitude) in K=3 setting than K=2 setting.

Efficiency Improvement: The accuracy-cost curves show that the cascading system matches the accuracy of larger models M_2 and M_3 at considerably lesser respective computation costs. For example, in case of QQP, cascading system matches the accuracy of model M_3 by using just 11.07% of M_3's computation cost and of model M_2 by using just 23.53% of M_2's computation cost. The magnitude of efficiency improvement in this setting is higher than that in the K=2 setting.

Accuracy Improvement: Cascading also achieves improvement in the overall accuracy. For example, on the CB dataset, cascading system achieves 83.93% accuracy that is even higher than the largest model M_3. Similar improvements are observed for other datasets also.

Comparing Cascading Approaches: Table 3 compares AUC values achieved by various cascading approaches. DTU clearly outperforms all other cascading methods as it achieves the highest AUC values. We attribute this to DTU's characteristic of utilizing the entire shape of the output probability distribution and not just the highest probability in computing its confidence.

Contribution of M_1, M_2, and M_3 in Cascade: Figure 5 shows the contribution of individual models M_1, M_2, and M_3 in the cascade for MNLI dataset when the cost is 4.8×10^9 FLOPs i.e. the point at which accuracy of cascade is equal to that of the largest model M_3 (where the horizontal red

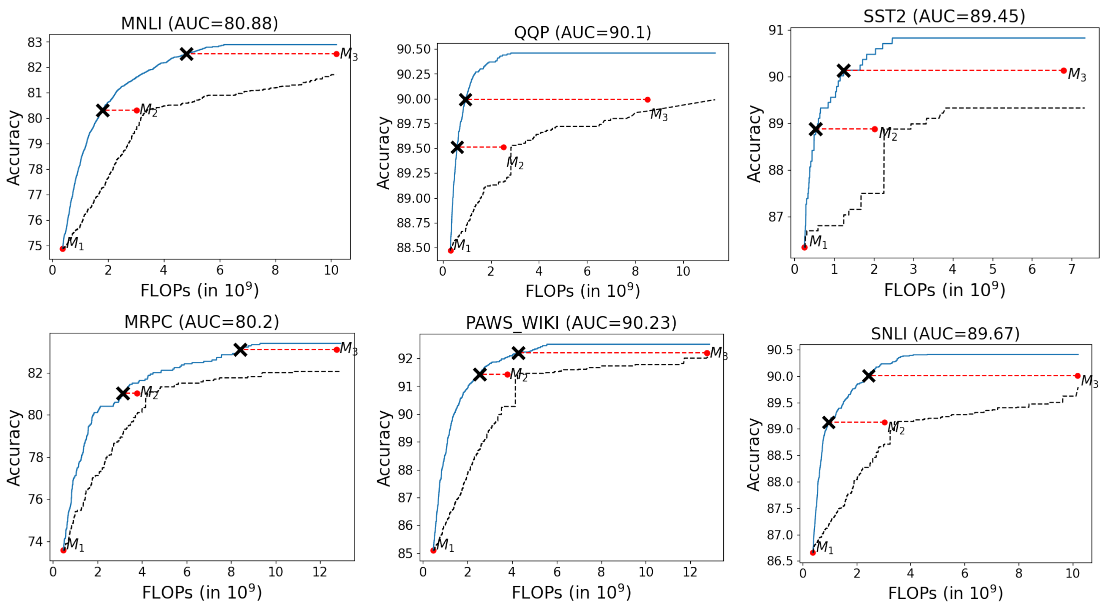

Figure 4: Accuracy-computation cost curves for cascading with MaxProb (in blue) and Random baseline (in black) methods in K=3 setting. Accuracy-cost values of individual models M_1, M_2, and M_3 are shown in red. Note that M_1 here is different from M_1 in Figure 2. *MaxProb* outperforms *Random* baseline as it achieves higher AUC.

