BE3R: BERT based Early-Exit using Expert Routing

Sourab Mangrulkar
smangrul@amazon.com
Amazon
Bengaluru, Karnataka, India

Ankith M S
ankiths@amazon.com
Amazon
Bengaluru, Karnataka, India

Vivek Sembium
viveksem@amazon.com
Amazon
Bengaluru, Karnataka, India

ABSTRACT

Pre-trained language models like BERT have reported state-of-the-art performance on several Natural Language Processing (NLP) tasks, but high computational demands hinder its widespread adoption for large scale NLP tasks. In this work, we propose a novel routing based early exit model called **BE3R** (**B**ERT based **E**arly-**E**xit using **E**xpert **R**outing), where we learn to dynamically exit in the earlier layers without needing to traverse through the entire model. **Unlike the exiting early-exit methods, our approach can be extended to a batch inference setting**. We consider the specific application of search relevance filtering in Amazon India marketplace services (a large e-commerce website). Our experimental results show that BE3R improves the batch inference throughput by **46.5%** over the BERT-Base model and **35.89%** over the DistilBERT-Base model on large dataset with 50 Million samples without any trade-off on the performance metric. We conduct thorough experimentation using various architectural choices, loss functions and perform qualitative analysis. We perform experiments on public GLUE Benchmark [28] and demonstrate comparable performance to corresponding baseline models with **23**% average throughput improvement across tasks in batch inference setting.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; • **Information systems** → **Web searching and information discovery**; • **Applied computing** → **Electronic commerce**.

KEYWORDS

Deep Learning; Natural Language Processing; Transformers; Attention Models; Relevance Classification; Product Search; E-Commerce

ACM Reference Format:
Sourab Mangrulkar, Ankith M S, and Vivek Sembium. 2022. BE3R: BERT based Early-Exit using Expert Routing. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534678.3539132

1 INTRODUCTION

Language models such as BERT [9] have achieved state-of-the-art performance on several NLP tasks. The fundamental building block of BERT is the self-attention layer. In this layer, the input word representation interacts with every other word in the input to generate a contextualised representation for the sequence. This interaction of all word pairs although captures deep semantic association between words, incurs a compute cost of $O(N^2)$, where N is the number of tokens in the input.

Consider the application of relevance filtering using (query, product) relevance classification model in large-scale e-commerce company like Amazon, Walmart etc,. BERT demonstrates a significant increase in performance [20], however, it comes with a significantly high compute cost. This prohibits widespread application of BERT based models. For example, for the task of relevance classification, it costs $440 to classify 1 Billion (query, product) pairs using BERT model. Big e-commerce websites deal with such traffic on a daily basis. For 1000 Billion pairs over the course of a year, it would cost $0.44M. In such scenarios, throughput gains of 40% while keeping the performance intact would decrease the compute costs by $0.13M while delivering the same customer experience. This enables feasibility of deploying BERT based model in diverse applications to better serve the customers.

Efficient attention based models is currently an active area of research. One approach is to have a smaller student model and perform knowledge distillation from bigger teacher model, these include models such as DistilBERT [25], TinyBERT [14] and MiniLM [30]. However, using these approaches the speedup comes at considerable performance drop on many downstream tasks. Another approach is to optimize the attention layer by reducing the $O(N^2)$ complexity, these include approaches such as Linformer [29], Performer [6] and BigBird [36]. However, when using these approaches the throughput gains are observed for long sequences or at the cost of considerable performance drop. Funnel Transformer [8] and PowerBERT [12] exploit the fact of knowledge diffusion through the layers to decrease the sequence length with depth thereby achieving speedup at comparable performance. For tasks involving sentence pairs, efficient architectures such as S-BERT (Sentence-Transformer) [24] and ColBERT [15] are used. These models achieve high scalability but aren't as performant as cross-encoder models.

Above we saw prominent approaches undertaken to improve the throughput in batch setting. Another active area of research questions the necessity of traversing through all layers for each sample. The motivation is to overcome the problem of overthinking [34, 37] in these overparameterized language models. The idea is that samples differ in the complexity with some requiring less compute for prediction when compared to others. As such, the samples can traverse through more layers if they are complex/difficult and traverse through few layers if they are simple. These includes approaches such as PABEE [37], FastBERT [19], DeeBERT [34] and PonderNet [4]. However, all of the existing early exit approaches are for adaptive inference which can be only used on individual samples with

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539132

batch size=1, causing high under-utilization of computational resources and vector optimizations of GPUs. This makes them highly inefficient at utilizing the hardware resources in batch inference setting. In this paper, (1) we overcome this problem and propose a novel routing based early exit model called **BE3R** (**B**ERT based **E**arly-**E**xit using **E**xpert **R**outing) capable of batch inference (2) carry out thorough experimentation, qualitative analysis & ablation studies on relevance classification task and evaluate the throughput gains on a large dataset of 50 Million (query, product) pairs in batch inference setting (3) perform thorough experiments on public GLUE benchmark and demonstrate comparable performance to corresponding baseline models with **23%** average throughput improvement across tasks in batch inference setting.

2 RELATED WORD

Knowledge Distillation and Compact Architectures: Knowledge Distillation [13] aims to train a small and inexpensive student network to mimic the behaviour of a large and expensive teacher model. It works by training a student model with a distillation loss over the soft target probabilities of the teacher. The key idea behind this is that soft probabilities from a teacher model is more informative than the hard labels. The most widely used Knowledge Distillation based BERT model is DistilBERT [25] which is a BERT model with 6 encoder layers. Other prominent models from this category are TinyBERT [14], MiniLM [30] and MobileBERT [27]. These model achieve great batch inference throughput. However, these models have considerable performance drop on many downstream tasks.

