Efficient and Interpretable Grammatical Error Correction with Mixture of Experts

Muhammad Reza Qorib[†], **Alham Fikri Aji**[‡] and **Hwee Tou Ng**[†]
[†]Department of Computer Science, National University of Singapore
[‡]NLP department, MBZUAI
mrqorib@u.nus.edu, alham.fikri@mbzuai.ac.ae, nght@comp.nus.edu.sg

Abstract

Error type information has been widely used to improve the performance of grammatical error correction (GEC) models, whether for generating corrections, re-ranking them, or combining GEC models. Combining GEC models that have complementary strengths in correcting different error types is very effective in producing better corrections. However, system combination incurs a high computational cost due to the need to run inference on the base systems before running the combination method itself. Therefore, it would be more efficient to have a single model with multiple sub-networks that specialize in correcting different error types. In this paper, we propose a mixture-of-experts model, MoECE, for grammatical error correction. Our model successfully achieves the performance of T5-XL with three times fewer effective parameters. Additionally, our model produces interpretable corrections by also identifying the error type during inference.[1]

1 Introduction

Grammatical error correction (GEC) is a task that aims to detect and correct any errors in a given text. Through scientific progress over the past decades, grammatical error correction has outgrown its name. GEC not only deals with grammatical errors but also includes the correction of misspellings, orthographic errors, semantic errors, and more (Bryant et al., 2023).

A GEC model receives a possibly erroneous text and should produce a corrected version of the text with minimal modification. An example is given in Table 1. In the early days of GEC, people approached this problem by building a specific classifier for each error type, which was later replaced by data-driven approaches adopted from the machine translation task (Chollampatt et al., 2016).

[1]The source code and models can be accessed at https://github.com/nusnlp/moece.

Even after adopting end-to-end approaches using neural network models, the error type information remains useful. Some recent models approach the problem by separating the error detection and correction models (Yuan et al., 2019; Li et al., 2023), training a single model for both detection and correction (Omelianchuk et al., 2020), or training a model to predict the edit/transformation tag, which can include the error type (Stahlberg and Kumar, 2020).

Error type information is useful not only for building the grammar correction model but also for reranking the outputs of GEC models (Sorokin, 2022; Chollampatt and Ng, 2018) and combining outputs of GEC models (Kantor et al., 2019; Lin and Ng, 2021; Qorib and Ng, 2023). GEC system combination has been very effective in improving the state of the art, and it has been repeatedly demonstrated that it works best when the base systems have complementary strengths (Susanto et al., 2014; Han and Ng, 2021), such as which error types they can correct more accurately. However, system combination is computationally expensive because it needs to run inference on each base system first before running the combination method itself.

It would be desirable to just have one model with multiple sub-networks that specialize in different aspects, such as which error types it can correct more accurately. The capability of a neural network to correct one error type is not necessarily transferable to correcting other error types, as Qorib et al. (2022) report that state-of-the-art GEC models are often less accurate in correcting certain error types than weaker GEC models. This could be caused by task interference during the training of the model. When fine-tuning T5-v1.1-Base for grammatical error correction, we observe that the model becomes less accurate at correcting punctuation (PUNCT) and preposition (PREP) error types when it has the best overall (ALL) performance (Figure 1). We also notice that the model's accuracy

Source	The rich people will buy a car but the poor people always need to use a bus or taxi .
Correction	Rich people will buy a car , but poor people always need to use a bus or taxi .
Edits	(0, 2, 'Rich', DET), (7, 7, ',', PUNCT), (8, 9, '', DET)

Table 1: Example of a GEC source sentence, its correction, and the edits. Each edit is represented by the start index, the end index, the replacement string, and the error type. DET denotes a determiner error while PUNCT denotes a punctuation error.

