The Right Tool for the Job: Matching Model and Instance Complexities

Roy Schwartz[◇♠] **Gabriel Stanovsky**[◇♠] **Swabha Swayamdipta**[◇]
Jesse Dodge[♣*] **Noah A. Smith**[◇♠]
[◇]Allen Institute for Artificial Intelligence
[♠]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♣]School of Computer Science, Carnegie Mellon University
{roys,gabis,swabhas,jessed,noah}@allenai.org

Abstract

As NLP models become larger, executing a trained model requires significant computational resources incurring monetary and environmental costs. To better respect a given inference budget, we propose a modification to contextual representation fine-tuning which, during inference, allows for an early (and fast) "exit" from neural network calculations for simple instances, and late (and accurate) exit for hard instances. To achieve this, we add classifiers to different layers of BERT and use their calibrated confidence scores to make early exit decisions. We test our proposed modification on five different datasets in two tasks: three text classification datasets and two natural language inference benchmarks. Our method presents a favorable speed/accuracy tradeoff in almost all cases, producing models which are up to five times faster than the state of the art, while preserving their accuracy. Our method also requires almost no additional training resources (in either time or parameters) compared to the baseline BERT model. Finally, our method alleviates the need for costly retraining of multiple models at different levels of efficiency; we allow users to control the inference speed/accuracy tradeoff using a single trained model, by setting a single variable at inference time. We publicly release our code.[1]

1 Introduction

The large increase in the size of artificial intelligence models often increases production costs (Amodei and Hernandez, 2018; Schwartz et al., 2019), and can also limit adoption on real-time devices. Compared to *training*, which is a one-time large investment, *inference* costs are incurred for every instance in production, and can thus add up significantly. For instance, Microsoft reports that using BERT (Devlin et al., 2019) to process Bing queries requires more than 2,000 GPUs concurrently.[2]

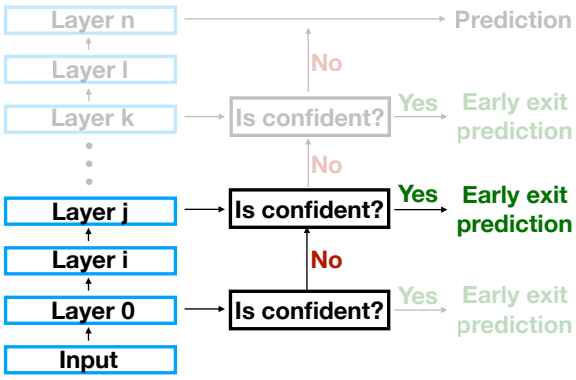


Figure 1: An illustration of our approach. Some layers of a BERT-large model are attached to output classifiers, which make their respective predictions. The confidence of each layer-wise prediction is computed. If high enough, the model takes an early exit, avoiding the computation associated with successive (higher) layers (grayed out). Otherwise, the model continues to the next layer/classifier.

We present a method to reduce the inference cost of today's common models in NLP: fine-tuned contextual word representations. Our method exploits variation along two axes: *models* differ in size and cost, and *instances* vary in difficulty. Our method assesses the complexity of each test instance and matches it with the most efficient model in our "toolbelt."[3] As a result, some instances, which we refer to in this paper as "easy" or "simple," can be solved by small models, leading to computational savings, while other instances (termed "hard" or "difficult") have access to larger models, thus

[*]Research completed during an internship at AI2.
[1]github.com/allenai/sledgehammer
[2]https://tinyurl.com/tzhj3o8
[3]Our approach should not be confused with model ensembles (Kuncheva and Whitaker, 2003), where the prediction of multiple models is combined, *on every instance*, in order to improve accuracy, at the expense of slower inference time.

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 6640–6651
July 5 - 10, 2020.

retaining good performance.

We apply our method to the BERT-large model, modifying its fine-tuning procedure by adding multiple output layers to some of its original $\ell = 24$ layers.[4] A classifier at the kth layer, is more efficient, though (presumably) less accurate than a classifier at a later ℓth layer (where $\ell > k$). At inference time, we run each instance on these classifiers in increasing order of depth. For each classification decision, we use its confidence as an inference-stopping criterion, continuing to the next, larger classifier only if the current classifier is not confident enough in its prediction. Since confidence scores play an important role, we use calibration techniques to make them more reliable. Associating classifiers with different layers of the same network allows us to reuse the computation performed by the simple classifiers for the complex ones. See Figure 1 for an illustration.

We experiment with three text classification benchmarks and two natural language inference (NLI) benchmarks. We consider each of our classifiers with different BERT layers as individual baselines. We find that using our method leads to a consistently better speed/accuracy tradeoff in almost all cases. In particular, in some cases, we obtain similar performance while being as much as five times faster than our strongest baseline (the original BERT-large mode with a single classification layer after the last layer).

Our approach, while allowing substantially faster inference compared to the standard BERT-large model, is neither slower to fine-tune nor significantly larger in terms of parameters, requiring less than 0.005% additional parameters. Moreover, our method is quite flexible: unlike other approaches for inference speed-up such as model distillation or pruning, which require training a different model for each point along the speed/accuracy curve, our method only requires training a single model, and by setting a single variable at inference time—the confidence threshold—supports each point along that curve. Finally, our method is orthogonal to compression methods such as model distillation (Hinton et al., 2014). Our experiments with a distilled version of BERT (Jiao et al., 2019) show that our method further improves the speed/accuracy curve on top of that model. We publicly release our code.[5]

2 Premise: Models Vary in Size, Examples Vary in Complexity

Our goal in this paper is to make model inference more efficient. Our premise relies on two general observations: first, as NLP models become bigger (e.g., in number of parameters), they become both better (in terms of downstream task accuracy), and slower to run. This trend is consistently observed, most notably in recent contextual representations work that compares different variants of the same model (Devlin et al., 2019; Radford et al., 2019; Raffel et al., 2019, *inter alia*).