Efficient Self Attention: This class of techniques attempt to tackle the major computational bottleneck of the Transformers, i.e. the self-attention mechanism which has quadratic time complexity. One technique is to introduce sparsity in the self-attention [5, 22, 36] by having each token attend to only a subset of other tokens in the sequence. Another approach proposed by [17] is to use local sensitive hashing. However, these approaches struggle in maintaining the prediction performances and don't report noticeable throughput gains for sequences with length < 2048.

Knowledge Diffusion: Knowledge diffusion is a phenomenon in large models like BERT where the various subparts of the model are encoded with the same information [12]. Specifically, in BERT, as the sequential input passes through the encoder pipeline, they tend to carry similar information as they go deeper in a network. PoWER-BERT [12] points out that the cause of this is the self-attention modules at each encoder layer. Hence we can shorten the sequence length of token-embeddings as we go deeper into the pipeline, thereby reducing the computation. Another prominent model falling in this category is the Funnel Transformer [8].

Architectural Innovation for sentence pair tasks: For tasks involving text-pair as input (for instance (query, product) pair in the relevance classification), traditional BERT based classifiers pack the sentence pair into a single sequence using a special token [SEP]. The segment embedding layer adds the information about the sentence a given token belongs to. The final hidden representation of the [CLS] token is passed to the classifier head for making predictions. Let this model be *RelBERT*. This model will be computationally very expensive for tasks such as Semantic Sourcing and Information Retrieval wherein one would have to evaluate all products for a given query. Approaches such as Sentence-BERT [24] and ColBERT [15] overcome this limitation by generating embeddings for unique products and caching them in offline setting. Now, in online setting, for a given query the model will generate embeddings, and then K nearest products can be retrieved using the similarity metric such as Cosine Similarity. These models have high scalability, however they have high performance drop in comparison to cross-encoder models like *RelBERT*.

Early Exit: Apart from the high computational cost, Language Models suffer from the overthinking problem, as pointed out by [34, 37]. For many input samples, shallow hidden states are adequate to make predictions, whereas the representations in the final layer are distracted by over-complicated and unnecessary features. So, early exit models infer samples using a variable number of layers depending on the input's complexity. These includes approaches such as PABEE [37], FastBERT [19], DeeBERT [34] and PonderNet [4]. In PABEE, depending on input sample's complexity, the model exits when the predictions from intermediate layers remain unchanged for predefined number of steps. In DeeBERT and FastBERT, sample exits from an intermediate layer when the entropy of the predictions is below a predefined threshold. PonderNet paper introduces an algorithm which takes into account the sample complexity to learn and adapt the amount of computation required. It introduces probabilistic halting policy wherein overall probability of halting at step is computed as geometric distribution based on predictions of halting node from current and previous steps. The loss is composed of (1) weighted average of step-wise loss wherein the weights are the halting probabilities for the corresponding steps and (2) regularization loss wherein KL-Divergence between the distribution of halting probabilities and a prior distribution (a geometric distribution) is computed. The importance of regularization loss is that it promotes exploration and biases the network towards expected number of steps. However, these existing early-exit approaches cannot be implemented for mini-batches and require inferencing on each sample (support for only batch_size=1); this is very inefficient and under-utilizes compute resources(GPUs) and vector optimizations. Therefore, the current early-exit approaches are inadequate in large scale production settings requiring mini-batch predictions on datasets containing billions of samples. We address this challenge in this work by leveraging inspiration from the PonderNet algorithm and the Switch layer based expert routing from the Switch Transformer [11] paper. Switch transformer uses a mixture of experts (MoE) paradigm, i.e. for a given input, the computations are carried out only through the subset of the model; hence, increasing the model parameters while keeping the inference cost constant. It replaces dense feed-forward layer in the transformer with a sparse Switch-FFN layer. Each token in the input sequence is independently routed to one of the expert FFN by the router module. Switch Transformer achieves sample efficiency during training and better performance due to MoE but has nothing to do with reduction in inference time or sample complexity based compute adaptation. This is because it leverages same architecture for experts which have same computational cost, i.e., homogeneous architecture.

Model Compression: Computational graph optimization, lower fp_16 precision weights and runtime hardware acceleration using

frameworks such as ONNX [3] and TensorRT [2] also provide significant speedups at comparable performance and these can be applied over all the above efficient modelling approaches including the model proposed in this work.

3 OUR APPROACH

We propose a novel routing based early exit model called **BE3R** (**B**ERT based **E**arly-**E**xit using **E**xpert **R**outing) capable of batch inference. This addresses the under-utilisation of computational resources in batch setting faced by the existing early exit methods. This is motivated by the intuition behind early-exit methods, switch layer from Switch Transformer [11] and loss components from PonderNet [4].

Switch Transformer replaces dense feed-forward layer in the transformer with a sparse Switch-Feed Forward Network (FFN) layer. Each token in the input sequence is independently routed to exactly one of the expert FFN by the router module. Unlike the token-level routing in Switch Transformer, BE3R is based on sequence-level expert routing, wherein the experts are n-layer BERT/DistilBERT [9, 25] based models. Thus, the router in BE3R alleviates the overthinking problem by routing samples to suitable expert of desired computational complexity. Here, we can observe that we are having heterogeneous architecture wherein experts are models with different number of layers thereby having different computational costs. We learn and adapt a switch layer which can route a sample based on its complexity to optimal expert preferably with less number of layers. By training experts of varying complexity, the model gains latency benefits by avoiding redundant computations in easier samples. This would result in alleviating overthinking problem and preserving the performance and at the same time increasing throughput. The switch layer partitions a given batch of samples into sub-batches based on routes assigned. These sub-batches can then be sent to respective experts for getting the predictions enabling batch inference. The switch layer and the expert models are learnt end-to-end simplifying training process.

