E-LANG: Energy-Based Joint Inferencing of Super and Swift Language Models

Mohammad Akbari, Amin Banitalebi-Dehkordi, Yong Zhang
Huawei Technologies Canada Co., Ltd.
{mohammad.akbari, amin.banitalebi, yong.zhang3}@huawei.com

Abstract

Building huge and highly capable language models has been a trend in the past years. Despite their great performance, they incur high computational cost. A common solution is to apply model compression or choose light-weight architectures, which often need a separate fixed-size model for each desirable computational budget, and may lose performance in case of heavy compression. This paper proposes an effective dynamic inference approach, called E-LANG, which distributes the inference between large accurate Super-models and light-weight Swift models. To this end, a decision making module routes the inputs to Super or Swift models based on the energy characteristics of the representations in the latent space. This method is easily adoptable and architecture agnostic. As such, it can be applied to black-box pre-trained models without a need for architectural manipulations, reassembling of modules, or re-training. Unlike existing methods that are only applicable to encoder-only backbones and classification tasks, our method also works for encoder-decoder structures and sequence-to-sequence tasks such as translation. The E-LANG performance is verified through a set of experiments with T5 and BERT backbones on GLUE, SuperGLUE, and WMT. In particular, we outperform T5-11B with an average computations speed-up of 3.3× on GLUE and 2.9× on SuperGLUE. We also achieve BERT-based SOTA on GLUE with 3.2× less computations. Code and demo are available here.

1 Introduction

With the introduction of influential language models such as BERT (Devlin et al., 2019), a trend in natural language processing (NLP) research has been to develop high capacity models and push their performance to new levels. Consequently, state-of-the-art (SOTA) results were achieved on various benchmarks using these models; GPT-3 (Brown et al., 2020), XLNet (Yang et al., 2019), RoBERTa (Liu et al., 2019), T5 (Raffel et al., 2020), ELECTRA (Clark et al., 2020), and DeBERTa (He et al., 2021) to name a few. A potential down-side, however, is that the number of parameters or floating point operations (FLOPs) for these models can get extremely large. For example, Gshard (Lepikhin et al., 2021) comes with 600B parameters with an enormous amount of computation. This in turn results in a higher inference latency, which is not desirable for latency-sensitive applications.

A common solution to speed-up the large language models is to apply model compression (Gupta et al., 2020). Although generally successful, compression does come with a trade-off on accuracy, and may lose performance if compression is heavy. In addition, these methods usually compress a model to a fixed smaller size, where a separate model is required for each possible computational budget. An alternative approach explored in the literature is to leverage dynamic inferencing in a way that examples may be routed to different (potentially lower cost) paths throughout the network. For example, a temporal early-exit model (Shen et al., 2017; Yu et al., 2018) terminates the procedure of reading the input sequence when sufficient evidence has been found for accurate predictions. Instance-wise early-exiting (Xin et al., ACL 2020) is another technique, which allows a sample to adaptively choose from multiple available exit nodes if some conditions are met. Consequently, earlier exists require less computation and lead to a lower latency. Adjusting the size of the model at the inference time by choosing adaptive width and depth is also another approach employed for dynamic inference (Kim and Cho, 2021; Hou et al., 2020). There is a variety of adaptive/dynamic inference approaches proposed, however, a general down-side for many of these methods is that often times they require a careful architecture design, manipulation of network modules, or even re-training.

In this paper, we propose a simple but rather effective approach of dynamically distributing the inference between the original large model (called the **Super** model) and a light-weight (e.g., compressed) model referred to as the **Swift** model. To this end, we design an energy-based decision making module that routes examples to the appropriate model based on the negative free energy of the latent space representations, such that the Swift model attains a high accuracy on the examples sent to it. The remaining samples are then forwarded to the Super model that is supposed to have a good performance on all examples. Since the Swift model can make highly accurate predictions over the majority of the samples, E-LANG significantly reduces the overall computational cost, while maintains the high accuracy of the Super model. Although simple, this strategy achieves SOTA results on multiple structures (e.g., T5 and BERT) and benchmarks (e.g., GLUE and SuperGLUE). Due to its desirable practical characteristics, this method is a strong candidate for the practical application of Super models. The main contributions of the paper are as follows:

- Combining Super models with high accuracy and latency and Swift models with lower accuracy and latency, to achieve **high accuracy and low latency**. In other words, by employing our method, we can achieve the high levels of accuracy provided by Super models, but at a lower computational cost. Our method is easily adoptable, architecture agnostic, and orthogonal to many other existing methods. It can be applied to black-box pre-trained models without a need for architectural manipulations, careful reassembling of modules, or re-training.

- An **energy-based routing mechanism** for directing examples to the Super or Swift. This provides a dynamic trade-off between the accuracy and computational cost that outperforms the previous works in both fixed-size and dynamic inference (with zero overhead for real-time adjustment of speed/accuracy). As such, E-LANG acts like a knob for adjusting the accuracy-latency trade-off in real-time during model serving.

- To the best of our knowledge, our method is the first generic approach to apply dynamic inference on both encoder-only and **encoder-decoder architectures** (e.g., T5) and also can extend the usage beyond classification tasks, to **sequence-to-sequence** tasks such as translation.

2 Related Works

As mentioned, compression is a widely used strategy to speed-up the large language models (Gupta et al., 2020; Gupta and Agrawal, 2022). This involves incorporating techniques such as quantization of weights and activations (Bai et al., 2021; Shen et al., 2020; Kim et al., 2021; Zhang et al., 2020; Jin et al., 2021), knowledge distillation (KD) (Hinton et al., 2015; Jiao et al., 2020; Sanh et al., 2019), pruning/sharing (Gordon et al., 2020; Chen et al., 2020), multi-device distribution (Banitalebi-Dehkordi et al., 2021), or a combination of these techniques (Cheng et al., 2017; Polino et al., 2018).

Among all the compression techniques, creating a fixed-size small version of large models along with distillation has been popular in the recent years. Sanh et al. (2019) introduced DistillBERT, which was a smaller version of BERT trained with distillation for general purposes. Another compact variant of BERT was proposed by MobileBERT (Sun et al., 2020) in which inverted bottleneck structures and progressive knowledge transfer were used. TinyBERT (Jiao et al., 2020) also presented a novel two-stage transformer distillation for both pre-training and task-specific fine-tuning. In (Iandola et al., 2020), the usage of grouped convolutions was studied to design SqueezeBERT. ELM (Jiao et al., 2021), a layer mapping search framework, was also proposed for improving downstream BERT distillation. A recent method, GhostBERT (Huang et al., 2021), employed softmax-normalized 1D convolutions as ghost modules to generate more features with cheap operations.

Although compression techniques in general are effective, they come with a trade-off on accuracy, and may lose performance in case of high ratio compression. In addition, an individual fixed-size model is required for each possible computational budget. As stated in the introduction, the alternative solution is dynamic inference, which can be achieved with either early-exit or length/depth-adaptive models. One of the first temporal early-exit strategies was proposed by ReasoNet (Shen et al., 2017), which stops its reading procedure when sufficient evidence has been found for answering a question. Similarly, in (Yu et al., 2018), an early stopping method applicable to classification tasks was presented. DeeBERT (Xin et al., ACL 2020) also proposed an instance-wise multi-exit method via the entropy of the output probability distribution to speed-up BERT inference.

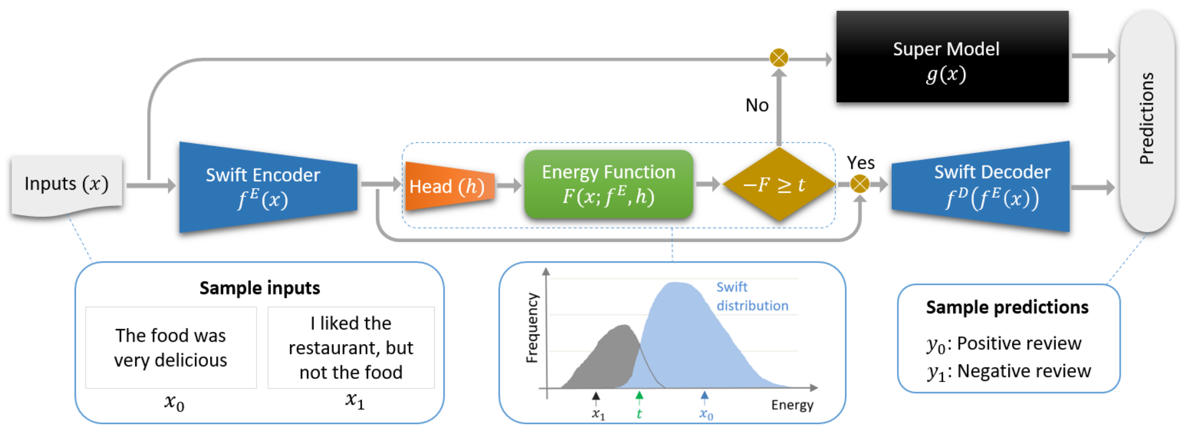


Figure 1: Overall framework of the proposed energy-based joint inference strategy (E-LANG).