Second, inputs are not equally difficult. For example, instances differ in length and wealth of linguistic phenomena, which affects the amount of processing required to analyze them. Consider the examples below for the task of sentiment analysis:

(1) The movie was awesome.

(2) I can't help but wonder whether the plot was written by a 12 year-old or by an award-winning writer.

Sentence 1 is short and simple to process. In contrast, Sentence 2 is long, contains misleading positive phrases ("award-winning writer"), and uses figurative speech ("the plot was written by a 12 year-old"). As a result, it is potentially harder to process.[6]

This work leverages these two observations by introducing a method to speed-up inference by matching simple instances with small models, and complex instances with large models.

3 Approach: The Right Tool for the Job

Motivation We assume a series of n trained models $m_1, \ldots, m_n$ for a given task, such that for each $1 < i \leq n$, m_i is both more accurate than m_{i-1} (as measured by a performance on validation data) and more expensive to execute. Current practice in NLP, which favors accuracy rather than efficiency (Schwartz et al., 2019), would typically run m_n on each test instance, as it would likely lead to the highest test score. However, many of the test instances could be solved by simpler (and faster)

[4] For simplicity, we refer to these output layers as classifiers, though our method can also be applied to non-classification tasks.

[5] github.com/allenai/sledgehammer

[6] Note that simplicity is task-dependent. For example, in topic classification, models often accumulate signal across a document, and shorter inputs (with less signal) may be more difficult than longer ones. See Section 6.

models; if we had an oracle that identifies the smallest model that solves a given instance, we could use it to substantially speed up inference. Our goal is to create an automatic measure which approximates the behavior of such an oracle, and identify the cheapest accurate model for each instance.

BERT-large To demonstrate our approach, we consider the BERT-large model (Devlin et al., 2019), based on a transformer architecture (Vaswani et al., 2017) with 24 layers. To apply BERT-large to some downstream task, an output layer is typically added to the final layer of the model, and the model is fine-tuned on training data for that task. To make a prediction using the classifier on the final layer, the computation goes through all the layers sequentially, requiring more computation than a shallower model with fewer layers, which would suffice in some cases.

Suite of models Our approach leverages BERT's multilayered structure by adding an output layer to intermediate layers of the model. For $k < \ell$, the output layer after k BERT layers exits the model earlier than a deeper output layer ℓ, and therefore yields a more efficient (but potentially less accurate) prediction.

Confidence scores for early exit decisions To make early exit decisions, we calculate the layer-wise BERT representations sequentially. As we reach a classification layer, we use it to make predictions. We interpret the label scores output by softmax as *confidence scores*. We use these confidence scores to decide whether to exit early or continue to the next (more expensive and more accurate) classifier. See Figure 1 for an illustration.

Training details To train the model, we use the standard way of applying BERT to downstream tasks—fine-tuning the pre-trained weights, while learning the weights of the randomly initialized classifier, where here we learn multiple classifiers instead of one. As our loss function, we sum the losses of all classification layers, such that lower layers are trained to both be useful as feature generators for the higher layers, and as input to their respective classifiers. This also means that every output layer is trained to perform well on all instances. Importantly, we do not perform early exits during training, but only during inference.

To encourage monotonicity in performance of the different classifiers, each classifier at layer k is given as input a weighted sum of all the layers up to and including k, such that the weight is learned during fine-tuning (Peters et al., 2018).[7]

Calibration Classifiers' confidence scores are not always reliable (Jiang et al., 2018). One way to mitigate this concern is to use calibration, which encourages the confidence level to correspond to the probability that the model is correct (DeGroot and Fienberg, 1983). In this paper we use temperature calibration, which is a simple technique that has been shown to work well in practice (Guo et al., 2017), in particular for BERT fine-tuning (Desai and Durrett, 2020). The method learns a single parameter, denoted *temperature* or T, and divides each of the logits $\{z_i\}$ by T before applying the softmax function:

$$\text{pred} = \arg\max_i \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

We select T to maximize the log-likelihood of the development dataset. Note that temperature calibration is monotonic and thus does not influence predictions. It is only used in our model to make early-exit decisions.

Discussion Our approach has several attractive properties. First, if m_i is not sufficiently confident in its prediction, we reuse the computation and continue towards m_{i+1} without recomputing the BERT layers up to m_i. Second, while our model is larger in terms of parameters compared to the standard approach due to the additional classification layers, this difference is marginal compared to the total number of trainable parameters: our experiments used 4 linear output layers instead of 1, which results in an increase of 6K (binary classification) to 12K (4-way classification) parameters. For the BERT-large model with 335M trainable parameters, this is less than 0.005% of the parameters. Third, as our experiments show (Section 5), while presenting a much better inference time/accuracy tradeoff, fine-tuning our model is as fast as fine-tuning the standard model with a single output layer. Moreover, our model allows for controlling this tradeoff by setting the confidence threshold at inference time, allowing users to better utilize the model for their inference budget.