3.1 Model Architecture and its variants

BE3R consists of n experts: $E =< expert_i >, i = 1, 2, 3, ..., n$ where $expert_i$ is a BERT/DistilBERT model with i encoder layers followed by a classifier/regression head which takes 'CLS' token output of the i^{th} encoder and predicts $\hat{y_z}$ for each input x_z. Let us assume that the number of experts $n = 6$ and consider DistilBERT as the backbone, which corresponds to $BE3R^{DistilBERT}$ model variants. It also contains a switch layer (S) which is a transformer encoder followed by an n-way softmax. Figure 1 is a visual representation of the $BE3R^{DistilBERT}$ model architectures.

Let $X =< x_i >, i = 1, 2, 3, ..., b$ be a mini-batch, where x_i is a sample. The samples X are passed to switch layer S, which performs a linear operation $h(x_i) = W \cdot x_i^{[CLS]}$, where $W \in R^{n \times d}$ and $x_i^{[CLS]}$ is the CLS token contextual embedding from encoder layer in S. Here, d is the embedding dimension of the backbone model. Figure 2 illustrates the switch layer. Then, $h(x_i)$ is passed to a softmax layer to calculate the route probabilities for $expert_j$ using $p_j(x_i) = \frac{e^{h_j(x_i)}}{\sum_{k=0}^{n} e^{h_k(x_i)}}$.

Finally, we partition X into subsets X_j going into $expert_j$ based on $X_j = \cup_{x \in X}\{\mathbb{1}(argmax\ p(x), j)\}$.

BE3R$_{\text{Sparse}}$: In this setting, the parameters are not shared between encoders of different experts. As such these models have large memory footprints. As observed in Switch Transformer paper [11], increasing the number of parameters adds robustness, improves sample efficiency and performance. Having independent parameters across experts helps the model separate out different learnable patterns that are prevalent to only a subsample space and hence train better generalizable models. Figure 1 (right) is a visual representation of the $BE3R_{Sparse}$ model architecture

BE3R$_{\text{Lean}}$: We have designed a lean setting for the BE3R model called $BE3R_{Lean}$ wherein parameters are shared between encoders of different experts. For example, encoder layer 2 of $expert_2$ and $expert_3$ would share parameters. This setting is trained on the same loss as that of BE3R. Figure 1 (left) is a visual representation of the $BE3R_{Lean}$ model architecture wherein the sequential arrows from Encoder 1 to Encoder 6 are only for visualization purposes to indicate that parameters are shared across experts. We will see that for larger models like BERT-Base, Bert-Large and Electra-Large, we cannot train $BE3R_{Sparse}$ variants because of the huge number of model parameters requiring model memory which won't fit on a single GPU. It would need model parallelism and isn't straightforward to train and tune. In such settings, we train the $BE3R_{Lean}$ variants wherein the models fit on single GPU because the parameters are shared.

3.2 Training

BE3R model involving switch layer and experts are trained in an end-to-end manner. We consider 2 kinds of training methodologies. They are explained in the Sub-Sections 3.2.1 and 3.2.2

3.2.1 Hard Routing loss objective. In this methodology, during training, we route each sample only to a single expert as per the switch layer. This approach also enables efficient training, the reason being that each sample is routed to exactly one expert during training. The default implementation led the switch layer to route to only a single expert as it had no incentive to route to different experts. To overcome this issue, similar to the [11] we incorporated the differentiable load balancing loss. This auxiliary loss encourages uniform distribution of samples across experts. The full objective is the combination of 2 loss functions where α is the hyperparameter weighing the auxiliary loss. The hard-routing objective is given in equation 1.

$$\ell_{hr} = \sum_{j=1}^{k} L(\hat{y}_j, y_j) + \alpha \cdot E \cdot \sum_{i=1}^{E} f_i \cdot P_i \tag{1}$$

where:
1. L is the cross-entopy loss for a classification task and mean-squared loss for a regression task; E is the number of experts; k is the number of samples in a given mini-batch B.
2. $f_i = 1/k \cdot \sum_{x \in B} 1\{argmax\ p(x), i\}$, the fraction of samples dispatched to expert i given mini-batch B with k samples.
3. $P_i = 1/k \cdot \sum_{x \in B} p_i(x)$, the fraction of the router probability allocated to expert i given mini-batch B with k samples.

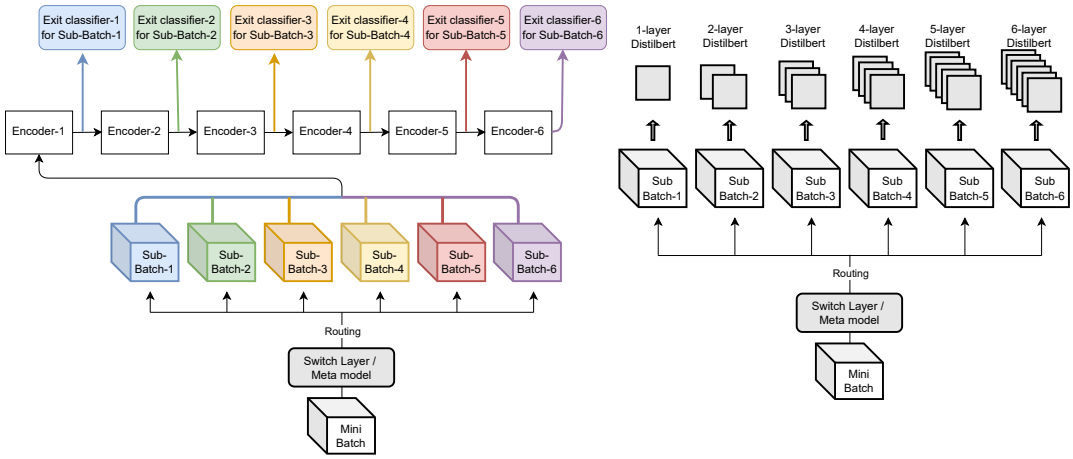

Figure 1: (left) $BE3R^{DistilBERT}_{Lean}$ architecture; (right) $BE3R^{DistilBERT}_{Sparse}$ architecture