Method	MNLI	QNLI	QQP	SST2	COLA	CB	DNLI	MRPC	PAWS	SNLI
Random	78.77	80.58	88.97	87.00	76.55	77.28	84.49	78.30	87.74	88.12
Heuristic	78.85	80.44	88.87	87.67	76.28	77.11	84.46	77.59	88.28	88.27
MaxProb	80.89	82.97	90.1	89.45	78.66	78.31	85.17	80.2	90.23	89.67
DTU	**80.98**	**83.28**	**90.15**	**89.6**	**78.87**	**78.52**	**85.20**	80.42	90.46	**89.72**
Routing	80.55	82.93	89.92	89.52	78.60	74.58	85.20	**80.97**	**90.68**	89.46

Table 3: Comparing AUC values of different cascading methods in K=3 setting. *Random* and *Heuristic* correspond to the cascading baselines. *DTU* outperforms other cascading methods on average.

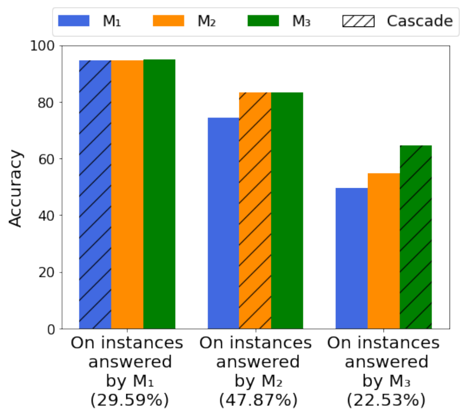

Figure 5: Comparing accuracy of individual models M_1, M_2, and M_3 on the instances answered by each model when used in the cascade for MNLI dataset.

dashed line drawn from M_3 intersects the accuracy-cost curve in Fig 4). The figure shows that the accuracy of M_1 on the instances that were passed to M_2 drops by 20.04% and accuracy of M_2 on instances that were passed to M_3 drops by 28.53%. This shows that the cascading system is good at identifying potentially incorrect predictions of M_1 and passes those instances to M_2 and similarly good at identifying potentially incorrect predictions of M_2 and passes those instances to M_3 .

Advantage of introducing another model in the Cascade: Comparing figure 5 for the K=3 setting with the figure 3 for K=2 setting, we find that by introducing a smaller model in the collection, the cascading system can be made more efficient. This is because the BERT-medium model answered 78% instances in K=2 setting and that portion got split into BERT-mini (smaller cost than medium) and medium models in K=3 setting while maintaining the accuracy. This suggests that the cascading technique utilizes the available models efficiently without sacrificing the accuracy. We analyze these results for other datasets in Appendix D.1.1.

5 Conclusion and Discussion

We systematically explored model cascading and proposed several methods for it. Through comprehensive experiments with 10 diverse NLU datasets, we demonstrated that cascading improves both the computational efficiency and the prediction accuracy. We also studied the impact of introducing another model in the collection and showed that it further improves the computational efficiency of the cascading system.

Selecting Optimal Operating Threshold: The selection of confidence threshold for models in the cascade is dependent on the computation budget of the system. A low-budget system can select low threshold for the low-cost models (so that low-cost models answer majority of the questions leading to less computation cost) and similarly, high-budget systems can afford to select high thresholds to achieve higher accuracy. In order to select thresholds in an application-independent manner, the ML's standard practice of using the validation data to tune the hyperparameters can be used.

Outlier/OOD Detection Techniques: Outlier/OOD detection techniques such as (Lee et al., 2018; Hsu et al., 2020; Liu et al., 2020) can also be explored to decide which instance to pass to the bigger models in the cascade.

Including Linear Models in the Cascade: This idea can be extended to include non-transformer based less expensive models like linear models or LSTM based models. Since the computation cost of these models is significantly lower than the transformer based models and yet they achieve non-trivial predictive performance, a cascading system with these models could achieve even more improvement in computational efficiency. We plan to explore this aspect in the future work.

Limitations

A potential downside of cascading is that it requires multiple models to be stored. However, we note that the additional space required for models (mini and medium) in K=3 setting is merely 0.44 times that required for base model (Table 1). Thus, it does not pose a serious concern.