[7] We also considered feeding the output of previous classifiers as additional features to subsequent classifiers, known as stacking (Wolpert, 1992). Preliminary experiments did not yield any benefits, so we did not further pursue this direction.

Name	#labels	Train	Val.	Test
AG	4	115K	5K	7.6K
IMDB	2	20K	5K	25K
SST	2	7K	0.9K	1.8K
SNLI	3	550K	10K	10K
MNLI	3	393K	9.8K	9.8K

Table 1: Number of labels and instances for the datasets in our experiments. The top set are text classification datasets, and the bottom set are NLI datasets.

4 Experiments

To test our approach, we experiment with three text classification and two natural language inference (NLI) tasks in English. NLI is a pairwise sentence classification task, where the goal is to predict whether a hypothesis sentence entails, contradicts or is neutral to a premise sentence (Dagan et al., 2005). Below we describe our datasets, our baselines, and our experimental setup.

Datasets For text classification, we experiment with the AG news topic identification dataset (Zhang et al., 2015) and two sentiment analysis datasets: IMDB (Maas et al., 2011) and the binary Stanford sentiment treebank (SST; Socher et al., 2013).[8] For NLI, we experiment with the SNLI (Bowman et al., 2015) and MultiNLI (MNLI; Williams et al., 2018) datasets. We use the standard train-development-test splits for all datasets except for MNLI, for which there is no public test set. As MNLI contains two validation sets (matched and mismatched), we use the matched validation set as our validation set and the mismatched validation set as our test set. See Table 1 for dataset statistics.

Baselines We use two types of baselines: running BERT-large in the standard way, with a single output layer on top of the last layer, and three efficient baselines of increasing size (Figure 2). Each is a fine-tuned BERT model with a single output layer after some intermediate layer. Importantly, these baselines offer a speed/accuracy tradeoff, but not within a single model like our approach.

As all baselines have a single output layer, they all have a single loss term, such that BERT layers $1, \ldots, k$ only focus on a single classification layer, rather than multiple ones as in our approach. As with our model, the single output layer in each of our baselines is given as input a learned weighted sum of all BERT layers up to the current layer.

As an upper bound to our approach, we consider a variant of our model that uses the *exact* amount of computation required to solve a given instance. It does so by replacing the confidence-based early-exit decision function with an oracle that returns the fastest classifier that is able to solve that instance, or the fastest classifier for instances that are not correctly solved by any of the classifiers.

Experimental setup We experiment with BERT-large-uncased (24 layers). We add output layers to four layers: 0, 4, 12 and 23.[9] We use the first three layer indices for our efficient baselines (the last one corresponds to our standard baseline). See Appendix A for implementation details.

For training, we use the largest batch size that fits in our GPU memory for each dataset, for both our baselines and our model. Our approach relies on discrete early-exit decisions that might differ between instances in a batch. For the sake of simplicity, we use a batch size of 1 during inference. This is useful for production setups where instances arrive one by one. Larger batch sizes can be applied using methods such as budgeted batch classification (Huang et al., 2018), which specify a budget for the batch and select a subset of the instances to fit that budget, while performing early exit for the rest of the instances. We defer the technical implementation of this idea to future work.

To measure efficiency, we compute the average runtime of a single instance, across the test set. We repeat each validation and test experiment five times and report the mean and standard deviation.

At prediction time, our method takes as an input a threshold between 0 and 1, which is applied to each confidence score to decide whether to exit early. Lower thresholds result in earlier exits, with 0 implying the most efficient classifier is always used. A threshold of 1 always uses the most expensive and accurate classifier.

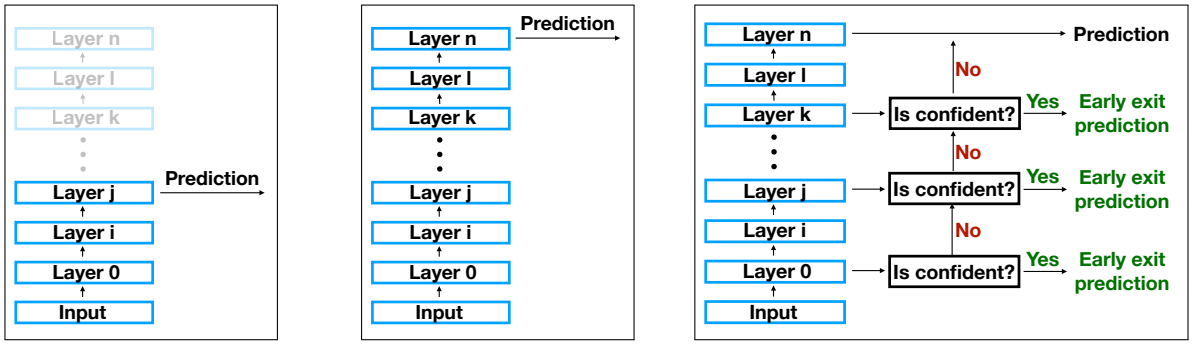
5 Results

A better speed/accuracy tradeoff. Figure 3 presents our test results.[10] The blue line shows our model, where each point corresponds to an increasingly large confidence threshold. The leftmost

[8] For SST, we only used full sentences, not phrases.

[9] Preliminary experiments with other configurations, including ones with more layers, led to similar results.

[10] For increased reproduciblity (Dodge et al., 2019a), we also report validation results in Appendix B.



