Learn To be Efficient: Build Structured Sparsity in Large Language Models

Haizhong Zheng [†] **Xiaoyan Bai**[†] **Xueshen Liu**[†]

Z. Morley Mao[†] **Beidi Chen**[‡] **Fan Lai**[§] **Atul Prakash** [†]

[†]University of Michigan [‡]Carnegie Mellon University
[§] University of Illinois Urbana-Champaign
{hzzheng, smallyan, liuxs, zmao, aprakash}@umich.edu ,
beidic@andrew.cmu.edu, fanlai@illinois.edu

Abstract

Large Language Models (LLMs) have achieved remarkable success with their billion-level parameters, yet they incur high inference overheads. The emergence of activation sparsity in LLMs provides a natural approach to reduce this cost by involving only parts of the parameters for inference. However, existing methods only focus on utilizing this naturally formed activation sparsity in a post-training setting, overlooking the potential for further amplifying this inherent sparsity. In this paper, we hypothesize that LLMs can *learn to be efficient* by achieving more structured activation sparsity. To achieve this, we introduce a novel training algorithm, Learn-To-be-Efficient (LTE), designed to train efficiency-aware LLMs to learn to activate fewer neurons and achieve a better trade-off between sparsity and performance. Furthermore, unlike SOTA MoEfication methods, which mainly focus on ReLU-based models, LTE can also be applied to LLMs like LLaMA using non-ReLU activations. Extensive evaluation on language understanding, language generation, and instruction tuning tasks show that LTE consistently outperforms SOTA baselines. Along with our hardware-aware custom kernel implementation, LTE reduces LLaMA2-7B inference latency by 25% at 50% sparsity. We make our code publicly available at GitHub[1].

1 Introduction

The exponential growth in data volumes and model sizes has catalyzed significant breakthroughs in large-scale models, enabling a wide range of applications [2, 47, 53, 48, 30]. Among them, large language models (LLMs), like GPT-3 [2], OPT [47], and LLaMA [38, 39], have demonstrated impressive natural language ability. However, the skyrocketing number of model parameters [21, 36] and dataset size [51, 45, 52, 44] has introduced challenges in further scaling those models. The exponential growth in model size has not only inflated the deployment costs of LLMs, due to their significant computational and

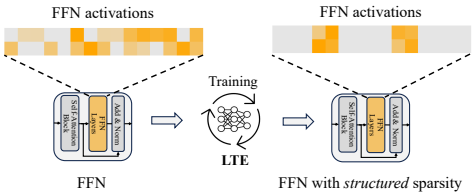

Figure 1: Learn-To-be-Efficient (LTE) performs efficiency-aware training to construct structured model contextual sparsity for fast inference. Activated neurons are in orange.

[1] https://github.com/haizhongzheng/LTE

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

memory demands during inference, but also affected the user experience in many latency-sensitive applications, such as chatbots [1] and autonomous driving [17, 35]. Recent advances have been leveraging sparsity to improve LLM inference efficiency, including model weight quantization [9] and pruning [22], token sparsity [50, 46], and activation sparsity [21, 49].

During LLM inference, Feed Forward Network (FFN) layers are the primary efficiency bottleneck, accounting for over 60% of the FLOPs and I/O operations [21], but they exhibit high activation sparsity, especially in ReLU-based LLMs [18]. For example, in a 175B OPT model, more than 95% of activations in FFN layers are zeros [21]. Recent advances leverage this activation sparsity via *MoEfication* [49, 25]: they convert FFN layers to MoE layers by grouping neurons in the FFN intermediate layer into n experts and then applying a router to select k most important experts for each input token. Unlike training the MoE model from scratch, MoEfication converts FFN layers in pretrained LLMs into MoE layers with relatively small training resources.

However, existing advances only focus on manipulating the pre-trained activation sparsity of neurons, overlooking the potential for further increasing this inherent sparsity. Moreover, they are limited to the historically ReLU-based LLMs, whereas emerging advanced models use soft activation functions for better model quality (e.g., SwiGLU in LLaMA [31] and GeGLU in Gemma [37]), exhibiting much lower natural activation sparsity. While existing works suggest replacing the model activation with ReLU to enable sparsity [49, 25], we show that this can hurt model performance (Section 5.2).

In this paper, we hypothesize that LLMs can learn to be efficient and achieve more structured activation sparsity. As shown in Figure 1, our key insight is that, without compromising model quality, *we can also train LLMs to be efficiency-aware by developing more structured sparsity, which is more friendly for achieving hardware speedup.* However, creating structured sparsity in pretrained LLMs is non-trivial due to two practical challenges:

1. *How to train routers more stably?* The widely-used Top-k Softmax routing can lead to a severe accuracy drop (Section3.2). How to jointly train the model and routers in a MoEfication setting is still open-ended.
2. *How to select the right experts in serving?* The number of experts needed in MoE layers depends on specific inputs and layers, implying a trade-off between model efficiency and quality.

To address these challenges, we introduce Learn-To-be-Efficient (LTE), a novel training algorithm to train efficiency-aware models. LTE integrates an efficiency loss penalty, encouraging models to activate fewer neurons in their FFN layers while keeping good task performance. Additionally, LTE adopts a threshold-based Sigmoid routing strategy to select experts and employs a two-stage training mechanism to improve training stability. LTE achieves a more flexible selection of experts instead of selecting a fixed number of experts for all layers and inputs. Same as Deja Vu [21] and moefication [49], LTE provides very structured sparsity. We further develop a custom CUDA kernel with Triton, a Python-like CUDA programming language, to enable wall-clock time speedup from such structured sparsity (Section 4.3).

We evaluate LTE on both encoder-based models (RoBERTa$_{\text{base}}$, RoBERTa$_{\text{large}}$ [20]) and decoder-based models (GPT2-Medium [28], LLaMA2-7B [38]) with the HuggingFace's transformers. Our extensive experiments on Natural Language Understanding (NLU) tasks, downstream Natural Language Generation (NLG) tasks, and instruction tuning tasks, show that LTE consistently outperforms state-of-the-art designs. For instance, LLaMA with LTE provides a 1.83x - 2.59x FLOPs speed-up on NLG tasks without compromising model quality. After integrating our hardware-efficient implementation of sparse matrix multiplication kernel, LTE reduces LLaMA2-7B 25% wall-clock time latency at around 50% sparsity. Our evaluation results demonstrate that LTE effectively builds more structured sparsity even on LLMs with soft activation functions.