Acknowledgement

We thank the anonymous reviewers for their insightful feedback. We also thank Tejas Gokhale for giving valuable suggestions. This research was supported by DARPA SAIL-ON and DARPA CHESS programs.

References

Yuki Arase and et al. 2021. Efficient NLP policy document, `https://bit.ly/EfficientNLP`.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained bert networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 15834–15846. Curran Associates, Inc.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D. Manning, and Quoc V. Le. 2019. BAM! born-again multi-task networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5931–5937, Florence, Italy. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Marie-Catherine de Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse.

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)*.

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

Demi Guo, Alexander Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. 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 4884–4896, Online. Association for Computational Linguistics.

Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *Proceedings of International Conference on Learning Representations*.

Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. 2020. Pretrained transformers improve out-of-distribution robustness. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2744–2751, Online. Association for Computational Linguistics.

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

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Zsolt Kira. 2020. Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10948–10957.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First quora dataset release: Question pairs. *data. quora. com*.

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.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Patrick Lewis, Ludovic Denoyer, and Sebastian Riedel. 2019. Unsupervised question answering by cloze translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4896–4910, Florence, Italy. Association for Computational Linguistics.

Lei Li, Yankai Lin, Deli Chen, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. 2021. CascadeBERT: Accelerating inference of pre-trained language models via calibrated complete models cascade. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 475–486, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. 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 4582–4597, Online. Association for Computational Linguistics.

Zheng Li, Zijian Wang, Ming Tan, Ramesh Nallapati, Parminder Bhatia, Andrew Arnold, Bing Xiang, and Dan Roth. 2022. DQ-BART: Efficient sequence-to-sequence model via joint distillation and quantization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 203–211, Dublin, Ireland. Association for Computational Linguistics.

Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. 2020. Energy-based out-of-distribution detection. In *Advances in Neural Information Processing Systems*, volume 33, pages 21464–21475. Curran Associates, Inc.

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

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. 2020. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5191–5198.

Swaroop Mishra and Bhavdeep Singh Sachdeva. 2020. Do we need to create big datasets to learn a task? In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 169–173, Online. Association for Computational Linguistics.

Ali Modarressi, Hosein Mohebbi, and Mohammad Taher Pilehvar. 2022. AdapLeR: Speeding up inference by adaptive length reduction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–15, Dublin, Ireland. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Pedro Rodriguez, Joe Barrow, Alexander Miserlis Hoyle, John P. Lalor, Robin Jia, and Jordan Boyd-Graber. 2021. Evaluation examples are not equally informative: How should that change NLP leaderboards? 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 4486–4503, Online. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics.

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.

Luca Soldaini and Alessandro Moschitti. 2020. The cascade transformer: an application for efficient answer sentence selection. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5697–5708, Online. Association for Computational Linguistics.

Sander Soo. 2014. Object detection using haar-cascade classifier. *Institute of Computer Science, University of Tartu*, 2(3):1–12.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. 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 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Chaofan Tao, Lu Hou, Wei Zhang, Lifeng Shang, Xin Jiang, Qun Liu, Ping Luo, and Ngai Wong. 2022. Compression of generative pre-trained language models via quantization. *arXiv preprint arXiv:2203.10705*.

Neeraj Varshney, Pratyay Banerjee, Tejas Gokhale, and Chitta Baral. 2022a. Unsupervised natural language inference using PHL triplet generation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2003–2016, Dublin, Ireland. Association for Computational Linguistics.

Neeraj Varshney, Swaroop Mishra, and Chitta Baral. 2022b. ILDAE: Instance-level difficulty analysis of evaluation data. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3412–3425, Dublin, Ireland. Association for Computational Linguistics.

Neeraj Varshney, Swaroop Mishra, and Chitta Baral. 2022c. Investigating selective prediction approaches across several tasks in IID, OOD, and adversarial settings. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1995–2002, Dublin, Ireland. Association for Computational Linguistics.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32. 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.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2020. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162, Online. Association for Computational Linguistics.