As a length-adaptive method, Kim and Cho (2021) introduced a dynamic inference framework with one-shot training of transformers for both sequence- and token-level classification. Also, in (Hou et al., 2020), an architecture named DynaBERT was proposed for adaptively adjusting the computations by choosing sub-networks of different widths and depths. Both Length-Adaptive and DynaBERT utilized knowledge distillation and data augmentation to improve their performance.

Although early-exit and adaptive methods have made significant progress and work well in practice, they often require architectural manipulation and re-training. In addition, they are only applicable to encoder-only backbones and classification tasks. In contrast, our method can work with out-of-the-box pre-trained models without a need for re-training and are also applicable for encoder-decoder structures and sequence-to-sequence tasks.

3 Proposed Method

We propose a new energy-based joint inference method called E-LANG, where a large/accurate language model (**Super**) is jointly employed with a small/fast one (**Swift**) to achieve efficient inference without sacrificing the accuracy. To this end, inspired by the method in (Akbari et al., 2021), a routing mechanism empowered by energy-based models (EBM) is introduced to dynamically distribute the input samples between the Super and Swift models. Similar to the out-of-distribution (OOD) detection problem, our goal is to identify the OOD samples that are hard to handle for the Swift and forward them to the Super model. On the other hand, we have the in-distribution data for which the Swift can make highly reliable and accurate predictions. In other words, the routing mechanism needs to detect whether or not the input data fits in the Swift's distribution (i.e., the one the Swift has been trained with). Inspired by the success of EBMs in dealing with OOD detection problems (Lee et al., 2019), the energy characteristics of data samples for an efficient and effective routing are investigated in our work. The overall framework of E-LANG is shown in Figure 1.

3.1 Energy-Based Models

The goal of EBM is to build an energy function denoted by $E(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}$ that maps an input data $\mathbf{x} \in \mathbb{R}^D$ to a non-probabilistic energy value $y \in \mathbb{R}$. To turn a collection of arbitrary energies for all possible outputs (denoted by Y) into a normalized probability distribution, Gibbs distribution can be used as follows (LeCun et al., 2006):

$$p(y|\mathbf{x}) = \frac{e^{-E(\mathbf{x},y)}}{\int_{y' \in Y} e^{-E(\mathbf{x},y')}}, \tag{1}$$

where the negative log of the denominator expresses the Helmholtz free energy (LeCun et al., 2006) defined as $F(\mathbf{x}) = -log\big(\int_{y' \in Y} e^{-E(\mathbf{x},y')}\big)$.

In machine learning, there is a deep relationship between the EBMs and discriminative models, which can be seen by connecting the Gibbs distribution in Equation (1) and the categorical distribution derived for a discriminative model. A discriminative classifier is defined as a function for mapping the input $\mathbf{x}$ to C real-valued logits (i.e., for C number of class labels): $f(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^C$. In order to derive a categorical distribution over C possible outputs, the softmax function is utilized:

$$p(y|\mathbf{x}) = \frac{e^{f_y(\mathbf{x})}}{\sum_i^C e^{f_i(\mathbf{x})}}, \tag{2}$$

where $f_y(\mathbf{x})$ denotes the logit (probability) of the yth class label. Based on the inherent connection between the Gibbs and categorical distributions defined in (1) and (2), the energy function for a given input $(\mathbf{x}, y)$ can be defined as $E(\mathbf{x}, y) = -f_y(\mathbf{x})$. The free energy function $F(\mathbf{x}; f)$ can then be obtained by taking the negative log of the categorical distribution denominator as:

$$F(\mathbf{x}; f) = -log \sum_{i}^{C} e^{f_i(\mathbf{x})}. \quad (3)$$

3.2 Energy-Based Joint Inference

Our goal is to detect the easy samples suitable for the Swift, which are indeed the ones with high likelihood in the density function. The energy-based density function for Swift is then defined as:

$$p(\mathbf{x}) = \frac{e^{-F(\mathbf{x};f)}}{\int_{\mathbf{x}} e^{-F(\mathbf{x};f)}}, \quad (4)$$

where the denominator is the normalized densities, which can be intractable to compute or estimate. By taking the logarithm of both sides, we obtain:

$$log\big(p(\mathbf{x})\big) = -F(\mathbf{x}; f) - log(\int_{\mathbf{x}} e^{-F(\mathbf{x};f)}). \quad (5)$$

The $log(\int_{\mathbf{x}} e^{-F(\mathbf{x};f)})$ term has no effect on the distribution of the overall energy values because it is constant for all $\mathbf{x}$. As a result, $-F(\mathbf{x}; f)$, i.e., the negative free energy, has a linear alignment with the log likelihood function, which makes it a well-suited solution to the easy vs. hard detection problem in our framework. To this end, lower energy values indicate higher likelihood and represent easier (more fit) samples for the Swift model.

More precisely, for a threshold δ on the density function such that $p(\mathbf{x}) < \delta$, then a threshold t on the negative free energy can be calculated according to (5) as $-F(\mathbf{x}; f) < t = log(\delta \int_{\mathbf{x}} e^{-F(\mathbf{x};f)})$. In practice, for a given input, an energy function is applied to the outputs of the Swift model during inference time to calculate the energy score. Then, if the negative energy value is smaller than a threshold, the input is identified as a bad sample for the Swift, and is sent to the Super model.

Given the energy threshold t, the Swift classifier $f(\mathbf{x})$, and the Super classifier defined as $g(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}^C$, the joint inference function $J(\mathbf{x}; f, g, t) \in [1, C]$ for a classification task with C classes can then be expressed by:

$$J(\mathbf{x}; f, g, t) = \begin{cases} f(\mathbf{x}) & \text{if } -F(\mathbf{x}; f) \geq t \\ g(\mathbf{x}) & \text{otherwise.} \end{cases} \quad (6)$$

3.2.1 Encoder-Decoder Architectures

The proposed energy-based joint inference solution can be directly applied to the encoder-only models such as BERT that are designed for text classification tasks. To this end, the energy scores corresponding to the BERT-based Swift model are obtained using Equation (3) and the joint inference is performed based on Equation 6.

On the other hand, for the encoder-decoder (auto-encoder) architectures such as T5, which are usually considered as generative models, some modifications are required. Encoder-decoder models are basically designed for sequence-to-sequence (e.g., text-to-text) problems such as translation or summarization. Although such models can also be employed for classification tasks, they still consider the task as a text generation (sequence-to-sequence) problem, where the target labels and the output predictions are treated as a sequence or a piece of text.

In Section 3.1, it was discussed that there is an inherent connection between the discriminative classifiers and the EBMs. In order to benefit from this characteristic for encoder-decoder architectures, we consider adding an extra classification head (i.e., a single linear layer) to the Swift model. As encoders are commonly considered as better feature extractors for training a classifier rather than the decoders, we place the extra head after the Swift encoder. While freezing the pre-trained encoder model (denoted by f^E), the extra energy head (denoted by h) is trained as a regular classifier head with C class labels. Note that the decoder is not required for training the head. The corresponding free energy function is then defined as follows:

$$F(\mathbf{x}; f^E, h) = -log \sum_{i}^{C} e^{h_i\left(f^E(x)\right)}, \quad (7)$$

where $f^E(x)$ denotes the outputs of the encoder's last hidden state. These features are then fed to the extra head h to obtain the logits for the ith class required for computing the energy scores.

In this approach, as the decoder part of the Swift model is not required for calculating the energy scores, less computations are involved and the joint inference is performed more efficiently.

For text-to-text (or sequence-to-sequence) problems such as translation, the output is a sequence of M word-pieces from a vocabulary/dictionary of size N. To still utilize the relationship of discriminative models and EBMs in designing and training the extra energy head, we can treat the text-to-text models as M multi-class classifiers. In this case, the number of class labels, i.e., C in (7), is equal to N. The final energy score is then calculated as the average of M energy values as follows:

$$F(\mathbf{x}; f^E, h) = -\frac{1}{M}\sum_m^M \Big(log \sum_i^C e^{h_{m,i}\left(f^E(x)\right)}\Big), \tag{8}$$

where $h_{m,i}(.)$ denotes the logits corresponding to the mth word in the sequence and ith class label.