(a) Efficient Baseline (b) Standard Baseline (c) Our approach

Figure 2: Illustration of our baselines. (2a) Efficient baseline: adding a single output layer to an intermediate layer, while not processing the remaining BERT layers. (2b) The standard model: adding a single output layer to the final BERT layer. (2c) Our approach: adding multiple output layers to intermediate BERT layers; running the corresponding classifiers sequentially, while taking early exits based on their confidence scores.

(rightmost) point is threshold 0 (1), with x-value showing the fraction of processing time relative to the standard baseline.

Our first observation is that our efficient baselines constitute a fast alternative to the standard BERT-large model. On AG, a classifier trained on layer 12 of BERT-large is 40% faster and within 0.5% of the standard model. On SNLI and IMDB a similar speedup results in 2% loss in performance.

Most notably, our approach presents a similar or better tradeoff in almost all cases. Our model is within 0.5% of the standard model while being 40% (IMDB) and 80% (AG) faster. For SST, our curve is strictly above two of the efficient baselines, while being below the standard one. In the two NLI datasets, our curve is slightly above the curve for the medium budgets, and below it for lower ones.

Finally, the results of the oracle baseline indicate the further potential of our approach: in all cases, the oracle outperforms the original baseline by 1.8% (AG) to 6.9% (MNLI), while being 4–6 times faster. These results motivate further exploration of better early-exit criteria (see Section 6). They also highlight the diversity of the different classifiers. One might expect that the set of correct predictions by the smaller classifiers will be contained in the corresponding sets of the larger classifiers. The large differences between the original baseline and our oracle indicate that this is not the case, and motivate future research on efficient ensemble methods which reuse much of the computation across different models.

Extreme case analysis Our results hint that combining the loss terms of each of our classifiers hurts their performance compared to our baselines, which use a single loss term. For the leftmost point in our graphs—always selecting the most efficient classifier—we observe a substantial drop in performance compared to the corresponding most efficient baseline, especially for the NLI datasets. For our rightmost point (always selecting the most accurate classifier), we observe a smaller drop, mostly in SST and MNLI, compared to the corresponding baseline, but also slower runtime, probably due to the overhead of running the earlier classifiers.

These trends further highlight the potential of our method, which is able to outperform the baseline speed-accuracy curves despite the weaker starting point. It also suggests ways to further improve our method by studying more sophisticated methods to combine the loss functions of our classifiers, and encourage them to be as precise as our baselines. We defer this to future work.

Similar training time Fine-tuning BERT-large with our approach has a similar cost to fine-tuning the standard BERT-large model, with a single output layer. Table 2 shows the fine-tuning time of our model and the standard BERT-large baseline. Our model is not slower to fine-tune in four out of five cases, and is even slightly faster in three of them.[11]

This property makes our approach appealing compared to other approaches for reducing runtime such as pruning or model distillation (Section 7). These require, in addition to training the full model, also training another model for each point along the speed/accuracy curve, therefore substantially increasing the overall training time required to gen-

[11] We note that computing the calibration temperature requires additional time, which ranges between 3 minutes (SST) to 24 minutes (MNLI).

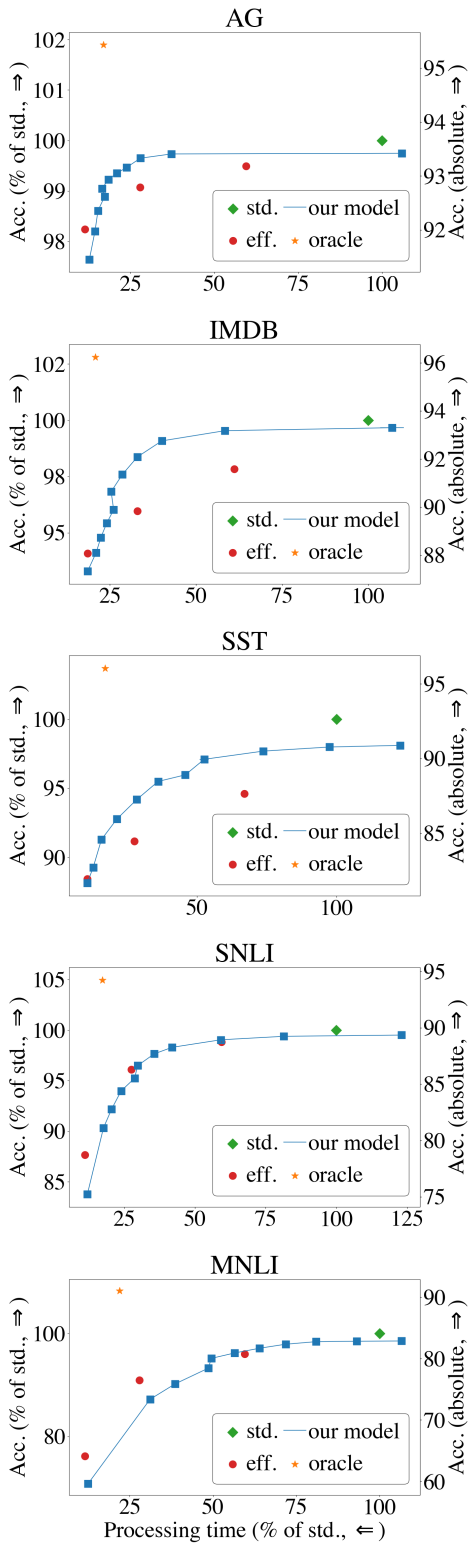


Figure 3: Test accuracy and processing time of our approach (blue squares, each point representing a different confidence threshold), our standard baseline (std., green diamond), efficient baselines (eff., red dots), and oracle baseline (orange star). Left and higher is better. Our method presents similar or better speed/accuracy tradeoff in almost all cases.