Zirui Wang, Adams Wei Yu, Orhan Firat, and Yuan Cao. 2021. Towards zero-label language learning. *ArXiv*, abs/2109.09193.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Sean Welleck, Jason Weston, Arthur Szlam, and Kyunghyun Cho. 2019. Dialogue natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3731–3741, Florence, Italy. Association for Computational Linguistics.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. 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.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware ultra-low bit BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 509–521, Online. Association for Computational Linguistics.

Yuan Zhang, Jason Baldridge, and Luheng He. 2019. PAWS: Paraphrase adversaries from word scrambling. 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 1298–1308, Minneapolis, Minnesota. Association for Computational Linguistics.

Ruiqi Zhong, Dhruba Ghosh, Dan Klein, and Jacob Steinhardt. 2021. Are larger pretrained language models uniformly better? comparing performance at the instance level. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3813–3827, Online. Association for Computational Linguistics.

Appendix

A Efficiency in NLP

With the introduction of large-scale pre-trained language models, the efficiency topic has attracted a lot of research attention. Efficiency is being studied from diverse lenses such as training data efficiency (Lewis et al., 2019; Schick and Schütze, 2021; Varshney et al., 2022a; Wang et al., 2021; Mishra and Sachdeva, 2020; Ben Zaken et al., 2022), evaluation efficiency (Rodriguez et al., 2021; Varshney et al., 2022b), parameter efficiency tuning methods (Li and Liang, 2021; Houlsby et al., 2019), and inference efficiency. In this work, we focus on inference efficiency and propose model cascading, a simple technique that utilizes a collection of models of varying capacities to accurately yet efficiently output predictions.

B Dataset Statistics

Table 4 shows the statistics of all evaluation datasets considered in this work. We consider a diverse set of NLU datasets spanning over several tasks, such as natural language inference, duplicate detection, and sentiment classification.

C Cascading with Two Models (K=2)

C.1 Other Model Combinations

C.1.1 Medium and Large

Figure 6 shows accuracy-cost curves with MaxProb (in blue) and Random (in black) as cascading approaches with M_1 as BERT-medium and M_2 as BERT-large. MaxProb approach clearly outperforms Random approach and achieves considerably higher AUC value.

C.1.2 Mini and Large

Figure 7 shows accuracy-cost curves with MaxProb (in blue) and Random (in black) as cascading approaches with M_1 as BERT-mini and M_2 as BERT-large. MaxProb approach clearly outperforms Random approach and achieves considerably higher AUC value.

C.2 Contribution of M_1 and M_2 in the Cascade

C.2.1 Medium and Base

Figure 8 shows the contribution of individual models M_1 and M_2 in the cascade when accuracy of M_2 is same as that of cascading system. We analyze this for MNLI dataset in section 4.2 and provide figures for a few other datasets here.