Denote the Swift's decoder by f^D, the joint inference function, $J(\mathbf{x}; f, g, h, t)$, based on energy scores in either Equation (7) or (8) is expressed as:

$$J = \begin{cases} f^D\big(f^E(x)\big) & \text{if } -F(\mathbf{x}; f^E, h) \geq t \\ g(\mathbf{x}) & \text{otherwise.} \end{cases} \tag{9}$$

3.3 Softmax and Entropy Mechanisms

In addition to energy, softmax and entropy (Xin et al., ACL 2020) scores can also be used for analyzing the Swift model's performance in the routing mechanism. In this sub-section, we study the mathematical connection of them with the energy score and their potential to solve our problem.

3.3.1 Softmax-Based Mechanism

The softmax score for a classifier is expressed by:

$$\max_y p(y|\mathbf{x}) = \max_y \frac{e^{f_y(\mathbf{x})}}{\sum_i^C e^{f_i(\mathbf{x})}} = \frac{e^{f_{max}(\mathbf{x})}}{\sum_i^C e^{f_i(\mathbf{x})}}. \tag{10}$$

By taking the logarithm of both sides, we see the connection between the log of the softmax and the free energy score formulated in Equation (3):

$$\begin{aligned} log \max_y p(y|\mathbf{x}) &= log(e^{f_{max}(\mathbf{x})}) - log \sum_i^C e^{f_i(\mathbf{x})} \\ &= f_{max}(\mathbf{x}) + F(\mathbf{x}; f), \end{aligned} \tag{11}$$

where all logits are shifted by their maximum $f_{max}(x)$. Plugging in the energy term to (5) yields:

$$\begin{aligned} log \max_y p(y|\mathbf{x}) &= -log(p(\mathbf{x})) + f_{max}(\mathbf{x}) \\ &\quad -log\Big(\int_{\mathbf{x}} e^{-F(\mathbf{x};f)}\Big). \end{aligned} \tag{12}$$

It is observed that for the samples with high likelihood of being in the Swift's distribution, the free energy goes lower, but the max logit tends to go higher. Due to this shifting, unlike the energy score, the softmax score is not well-aligned with the probability density $p(\mathbf{x})$. As a result, the softmax score is less reliable for our routing module to analyze the performance of the Swift.

3.3.2 Entropy-Based Mechanism

The entropy score is a measure of randomness in the processed information, and is calculated as:

$$H(\mathbf{x}; f) = -\sum_i^C f_i.log(f_i), \tag{13}$$

where $f_i(\mathbf{x})$ is the probability (logit) corresponding to the ith class label. Let U be the internal energy, i.e., the expectation value of the energy function (Oh et al., 2020), defined by:

$$U(\mathbf{x}; f) = \sum_i^C E(\mathbf{x}, i) f_i. \tag{14}$$

According to Oh et al. (2020), the entropy can be defined in terms of the internal and free energy functions as: $H(\mathbf{x}; f) = U(\mathbf{x}; f) - F(\mathbf{x}; f)$, where all logits are shifted by the internal energy U. Substituting the free energy from (5) yields:

$$H(\mathbf{x}; f) = log(p(\mathbf{x})) + U(\mathbf{x}; f) + log\big(\textstyle\int_{\mathbf{x}} e^{-F(\mathbf{x};f)}\big), \tag{15}$$

which shows, due to the shifting caused by internal energy, the entropy is not reliably aligned with the probability density $p(\mathbf{x})$. Thus, it is a less suitable routing mechanism unlike the energy score.

4 Experimental Results

In this section, the performance of E-LANG on different architectures such as T5 and BERT; and benchmarks such as GLUE (Wang et al., 2019b), SuperGLUE (Wang et al., 2019a), and WMT (Bojar et al., 2016) is evaluated and compared with the Super models and previous works.

4.1 T5-Based Joint Inference

In Table 1, the T5-based results on GLUE, SuperGLUE, and WMT benchmarks are reported. For all the tasks, we use T5-11B (with 87×10^{11} FLOPs) and T5-large (with 4.25×10^{11} FLOPs) as our Super and Swift models, respectively. The average GPU-based running time and accuracy of both models compared with E-LANG are also summarized in the table. Note that the T5 models used in this experiment have been separately fine-tuned on each of the downstream tasks given in Table 1. The extra energy head for each of these tasks was also separately trained and used based on the task-specific number of classes, i.e., C in Equation (7).

		GLUE						SuperGLUE							WMT
		MNLI	QNLI	SST2	RTE	MRPC	COLA	RTE	BoolQ	MRC	COPA	CB	WIC	WSC	EnRo
Swift (Large)	Time (ms)	216	283	57	263	160	56	287	303	201	96	223	185	133	1609
	Accuracy (%)	89.7	93.9	95.5	90.3	90.9	62.7	88.5	84.3	80.7	81.0	92.0	72.7	86.5	28.6
Super (11B)	Time (ms)	821	980	281	964	433	213	818	3205	1731	268	844	671	2211	3041
	Accuracy (%)	91.7	95.9	96.6	92.4	91.7	69.1	93.1	**89.4**	84.9	**93.0**	93.1	77.4	89.4	28.9
E-LANG	Accuracy (%)	**91.7**	**96.0**	**96.6**	**92.4**	**92.2**	**69.5**	**93.2**	88.7	**84.9**	90.0	**93.1**	**78.1**	**89.4**	**28.9**
	FLOPs ($\times 10^{11}$)	47.8	25.7	29.5	50.4	11.5	39.9	42.0	50.8	46.9	52.6	13.4	40.3	20.6	63.4
	Time (ms)	582	495	132	716	190	147	671	1978	1022	222	302	447	545	2800
	Swift Ratio (%)	49	75	70	46	91	58	56	45	50	43	89	57	81	30
	Speed-up (FLOPs)	1.8X	3.4X	2.9X	1.7X	7.6X	2.2X	2.1X	1.7X	1.9X	1.7X	6.5X	2.2X	4.2X	1.4X
	Speed-up (time)	1.4X	2.0X	2.1X	1.4X	2.3X	1.5X	1.2X	1.6X	1.7X	1.2X	2.8X	1.5X	4.1X	1.1X

Table 1: Joint inference results with T5 architecture on GLUE and SuperGLUE development sets, and WMT's English-to-Romanian translation. The FLOPs for Super and Swift are respectively 87×10^{11} and 4.25×10^{11}.

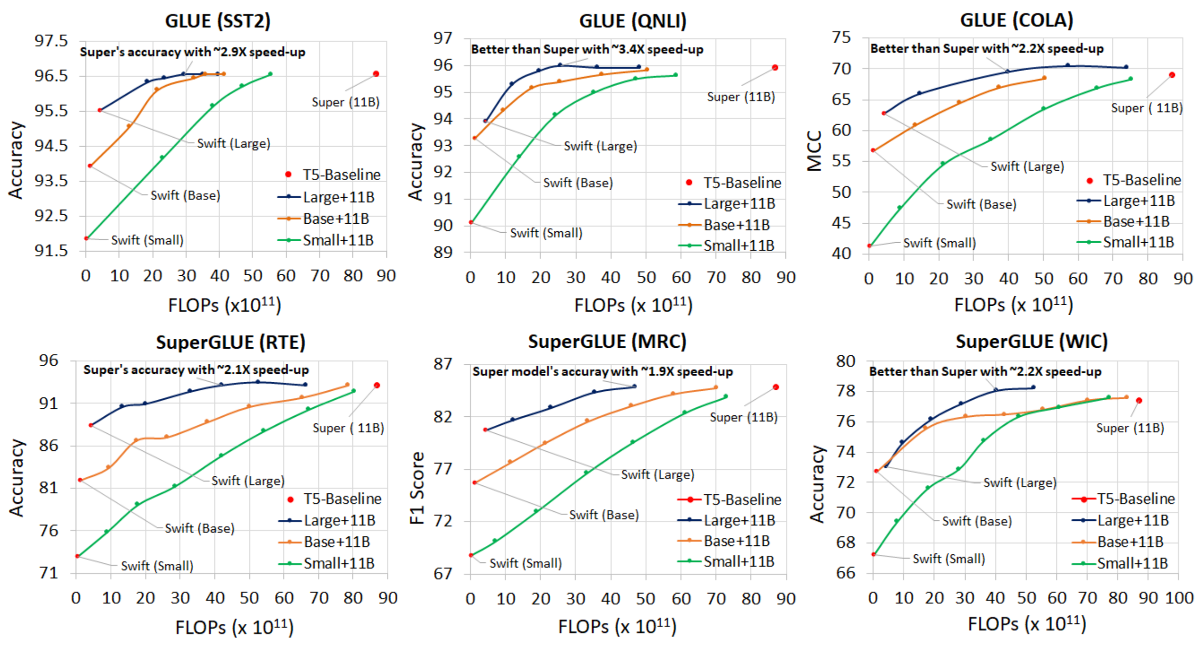


Figure 2: Joint inference trade-off curves with T5 architecture on GLUE and SuperGLUE development sets. Each point is obtained with a different energy threshold.