Dataset	Training Time	
	Ours	Standard
AG	52	53
IMDB	56	57
SST	4	4
SNLI	289	300
MNLI	852	835

Table 2: Fine-tuning times (in minutes) of our model compared to the most accurate baseline: the standard BERT-large model with a single output layer.

erate a full speed/accuracy tradeoff. In contrast, our single model allows for full control over this tradeoff by adjusting the confidence threshold, without increasing the training time compared to the standard, most accurate model.

Combination with model distillation A key property of our approach is that it can be applied to any multi-layer model. Particularly, it can be combined with other methods for making models more efficient, such as model distillation. To demonstrate this, we repeat our IMDB experiments with tinyBERT (Jiao et al., 2019), which is a distilled version of BERT-base.[12] We experiment with the tinyBERT v2 6-layer-768dim version.[13]

Figure 4 shows our IMDB results. Much like for BERT-large, our method works well for tinyBERT, providing a better speed/accuracy tradeoff compared to the standard tinyBERT baseline and the efficient tinyBERT baselines.

Second, while tinyBERT is a distilled version of BERT-*base*, its speed-accuracy tradeoff is remarkably similar to our BERT-*large* efficient baselines, which hints that our efficient baselines are a simpler alternative to tinyBERT, and as effective for model compression. Finally, our method applied to BERT-large provides the best overall speed-accuracy tradeoff, especially with higher budgets.

6 A Criterion for "Difficulty"

Our approach is motivated by the inherent variance in the level of complexity of text instances, and leverages this variance to obtain a better

[12] While we experimented with BERT-large and not BERT-base, the point of this experiment is to illustrate the potential of our method to be combined with distillation, and not to directly compare to our main results.

[13] Jiao et al. (2019) also suggested a task-specific version of tinyBERT which distills the model based on the downstream task. For consistency with our BERT-large experiments, we use the general version.

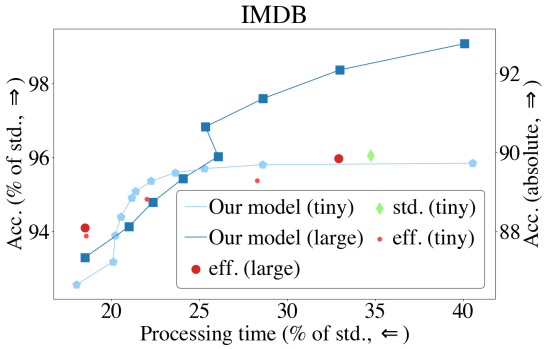


Figure 4: Experiments with tinyBERT. Our method (light-blue pentagons) provides a better speed-accuracy tradeoff compared to the standard (light-green diamonds) and efficient (small light-red dots) baselines. For comparison, we also show the results of our method (blue squares) and our efficient baselines (large red dots) with BERT-large. Our method applied to BERT-large provides the overall best tradeoff.

Dataset	Length	Consistency
AG	0.13	0.37
IMDB	–0.17	0.47
SST	–0.19	0.36
SNLI	–0.08	0.44
MNLI	–0.13	0.39

Table 3: Spearman's ρ correlation between confidence levels for our most efficient classifier and two measures of difficulty: document length and consistency. Confidence is correlated reasonably with consistency across all datasets. For all datasets except AG, confidence is (loosely) negatively correlated with document length. For the AG topic classification dataset, confidence is (loosely) positively correlated. Results for the other layers show a similar trend.

speed/accuracy tradeoff compared to our baselines. Our method also automatically identifies instances on which smaller models are highly confident in their predictions. Here we analyze our data using other definitions of difficulty. Perhaps surprisingly, we find that the various definitions are not strongly correlated with ours. The results we observe below, combined with the performance of our oracle baseline (Section 5), motivate further study on more advanced methods for early exiting, which could potentially yield even larger computational gains.

Shorter is easier? We first consider the length of instances: is our model more confident in its decisions on short documents compared to longer ones? To address this we compute Spearman's ρ correlation between the confidence level of our most efficient classifier and the document's length.

The results in Table 3 show that the correlations across all datasets are generally low ($|\rho| < 0.2$). Moreover, as expected, across four out of five datasets, the (weak) correlation between confidence and length is negative; our model is somewhat more confident in its prediction on shorter documents. The fifth dataset (AG), shows the opposite trend: confidence is positively correlated with length. This discrepancy might be explained by the nature of the tasks we consider. For instance, IMDB and SST are sentiment analysis datasets, where longer texts might include conflicting evidence and thus be harder to classify. In contrast, AG is a news topic detection dataset, where a conflict between topics is uncommon, and longer documents provide more opportunities to find the topic.