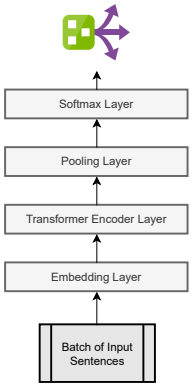

Figure 2: BE3R: Switch Layer Architecture

Due to the hard-routing during training, switch layer couldn't effectively learn the optimal layer for samples as it didn't have counter-factual information about whether the loss would have decreased or increased if it considered another expert unless it routed that sample to each and every expert atleast once. Also, the load-balancing is very restrictive to uniform distribution of samples across experts. These reasons led to the soft-routing objective with flexible auxiliary loss. These changes lead to better performance and throughput as we will observe in Section 4.3.

3.2.2 Soft Routing loss objective. During training, instead of restricting each sample to only a single expert, we route the sample to all experts, compute expert-wise loss for each sample and take a weighted average based on the route probabilities computed by the switch layer. This enables the switch layer to learn optimal layer for a given sample efficiently and improves the performance in comparison to hard-routing based objective. In PonderNet [4], we observe that a regularization loss wherein KL-Divergence between the distribution of halting probabilities and a prior distribution (a geometric distribution) is computed. The importance of regularization loss is that it promotes exploration and biases the network towards expected number of steps as per the prior distribution. However, PonderNet chose only Geometric distribution as prior distribution inline with their primary loss based on geometric halting probabilities. This regularization loss is more fleixible in comparison to load-balancing based loss and we extend it consider different prior distribution based on problem and expected performance-throughput trade-off. A key observation is that starting experts with less layers are less performant but have faster inference; ending experts with more layers are highly performant but have slower inference. Therefore, we can get good tradeoff between performance and inference time if most samples are appropriately routed to middle experts. Therefore, having Gaussian distribution as the prior distribution for KL Divergence loss should achieve high speed-ups without loss in performance and we will observe this in Section 4.3. If having high throughput is critical at acceptable performance drop, then having Geometric distribution can be considered as the prior distribution for KL Divergence loss; this will ensure most samples are routed to starting experts having fast inference. The full objective is the combination of 2 loss functions where β is the hyperparameter weighing the auxiliary loss. The soft-routing objective is given in equation 2:

$$\ell_{sr} = \sum_{j=1}^{k} \left(\sum_{i=1}^{E} L(\hat{y}_j^{e_i}, y_j) \cdot P_j^{e_i} + \beta \cdot KL\_Div(P_j^e, P^{target}) \right) \quad (2)$$

where:
1. L is the cross-entopy loss for a classification task and mean-squared loss for a regression task; E is the number of experts; k is the number of samples in a given mini-batch B.
2. P_j^e is the Probability distribution over different experts which is the output of Switch layer for sample j.

3. $P^{target} \in$ {Gaussian Distribution,Geometric Distribution,Uniform Distribution}

3.3 Inference

During Inference, we perform hard-routing wherein the switch layer predicts the expert a given sample should be routed to. In applications containing Millions/Billions of samples, we first pass them through the switch layer and group the samples based on the expert route. Then, we can either pass these sub-batches to the respective expert models in sequential or parallel manner based on hardware availability.

4 EXPERIMENTS AND RESULTS

In this Section, we analyse and study the impact of various architectural choices and training methodologies on relevance classification task as well as GLUE benchmark.

4.1 Datasets and evaluation metrics

Relevance Classification Dataset: Human-annotated relevance data from English speaking Amazon marketplace services comprising of (Query, Product Title, Relevance label) tuples used for learning binary relevance classification. We sample 60K instances (50k + | 10k -) as test data. Train and test datasets are based on random splits and no samples are common between train and test splits. For evaluating throughput, we sample 50M (Query, Product Title) samples from one day logs of Amazon India marketplace services. For the relevance classification task which is highly skewed, we report ROC-AUC score. For throughput analysis, we report the number of Million samples scored per hour by a given model/approach on a single GPU.

GLUE Benchmark: We test our proposed approaches on following tasks of GLUE benchmark [28]: Multi-Genre Natural Language Inference Matched/Mismatched (MNLI-m/MNLI-mm) [32], Quora Question Pairs (QQP) [1], Question Natural Language Inference (QNLI) [23], Stanford Sentiment Treebank (SST-2) [7], Corpus of Linguistic Acceptability (CoLA) [31], Semantic Textual Similarity Benchmark (STS-B) [26], Microsoft Research Paraphrase Corpus (MRPC) [10] and Recognizing Textual Entailment (RTE) [28]. WNLI [18] was excluded following previous works [9, 14, 35]. These span different types of tasks with varying difficulties and challenges covering natural language inference task, paraphrase similarity matching task and linguistic acceptability task. It involves classification as well as regression tasks wherein STS-B being the regression task. We report F1 score for MRPC and QQP tasks, Matthew's Correlation for CoLA task, Spearman Correlation for STS-B task and Accuracy for the remaining tasks. For speedup analysis, we carry out batch inference for each task on minimum 100K samples by upsampling test dataset if the number of samples in test dataset is less than 100K. We report percentage reduction in time taken for scoring as the speed-up.