The total FLOPs for our method is measured as a weighted average of the Super and Swift FLOPs based on their usage frequency as:

$$FLOPs = \frac{1}{N_{sw}+N_{su}}\big(N_{sw}.(F^{E}_{sw}+F_h+F^{D}_{sw}) + N_{su}.(F^{E}_{sw}+F_h+F_{su})\big), \quad (16)$$

where N_{su} and N_{sw} are respectively the number of samples processed by the Super (with F_{su} FLOPs) and Swift (with F^{E}_{sw}, F^{D}_{sw}, and F_h FLOPs for the encoder, decoder, and energy head). Note that F_h is equal to $\approx$0.00001$\times10^{11}$, which has a very insignificant overhead in our framework.

As presented in Table 1, E-LANG can reach the Super model's accuracy on all GLUE tasks with an average 3.3X in FLOPs and 1.8X in running time speed-ups. For some tasks such as QNLI, MRPC, and COLA, we even outperform the Super model, which leads to a higher average accuracy of 89.7% than the Super model with 89.5% on GLUE. For the SuperGLUE benchmark, with an average FLOPs and running time speed-up of 2.9X and 2.0X, our method achieves the same accuracy as the Super model on MRC and CB; and better accuracy on RTE and WIC. On BoolQ and COPA, although 99% and 97% of the Super's accuracy are respectively obtained, it is with 1.7X and 1.4X less FLOPs and latency, on average.

In order to analyze the generality of E-LANG to other NLP problems rather than text classification (Section 3.2.1), we also apply our method to two text-to-text tasks including SuperGLUE's WSC and WMT's English-to-Romanian (EnRo) translation. As given in the table, our method achieves the Super model's accuracy on both WSC and EnRo with 4.2X and 1.4X less FLOPs, respectively.

Figure 2 illustrates the accuracy vs. FLOPs trade-off curves for some tasks in GLUE and Super-

GLUE benchmarks. The curves related to all tasks are given in the supplementary materials. The trade-off points on the curves are dynamically achieved at the inference time by selecting different thresholds, i.e., t in Equations (6) and (9). Larger values for t will result in routing more input data to the Super model, which consequently provides more accurate, but slower inference. As the Swift is able to make accurate predictions for the majority of input data, the dynamic inference with a small enough t can reach the Super model's accuracy but with a much lower computational cost and latency.

Figure 3 illustrates the distribution of the energy scores across the input samples in GLUE tasks. For each task, the distributions of the samples processed by the Super and the Swift models are plotted. As shown, the samples routed to the Super model tend to have lower energy scores that are indeed considered as out-of-distribution samples for the Swift. On the other hand, in overall, higher scores are observed for the Swift distribution, that is for the samples handled by the Swift only. For some tasks such as MRPC and QNLI, the Swift is shown to be highly capable of handling the majority of the input samples. This is also supported by the results in Table 1 and Figure 2, where 91% (for MRPC) and 75% (for QNLI) of the samples are accurately processed by the Swift. In contrast, for other datasets including RTE and MNLI with Swift ratio of less than 50%, most of the samples are hard for the Swift, which are transferred to the Super model. Based on our experiments, the most optimal results for our joint inference framework is achieved when the crossing point of the two distributions (highlighted in green in the figures) is chosen as the threshold t in Equation (9).

4.1.1 Ablation Studies

In Sections 3.3.1 and 3.3.2, the possibility of using softmax and entropy scores instead of energy score was theoretically analyzed. To support that analysis and also experimentally evaluate the performance of different routing mechanisms, an ablation study on GLUE is performed, which is presented in Table 2. In this study, we report the joint inference results based on softmax, entropy, and random scores (i.e., randomly distributing the samples between Super and Swift). Our experiments show that, compared to the random score, softmax and entropy can result in satisfactory performance on routing the samples. However, as also discussed in Sections 3.3.1 and 3.3.2, the energy score is still a better mechanism with about 14% less FLOPs. Another potential mechanism is the perplexity (Chen et al., 1998), but since it provides the same information as entropy, we did not add any extra experiment on it.

The results with the usage of different Swift models including T5-small (with 0.33×10^{11} FLOPs) and T5-base (with 1.24×10^{11} FLOPs) are also given in Table 2. Using these models as Swifts can lead to good performance on some tasks, but not all of them. For example, on SST2, the joint inference with T5-small and T5-base Swifts can respectively reach the Super's accuracy with 1.9X and 2.X less computations. In general, although these models are smaller and require less FLOPs, our results in Table 2 indicate that they perform worse than T5-large in our joint inference structure. In Figure 2, the trade-off curves for different Swift models are shown for GLUE and SuperGLUE.

Moreover, to show the effectiveness of the extra energy head for the Swift encoder, the E-LANG results based on last linear layer of the Swift decoder is also given and compared in Table 2. As reported, the E-LANG empowered by the energy head on the Swift encoder outperforms the case with the decoder's head in both FLOPs (36.8% less) and accuracy (0.7% better). As explained in Section 3.2.1, this shows the deep connection between the encoder's features, discriminative models, and the proposed routing mechanism via the energy head.

We observed that E-LANG can achieve a high performance even when applied to individually pre-trained Super and Swift models. But, more improvement can still be obtained by performing KD from the Super to the Swift model, especially at the fine-tuning process for downstream tasks. To study this, we apply the KD technique in (Sanh et al., 2019) to the Super and Swift models for some GLUE tasks. As summarized in Table 3, the Super model's accuracy for QNLI, SST2, and COLA is respectively attained by the distillation-based E-LANG with 29.2%, 48.5%, and 14.3% less FLOPs than E-LANG (without distillation). The results show the effectiveness of E-LANG along with other compression techniques such as distillation. The trade-off curves for this experiment will be provided in the supplementary materials.

4.2 BERT-Based Joint Inference

In this section, the proposed energy-based joint inference method is applied to the BERT architecture (Devlin et al., 2019) and compared with

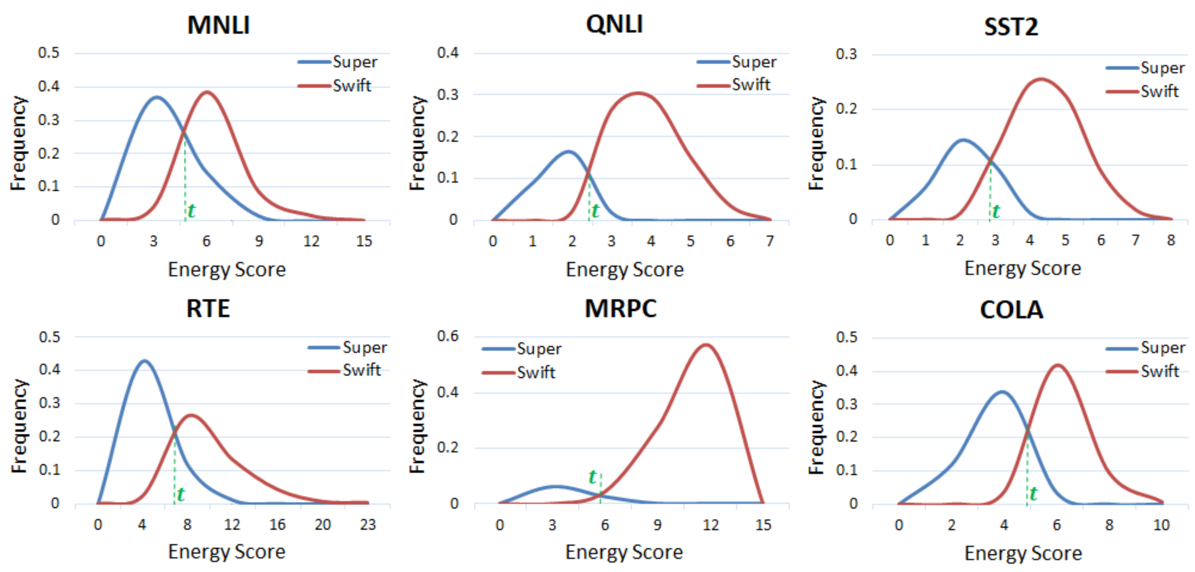


Figure 3: Energy score distribution for GLUE tasks. t shows the optimal threshold.