2 Related Work

Mixture of Experts (MoE). MoE was proposed by [14] a few decades ago to build an integral system with subset networks. Recently, MoE layers have been used in the Transformer architecture as a substitute for the MLP block [32, 7], with the recent Mistral-7B model [15] as a prominent example. In MLP blocks of MoE transformers, instead of using an FFN layer for calculating output, MoE layers construct multiple smaller FFN layers and employ a router to choose a subset of experts to do conditional computation. Even though transformers can have billions of parameters, only a subset of

experts are activated for each token, thus effectively increasing model activation sparsity (i.e., the number of skipped neurons in execution).

LLM Contextual Sparsity. Recent studies [18, 21] show that trained ReLU-based transformers naturally show a great sparsity in the activation of FFN layers. For example, in a 175B OPT model, more than 95% activations in FFN layers are zeros [21]. While SOTA models are increasingly using diverse soft activations, like GeLU and SwiGLU, existing work shows that replacing soft activations with ReLU and fine-tuning the model can increase the activation sparsity without greatly hurting model quality [49, 25, 18]. Yet, we find that this benefit is not consistent for all datasets (Section 5.2). This emerging sparsity indicates that a large portion of neurons are unnecessary in LLM inference (i.e., sparsity), which we can leverage to improve LLM inference efficiency like using MoEfication [49]. Recent papers [21, 49] show that a single FFN layer in pretrained LLMs can be converted to an MoE layer, by splitting the matrices of FFN layers into exclusive sub-matrices and employing a router to activate only neurons in a subset of experts. Similar to MoE models, MoEfication can effectively accelerate LLM inference by only using part of the parameters [49, 21].

Model Weight Sparsity. Another orthogonal direction regarding model sparsity is static model weight sparsity [8, 36], which often prunes the model weights and keeps them static during inference. For instance, Wanda [36] estimates model weight importance and prunes unimportant weights, so all inputs will use the same subset of weights. In contrast, contextual sparse models select a different subset of weights for different inputs. Nonetheless, model pruning can be applied to contextual sparse models too, meaning a complementary optimization to contextual sparsity.

3 Background and Motivation

In this section, we first briefly introduce the MoEfication of FFN layers in transformers. Then, we present a study on the limitation of applying noisy top-K Softmax routing in the Moefication setting.

3.1 Background: MoEfication

MoEfication [49] is a way to group neurons in FFN layers, thereby converting FFN layers to MoE layers for better execution speedup. For a transformer whose hidden states and FFN intermediate dimension are d_{model} and d_{FFN}, respectively, the FFN layers process the input as follows:

$$h = xW_1 + b_2$$
$$FFN(x) = \sigma(h)W_2 + b_2$$

where $x \in \mathbb{R}^{d_{model}}$, W_1, W_2 are weight metrics, b_1, b_2 are biased term and σ is an activation function. MoEfication aims to group d_{FFN} neurons in the FFN intermediate layer into n experts. Then, a router is trained for each FFN layer to select k experts (k < n), thus reducing activation load, to speed up inference. A recent work, Deja Vu [21], uses a similar setting but treats each neuron as an expert.

However, SOTA MoEfication methods overlook the potential for further optimizing the activation sparsity of LLMs. Besides, for LLMs employing soft activations such as GeLU [12] and SwiGLU [31], MoEfication [49] proposes to replace soft activations with ReLU and fine-tune the model to improve activation sparsity. Although recent works [49, 25] show that this replacement has a marginal impact on model performance, we notice that this hypothesis is not widely applicable (Section 5.2).

3.2 Limitations of Noisy Top-K Softmax Routing

One potential solution to train and convert the model into MoE layers is noisy top-K Softmax routing [7, 33]. It selects k experts based on the highest router outputs and then calculates Softmax values for these selected outputs. The output of the MoE layer is determined by summing the products of these softmax values with their corresponding experts' outputs. Additionally, a small Gaussian noise is added to the router's outputs during training to encourage exploration across different experts, which addresses the issue of unselected experts being non-differentiable.

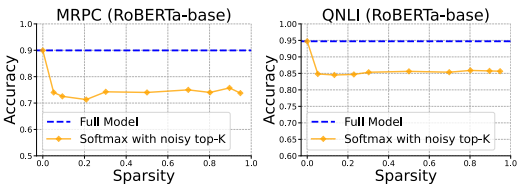

Figure 2: Models trained with noisy top-K Softmax routing experience severe accuracy drops, even at very low levels of sparsity.

To study the effectiveness of noisy top-K Softmax routing, we follow the standard MoEfication setting [49] to form experts in FFN layers: we fine-tune a model on a downstream dataset and group neurons into experts via parameter clustering. Subsequently, we integrate Softmax routers into models and jointly train both the model and routers. As shown in Figure 2, we observe noticeable accuracy drops even at very low sparsity levels. When diving into the expert scores produced by routers, we observe a very biased expert score allocation: only one expert in an MoE layer has an expert score close to 1, while other experts receive nearly 0 scores. This results in only one expert contributing to the inference in each MoE layer, leading to performance drops.

A potential reason behind this biased allocation is that the sum of expert scores in Softmax routing is constrained to 1. Unlike traditional MoE models, which typically select 1-2 experts [32, 7], MoEfied models need to select a much larger number of experts [21, 49], but the sum of experts score is still 1; this can complicate the allocation of the Softmax budget and cause a biased allocation.

4 Methodology: Learn To be Efficient

In this section, we present LTE (Learn To be Efficient) to learn more structured MoE in FFN layers for fast inference. Specifically, we tackle two key challenges toward practical MoEfication for diverse LLMs: (1) How to train routers more stably? (Section 4.2); and (2) How to select the right experts in serving? (Section 4.2.2).

4.1 Experts Grouping

The first step of LTE is to group neurons in FFN layers to form experts. Expert grouping aims to group neurons that are often activated together, thereby improving efficiency by reducing the total number of activated neurons. Here, constructing too many experts (e.g., each neuron as an expert) can significantly increase the cost, while too few can lead to poor model quality. Moreover, we need to group neurons without a significant performance drop on the pre-trained model.

Following the previous work [49], we group 32 neurons as an expert in FFN layers, and use parameter clustering to group MLP neurons into different experts. Fundamentally, in FFN layers, each neuron is associated with a column of W_1 (which is a d_{model} vector). Parameter clustering first treats the corresponding vector in W_1 as the feature for a neuron, then applies the balanced K-Means algorithm [23] to cluster neurons into n clusters. Each cluster has 32 neurons by default and will be treated as an individual MoE expert.[2] We also conduct an ablation study on other alternative grouping strategies in Section 5.5.

4.2 Adaptive Expert Routing

4.2.1 Router Design

After constructing the experts, we need to decide on the right routing strategy. Similar to existing MoE designs, we use a fully-connected layer as the router network. The input to the router is the output of the preceding self-attention block, and the router output is the expert score for each expert.