4.2 Implementation Details

We implemented all the experiments using PyTorch [21] and HuggingFace [33]. For the relevance classification, we experiment with DistilBERT and BERT-base backbones. For BERT-base backbone, we experiment with the lean architecture. The reason being that sparse architecture for BERT-base has huge number of parameters which won't fit on a single GPU with 16GB memory whereas lean variant can be efficiently trained on a single GPU. For hard-routing objectives, alpha is set to 0.05. For soft-routing objectives, beta is set to 0.05. The small values for auxiliary losses ensure that they don't overwhelm the primary loss objective. For Gaussian prior distribution, mean (μ) is set to $(num\_layers + 1)/2$ and standard deviation (σ) is set to $\lfloor num\_layers/2 \rfloor$. Hence, ($\mu$, σ) is set to (3.5, 3) and (6.5, 6) for DistilBERT and BERT-Base backbones, respectively. For Geometric prior distribution, λ is set to $1/num\_layers$ so that the expected number of layers ($1/\lambda$) will be num_layers. For Uniform prior distribution, probability of each expert will be $1/num\_layers$. Relevance classification models were trained for 3 epochs using Adam optimiser [16] with learning rate 5e-5, batch size 64 and L2 normalization with weight decay set to 0.1 for parameters except bias and layernorm weights. Rest of the hyperparameters are same as the backbone architecture.

For the GLUE benchmark, we experiment with BERT-Large and Electra-Large backbones. We experiment with the lean architecture for these large model backbones so that we can efficiently train by fitting model on a single GPU. We experiment with the soft-routing objective with Gaussian prior distribution. The reason being better performance and throughput as reported in Section 4.3. For BERT-Large experiments, β was set to 0.2 and (μ, σ) were set to (12.5, 12) for all tasks except for SST-2 where (μ, σ) were set to (20, 4). Batch size was set to 16 for QNLI task, 8 for RTE task and 32 for remaining tasks. Adam optimizer was used with learning rate 3e-5 for MNLI and QNLI tasks, 2e-5 for the remaining tasks. Number of epochs was set to 1 for MNLI, QNLI and SST-2 tasks, 5 for CoLA task and 3 for remaining tasks. For BERT-Large, we carried out 5 runs with different seeds and report the median performance metrics and average speed-up. This is done as few datasets of GLUE Benchmark are too small and have high variability in performance metrics. For performance comparison in a given run, we compare the metrics of last expert which will be the entire backbone architecture with all layers with the metrics from our proposed BE3R model. For Electra-Large experiments, we report the metrics and speed-up for all tasks for a single run due to computational constraints. β was set to 0.05 for SST-2 task, 0.1 for CoLA task, 1 for RTE task, 0.4 for STS-B task and 0.2 for remaining tasks. (μ, σ) were set to (18, 6) for QNLI task, (20, 4) for SST-2 task and (12.5, 12) for remaining tasks. For CoLA and STS-B tasks, the shared parameters between experts were frozen for 1st epoch and only the weights of switch layer and classifiers of each expert were trainable; for subsequent epochs all parameters were trainable. This improved model stability for these 2 tasks. Batch size was set to 16 for QNLI task, 30 for STS-B task and 32 for remaining tasks. Adam optimizer was used with learning rate 3e-5 for MNLI task, 5e-5 for RTE task and 2e-5 for the remaining tasks. Number of epochs was set to 5 for MRPC and RTE tasks, 10 for CoLA and STS-B tasks, 1 for SST-2 task and 3 for remaining tasks.

All experiments were performed on p3.16xlarge EC2 instance on AWS wherein we use only 1 GPU. For all experiments, sequence length was fixed at 128. All hyperparameters were chosen empirically based on experiments performed.

4.3 Results

In this Section, we first report experimental results on product search relevance filtering application. In this regard, we compare various BE3R variants and baseline models on the task of relevance classification using ROC-AUC as performance metric and throughput gains in batch-inference setting. Then, we map the throughput gains to cost savings when applied for scoring 1 Billion samples in a batch setting. **We demonstrate the evidential results with respect to successful application of our proposed BE3R model to a large-scale real-world problem.**

On GLUE benchmark, we report similar performance-speed metrics and demonstrate the effectiveness of our proposed BE3R model across various NLP tasks, for classification as well as regression tasks and using 2 backbone models. This demonstrates that our proposed BE3R model is generic and can be applied to various tasks to get comparable performance at considerable speed-up.

In Sub-Section 4.4, we carry out thorough ablation studies pondering over important aspects of the modelling choices and answering them through experimental results.

Relevance Classification Task: We consider BERT-Base model as a baseline model. DistilBERT model has considerable performance drop on GLUE benchmark as stated in the [25]. However, for the relevance clasification task we didn't observe any performance drop when using DistilBERT in comparison to BERT-Base model. This is a great insight demonstrating that compact model performance can be at par with big models depending on the task. Hence, we used DistilBERT model as another baseline model. Table 1 shows the ROC-AUC scores on test dataset for baseline models as well as the BE3R variants using the corresponding baseline models as backbone. It also shows the throughput gains as measured using 50 M (Query, Product Title) pairs mentioned in Sub-Section 4.1. The cost for scoring 1 Billion samples in a batch setting are reported.