	MNLI	QNLI	SST2	RTE	MRPC	COLA	Average
Super (11B)	87.0 / 91.7	87.0 / 95.9	87.0 / 96.6	87.0 / 92.4	87.0 / 91.7	87.0 / 69.1	87.0 / 89.5
Random (Encoder)	78.5 / 91.5	61.9 / 95.3	58.7 / 96.3	60.2 / 91.2	47.5 / 91.9	61.6 / 67.2	61.4 / 88.9
Softmax (Encoder)	57.7 / 91.6	36.5 / 95.9	34.6 / 96.5	52.0 / 92.3	13.8 / 92.1	45.7 / 69.3	40.1 / 89.6
Entropy (Encoder)	55.7 / 91.6	27.1 / 96.0	40.2 / 96.5	50.7 / 92.0	23.0 / 92.2	48.1 / 69.3	40.8 / 89.6
Energy (Swift$_{small}$)	71.3 / 91.0	58.8 / 95.6	47.0 / 96.6	71.2 / 88.5	55.0 / 91.4	75.3 / 68.3	63.1 / 88.5
Energy (Swift$_{base}$)	54.5 / 91.5	50.5 / 95.8	35.9 / 96.6	55.8 / 90.6	44.0 / 91.9	50.6 / 68.4	48.5 / 89.1
Energy (Decoder)	57.9 / 90.6	68.1 / 95.5	75.8 / 96.3	60.5 / 91.5	20.2 / 90.9	45.1 / 69.3	54.6 / 89.0
Energy (Encoder)	**47.8 / 91.7**	**25.7 / 96.0**	**32.0 / 96.6**	**50.4 / 92.4**	**11.5 / 92.2**	**39.9 / 69.5**	**34.5 / 89.7**

Table 2: Ablation study on different T5-based scenarios. Each cell shows FLOPs/Accuracy.

	QNLI	SST2	COLA
Super (11B)	87.0 / 95.9	87.0 / 96.6	87.0 / 69.1
Swift (Large)	4.25 / 93.9	4.25 / 95.5	4.25 / 62.7
+ Distillation	4.25 / 95.0	4.25 / 95.7	4.25 / 63.3
Ours	25.7 / 96.0	29.5 / 96.6	39.9 / 69.5
+ Distillation	**18.2 / 96.0**	**15.2 / 96.6**	**34.2 / 69.5**

Table 3: Distillation-based results with T5 in terms of FLOPs/Accuracy.

BERT-based SOTA in both fixed-size and dynamic inference. The majority of the previous methods employ knowledge distillation and data augmentation techniques for training their student models. For a fair comparison, we follow the same practice and use the transformer distillation and augmentation strategies in TinyBERT (Jiao et al., 2020) to train and prepare our Swift model (i.e., BERT$_{Tiny}$ with 1.2×10^9 FLOPs). Moreover, similar to the other works, we use BERT$_{Base}$ (with 21.8×10^9 FLOPs) as our Super (i.e., teacher) model.

In Table 4, the comparison results with the baseline BERT$_{Base}$ and SOTA on GLUE benchmark are presented in terms of accuracy, FLOPs, and latency. Compared to the Super model, E-LANG delivers better accuracy on SST2 and RTE with 3.5X and 2.0X FLOPs speed-up; and the same accuracy on QNLI, MRPC, and QQP with 2.4X, 2.7X, and 7.0X FLOPs speed-up, respectively. On MNLI and COLA, 99.8% and 97.3% of the Super model's accuracy are achieved, but with an average FLOPs speed-up of 2.3X. On average, E-LANG outperforms the Super model with 0.1% higher accuracy, 3.2X less FLOPs, and 1.6X less latency.

Compared with SOTA, our method achieves the best performance on all GLUE tasks, except MRPC for which SqueezeBERT outperforms all due to having a more accurate teacher (Iandola et al., 2020). There are some works such as ELECTRA (Clark et al., 2020) and MobileBERT (Sun et al., 2020) that require less FLOPs than our method, but they only reach 95% of the baseline's accuracy. Compared to other methods, GhostBERT (Huang et al., 2021) and DynaBERT (Hou et al., 2020) give the closest performance to the baseline and even the same as ours on some tasks such as QNLI. However, on average, they still need about 30% more FLOPs on GLUE compared to E-LANG.

The E-LANG accuracy vs. FLOPs trade-off curves compared to SOTA on some of GLUE tasks are shown in Figure 4. The trade-off curves for all the tasks are reported in the supplementary materials. Among the SOTA methods presented in Table

		MNLI (m/mm)	**QNLI**	**SST2**	**RTE**	**MRPC**	**COLA**	**QQP**	**Avg.**	**FLOPs** (G)	**Time** (ms)
	BERT$_{Tiny}$ (Swift)	82.8 / 82.9	87.9	92.6	65.7	85.8	49.7	90.5	78.5	1.2	7
	BERT$_{Base}$ (Super)	**84.9 / 85.5**	92.2	93.5	71.1	87.3	**60.3**	91.5	83.3	21.8	20
Previous works	DistillBERT	82.2 / -	89.2	92.7	59.9	87.5	51.3	88.5	78.8	11.3	-
	ELECTRA	78.9 / -	87.9	88.3	68.5	84.4	56.8	88.3	79.0	3.7	-
	DeeBERT	83.9 / 82.9	90.9	93.4	69.5	-	-	-	-	-	17
	MobileBERT	84.3 / -	91.5	92.5	70.4	87.0	51.1	-	79.5	5.7	-
	SqueezeBERT	82.5 / 82.9	90.9	92.2	71.8	**89.8**	53.7	89.5	81.7	7.4	-
	Len-Adaptive	84.4 / -	-	93.1	-	-	-	-	-	8.8	-
	TinyBERT	84.5 / 84.5	91.8	93.0	69.3	87.2	54.0	91.0	81.9	11.3	10
	ELM	84.2 / -	90.8	92.7	72.2	89.0	54.2	91.1	82.0	10.9	-
	GhostBERT	84.7 / -	92.2	92.9	72.2	87.3	58.1	91.2	82.7	11.3	-
	DynaBERT	84.7 / 85.2	92.2	93.3	73.0	84.8	58.4	91.3	82.9	10.9	16
E-LANG	**Accuracy** (%)	84.7 / 85.4	**92.2**	**93.7**	**73.3**	87.3	58.7	**91.5**	**83.4**	-	-
	FLOPs (G)	**9.1**	**9.2**	**6.3**	**10.8**	8.2	**9.9**	**3.1**	8.1	-	-
	Time (ms)	14	14	11	16	13	15	9	13	-	-
	Swift Ratio (%)	64	63	77	56	68	60	91	68	-	-
	Speed-up (FLOPs)	2.4X	2.4X	3.5X	2.0X	2.7X	2.2X	7.0X	3.2X	-	-
	Speed-up (time)	1.4X	1.4X	1.8X	1.3X	1.5X	1.3X	2.2X	1.6X	-	-

Table 4: Joint inference results with BERT architecture on GLUE development set compared with SOTA.

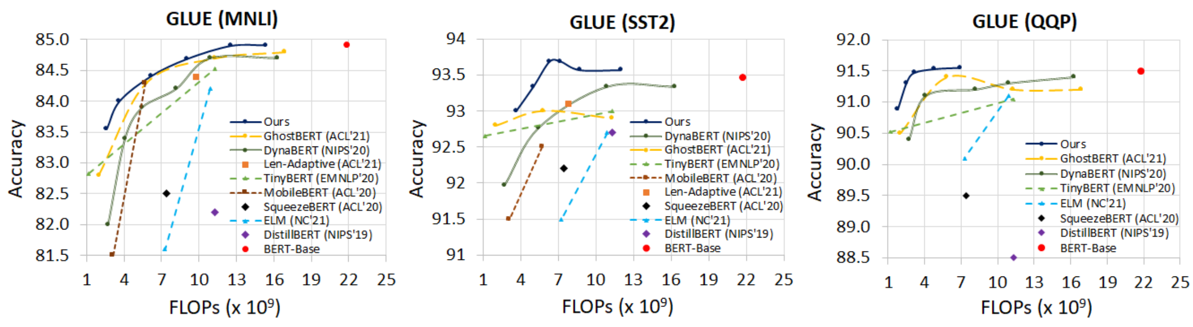


Figure 4: Joint inference trade-off curves with BERT on GLUE development set compared with SOTA.

4 and Figure 4, only DeeBERT (Xin et al., ACL 2020), Length-Adaptive (Kim and Cho, 2021), and DynaBERT (Hou et al., 2020) are in the category of dynamic inference, where a single model can operate at different trade-off points between accuracy and computational cost. The other approaches propose fixed-size smaller versions of BERT$_{Base}$, which require re-training for every trade-off point.