Consistency and difficulty Our next criterion for "difficulty" is the consistency of model predictions. Toneva et al. (2019) proposed a notion of "unforgettable" training instances, which once the model has predicted correctly, it never predicts incorrectly for the remainder of training iterations. Such instances can be thought of as "easy" or memorable examples. Similarly, Sakaguchi et al. (2019) defined test instances as "predictable" if multiple simple models predict them correctly. Inspired by these works, we define the criterion of consistency: whether all classifiers in our model agree on the prediction of a given instance, regardless of whether it is correct or not. Table 3 shows Spearman's ρ correlation between the confidence of the most efficient classifier and this measure of consistency. Our analysis reveals a medium correlation between confidence and consistency across all datasets ($0.37 \leq \rho \leq 0.47$), which indicates that the measure of confidence generally agrees with the measure of consistency.

Comparison with hypothesis-only criteria Gururangan et al. (2018) and Poliak et al. (2018) showed that some NLI instances can be solved by only looking at the hypothesis—these were artifacts of the annotation process. They argued that such instances are "easier" for machines, compared to those which required access to the full input, which they considered "harder." Table 4 shows the correlation between the confidence of each of our classifiers on the SNLI and MNLI dataset with the confidence of a hypothesis-only classifier. Simi-

Layer	SNLI		MNLI	
	Hyp.-Only	IAC	Hyp.-Only	IAC
0	0.39	0.14	0.37	0.08
4	0.31	0.25	0.35	0.21
12	0.31	0.31	0.32	0.27
23	0.28	0.32	0.30	0.32

Table 4: Spearman's ρ correlation between confidence levels for our classifiers (of different layers) on the validation sets of SNLI and MNLI, and two measures of difficulty: hypothesis-only classifier predictions (Hyp.-Only) and inter-annotator consensus (IAC).

larly to the consistency results, we see that the confidence of our most efficient classifier is reasonably correlated with the predictions of the hypothesis-only classifier. As expected, as we move to larger, more accurate classifiers, which presumably are able to make successful predictions on harder instances, this correlation decreases.

Inter-annotator consensus Both NLI datasets include labels from five different annotators. We treat the inter-annotator consensus (IAC) as another measure of difficulty: the higher the consensus is, the easier the instance. We compute IAC for each example as the fraction of annotators who agreed on the majority label, hence this number ranges from 0.6 to 1.0 for five annotators. Table 4 shows the correlation between the confidence of our classifiers with the IAC measure on SNLI and MNLI. The correlation with our most efficient classifiers is rather weak, only 0.08 and 0.14. Surprisingly, as we move to larger models, the correlation increases, up to 0.32 for the most accurate classifiers. This indicates that the two measures perhaps capture a different notion of difficulty.

Confidence across labels Figure 5 shows the proportion of instances in our validation set that are predicted with high confidence by our calibrated model (90% threshold) for each dataset, label, and model size. We first note that across all datasets, and almost all model sizes, different labels are not predicted with the same level of confidence. For instance, for AG, the layer 0 model predicts the *tech* label with 87.8% average confidence, compared to 96.8% for the *sports* label. Moreover, in accordance with the overall performance, across almost all datasets and model sizes, the confidence levels increase as the models get bigger in size. Finally, in some cases, as we move towards larger models, the gaps in confidence close (e.g., IMDB and SST), although the relative ordering hardly ever changes.

Two potential explanations come up when observing these results; either some labels are easier to predict than others (and thus the models are more confident when predicting them), or the models are biased towards some classes compared to others. To help differentiate between these two hypotheses, we plot in Figure 6 the average confidence level and the average F_1 score of the most efficient classifier across labels and datasets.

The plot indicates that both hypotheses are correct to some degree. Some labels, such as *sports* for AG and *positive* for IMDB, are both predicted with high confidence, and solved with high accuracy. In contrast, our model is overconfident in its prediction of some labels (*business* for AG, *positive* for SST), and underconfident in others (*tech* for AG, *entailment* for MNLI). These findings might indicate that while our method is designed to be globally calibrated, it is not necessarily calibrated for each label individually. Such observations relate to existing concerns regarding fairness when using calibrated classifiers (Pleiss et al., 2017).

7 Related Work

Methods for making inference more efficient have received considerable attention in NLP over the years (Eisner and Satta, 1999; Goldberg and Elhadad, 2010, *inter alia*). As the field has converged on deep neural architecture solutions, most efforts focus on making models smaller (in terms of model parameters) in order to save space as well as potentially speed up inference.

In *model distillation* (Hinton et al., 2014) a smaller model (the student) is trained to mimic the behavior or structure of the original, larger model (the teacher). The result is typically a student that is as accurate as the teacher, but smaller and faster (Kim and Rush, 2016; Jiao et al., 2019; Tang et al., 2019; Sanh et al., 2019). *Pruning* (LeCun et al., 1990) removes some of the weights in the network, resulting in a smaller, potentially faster network. The basic pruning approach removes individual weights from the network (Swayamdipta et al., 2018; Gale et al., 2019). More sophisticated approaches induce structured sparsity, which removes full blocks (Michel et al., 2019; Voita et al., 2019; Dodge et al., 2019b). Liu et al. (2018) and Fan et al. (2020) pruned deep models by applying dropout to different layers, which allows dynamic control of

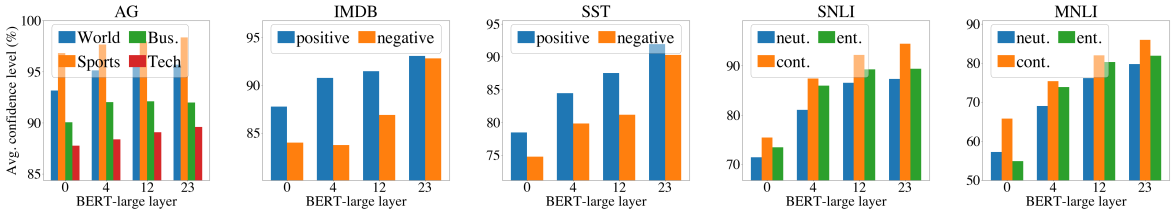


Figure 5: Instances with different labels are predicted with different degrees of confidence.