Routing function. To address the biased expert score issue (Section 3.2), we employ the Sigmoid function on router outputs to get expert scores. Unlike the Softmax function, the Sigmoid function computes each expert score independently, which circumvents the biased expert score due to the constraint on the sum of outputs in the Softmax function:

$$G(x)_i = Sigmoid(x \cdot W_{g,i}) = \frac{1}{1 + e^{-x \cdot W_{g,i}}}, \tag{1}$$

where x is the input to the corresponding FFN layer, i is the index for the expert, and $W_{g,i}$ is the router network weight for the i-th expert.

Threshold-based experts selection. SOTA methods [49, 7] select a fixed number of experts for all layers and inputs. However, the necessary number of experts may differ depending on inputs and layers. Here, we propose to select experts more adaptively for different inputs and layers for a

[2] For FFN layers using SwiGLU (e.g., LLaMA), there are two linear networks to calculate neuron values: $SwiGLU(x) = (Swish(xW) \otimes xV)W_2$. We use the weight of the gate network, W, to cluster neurons.

better trade-off between model sparsity and quality, by leveraging a threshold-based method to enable adaptive selection. The MoEfied FFN layers become:

$$FFN(x) = \sum_{i=1}^{n} \mathbf{1}_{\{G(x)_i > \tau\}}(x) E(x)_i, \tag{2}$$

where $E(x)_i$ is the output for the i-th expert, $\mathbf{1}(\cdot)$ is the indicator function, and τ is a predefined threshold. Only experts with score larger than τ will be activated. Consequently, within the same MoE layer, as the router generates larger expert scores, more experts are selected.

4.2.2 Two-stage LTE Training

Training the router poses three practical challenges. Firstly, as we consider training experts independently to mitigate bias (e.g., using a Sigmoid routing function), it is difficult to tease apart important experts from the rest. Secondly, the non-differentiability of threshold-based expert selection further complicates router training. Lastly, setting thresholds for each MoE layer is non-trivial.

To tackle these challenges, we next propose a novel two-stage training algorithm with an efficiency loss penalty to MoEfy LLMs. Our training algorithm consists of two training stages: *a) model-router training stage* and *b) model adaption stage*.

Stage 1: Model-router training. In this stage, we jointly train routers and model parameters to capture the importance of experts for given inputs in the Sigmoid routers. To address the non-differentiability issue, we switch the expert selection to a "soft" mode:

$$FFN_{soft}(x) = \sum_{i=1}^{n} G(x)_i E(x)_i. \tag{3}$$

Instead of selecting a subset of experts, we always select all experts and multiply expert outputs $E(x)_i$ with the corresponding expert score $G(x)_i$ to make both router and model differentiable. Since there is no discrete selection in MoE layers, all parameters are trained for each iteration.

Compared to Softmax routers, Sigmoid routers score each expert independently and do not introduce any competition on expert scores among experts. This makes Sigmoid routers alone cannot identify more important experts in MoE layers. As such, we design an efficiency loss penalty, $\mathcal{L}_{efficiency}$, to introduce competition among experts for Sigmoid routers:

$$\mathcal{L}_{efficiency} = \frac{1}{LN} \sum_{l=1}^{L} \sum_{i=1}^{N} |G_l(x)_i|^2, \tag{4}$$

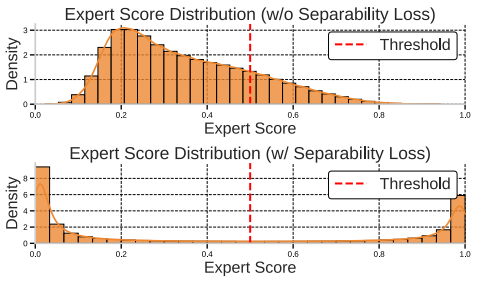

Figure 3: Expert distribution w/ and w/o separability loss. The expert scores in the model trained with the separability loss are much more separable than the model trained without the separability loss.

where L is the number of layers, and N is the number of experts in each layer. The efficiency loss calculates the mean of expert scores across all layers. This efficiency loss penalizes the output magnitudes of routers, driving routers to selectively allocate lower expert scores to less important experts, which helps distinguish experts with different importance. Moreover, instead of assigning the same sparsity for all layers, the efficiency loss also induces inter-layer orchestration, allowing LTE-trained models to allocate adaptive sparsity for different layers. We show that LTE models are capable of adaptively allocating sparsity across different layers (Figure 16).

For the threshold in Equation 2, instead of choosing a threshold for each router, we propose to select a predetermined fixed threshold and then train models to fit this threshold, i.e., train models to be threshold-aware. Specifically, we use a threshold separability regularizer to drive models to make expert scores more separable for this given threshold:

$$\mathcal{L}_{separability} = \frac{1}{LN} \sum_{l=1}^{L} \sum_{i=1}^{N} \frac{1}{(G_l(x)_i - \tau)^2}, \tag{5}$$

where τ is a predefined threshold for all routers (we set $\tau = 0.5$ for all experiments). The separability loss encourages router outputs to diverge from the threshold τ, which makes the router outputs more separable. An example is illustrated in Figure 3. The only difference between the models in the two figures is if the model is trained with a separability loss. We find that the expert scores are not separable when training without the separability loss, making it hard to decide the threshold. However, training with the separability loss markedly improves the separability of expert scores, allowing us to choose experts with the predefined threshold.

Combined with the task performance loss L_{task}, our training loss in model-router training stage is:

$$\mathcal{L}_{s1} = L_{task} + \eta L_{efficiency} + \lambda L_{separability}. \tag{6}$$

The efficiency coefficient, η, is used to control the trade-off between inference efficiency and task performance, and η can be used to control the sparsity of trained models. A higher η indicates a larger inference cost penalty, leading to a more sparse model. λ is the separability loss, we set $\lambda = 0.5$ for our evaluation. We conduct an ablation study on both hyperparameters in Section 5.5.

Stage 2: Model Adaptation. To accelerate model inference, we need to switch routers to discrete selection mode (Equation 2), and the model trained in Stage 1 needs further training to adapt to the changes in router outputs. In the model adaptation stage, we freeze the router parameters, switch routers to discrete selection mode (Equation 2), and fine-tune the model to adapt to the discrete routing with the task performance loss: $\mathcal{L}_{s2} = L_{task}$.

4.3 Hardware-efficient Implementation

As illustrated in Figure 4, same as Deja Vu [21] and moefication [49], LTE provides very structured sparsity. After selecting neurons in intermediate layers, a CUDA kernel needs only to load the relevant columns and rows of W_{up} and W_{down} from GPU global memory to SRAM to compute dot product, thus reducing memory-I/O as well as computational overheads. Thanks to the structured sparsity provided by LTE, the kernel avoids non-coalesced memory access by storing column-major W_{up} and row-major W_{down}. In this paper, we use Triton 2.3.0 to implement a customized MLP layer to translate the sparsity to wall-clock time latency reduction. The evaluation of our custom kernel is presented in Section 5.3.