To investigate the orthogonality of E-LANG with others, we integrate our energy-based joint inference strategy with DynaBERT that is SOTA in BERT-based adaptive inference. In other words, we analyze whether E-LANG can be added on top of other efficient methods to benefit both from their designs and our approach. In this experiment, the DynaBERT configurations with the highest accuracy (i.e., width=0.75 & depth=1.0) and the lowest FLOPs (i.e., width=0.5 & depth=0.25) are respectively employed as the Super and Swift models in our framework. The corresponding joint inference results on MNLI, SST2, and QQP are reported in Table 5. As observed, we accomplish the DynaBERT Super's accuracy for MNLI and SST2 with 1.7X and 3.1X less FLOPs. For QQP, our method combined with DynaBERT even outperforms DynaBERT by 0.1% with 2.6X FLOPs speed-up.

	MNLI	**SST2**	**QQP**
DynaBERT (Swift) (w=0.5, d=0.25)	2.7 / 82.0	2.7 / 91.9	2.7 / 90.4
DynaBERT (Super) (w=0.75, d=1.0)	16.3 / 84.7	16.3 / 93.3	16.3 / 91.4
Ours+DynaBERT	**9.4 / 84.7**	**5.2 / 93.3**	**6.2 / 91.5**

Table 5: Orthogonality of E-LANG (ours) with DynaBERT in terms of FLOPs/Accuracy.

5 Conclusion

In this paper, we introduced E-LANG, an energy-based joint inference approach, which integrates Super and Swift language models for achieving efficient inference without sacrificing the accuracy. Our method can work with both encoder-only (e.g., BERT) and encoder-decoder (e.g., T5) architectures, and is also applicable for text classification and sequence-to-sequence problems. The proposed joint inference strategy was theoretically and experimentally analyzed with an extensive set of experiments and ablation studies. Our results showed that E-LANG outperforms SOTA in both fixed-size and dynamic inference over different benchmarks such as GLUE and SuperGLUE. One future direction to this work is to apply E-LANG to multiple Super and Swift models with different sizes.

References

Mohammad Akbari, Amin Banitalebi-Dehkordi, and Yong Zhang. 2021. EBJR: Energy-based joint reasoning for adaptive inference. *BMVC 2021*.

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jin Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2021. BinaryBERT: Pushing the limit of bert quantization. In *ACL 2021*, pages 4334–4348.

Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. 2021. Auto-split: A general framework of collaborative edge-cloud ai. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2543–2553.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, et al. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Stanley F Chen, Douglas Beeferman, and Roni Rosenfeld. 1998. Evaluation metrics for language models.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pretrained bert networks. *Advances in neural information processing systems*.

Y. Cheng, D. Wang, P. Zhou, and T. Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv:1710.09282*.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *International Conference on Learning Representations, ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*.

Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing bert: Studying the effects of weight pruning on transfer learning. *ACL 2020*, page 143.

Manish Gupta and Puneet Agrawal. 2022. Compression of deep learning models for text: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 16(4):1–55.

Manish Gupta, Vasudeva Varma, Sonam Damani, and Kedhar Nath Narahari. 2020. Compression of deep learning models for nlp. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management, CIKM*, pages 3507–3508.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. *International Conference on Learning Representations, ICLR*.

G. E. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33.

Zhiqi Huang, Lu Hou, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2021. Ghostbert: Generate more features with cheap operations for bert. *ACL/IJCNLP*.

Forrest Iandola, Albert Shaw, Ravi Krishna, and Kurt Keutzer. 2020. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 124–135. Association for Computational Linguistics.

Xiaoqi Jiao, Huating Chang, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2021. Improving task-agnostic bert distillation with layer mapping search. *Neurocomputing*, 461:194–203.

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 *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4163–4174.

Jing Jin, Cai Liang, Tiancheng Wu, Liqin Zou, and Zhiliang Gan. 2021. KDLSQ-BERT: A quantized bert combining knowledge distillation with learned step size quantization. *arXiv preprint arXiv:2101.05938*.

Gyuwan Kim and Kyunghyun Cho. 2021. Length-adaptive transformer: Train once with length drop, use anytime with search. *ACL 2021*.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR.

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. 2006. A tutorial on energy-based learning. *Predicting structured data*, 1(0).

Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. 2019. An energy and gpu-computation efficient backbone network for real-time object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. Gshard: Scaling giant models with conditional computation and automatic sharding. *International Conference on Learning Representations, ICLR*.

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

Sangchul Oh, Abdelkader Baggag, and Hyunchul Nha. 2020. Entropy, free energy, and work of restricted boltzmann machines. *Entropy*, 22(5):538.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. In *International Conference on Learning Representations*.

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.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS 2019*.

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.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2017. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1047–1055.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019a. Superglue: a stickier benchmark for general-purpose language understanding systems. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 3266–3280.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019b. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019*.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. ACL 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.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Keyi Yu, Yang Liu, Alexander G Schwing, and Jian Peng. 2018. Fast and accurate text classification: Skimming, rereading and early stopping. In *6th International Conference on Learning Representations, ICLR 2018*.

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.

A Supplementary Materials

This section contains the supplementary materials.

A.1 Code and Demo

We shared our code to make it easy to reproduce our BERT-based results. In addition to the code, we included a video demo that contains a demonstration of T5-based E-LANG. The BERT-based E-LANG source-code with the detailed running instructions and the T5-based E-LANG demo are available here[1].

Please note that the demo is based on screen recording of a web application we built to show the use-cases of our method in real-world scenarios. Figure 5 shows a screenshot of the demo application.

The T5-based E-LANG code with detailed instructions is also shared in a 'code' directory in the supplementary materials file.

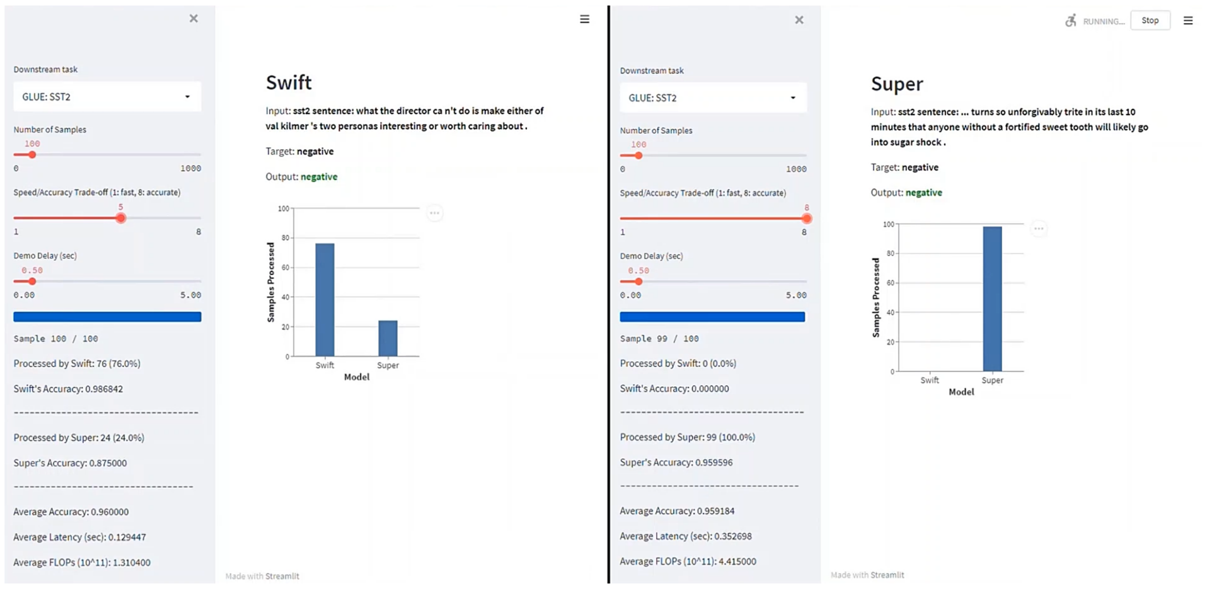
A.2 Additional Results and Visualizations

The trade-off curves (for the experiments given in Table 1) with T5 architecture on GLUE and SuperGLUE tasks are respectively shown in Figures 6 and 7. The ablation over different Swift models are also given in the figures.

In Figure 8, the accuracy vs. FLOPs trade-off curves for distillation-based experiments (reported in Table 3) are also given. On QNLI, distillation-based E-LANG (denoted by DE-LANG) with 4.8× less computations than the Super model outperforms E-LANG with 3.4× FLOPs speed-up, although both methods performs 0.1% more accurate than the Super model. DE-LANG on SST2 can also achieve the Super model's accuracy with 5.7× less computations, while the original E-LANG achieves the same performance with only 2.9× speed-up. Moreover, DE-LANG can improve the Super model's accuracy by 0.1% with 2.9× speed-up on SST2. For COLA, DE-LANG achieves a better FLOPs speed-up of 2.5× than E-LANG with 2.2× speed-up, where both outperform the Super model's accuracy by 0.4%.