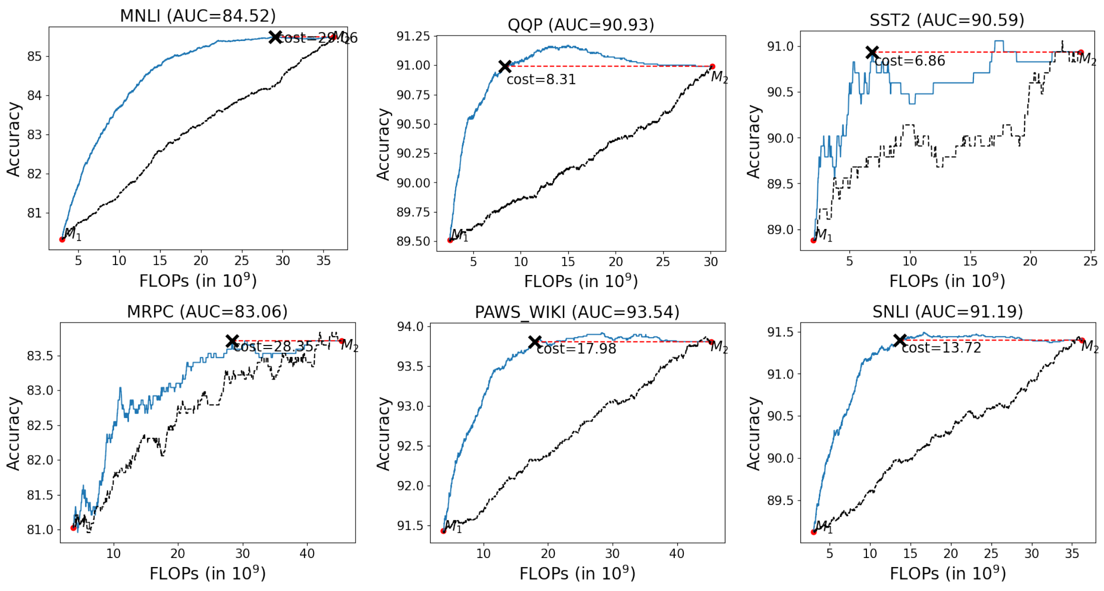

Figure 6: Accuracy-Cost curves for K=2 setting with M_1 as BERT-medium and M_2 as BERT-large models.

Dataset	Size	Dataset	Size
MNLI	19645	QNLI	5650
QQP	40371	SST2	872
COLA	1042	CB	56
DNLI	16396	MRPC	1639
PAWS	7994	SNLI	9840

Table 4: Statistics of evaluation datasets considered in this work.

D Cascading with Three Models (K=3)

D.1 Contribution of M_1, M_2, and M_3 in the Cascade

D.1.1 Mini, Medium, and Base

Figure 9 shows the contribution of individual models M_1, M_2, and M_3 in the cascade when accuracy of M_3 is same as that of cascading system. We analyze this for MNLI dataset in section 4.3 and provide figures for a few other datasets here.

D.2 Overall Efficiency and Accuracy Improvement

Figure 10 (left) illustrates efficiency improvements achieved by a cascading method over the largest model (M_3) in K=3 setting. For example, in case of QQP dataset, the cascading system achieves 88.93% computation improvement over M_3 i.e. it requires just 11.07% of the computation cost of model M_3 to attain equal accuracy. Then, we show that cascading also achieves improvement in prediction performance. Figure 10 (right) illustrates the accuracy improvements achieved over M_3 in K=3 setting. For example, on CB dataset, the cascading system achieves 2.18% accuracy improvement over M_3.