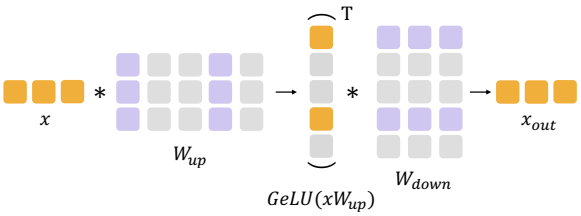

Figure 4: Structural sparsity provided by LTE in FFN layer. After selecting neurons, unselected rows and columns won't be loaded and used.

5 Experiment

In this section, we conduct extensive evaluationS to verify the effectiveness of LTE. For instance, in Section 5.2, LLaMA with LTE provides a 1.83x - 2.59x FLOPs speed-up on NLG tasks. Along with our hardware-aware custom kernel, LTE reduces 25% LLaMA2-7B inference latency at 50% sparsity.

5.1 Experimental Setting

Models. We implement LTE on both encoder-based models (RoBERTa$_{\text{base}}$, RoBERTa$_{\text{large}}$) and decoder-based models (GPT2-Medium, and LLaMA-7B) with the HuggingFace's transformers. Following previous work [49], we set each expert to contain 32 neurons in FFN layers for all models.

Datasets: 1) Natural Language Understanding (NLU): we evaluate on eight tasks from the GLUE dataset [40] SST-2 [34], RTE [5], CoLA [42], MNLI [43], QNLI [29], QQP [13], STS-B [3], and MPRC [6]. We report the Pearson correlation coefficient for STS-B, the Matthews correlation coefficient for CoLA, and the accuracy for the other six datasets. **2) Natural Language Generation (NLG)**: we evaluate LTE on E2E [27], XSum [26], and Wikitext103 [24]. For both E2E and XSum, we report the ROUGE-L [19] to measure the generation text quality and report the perplexity performance for the Wikitext dataset. **3) Instruction Tuning**: besides downstream tasks, we also

evaluate LTE on instruction tuning tasks to evaluate LTE's generalization capabilities. We use Tulu dataset [41] to perform instruction tunning with LTE, and we evaluate models with MMLU benchmark [11]). MMLU benchmark is a comprehensive evaluation dataset designed to evaluate the generalization capabilities of LLMs and consists of 15908 questions from 57 distinct tasks.

Baselines. We compare two baselines with LTE in our paper: (1) *MoEfication* [49]: MoEfication proposes to replace soft activations with ReLU and then fine-tune the model on downstream datasets. In our evaluation, we report the number of MoEfication with parameter clustering and ground-truth expert selection. (2) *Deja Vu* [21]: Deja Vu employs predictors to estimate the values of neurons in intermediate layers, pruning neurons with absolute magnitudes (absolute values for non-ReLU activations). The models used in Deja Vu evaluation are models fine-tuned on specific tasks as well.

5.2 End-to-end Performance Comparison

LTE, MoEfication, and Deja Vu provide the same type of structured sparsity in FFN layers. For the convenience of comparison, we use FFN neuron sparsity as a measure of efficiency in this section. We will further discuss how our hardware-efficient implementation translates this sparsity into wall-clock time latency reduction in Section 5.3. For our method (LTE), we get models with different sparsity by tuning efficiency loss η in Equation 6.

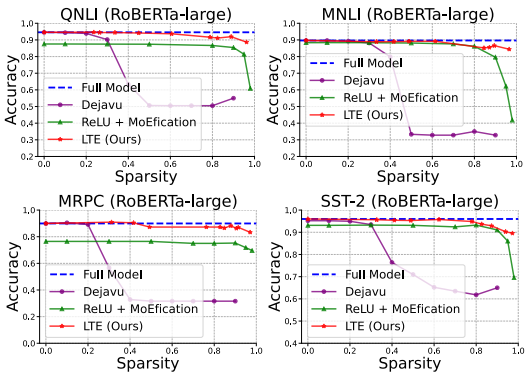

Figure 5: LTE consistently outperforms other baselines on 4 NLU datasets.

No performance drop with 80-95% sparsity on NLU tasks. We first evaluate our methods on eight NLU Tasks in GLUE datasets[40] with two encoder-based models ($\text{RoBERTa}_{\text{base}}$ and $\text{RoBERTa}_{\text{large}}$). Due to the space limit, we report four datasets (MRPC, SST2, QNLI, and MNLI) on $\text{RoBERTa}_{\text{large}}$ in Figure 5 and other results in Figure 14 in Appendix C.

As shown in Figure 5 and Figure 14, our method (LTE) consistently outperforms the baselines on all datasets. MoEfication exhibits a performance drop when sparsity reaches 70%–80%, and Deja Vu experiences performance drop at a relatively low sparsity level (e.g., 30%), while LTE maintains good performance even at higher sparsity levels (>90%). Moreover, we notice that while ReLU-based models achieve high sparsity with a slight decrease in performance, replacing GeLU with ReLU in RoBERTa can affect performance on some datasets. For instance, replacing GeLU with ReLU in a RoBERTa model results in around 10% accuracy drop on the MRPC dataset.

Table 1: GFLOPs per token for LLaMA-7B on different datasets with permitting quality drops. FLOPs of routers are also included in our method. N/A stands for the method failing to achieve the expected performance.

	XSum	E2E	Wiki
Full	12.06	12.06	12.06
Deja Vu	7.92	6.42	7.92
MoEfication	10.45	N/A	N/A
R-LLaMA+MoE	8.27	7.39	11.1
LTE (Ours)	**5.38**	**4.65**	**6.59**

50% FLOPs saving on NLG tasks. We next evaluate LTE with decoder-based models on three types of generation tasks. Similar to NLU tasks, Figure 6 shows that LTE outperforms all other baselines. NLG tasks are more challenging: all methods start to have a performance drop at a lower sparsity level compared to NLU tasks. Yet, LTE-trained models have better resilience to the sparsity increase. Especially at a high sparsity level, while other baselines fail to generate meaningful content, LTE still generates content with a marginal quality drop. Similarly, we also observe the performance drop caused by replacing soft activation with ReLU.