Figure 9 also illustrates the corresponding curves for the BERT-based results of Table 4, which are compared with previous works in fixed-size and adaptive inference.

[1]https://developer.huaweicloud.com/develop/aigallery/notebook/detail?id=64199726-9aaf-4905-8f6f-4cae290df874


Downstream task

GLUE: SST2

Number of Samples: 100 (0 – 1000)

Speed/Accuracy Trade-off (1: fast, 8: accurate): 5 (1 – 8)

Demo Delay (sec): 0.50 (0.00 – 5.00)

```
Sample 100 / 100
```

Processed by Swift: 76 (76.0%)

Swift's Accuracy: 0.986842

Processed by Super: 24 (24.0%)

Super's Accuracy: 0.875000

Average Accuracy: 0.960000

Average Latency (sec): 0.129447

Average FLOPs (10^11): 1.310400

Swift

Input: **sst2 sentence: what the director ca n't do is make either of val kilmer 's two personas interesting or worth caring about .**

Target: **negative**

Output: **negative**

Downstream task

GLUE: SST2

Number of Samples: 100 (0 – 1000)

Speed/Accuracy Trade-off (1: fast, 8: accurate): 8 (1 – 8)

Demo Delay (sec): 0.50 (0.00 – 5.00)

```
Sample 99 / 100
```

Processed by Swift: 0 (0.0%)

Swift's Accuracy: 0.000000

Processed by Super: 99 (100.0%)

Super's Accuracy: 0.959596

Average Accuracy: 0.959184

Average Latency (sec): 0.352698

Average FLOPs (10^11): 4.415000

Super

Input: **sst2 sentence: ... turns so unforgivably trite in its last 10 minutes that anyone without a fortified sweet tooth will likely go into sugar shock .**

Target: **negative**

Output: **negative**

Figure 5: A demo application to show-case the adaptive inference with the proposed method.

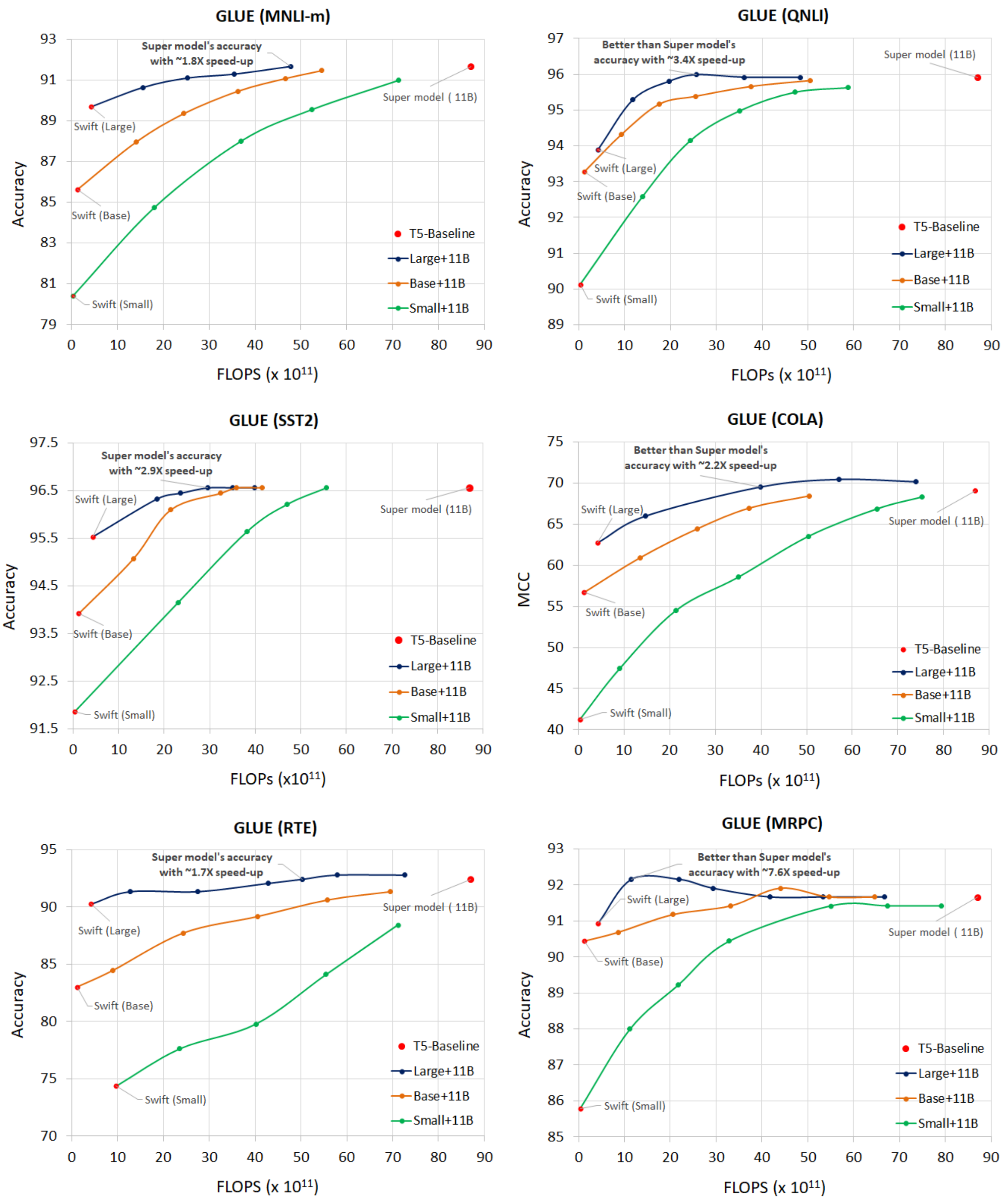


Figure 6: Trade-off curves with T5 backbone on GLUE tasks.

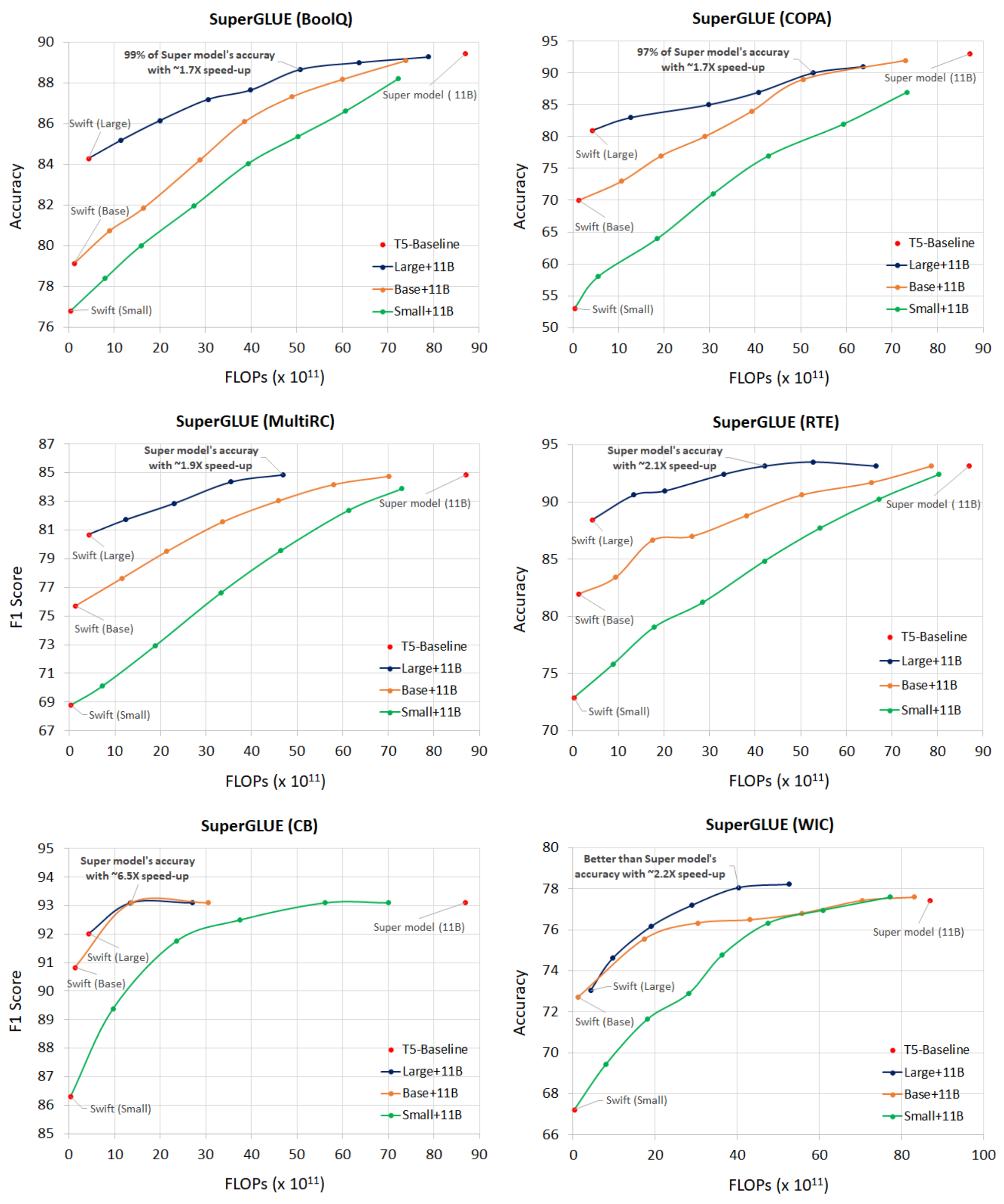


Figure 7: Trade-off curves with T5 backbone on SuperGLUE tasks.