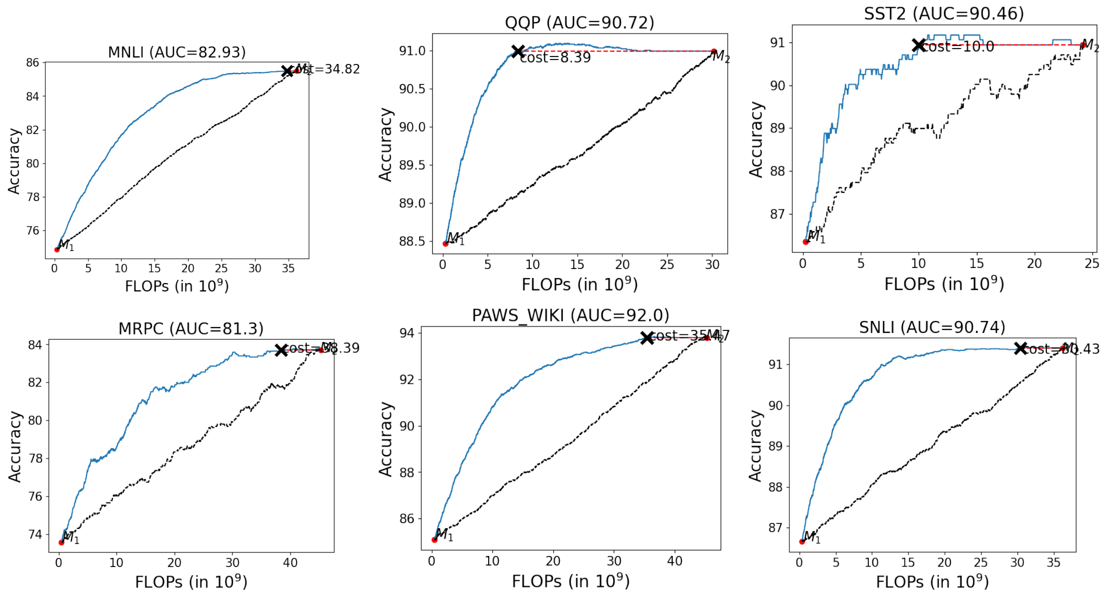

Figure 7: Accuracy-Cost curves for K=2 setting with M_1 as BERT-mini and M_2 as BERT-large models.

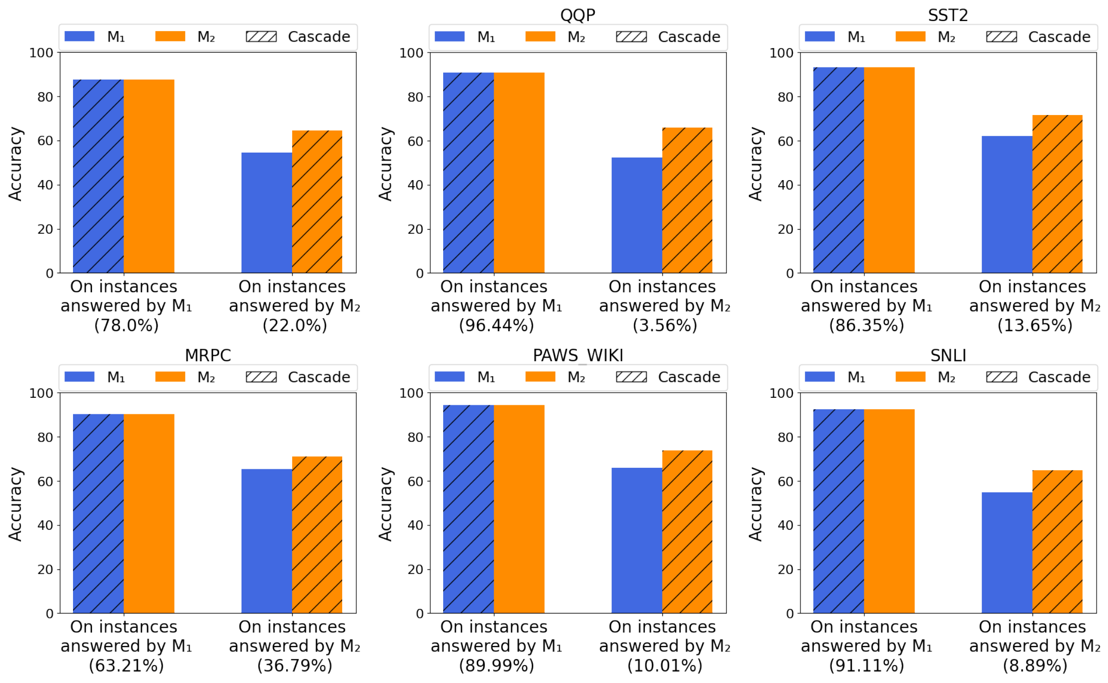

Figure 8: Contribution of M_1 and M_2 for K=2 setting with M_1 as BERT-medium and M_2 as BERT-base.