Additionally, to better understand the computation saving of LTE, we compare the FLOPs per token of different methods on LLaMA-7B, allowing an up to 0.05 ROUGE-L decrease for XSum and E2E, and an up to 0.5 perplexity (PPL) increase for WikiText-103. It is noteworthy that the FLOPs for routers are included in our method, constituting approximately 1% of the total FLOPs of FFN layers. The results are presented in Table 1. The evaluation results show that LTE provides a 1.83x - 2.59x FLOPs speed-up and gives the most FLOPs savings among all methods.

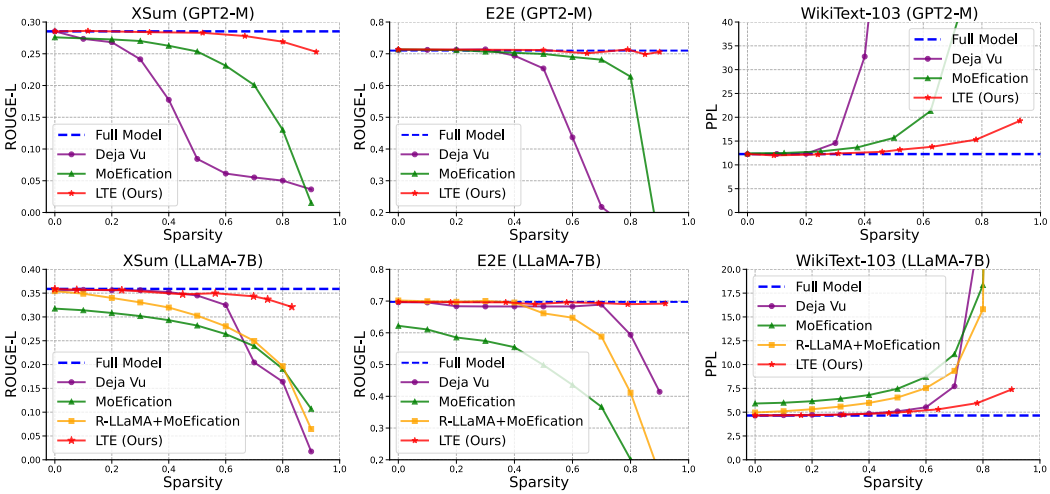

Figure 6: Performance comparison across three NLG datasets (each column). We compare LTE (Ours) with other baselines on two decoder-based models: GPT2-M and LLaMA-7B (each row).

LTE Outperforms SOTA on instruction tuning. To further evaluate the effectiveness of LTE, we evaluate LTE as a chatbot and verify model performance in a few-shot learning scenario. We fine-tune LTE on the Tulu instruction tunning dataset [41], and then evaluate the few-shot performance on MMLU benchmark [11]. We use LLaMA2-7B as the base model for LTE training. For a fair comparison, we also fine-tune base models with the Tulu dataset for other baselines. Specifically, for Deja Vu, we use Tulu-fine-tuned LLaMA2-7B as the base model. For Moefication, we use Tulu-fine-tuned ReLULLaMA-7B as the base model to ensure the best possible baseline performance. We present the 5-shot MMLU accuracy comparison in Figure 7. The evaluation results show that LTE outperforms other baselines across all sparsity levels. Yet, LTE shows a weaker sparsity-performance trade-off compared to downstream tasks. Our hypothesis is that LTE routers change the structure of the original LLaMA model, which influences general language patterns learned by models in pretraining. Instruction tuning alone is not enough to fully recover this ability (considering instruction tuning has much smaller datasets). We believe that training with a wider variety of data (like pretraining data) will further improve LTE performance.

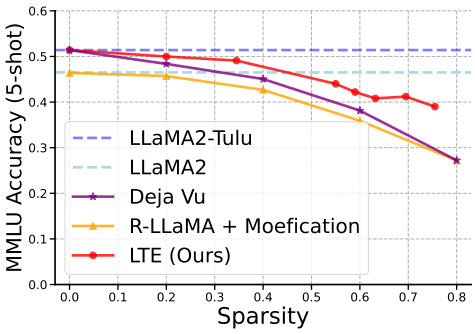

Figure 7: LTE outperforms other baselines on 5-shot MMLU benchmark.

5.3 Translate Sparsity to Wall-clock Time Speedup

25% wall-clock time latency reduction. As discussed in Section 4.3, the sparsity introduced by LTE is highly structured and can be easily translated to latency reduction. This section demonstrates that our custom Triton kernel effectively translates the LTE sparsity to wall-clock time speed up. We evaluate wall-clock time speed up using LLaMA2-7B on a single 3090Ti. We also reported vanilla Pytorch indexed matrix multiplication. Due to the requirement for

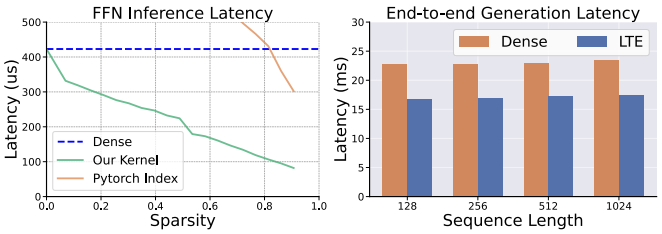

Figure 8: Wall-clock time latency comparison on dense FFN blocks and our custom Triton kernel (Left). End-to-end generation latency comparison between dense model and LTE with 50% sparsity (Right).

Table 2: Performance comparison with model pruning. We apply Wanda [36] on WikiText103 fine-tuned LLaMA-7B with the author-implemented code. Besides using C4 as the calibration data suggested by the Wanda paper, we also try to use WikiText for calibration to improve this baseline. The sparsity reported here is the overall sparsity of the entire model, which is different from the FFN sparsity. The evaluation results show that LTE achieves a lower perplexity than Wanda.

Method	Wanda (2:4)	Wanda (4:8)	Wanda (Unstructured)	Wanda (2:4)	Wanda (4:8)	Wanda (Unstructured)	LTE ($\eta = 2$)
Overall Sparsity	50%	50%	50%	50%	50%	50%	52%
Calibration Data	C4	C4	C4	Wiki	Wiki	Wiki	-
PPL	11.45	8.39	7.04	10.82	8.17	6.87	**5.95**

new memory allocations, slicing in Pytorch is very time-consuming. This makes indexed matrix multiplication even slower than dense matrix multiplication at most sparsity levels. We report the wall-clock time latency comparison in Figure 8 (noting that the router inference latency is already included in LTE latency). Compared to a dense FFN block, our custom Triton kernel achieves nearly linear speed-up with respect to sparsity (Left). At around 50% sparsity, LTE reduces end-to-end generation latency by approximately 25%.