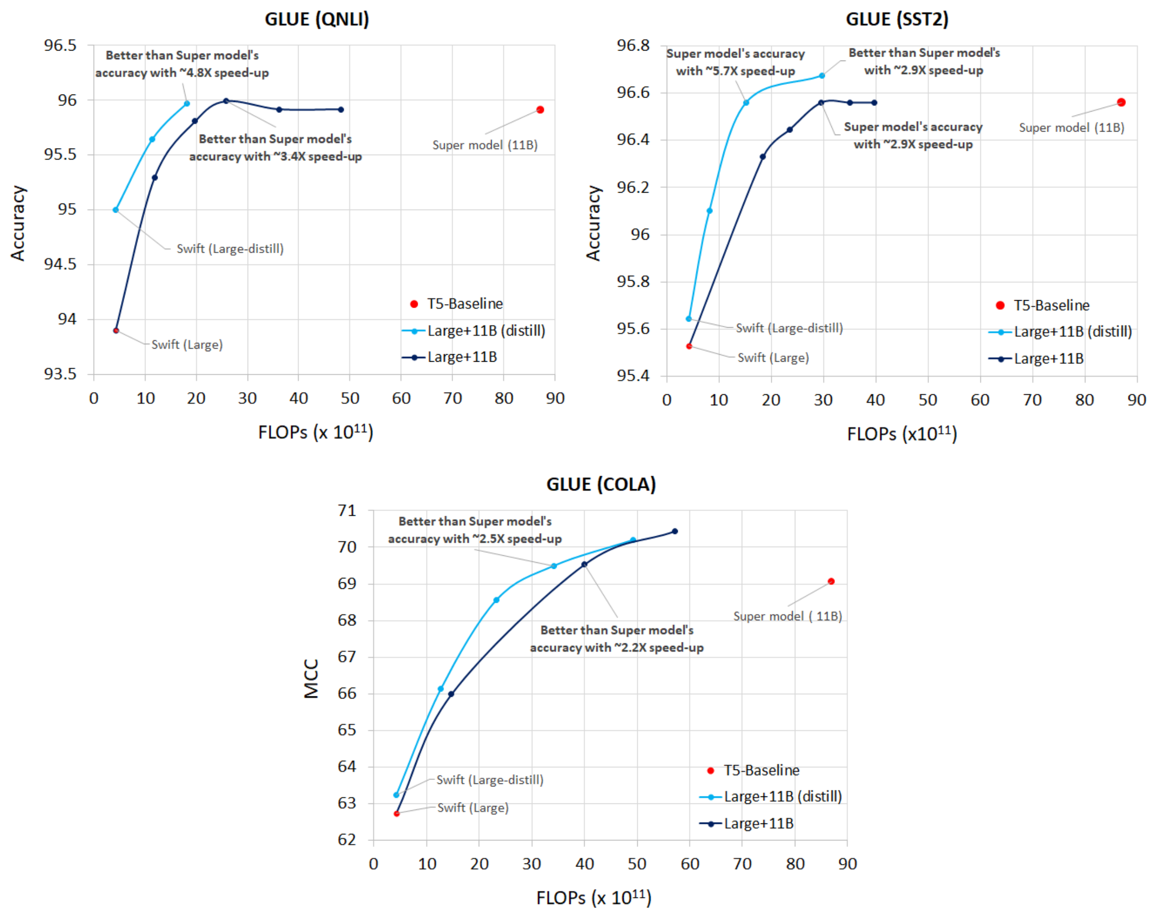


Figure 8: Distillation-based trade-off curves with T5 backbone on some GLUE tasks.

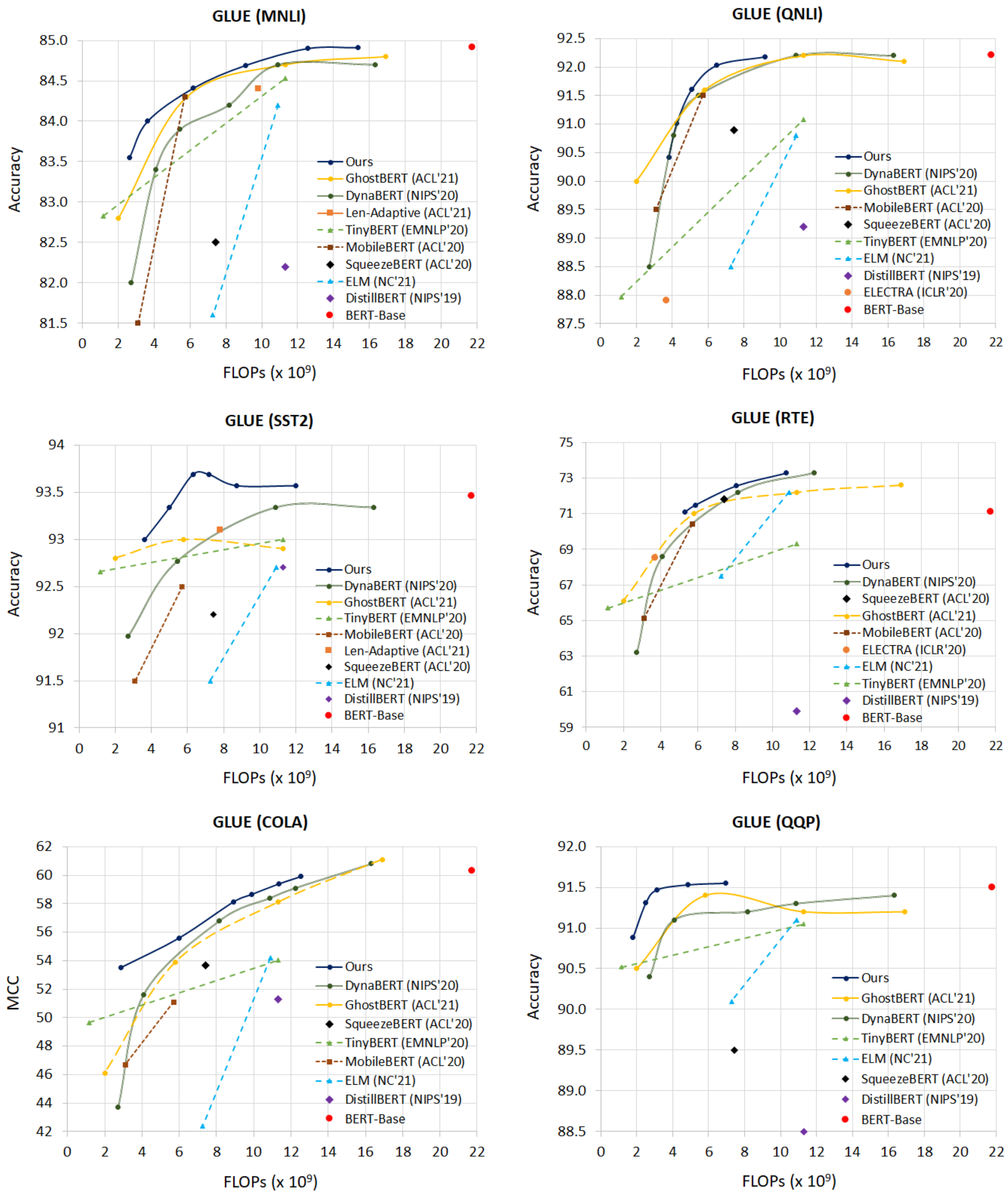


Figure 9: Trade-off curves compared with BERT-based SOTA on GLUE tasks.