For DistilBERT baseline model, we used it as backbone to train BE3R variants. From Table 1 we can observe that Hard routing methodology had drop in ROC-AUC scores of about -0.78% for Sparse architecture and -2.13% for Lean architecture. On the contrary, in Soft routing methodology, there is no performance drop for Gaussian prior distribution based regularization loss for both Sparse and Learn architectures. For Uniform prior distribution based regularization loss in Soft routing methodology, we observe no performance drop in Sparse architecture and a slight drop of -0.14% in Lean architecture. For Geometric prior distribution based regularization loss in Soft routing methodology, we observe slight drop in ROC-AUC scores of about -0.5% and -0.71% for Sparse and Lean architectures, respectively. We can observe that Gaussian prior distribution based regularization loss is able to maintain the performance and achieve significant throughput gains by avoiding starting experts which are less performant and avoiding ending experts which have high latency . $BE3R_{Sparse}^{DistilBERT} - Gaussian$ achieves **35.89%** throughput gains over backbone DistilBERT baseline. $BE3R_{Lean}^{DistilBERT} - Gaussian$ achieves **23.93%** throughput gains over backbone DistilBERT baseline. Lowest throughput gains are achieved by Uniform prior distribution which is inline with expectation as considerable number of samples are routed to ending experts. Highest throughput gains are achieved by Geometric prior distribution as considerable number of samples are routed to starting experts. This explains the slight drop in ROC-AUC scores as starting experts are less performant due to lesser number of layers.

For BERT-Base baseline model, we used it as backbone to train BE3R variants using Soft routing methodology as it was having better performance when compared to Hard routing methodology. We train the Lean architecture as stated in Sub-Section 4.2. From Table 1 we can observe that $BE3R_{Lean}^{BERT} - Gaussian$ and $BE3R_{Lean}^{BERT} - Uniform$ have no drop in ROC-AUC scores while achieving significant throughput gains of **46.39%** and **10.24%** over backbone BERT-Base. $BE3R_{Lean}^{BERT} - Geometric$ has slight drop in ROC-AUC score of -0.93% but has astonishing throughput gains of **150.48%**.

Over the period of a year, large e-commerce website would score tens of Billions to Trillions of samples for a single application. This would mean, with respect to DistilBERT baseline model, the highest cost saving without performance drop is achieved by $BE3R_{Sparse}^{DistilBERT} - Gaussian$ which is **$ 0.28M** on 1 Trillion samples. With respect to BERT-Base baseline model, the highest cost saving without performance drop is achieved by $BE3R_{Lean}^{BERT} - Gaussian$ which is **$ 0.14M** on 1 Trillion samples. This is single application of transformer model in e-commerce website. The cost savings will further increase when applied to other applications such as review sentiment classification, named entity recognition (NER), product category prediction etc. Therefore, evident from the experimental results, BE3R model based on Sparse architecture and Soft routing methodology using Gaussian prior distribution has the highest throughput gains while maintaining the model performance. These significant cost savings without impacting the customer experience demonstrates the effectiveness of our proposed BE3R model in batch inference setting.

GLUE Benchmark: We consider BERT-Large and Electra-Large models as baselines to consider diverse models which weren't experimented in relevance classification task to analyse the generality of BE3R models. Based on learnings from relevance classification experiments, we experiment using Soft routing methodology using Gaussian prior distribution and use Lean architecture as these are large models. Implementation details are outlined in Sub-Section 4.2. Table 2 reports the task-wise and overall performance metrics of BE3R models and backbone/baseline models along with Speed-up of BE3R models over respective backbone/baseline models. We can observe that $BE3R_{Lean}^{BERT-Large} - Gaussian$ has comparable performance with respect to $BERT - Large$ model with a slight drop in Macro Score of -0.38% on Dev dataset and -0.8% on Test dataset. For certain tasks such as SST-2 and STS-B, there is no performance drop. In terms of Speed-up, we observe average gains of **23.82%** across all tasks. The highest throughput gains of **35.9%** is observed on QQP task which has the second highest amount of training dataset (363K) and highest amount of test dataset (390k). Similarly, $BE3R_{Lean}^{Electra-Large} - Gaussian$ has comparable performance with respect to $Electra - Large$ model with a slight drop in Macro Score of -0.21% on Dev dataset and -0.7% on Test dataset. For certain tasks such as CoLA and STS-B, there is no performance drop. In terms of Speed-up, we observe average gains of **22.19%** across all tasks. For STS-B, we observed peculiar behaviour wherein all samples were being routed to last expert on Dev dataset. The switch layer learnt that to maintain the optimal performance all the layers were

Table 1: ROC-AUC scores of BE3R variants and their respective backbone models on relevance classification test dataset. Throughput is measured on 50M samples.

Model	Prediction		Inference			Cost ($) per Billion samples
	ROC-AUC	Gain (%)	Million Samples/hr	Gain (%) over BERT-Base	Gain (%) over DistilBERT	
BERT-Base (Baseline)	0.965	-	1.66	-	-	440.00
DistilBERT (Baseline)	0.9655	0.06	3.26	-	-	224.05
Hard Routing						
$BE3R_{Sparse}^{DistilBERT}$	0.9575	-0.78	4	-	22.70	182.60
$BE3R_{Lean}^{DistilBERT}$	0.9444	-2.13	3.65	-	11.96	200.11
Soft Routing						
$BE3R_{Lean}^{BERT} - Gaussian$	0.96551	0.053	2.43	46.39	-	300.58
$BE3R_{Lean}^{BERT} - Uniform$	0.96774	0.28	1.83	10.24	-	399.13
$BE3R_{Lean}^{BERT} - Geometric$	0.956	-0.93	4.12	150.48	-	177.28
$BE3R_{Lean}^{DistilBERT} - Gaussian$	0.96538	0.04	4.04	-	23.93	180.79
$BE3R_{Lean}^{DistilBERT} - Uniform$	0.96369	-0.14	3.63	-	11.35	201.21
$BE3R_{Lean}^{DistilBERT} - Geometric$	0.96021	-0.50	5.05	-	54.91	144.63
$BE3R_{Sparse}^{DistilBERT} - Gaussian$	0.96572	0.07	4.43	-	35.89	164.88
$BE3R_{Sparse}^{DistilBERT} - Uniform$	0.96612	0.12	3.92	-	20.25	186.33
$BE3R_{Sparse}^{DistilBERT} - Geometric$	0.95811	-0.71	6.03	-	84.97	121.13

Table 2: Performance of BE3R models and their respective backbone models on GLUE benchmark. Speed-up of BE3R models over their respective backbone models.* & † indicates median and mean of 5 runs with different initialization seeds, respectively.