5.4 Addtional Comparison

Model pruning. As discussed in Section 2, besides dynamic contextual sparsity provided by LTE, static sparsity provided by model pruning is also a common method to accelerate model inference. In this section, we present a performance comparison between Wanda [36] and LTE in Table 2. The sparsity reported here is the overall sparsity of the entire model, which is different from the FFN sparsity reported in Seciton 5.2. For LTE, we recalculate the overall sparsity based on the FFN sparsity. The evaluation results show that LTE achieves a lower perplexity than Wanda. The evaluation results show that, given a similar sparsity, LTE achieves better perplexity.

5.5 Ablation Study

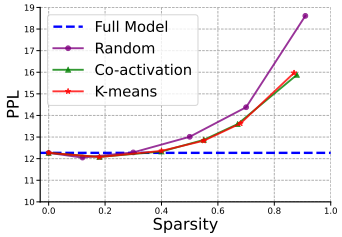

Figure 9: Performance comparison between different experts grouping algorithms.

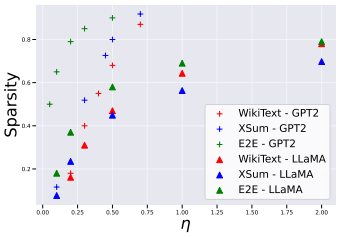

Figure 10: The relation between η and sparsity on different models and tasks.

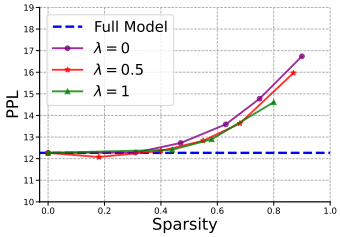

Figure 11: Performance comparison between different λ on WikiText-103 (GPT2-M).

Experts grouping algorithms. Three different expert grouping algorithms are explored in [49]: random grouping, parameter K-means clustering, and co-activation graph split. The co-activation graph split algorithm constructs a graph where each neuron represents a node and the edges denote the frequency of co-activation between neuron pairs. This graph is subsequently divided into subgraphs using the graph split algorithm [16], with each subgraph representing a group of neurons. To study the effectiveness of these grouping algorithms, we compare their LTE performance by fine-tuning GPT2 on the WikiText-103 dataset. As shown in Figure 9, both co-activation graph split and parameter K-means grouping have very similar performance (similar to finding in [49]) and outperform random grouping. Given that parameter K-means grouping is simpler to implement, as co-activation graph split requires collecting activation data, we use K-means grouping in our paper.

The relation between efficiency loss hyperparameter η and sparsity. As discussed in Section 4.2.2, efficiency loss hyperparameter η is used to balance the trade-off between task performance and sparsity (efficiency). We illustrate the relation between η and sparsity across different models and tasks in Figure 10. For a given task and model, increasing η results in greater sparsity.

Ablation study on separability loss. Another hyperparameter in LTE training, λ, is used to encourage the separability of router outputs, which facilitates us to choose the threshold to pick neurons (as shown in Figure 3). We compare models trained with different lambda in Figure 11. When increasing λ from 0 (i.e., no separability loss) to 0.5, we observe a constant perplexity drop for all sparsity. However, if we keep increasing λ, perplexity does not further decrease.

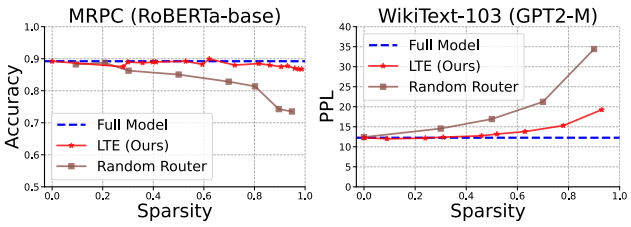

Figure 12: Performance comparison between random routers and routers trained with LTE. Models with LTE-trained routers consistently outperform models with random routers.

Effectiveness of LTE-trained routers. We next demonstrate the effectiveness of LTE's first training stage, the model-router training stage, in helping LLMs gain better sparsity. To build a baseline, instead of using routers trained in the model-router training stage, we use randomly initialized networks as routers and train LLMs to adapt to random routers. Since we cannot control the sparsity of LLMs with a threshold in a random router, we decide the sparsity for MoE layers by picking K top experts: we first fix a K to decide the sparsity and then start to train the model. Figure 12 compares the performance of models with different routers. We observe that routers trained with LTE significantly contribute to achieving a better trade-off between sparsity and model performance, which proves the effectiveness of the model-router training stage.

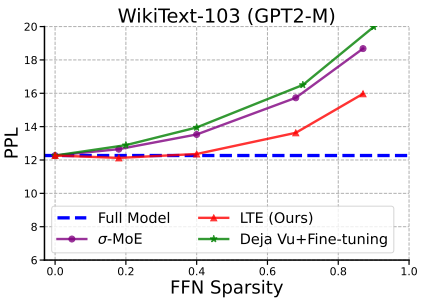

Figure 13: Performance comparison with new baselines: 1) σ-MoE [4] and 2) Deja VU [21] with further fine-tuning. We evaluate two baselines by fine-tuning GPT2 medium with WikiText103. LTE outperforms the other two baselines across different sparsity.

Other baselines with further fine-tuning. A key distinction between LTE and other baselines is that LTE requires additional fine-tuning to make the model efficiency-aware. In this section, we assess whether further fine-tuning can enhance the performance of other baselines, aiming to understand the contribution of additional training to LTE's effectiveness. Specifically, we fine-tune two baselines on GPT2-Medium using the WikiText103 dataset: 1) we first fine-tune GPT2-M with Wikitext and apply Deja Vu on the fine-tuned models. Then, we further fine-tune MoEfied models with WikiText103. 2) We implement σ-MoE [4] on GPT2-Medium model and fine-tune models with the WikiText103 dataset. For both baselines, we train models for the same training time as we train LTE models. The evaluation results shown in Figure 13 show that LTE outperforms the other two baselines across different sparsity levels.

6 Conclusion

In this paper, we explore how to enable better activation sparsity for fast LLM inference. We introduce a novel algorithm, LTE, for the model to automatically learn to activate fewer neurons, by grouping neurons into different experts and then adaptively selecting important ones for specific inputs and layers. Our extensive evaluations, spanning four LLMs and eleven datasets, show that LTE can achieve 1.83x - 2.59x FLOPs speed-up on LLaMA, thus execution efficiency, outperforming state-of-the-art designs. We believe that our work can inspire more research on designing more efficiency-aware training methods, making LLMs more accessible to the broad community.

Acknowledgement

My work was partially supported by DARPA Grant HR00112020008, National Science Foundation Grant No. 2039445, and Cisco Research Grant. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of our research sponsors.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

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

[3] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Association for Computational Linguistics, 2017.

[4] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. Approximating two-layer feedforward networks for efficient transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 674–692, 2023.