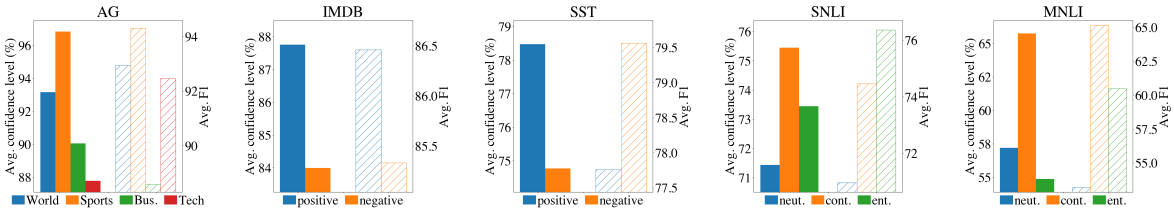


Figure 6: Comparing confidence levels and F_1 scores of our most efficient classifier across datasets and labels. High confidence by the model is sometimes explained by "easy" classes that are predicted with high F_1 (e.g., *sports* in AG). Other cases might stem from biases of the model which make it overconfident despite the label being harder than other labels (e.g., *positive* in SST).

the speed/accuracy tradeoff of the model without retraining. Our method also allows for controlling this tradeoff with a single training pass, and yields computational savings in an orthogonal manner: by making early exit decisions.

Quantization is another popular method to decrease model size, which reduces the numerical precision of the model's weights, and therefore both speeds up numerical operations and reduces model size (Wróbel et al., 2018; Shen et al., 2019; Zafrir et al., 2019).

Some works introduced methods to allocate fewer resources to certain parts of the input (e.g., certain words), thereby potentially reducing training and/or inference time (Graves, 2016; Seo et al., 2018). Our method also puts less resources into some of the input, but does so at the document level rather than for individual tokens.

A few concurrent works have explored similar ideas for dynamic early exits in the transformer model. Elbayad et al. (2020) and Dabre et al. (2020) introduced early stopping for sequence-to-sequence tasks (e.g., machine translation). Bapna et al. (2020) modify the transformer architecture with "control symbols" which determine whether components are short-circuited to optimize budget. Finally, Liu et al. (2020) investigated several inference-time cost optimizations (including early stopping) in a multilingual setting.

Several computer vision works explored similar ideas to the one in this paper. Wang et al. (2018) introduced a method for dynamically skipping convolutional layers. Bolukbasi et al. (2017) and Huang et al. (2018) learned early exit policies for computer vision architectures, observing substantial computational gains.

8 Conclusion

We presented a method that improves the speed/accuracy tradeoff for inference using pre-trained language models. Our method makes early exits for simple instances that require less processing, and thereby avoids running many of the layers of the model. Experiments with BERT-large on five text classification and NLI datasets yield substantially faster inference compared to the standard approach, up to 80% faster while maintaining similar performance. Our approach requires neither additional training time nor significant number of additional parameters compared to the standard approach. It also allows for controlling the speed/accuracy tradeoff using a single model, without retraining it for any point along the curve.

Acknowledgments

The authors thank the members of Noah's ARK at the University of Washington, the researchers at the Allen Institute for AI, and the anonymous reviewers for their valuable feedback.

References

Dario Amodei and Danny Hernandez. 2018. AI and compute. Blog post.

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. 2020. Controlling computation versus quality for neural sequence models. arXiv:2002.07106.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. In *Proc. of ICML*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proc. of EMNLP*.

Raj Dabre, Raphael Rubino, and Atsushi Fujita. 2020. Balancing cost and benefit with tied-multi transformers. arXiv:2002.08614.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Proc. of MLCW*.

Morris H. DeGroot and Stephen E. Fienberg. 1983. The comparison and evaluation of forecasters. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 32(1-2):12–22.

Shrey Desai and Greg Durrett. 2020. Calibration of pre-trained transformers. arXiv:2003.07892.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL*.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. 2019a. Show your work: Improved reporting of experimental results. In *Proc. of EMNLP*.

Jesse Dodge, Roy Schwartz, Hao Peng, and Noah A. Smith. 2019b. RNN architecture learning with sparse regularization. In *Proc. of EMNLP*.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. of ACL*.

Maha Elbayad, , Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *Proc. of ICLR*.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *Proc. of ICLR*.

Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The state of sparsity in deep neural networks. arXiv:1902.09574.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proc. of NLP-OSS*.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Proc. of NAACL*.

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

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. In *Proc. of ICML*.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. In *Proc. of NAACL*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2014. Distilling the knowledge in a neural network. In *Proc. of NeurIPS Deep Learning Workshop*.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. 2018. Multi-scale dense networks for resource efficient image classification. In *Proc. of ICLR*.

Heinrich Jiang, Been Kim, Melody Y. Guan, and Maya Gupta. 2018. To trust or not to trust a classifier. In *Proc. of NeurIPS*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TinyBERT: Distilling BERT for natural language understanding. arXiv:1909.10351.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proc. of EMNLP*.