Model	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Macro Score
Dev Set										
$BERT - Large^*$	86.03	86.28	88.58	92.35	93.58	59.59	89.30	90.60	72.20	**84.28**
$BE3R_{Lean}^{BERT-Large} - Gaussian^*$	85.63	85.86	88.08	91.87	93.23	59.81	89.43	90.27	71.48	**83.96**
$Electra - Large$	90.18	90.19	89.85	93.79	96.79	67.83	92.28	91.44	86.64	**88.78**
$BE3R_{Lean}^{Electra-Large} - Gaussian$	89.56	89.29	89.61	93.74	96.79	68.26	92.28	91.47	86.28	**88.59**
Test Set										
$BERT - Large^*$	86.1	85.5	71.7	92.4	94.3	58.1	85.1	88.2	68.4	**81.1**
$BE3R_{Lean}^{BERT-Large} - Gaussian^*$	85.6	85.1	71.1	91.5	94.3	55.3	85.2	88.2	67.6	**80.4**
$Electra - Large$	89.8	89.1	73.7	94	96.9	70	91.6	90.6	85.7	**86.82**
$BE3R_{Lean}^{Electra-Large} - Gaussian$	89.3	88.6	73.2	93.9	96.8	70.3	91.6	90.1	82.1	**86.21**
Speed-Up with respect to backbone model										
$BE3R_{Lean}^{BERT-Large} - Gaussian\dagger$	17.23	22.21	35.90	28.11	11.96	26.17	24.97	22.31	25.50	**23.82**
$BE3R_{Lean}^{Electra-Large} - Gaussian$	26.74	26.35	36.21	27.16	13.92	25.96	0	34.41	8.99	**22.19**

necessary for all type of samples and hence the Speed-up was 0% for this task. The highest throughput gains of **36.21%** is observed on QQP task similar to $BE3R^{BERT-Large}_{Lean} - Gaussian$ model.

4.4 Ablation Studies

Soft Routing vs Hard Routing based loss function: From Table 1, we can observe that Hard routing methodology is having performance drop in contrast to Soft routing methodology with Gaussian/Uniform prior distribution which maintains the performance. This demonstrates that switch layer is able to learn the optimal expert for a given sample when it is provided with the loss from each expert. It helps the switch layer to update the route probabilities appropriately and in accordance with the chosen prior distribution.

The load balancing regularization loss for Hard routing methodology allows only for uniform distribution of samples across experts. This leads to low throughput gains in comparison to Soft routing methodology with Gaussian/Geometric prior distribution for KL-Div regularization loss. Specifically, Soft routing methodology with Gaussian/Geometric prior distribution have **13%-62%** higher throughput in comparison to Hard routing methodology.

Importance of Prior Distribution in Regularization loss: Gaussian prior achieves the best performance-throughput tradeoff. It does so by avoiding starting experts that are less performant and avoiding ending experts that have higher latency. As observed from Table 1 and Table 2, there is no performance drop in relevance classification task and 2-3 GLUE tasks. There is a marginal performance drop of -0.1% to -1% on GLUE tasks with training datasets containing more than 100K samples. We see throughput gains of **23.93%-46.39%** for the relevance classification task and average speed-up of **22.19%-23.82%** across GLUE tasks. Based on Table 1, Uniform prior has no performance drop except in lean setting with DistilBERT backbone where in there is marginal drop of -0.14%. It has the least throughput gains when compared to other priors. This is because considerable number of samples are routed to the ending experts having higher latencies. Geometric prior has the highest throughput gains but has slight performance drop of -0.5% to -0.93% on relevance classification task. Results being inline with the expectations that most samples are routed to initial experts having lower latencies and being less performant.

Sparse vs Lean Models: Lean models can be efficiently trained using single GPU for large models in contrast to Sparse models which are only limited to compact architectures like DistilBERT backbone. However, Sparse models have better performance in contrast to Lean models as can be observed for Table 1. Sparse models have **10%-30%** higher throughput over Lean models.

Insights into samples being routed across different experts: Table 3 shows the distribution of samples (%) across experts for various BE3R variants on relevance classification test dataset. Here, the BE3R variants trained using Soft routing methodology have been considered. Inline with the expectations, the distribution mimics the corresponding prior distribution to some extent. Qualitative analysis of samples revealed an interesting pattern that switch layer is routing based on categories of (query, product) pairs. It is learning that some categories are easier to make relevance judgements on and hence it routes them to experts with less layers. Remaining categories which are difficult to make relevance judgements on are being routed to experts with more layers. The difficulty of samples belonging to a category can be explained by the selection available in terms of brands, variety, attributes, age group and gender. E.g., with respect to $BE3R^{DistilBERT}_{Sparse} - Gaussian$ model, queries and products pertaining to lipsticks, TV, gaming, sports, diapers and sanitary pads are going to expert 0; men's watches/footwear/clothing/wallets, bluetooth speakers, Oppo and vivo phones are going to expert 1; jewellery, women's clothing, beauty products are going to expert 2; electronic appliances, power supplies and batteries, women's undergarments/footware/clothing, realme/redmi/apple phones and several other categories are going to expert 3; kids clothing/food supplements/car & bike accessories are going to expert 4.