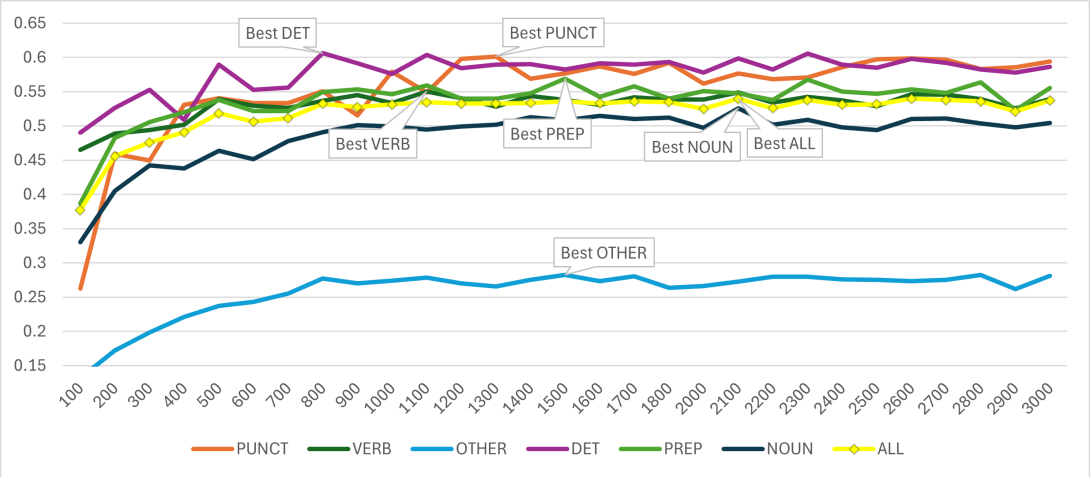


Figure 1: $F_{0.5}$ scores of a T5-v1.1-Base model on the six most frequent error types and all error types (ALL) in the BEA-2019 development set at different numbers of training steps.

in correcting preposition errors decreases when its accuracy in correcting punctuation errors increases (e.g., training step 1600 to 2000 in Figure 1).

As such, a neural network that has separate sets of parameters that specialize in correcting different error types could be beneficial in modeling grammatical error correction. Such a model is computationally more efficient as it does not need to go through multiple threads of inference necessary in system combination. For that purpose, we propose a mixture-of-experts model for grammatical error correction. To the best of our knowledge, we are the first to do so. This study focuses on building an MoE model for GEC through transfer learning from a non-MoE (dense) sequence-to-sequence model.

2 Mixture of Experts

Mixture of experts (MoE) is a learning procedure for a system that can be decomposed into multiple separate networks, with each network representing an expert at a particular task (Jacobs et al., 1991). Mixture of experts has been reported to perform well on tasks that can be broken down into different sub-tasks, such as multilingual machine translation (Kudugunta et al., 2021), and single tasks that have multiple objectives, such as video recommendation (Li et al., 2020).

When used with the transformer architecture, MoE is typically applied by replacing the feed-forward layer $f(\boldsymbol{x})$ in a transformer block with an MoE layer[2] $f'(\boldsymbol{x})$. MoE is applied independently to some or all transformer blocks in the network. An MoE layer consists of a set of experts $\{E_1, E_2, \ldots, E_M\}$ with parameters $\phi_i, i \in \{1, \ldots, M\}$, a router or a gating function r that selects a subset of K experts, and optionally an aggregation function g_i that produces the weight for expert i. The aggregation function can also be the same as the router. Let $\boldsymbol{x}$ denote the vector representation of a token from the previous layer. We formalize the MoE layer in Equation 1 below:

$$f'(\boldsymbol{x}) = \sum_{k=1}^{K} \frac{g_k(\boldsymbol{x})}{\sum_{j=1}^{K} g_j(\boldsymbol{x})} E_k(\boldsymbol{x}; \boldsymbol{\phi_k}) \quad (1)$$

One problem that may arise in MoE is one expert getting chosen more often than others, making it receive more gradient updates and subsequently making it more preferred by the router, a self-reinforcing problem. To avoid this issue, the number of tokens in a sequence $\boldsymbol{X}$ =

[2]The notation of $f'(\boldsymbol{x})$ represents a function that replaces $f(\boldsymbol{x})$, rather than the derivative of $f(\boldsymbol{x})$.