Ludmila I. Kuncheva and Christopher J. Whitaker. 2003. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207.

Yann LeCun, John S. Denker, and Sara A. Solla. 1990. Optimal brain damage. In *Proc. of NeurIPS*.

Liyuan Liu, Xiang Ren, Jingbo Shang, Xiaotao Gu, Jian Peng, and Jiawei Han. 2018. Efficient contextualized representation: Language model pruning for sequence labeling. In *Proc. of EMNLP*.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. 2020. FastBERT: a self-distilling BERT with adaptive inference time. In *Proc. of ACL*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proc. of ACL*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Proc. of NeurIPS*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q. Weinberger. 2017. On fairness and calibration. In *Proc. of NeurIPS*.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. 2018. Hypothesis Only Baselines in Natural Language Inference. In *Proc. of *SEM*.

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

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

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. WinoGrande: An adversarial winograd schema challenge at scale. arXiv:1907.10641.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *Proc. of EMC2*.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2019. Green AI. arXiv:1907.10597.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Neural speed reading via skim-RNN. In *Proc. of ICLR*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. Q-BERT: Hessian based ultra low precision quantization of BERT. arXiv:1909.05840.

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 *Proc. of EMNLP*.

Swabha Swayamdipta, Ankur P. Parikh, and Tom Kwiatkowski. 2018. Multi-mention learning for reading comprehension with neural cascades. In *Proc. of ICLR*.

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:1903.12136.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. 2019. An empirical study of example forgetting during deep neural network learning. In *Proc. of ICLR*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of NeurIPS*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proc. of ACL*.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018. SkipNet: Learning dynamic routing in convolutional networks. In *Proc. of ECCV*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proc. of NAACL*.

David H. Wolpert. 1992. Stacked generalization. *Neural Networks*, 5:241–259.

Krzysztof Wróbel, Marcin Pietroń, Maciej Wielgosz, Michał Karwatowski, and Kazimierz Wiatr. 2018. Convolutional neural network compression for natural language processing. arXiv:1805.10796.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit BERT. In *Proc. of EMC2*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proc. of NeurIPS*.

A Implementation Details

We fine-tune both our model and our baselines with dropout 0.1. We run all our experiments on a single Quadro RTX 8000 GPU. Our model is implement using the AllenNLP library (Gardner et al., 2018).[14] Our calibration code relies on the implementation of Guo et al. (2017).[15]

We fine-tune text classification models for 2 epochs and NLI models for 4 epochs. We run ten trials of random search on the validation set for both our model and our baselines to select both a learning rate among {0.00002, 0.00003, 0.00005} and a random seed. For our baselines, we select the highest performing model on the validation set among the ten runs. For our model, we select the one with the highest performance averaged across all thresholds explored (we use 0% and 5% intervals in the range [55%, 100%]) on the validation set.

B Validation Results

Figure 7 shows the validation results of our experiments.

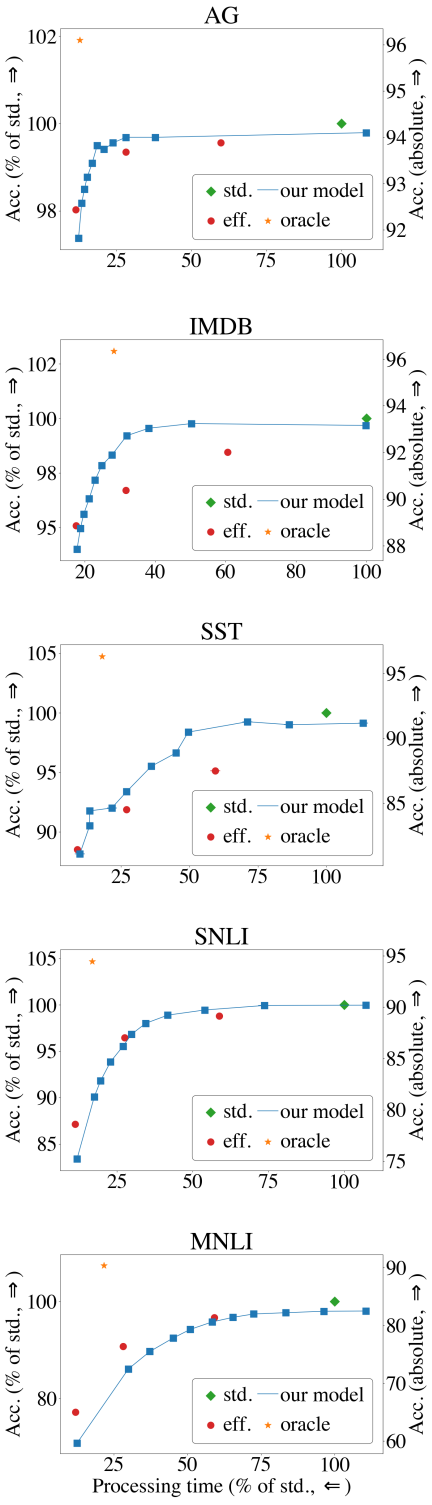


Figure 7: Validation accuracy and processing time of our approach (blue line) and our standard baseline (std., green diamond), our efficient baselines (eff., red dots) and our oracle (orange star). Left and higher is better.

[14] https://allennlp.org

[15] https://github.com/gpleiss/temperature_scaling