[5] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, MLCW'05, page 177–190, Berlin, Heidelberg, 2005. Springer-Verlag.

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

[7] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

[8] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR, 2023.

[9] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[10] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, 2021.

[11] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[12] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[13] Shankar Iyer, Nikhil Dandekar, Kornél Csernai, et al. First quora dataset release: Question pairs. *data. quora. com*, 2017.

[14] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[15] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[16] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[17] Xin Li, Yeqi Bai, Pinlong Cai, Licheng Wen, Daocheng Fu, Bo Zhang, Xuemeng Yang, Xinyu Cai, Tao Ma, Jianfei Guo, et al. Towards knowledge-driven autonomous driving. *arXiv preprint arXiv:2312.04316*, 2023.

[18] Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *Conference on Parsimony and Learning (Recent Spotlight Track)*, 2023.

[19] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

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

[21] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR, 2023.

[22] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

[23] Mikko I Malinen and Pasi Fränti. Balanced k-means for clustering. In *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop, S+ SSPR 2014, Joensuu, Finland, August 20-22, 2014. Proceedings*, pages 32–41. Springer, 2014.

[24] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[25] Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models. *arXiv preprint arXiv:2310.04564*, 2023.

[26] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

[27] Jekaterina Novikova, Ondrej Dušek, and Verena Rieser. The E2E dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, Saarbrücken, Germany, 2017. arXiv:1706.09254.

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[29] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[31] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[32] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018.

[33] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2016.

[34] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[35] Jiachen Sun, Haizhong Zheng, Qingzhao Zhang, Atul Prakash, Zhuoqing Mao, and Chaowei Xiao. Calico: Self-supervised camera-lidar contrastive pre-training for bev perception. In *ICLR*, 2024.

[36] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[37] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[38] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[39] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[40] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[41] Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. How far can camels go? exploring the state of instruction tuning on open resources. *Advances in Neural Information Processing Systems*, 36, 2023.

[42] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.

[43] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference, 2018.

[44] Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. Self-play preference optimization for language model alignment. *arXiv preprint arXiv:2405.00675*, 2024.

[45] Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*, 2024.

[46] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[47] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[48] Xiaokuan Zhang, Haizhong Zheng, Xiaolong Li, Suguo Du, and Haojin Zhu. You are where you have been: Sybil detection via geo-location analysis in osns. In *2014 IEEE Global Communications Conference*, pages 698–703. IEEE, 2014.

[49] Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 877–890, 2022.

[50] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

[51] Haizhong Zheng, Rui Liu, Fan Lai, and Atul Prakash. Coverage-centric coreset selection for high pruning rates. In *ICLR*, 2023.

[52] Haizhong Zheng, Jiachen Sun, Shutong Wu, Bhavya Kailkhura, Zhuoqing Mao, Chaowei Xiao, and Atul Prakash. Leveraging hierarchical feature sharing for efficient dataset condensation. In *ECCV*, 2024.

[53] Haizhong Zheng, Minhui Xue, Hao Lu, Shuang Hao, Haojin Zhu, Xiaohui Liang, and Keith Ross. Smoke screener or straight shooter: Detecting elite sybil attacks in user-review social networks. In *NDSS*, 2018.

Broader Impact

This paper presents the Learn-To-be-Efficient (LTE) algorithm to accelerate the LLM inference. A more efficient LLM inference can significantly enhance the accessibility and sustainability of LLMs. By reducing computational demands, it addresses environmental concerns through lower energy consumption and helps democratize access to advanced AI technologies. We believe that our work can better help smaller organizations, educational institutions, and researchers, who previously faced barriers due to resource limitations, access LLMs more easily. To sum up, the LTE algorithm not only advances the field technically but also broadens the scope of AI's benefits to a wider, more inclusive community.

Limitations

While LTE outperforms other baselines, LTE needs further fine-tuning to get contextual sparse models. (Yet, training is a one-time investment that yields long-term benefits for LLM serving.) A future study is to combine LTE with other parameter-efficient fine-tuning techniques to reduce LTE training overhead. Another limitation is our computational resource, which prevents us from training LTE on pretraining or RLHF. As discussed in Section 5.2, we believe that training with a wider variety of data will further improve LTE performance. We leave this for future exploration.

A Code and Datasets License

Codebase. Our model implementation is based on Huggingface transformer repo: Apache-2.0 license

Datasets. We list the license of used datasets as follow:

GLUE dataset [40]: Custom License;

WikiText103 [24]: CC BY-SA 3.0;

XSum [26]: MIT License;

E2E [27]: CC4.0-BY-SA;

Tulu [41]: ODC-BY;

MMLU [11]: Custom License.

B Experiment Settings

We presented all training hyperparameters in Table 3, 4, 5, and 6. For hyperparameters presented in { }, we select the best hyperparameter for each task. We follow the settings in RoBERTa paper [20] to fine-tune RoBERTa on GLUE datasets. We set the coefficient for the separability loss (λ in Equation 6) to be 0.5 for all stage 1 training. We use different η to control the sparsity of trained models (Figure 10). Hardware: All models are trained and evaluated on A100, A40, and 3090Ti, depending on memory usage and availability.

C Additional Evaluation Results

C.1 NLU Performance Comparison

In Figure 14, we present the evaluation results of the rest four NLU datasets. We have similar findings as we discussed in Section 5.2: LTE achieves a better trade-off between sparsity and task performance. However, we notice that, LTE outperforms MoEfication on the CoLA dataset, but has worse performance than KLA at low sparsity level. The size of CoLA is relatively small and may not be efficient for models to learn good routers.

C.2 Further Analysis on Sparsity Provided by LTE

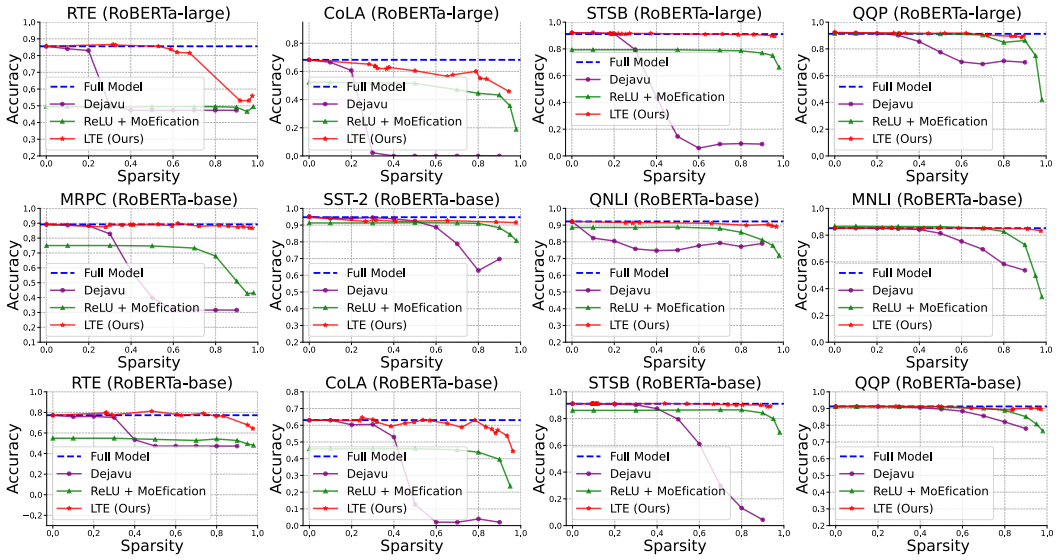

Figure 14: Performance comparison across four NLU datasets from GLUE dataset. The evaluation results show that LTE consistently outperforms other baselines in each dataset for both models.