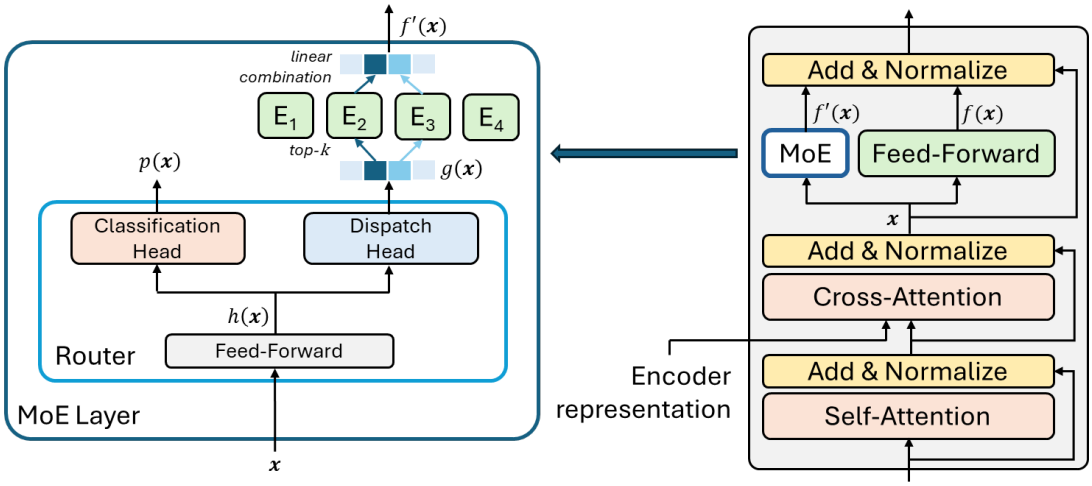


Figure 2: Illustration of a transformer block with an MoE layer with $M = 4$ and $K = 2$. To simplify the notation, $\boldsymbol{x}$ represents the input of the MoE layer and the feed-forward layer instead of the transformer block.

$\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ that an expert can process is often limited to a threshold called expert capacity, and a load balancing loss is used to encourage a more balanced expert allocation. If an expert already reaches the maximum capacity, subsequent tokens that are assigned to that expert do not go through the expert layer. The load balancing losses are calculated based on the fraction of the input that goes into expert i, represented by w_i.

2.1 GShard

Lepikhin et al. (2021) propose to penalize the model based on the mean square of w_i, but the top-k operation to select the experts is not differentiable. As such, it approximates the loss function by multiplying the average routing score for expert i on all tokens (v_i) with the fraction of tokens that are allocated to it (w_i), then takes the average of that score from all experts. The loss function is formalized in Equation 2 below.

$$v_i = \frac{1}{N} \sum_{\boldsymbol{x} \in \boldsymbol{X}} g_i(\boldsymbol{x})$$
$$\mathcal{L}_b = \frac{1}{M} \sum_{i=1}^{M} (w_i \times v_i) \quad (2)$$

In GShard, the router chooses the top two experts (E_{i_1}, E_{i_2}) for each token. However, the second expert will be ignored if its routing score is too small. The router sends the token $\boldsymbol{x}$ to the second expert with a probability proportional to the routing score $g_{i_2}(\boldsymbol{x})$.

2.2 SwitchTransformer

SwitchTransformer (Fedus et al., 2022b) follows the balancing loss of GShard, but instead of taking the average, they take the sum and then multiply it again with the number of experts, as shown in Equation 3 below.

$$\mathcal{L}_b = M \times \sum_{i=1}^{M} (w_i \times v_i) \quad (3)$$

The reason is to keep the loss constant and invariant to the number of experts. In the case of uniform routing, $w_i = \frac{1}{M}$ and $g_i = \frac{1}{M}$, so $\mathcal{L}_b = 1$.

Fedus et al. (2022b) argue that sending each token to only one expert is sufficient. This challenges the conjecture of Shazeer et al. (2017) that sending a token to more than one expert is necessary. They report that choosing one expert for each token preserves model quality, reduces routing computation, and performs better.

3 Method

In this section, we explain how we build MoECE (**M**ixture **o**f **E**rror **C**orrection **E**xperts), a mixture-of-experts model for grammatical error correction.