Table 3: Distribution of samples (%) across experts for various BE3R variants on relevance classification test dataset

Model	Distribution of samples (%) across experts
$BE3R^{BERT}_{Lean} - Gaussian$	[0.0, 1.01, 0.02, 0.04, 0.38, 20.37, 76.56, 1.63, 0.0, 0.0, 0.0, 0.0]
$BE3R^{BERT}_{Lean} - Uniform$	[0.31, 0.65, 3.15, 0.93, 2.35, 1.75, 4.58, 8.0, 5.28, 1.63, 45.29, 26.08]
$BE3R^{BERT}_{Lean} - Geometric$	[21.4, 1.27, 7.93, 26.1, 28.44, 14.82, 0.04, 0.0, 0.0, 0.0, 0.0, 0.0]
$BE3R^{DistilBERT}_{Lean} - Gaussian$	[0.52, 0.61, 14.29, 84.58, 0.0, 0.0]
$BE3R^{DistilBERT}_{Lean} - Uniform$	[6.33, 4.65, 4.88, 29.39, 33.44, 21.3]
$BE3R^{DistilBERT}_{Lean} - Geometric$	[13.2, 12.9, 50.0, 23.9, 0.0, 0.0]
$BE3R^{DistilBERT}_{Sparse} - Gaussian$	[0.37, 0.46, 21.03, 76.89, 1.25, 0.0]
$BE3R^{DistilBERT}_{Sparse} - Uniform$	[5.29, 4.97, 4.66, 31.85, 39.36, 13.88]
$BE3R^{DistilBERT}_{Sparse} - Geometric$	[18.27, 32.88, 41.8, 6.25, 0.08, 0.72]

5 CONCLUSION

Pre-trained language models like BERT achieve good performance but have high computational costs hindering its widespread applicability. The existing early-exit methods are only applicable in adaptive setting, i.e., batch_size=1. This leads to poor hardware utilization making them highly inefficient for batch inference tasks involving Billions of samples. In this paper, we presented a novel deep learning based **BE3R** (**B**ERT based **E**arly-**E**xit using **E**xpert **R**outing) model which achieves significant throughput gains while maintaining performance with respect to critical task of search relevance filtering on a popular e-commerce website. We carry out thorough experiments of BE3R models on GLUE benchmark and demonstrate significant speed-up at comparable performance. Ablation studies are performed to better understand the importance of architectural and loss objective choices. Experimental results demonstrate the generality of BE3R model across tasks and thereby

enabling widespread applicability of large pre-trained language models at scale in batch inference setting.

REFERENCES

[1] Quora question pairs. https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs, 2018.

[2] Tensorrt. https://developer.nvidia.com/tensorrt., 2018.

[3] Bai, J., Lu, F., Zhang, K., et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019.

[4] Banino, A., Balaguer, J., and Blundell, C. Pondernet: Learning to ponder. *ArXiv abs/2107.05407* (2021).

[5] Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers, 2019.

[6] Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., and Weller, A. Rethinking attention with performers, 2021.

[7] Conneau, A., and Kiela, D. SentEval: An evaluation toolkit for universal sentence representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)* (Miyazaki, Japan, May 2018), European Language Resources Association (ELRA).

[8] Dai, Z., Lai, G., Yang, Y., and Le, Q. V. Funnel-transformer: Filtering out sequential redundancy for efficient language processing, 2020.

[9] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (Minneapolis, Minnesota, June 2019), Association for Computational Linguistics, pp. 4171–4186.

[10] Dolan, W. B., and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)* (2005).

[11] Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

[12] Goyal, S., Choudhury, A. R., Raje, S. M., Chakaravarthy, V. T., Sabharwal, Y., and Verma, A. Power-bert: Accelerating bert inference via progressive word-vector elimination, 2020.

[13] Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network, 2015.

[14] Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. Tinybert: Distilling bert for natural language understanding, 2020.

[15] Khattab, O., and Zaharia, M. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*. Association for Computing Machinery, New York, NY, USA, 2020, p. 39–48.

[16] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Y. Bengio and Y. LeCun, Eds.

[17] Kitaev, N., Łukasz Kaiser, and Levskaya, A. Reformer: The efficient transformer, 2020.

[18] Levesque, H. J., Davis, E., and Morgenstern, L. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning* (2012), KR'12, AAAI Press, p. 552–561.

[19] Liu, W., Zhou, P., Zhao, Z., Wang, Z., Deng, H., and Ju, Q. Fastbert: a self-distilling bert with adaptive inference time, 2020.

[20] Nogueira, R., and Cho, K. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085* (2019).

[21] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[22] Qiu, J., Ma, H., Levy, O., tau Yih, S. W., Wang, S., and Tang, J. Blockwise self-attention for long document understanding, 2020.

[23] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (Austin, Texas, Nov. 2016), Association for Computational Linguistics, pp. 2383–2392.

[24] Reimers, N., and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[25] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[26] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (Seattle, Washington, USA, Oct. 2013), Association for Computational Linguistics, pp. 1631–1642.

[27] Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., and Zhou, D. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (Online, July 2020), Association for Computational Linguistics, pp. 2158–2170.

[28] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP* (Brussels, Belgium, Nov. 2018), Association for Computational Linguistics, pp. 353–355.

[29] Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity, 2020.

[30] Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020.

[31] Warstadt, A., Singh, A., and Bowman, S. R. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics 7* (2019), 625–641.

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

[33] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. Huggingface's transformers: State-of-the-art natural language processing. *CoRR abs/1910.03771* (2019).

[34] Xin, J., Tang, R., Lee, J., Yu, Y., and Lin, J. Deebert: Dynamic early exiting for accelerating bert inference, 2020.

[35] Xu, C., Zhou, W., Ge, T., Wei, F., and Zhou, M. Bert-of-theseus: Compressing bert by progressive module replacing. In *EMNLP* (2020).

[36] Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. Big bird: Transformers for longer sequences, 2021.

[37] Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., and Wei, F. Bert loses patience: Fast and robust inference with early exit, 2020.