Contextual Sparsity for Larger Batches. Similar to previous contextual sparsity work [21], we observe that as the batch size increases, more neurons are activated, leading to a reduction in union sparsity. These dynamics are illustrated in Figure15. Similar to Deja Vu [21], the number of activated neurons does not grow linearly with the batch size, indicating a non-uniform distribution of parameter access from different inputs. This property can extend LTE for a high-throughput batch setting by batching inputs activating similar neurons together to achieve a high union sparsity.

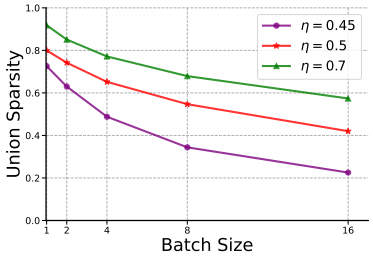

Figure 15: Union sparsity changes w.r.t. batch size.

Adaptive Sparsity across Layers. Instead of assigning a fixed sparsity to each layer, LTE uses the efficiency loss to introduce competition across different layers to allocate the computation budget to more important layers. Here, we conduct a study on the distribution of sparsity across layers in models trained with LTE. As shown in Figure 16, we observe that LTE achieves different sparsity levels across different layers. For example, the MRPC (RoBERTa$_{\text{base}}$) model with 0.46 sparsity has 0.2 sparsity at layer 2, but has more than 0.8 sparsity at layers 10 and 11.

Another intriguing finding is that RoBERTa$_{\text{base}}$ and GPT2-M have very different sparsity patterns. In RoBERTa, the sparsity of MoE layers significantly increases with depth, contrasting with GPT2-M, where MoE layer sparsity decreases as layer depth increases. This difference can stem from differences in the design of these two transformers and tasks. Given that MRPC is a sentence classification task, encoder-based transformers like RoBERTa primarily rely on the "[CLS]" token for classification. Consequently, other tokens become less important in deeper layers, allowing sparser FFN layers. In contrast, GPT2, a decoder-based transformer designed for language generation, requires all tokens to generate the next token. Furthermore, recent research [10] on interpretability suggests that deeper FFN layers store more complex semantic information, which may drive deep FFN layers to be less sparse.

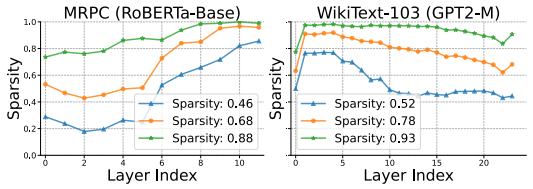

Figure 16: Sparsity across different layers. We present the sparsity of different layers in an encoder-based model (RoBERTa$_{\text{base}}$) for MRPC and a decoder-based model (GPT2-M) for WikiText103. Different layers learn to have different sparsity, and RoBERTa$_{\text{base}}$ and GPT2-M have different sparsity patterns across layers.

Table 3: RoBERTa-base hyperparameter for LTE training on GLUE dataset.

Hyperparameter	LTE-Stage 1	LTE-Stage 2
Learning rate	{2e-5, 5e-5}	{5e-5}
Training batch size	{16, 32}	{16, 32}
Training epochs	{1, 10}	{1, 3, 10}
Weight decay	0	0
Warm up ratio	0.06	0.06

Table 4: RoBERTa-large hyperparameter for LTE training on GLUE dataset.

Hyperparameter	LTE-Stage 1	LTE-Stage 2
Learning rate	{2e-6, 5e-6, 3e-5}	{2e-6, 5e-6, 3e-5}
Training batch size	{16, 32}	{16, 32}
Training epochs	{1, 2, 3, 10}	{3, 5, 10}
Weight decay	0	0
Warm up ratio	0.06	0.06

Table 5: GPT2-Medium hyperparameter for fine-tuning and LTE training on XSum, E2E, and WikiText-103 datasets. (Numbers in the bracket correspond to three datasets in order.)

Hyperparameter	Fine-tuning	LTE-Stage 1	LTE-Stage 2
Learning rate	5e-5	5e-5	5e-5
Training batch size	8	8	8
Training epochs	3	1	(1, 1, 3)
Weight decay	0	0	0
Warm up ratio	0.06	0.06	0.06

Table 6: LLaMA hyperparameter for fine-tuning and LTE training on XSum, E2E, and WikiText-103 datasets.

Hyperparameter	Fine-tuning	LTE-Stage 1	LTE-Stage 2
Learning rate	1.5e-5	1.5e-5	1.5e-5
Training batch size	16	16	16
Training epochs	3	1	1
Weight decay	0	0	0
Warm up ratio	0.06	0.06	0.06

Table 7: Hyperparameters for instruction tunning with Tulu dataset.

Hyperparameter	Fine-tuning	LTE-Stage 1	LTE-Stage 2
Learning rate	2e-5	2e-5	2e-5
Training batch size	128	128	128
Training epochs	3	1	4
Weight decay	0	0	0
Warm up ratio	0.06	0.06	0.06

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the abstract and introduction accurately summarize the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations of our work in the Appendix.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: No theory is included in the paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We introduced detailed implementation details and hyperparameters used in the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We upload the code in the Supplementary files.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We introduced detailed implementation details and hyperparameters used in the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Training LLMs incurs significant costs. To ensure the reliability of our evaluation and avoid the potential bias of cherry-picking results, we employ entirely random seeds. Furthermore, the consistency of our conclusions across multiple models and benchmarks can also be treated as a form of repeated evaluation and support the robustness of our findings.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We introduced compute resources used in the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes, we check that our paper conform with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We included broader impacts in the Appendix.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: No model or dataset is released in this paper.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We include the license for the used code and datasets in the Appendix.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.