3.1 MoE Layer

We follow the common approach of building MoE explained in Section 2 with one modification. Instead of replacing the feed-forward layer with an MoE layer, we augment the feed-forward layer with an MoE layer (Figure 2). The output of the i-th transformer block in a standard transformer is the normalized $\boldsymbol{x}_i + f_i(\boldsymbol{x}_i)$ and the output of the transformer block in common MoE approaches is the

normalized $\boldsymbol{x}_i + f'_i(\boldsymbol{x}_i)$. In our model, the output of the transformer block with an MoE layer is the normalized $\boldsymbol{x}_i + f_i(\boldsymbol{x}_i) + f'_i(\boldsymbol{x}_i)$. While this design increases the computation cost during training, no additional cost is incurred during inference by merging the weights of the feed-forward layer with the experts.

3.2 Router

We hypothesize that an optimal expert allocation in a mixture of experts for grammatical error correction involves having each expert focus on correcting particular error types. With this goal in mind, the router needs to know what error type the current token has, so that the router can direct it to the appropriate expert. We approach this by adding a classification head to the router and training it with an auxiliary loss. As such, for every input token, the router predicts the error type through the classification head and determines the expert allocation through the dispatch head.

We design the router in each MoE layer to be a 2-layer neural network with two outputs (Figure 2). The input dimension of the gate is the same as the model dimension of the transformer block. The output dimension of the classification head is the number of possible error types $|T|$, while the output dimension of the dispatch head is the number of experts $|M|$. Let $\boldsymbol{x}$ denote an input token from the previous layer, and let σ denote the softmax function. We formally describe the routing function in Equation (4 – 5) below.

$$h(\boldsymbol{x}) = \boldsymbol{W}_h \times \boldsymbol{x} + \boldsymbol{b}_h$$
$$p(\boldsymbol{x}) = \sigma(\boldsymbol{W}_p \times h(\boldsymbol{x}) + \boldsymbol{b}_p) \quad (4)$$
$$g(\boldsymbol{x}) = \sigma(\boldsymbol{W}_g \times h(\boldsymbol{x}) + \boldsymbol{b}_g) \quad (5)$$

While the classification head guides the hidden representation of the router h to route the token based on the error type, the dispatch head may unsatisfactorily route tokens with different error types to a single expert. To avoid that, we employ the expert capacity limitation and load balancing loss of GShard and SwitchTransformer. Similar to previous work, we also use the routing function g as the aggregation function.

3.3 Training Objective

The model is trained to minimize the combination of three loss functions, the cross-entropy loss $\mathcal{L}_c$ on the prediction of the corrected text, the cross-entropy loss of the error type $\mathcal{L}_e$ from the router, and the load balancing loss $\mathcal{L}_b$ from the router. The contribution of the error type loss is controlled by a hyper-parameter α while the contribution of the load balancing loss is controlled by another hyper-parameter β. Let L denote the number of MoE layers. The final loss function is given below.

$$\mathcal{L} = \mathcal{L}_c + \alpha \times \frac{1}{L}\sum_{l=1}^{L}\mathcal{L}_{e_l} + \beta \times \frac{1}{L}\sum_{l=1}^{L}\mathcal{L}_{b_l} \quad (6)$$

4 Experiments

We transformed the pre-trained T5-v1.1 (Raffel et al., 2020) language models into mixture-of-experts models. We trained two sizes of the model, a base model based on T5-v1.1-Base and a large model based on T5-v1.1-Large. For each model size, we trained two models with different routers, one with the GShard router (MoECE-GS) and another with the SwitchTransformer router (MoECE-ST). We apply an MoE layer with 7 experts to all transformer blocks in the decoder except the first block and share the parameters of the routers in all transformer blocks.

Name	Type	# sent	# ref
cLang-8	Train	2,372,119	1
BEA-2019	Dev	4,384	1
CoNLL-2014	Test	1,312	2
BEA-2019	Test	4,477	5
CWEB-G	Test	3,981	2
CWEB-S	Test	2,864	2

Table 2: GEC datasets used in our experiments. # sent refers to the number of sentences while # ref refers to the number of reference annotations. Dev refers to the BEA-2019 development set.

4.1 Implementation

We implement our T5 model using the modification of the fairseq framework (Ott et al., 2019) by Applica AI[3] and augment it with the MoE implementation from Fastmoe (He et al., 2021). In Fastmoe, the GShard loss function is multiplied by the square of the number of experts (M^2) to make the loss magnitude not affected by the number of experts in the layer. We follow this implementation to standardize the experiments. As such, the difference between the GShard and SwitchTransformer variants in our model is in the number of experts selected per token and when the loss is calculated.