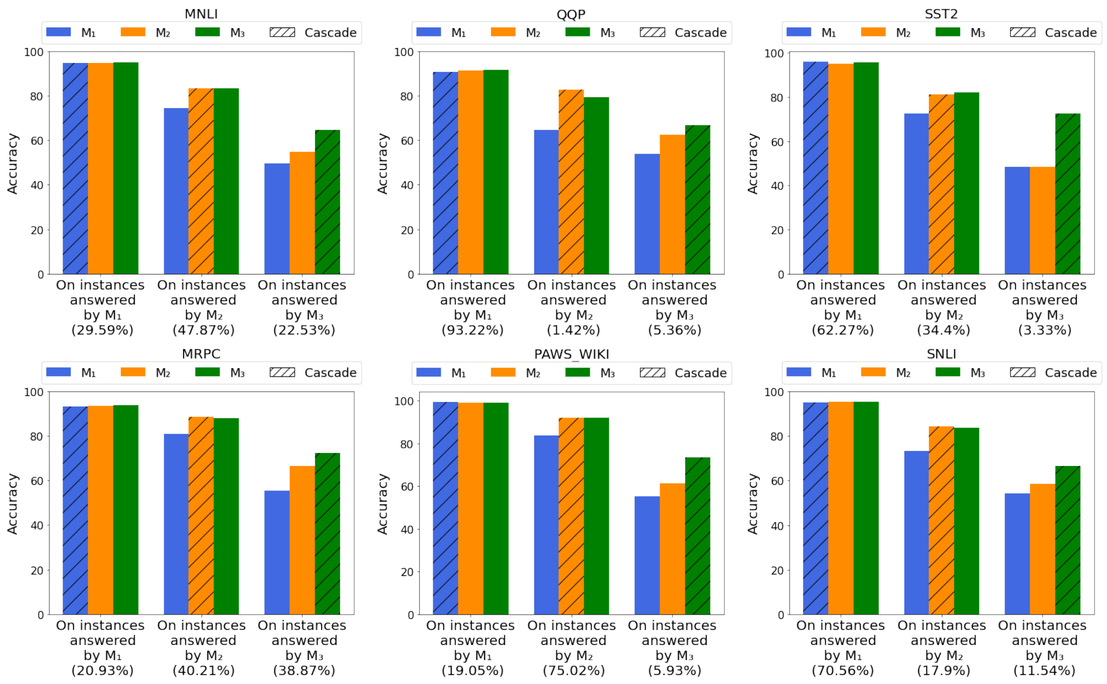

Figure 9: Contribution of M_1, M_2, and M_3 for K=3 setting with M_1 as BERT-mini, M_2 as BERT-medium, and M_3 as BERT-base.

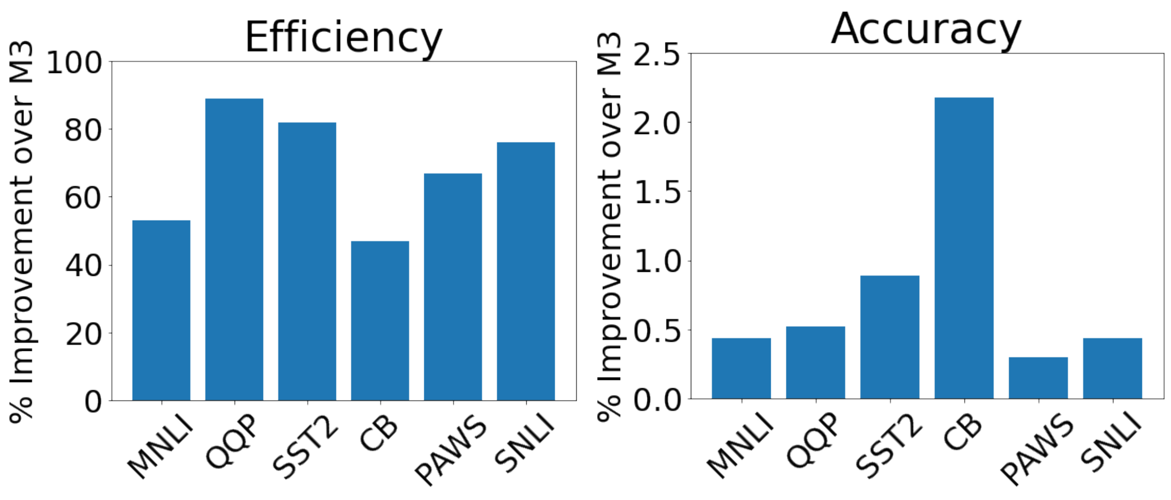

Figure 10: Efficiency and accuracy improvement achieved by the cascading system (using *DTU* method (3.2)) over the largest model M_3 in K=3 setting.