[3] https://github.com/applicaai/fairseq/tree/applica-t5

Model	EPC	BEA-2019 Dev ($F_{0.5}$)	CoNLL-2014 Test P	CoNLL-2014 Test R	CoNLL-2014 Test $F_{0.5}$	BEA-2019 Test P	BEA-2019 Test R	BEA-2019 Test $F_{0.5}$
T5-v1.1-Base	248M	53.97	72.43	48.38	65.88	72.82	62.11	70.39
MoECE-GS-Base	282M	**55.28**	73.00	48.53	66.31	74.59	62.11	**71.71**
MoECE-ST-Base	248M	54.84	72.75	49.31	**66.43**	73.99	62.00	71.24
T5-v1.1-Large	783M	55.85	73.18	49.46	66.78	75.65	64.59	73.15
MoECE-GS-Large	917M	56.42	74.29	50.21	**67.79**	76.91	64.54	**74.07**
MoECE-ST-Large	784M	**56.68**	73.60	51.25	67.69	75.95	65.78	73.67

Table 3: MoECE performs better than the comparable dense model on the BEA-2019 development set, CoNLL-2014 test set, and BEA-2019 test set. EPC is the effective parameter count.

In GShard, the loss is calculated before limiting the expert by its capacity, while in SwitchTransformer it is calculated after.

4.2 Datasets and Evaluation

Following Rothe et al. (2021), we train the model with the cLang-8 dataset. The cLang-8 corpus is a GEC dataset that was made by relabeling the raw version of the Lang-8 dataset with the output of a big multilingual GEC model. Training a GEC model with cLang-8 is reported to produce a better GEC model than training it with the original Lang-8 dataset (Sorokin, 2022). We augment the training data with the error types produced by ERRANT (Bryant et al., 2017). During the development of the model, we use the BEA-2019 development set (Bryant et al., 2019) as the validation data to choose the hyper-parameters.

We evaluate the model on standard GEC benchmarks, the CoNLL-2014 test set and the BEA-2019 test set. The model is evaluated on the $F_{0.5}$ score produced by the M2Scorer (Dahlmeier and Ng, 2012) for the CoNLL-2014 test set and the $F_{0.5}$ score produced by the official blind scorer[4] for the BEA-2019 test set. In addition to the standard benchmarks, we also evaluate our model on out-of-domain datasets, which are CWEB-G and CWEB-S (Flachs et al., 2020). Most of GEC training and test data were made from student essays in an academic setting, but CWEB datasets were made from texts from various websites on the Internet. The CWEB-G test set was made from generic websites while the CWEB-S test set was made from websites of official institutions, such as governments, schools, and museums. We report the statistics of the datasets in Table 2. We perform a bootstrap resampling test on 100 samples of the models' outputs to measure statistical significance.

[4] https://codalab.lisn.upsaclay.fr/competitions/4057

We compare our model to baselines and previous work that have comparable effective parameter count (EPC) (Fedus et al., 2022a). The effective parameter count only considers the number of parameters that are active or used in a single forward pass. Since in MoE only a few experts are used at a time to generate the corrections, the unused experts do not contribute to the computational cost. Therefore, it is more appropriate to compare an MoE model based on the effective parameter count rather than the total number of parameters.

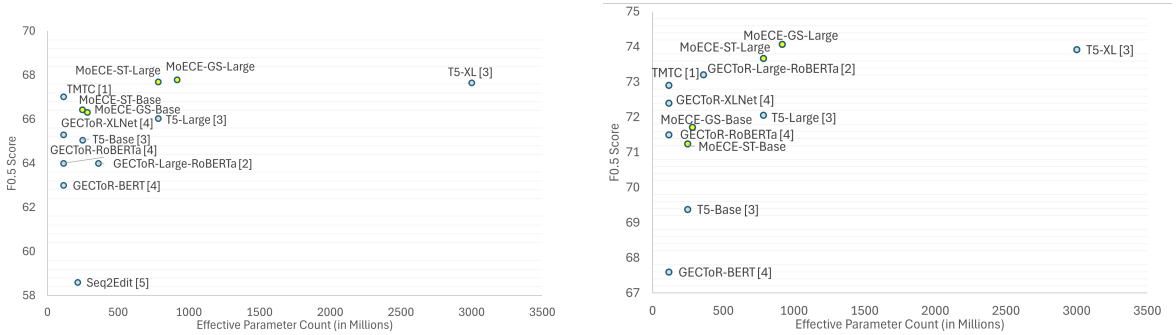
Model	CWEB-G	CWEB-S
T5-v1.1-Base	36.78	26.83
MoECE-GS-Base	39.22	27.77
MoECE-ST-Base	**39.37**	**27.90**
T5-v1.1-Large	42.08	26.14
MoECE-GS-Large	42.82	27.14
MoECE-ST-Large	**43.06**	**27.48**

Table 4: MoECE achieves higher $F_{0.5}$ scores than the dense model on the CWEB-G and CWEB-S test sets with a relatively small overhead for the computation in the router.

5 Results

We report the scores of our models and the comparable dense model in Table 3 and Table 4. Our MoECE-GS-Base model successfully improves the $F_{0.5}$ score by 0.43 points on the CoNLL-2014 test set and 1.32 points on the BEA-2019 test set, while our MoECE-ST-Base improves the $F_{0.5}$ score by 0.55 points on the CoNLL-2014 test set and 0.85 points on the BEA-2019 test set. The improvements are statistically significant with $p < 0.01$.

Without re-training or changing the hyper-parameters, MoECE also brings improvements over the dense model on out-of-domain datasets (Table 4). MoECE-GS-Base improves the $F_{0.5}$ score on the CWEB-G test set and CWEB-S test set



(a) CoNLL-2014 test set

(b) BEA-2019 test set

Figure 3: $F_{0.5}$ scores of comparable models that produce the error type of corrections according to the effective parameter count (in millions). Legends: [1] Lai et al. (2022) [2] Tarnavskyi et al. (2022), [3] Rothe et al. (2021), [4] Omelianchuk et al. (2020), [5] Stahlberg and Kumar (2020). We do not compare against (Sun and Wang, 2022) and (Bout et al., 2023) which only produce the correction tokens.

by 2.44 points and 0.94 points respectively, while MoECE-ST-Base improves the $F_{0.5}$ score on the same test sets by 2.59 points and 1.07 points. Our method still improves the $F_{0.5}$ scores on all test sets when we scale up the models to MoECE-GS-Large and MoECE-ST-Large, and the improvements are statistically significant with $p < 0.01$.

Compared to previous work (Figure 3), the performance of our MoECE-GS-Large model is comparable to T5-XL (Rothe et al., 2021) despite having vastly fewer effective parameters. T5-XL with 3B effective parameters has an $F_{0.5}$ score of 67.65[5] on the CoNLL-2014 test set and 73.92 on the BEA-2019 test set, while MoECE-GS-Large that has three times fewer effective parameters has an $F_{0.5}$ score of 67.79 on the CoNLL-2014 test set and 74.07 on the BEA-2019 test set (Table 3). Even if we include the unused experts, MoECE-GS-Large with 1.7B parameters is still smaller than T5-XL. This shows that our model is more efficient in terms of computation and memory usage than T5-XL.

6 Analysis

6.1 Routing Policy

We analyze whether the error type loss helps the router in choosing the experts based on the error type of the token. We take the average routing score that the last router produces for each error type. We find that with the error type loss, the router chooses different combinations of experts according to the token error type. In Figure 4, we can see that the router mainly chooses experts #0 and #6 when correcting punctuation errors (PUNCT) and experts #2 and #5 when correcting preposition errors (PREP). On the other hand, when the error type loss is not used, experts #1 and #2 dominate the routing policy for correcting almost any error type.

Figure 4: Average routing score of MoECE-GS-Base for each token based on the error type.

The routing process also leads to the specialization of experts (Figure 5). For example, in correcting punctuation errors, we observe that expert #6 and expert #0 achieve the highest accuracy, at 83.7% and 82.6%, respectively. In contrast, the accuracy of expert #3, expert #2, and expert #4 is 33.3%, 38.1%, and 50.7%, respectively. Similarly, for orthographical errors, expert #1 achieves 80.9% accuracy, while expert #3 achieves 66.7% accuracy. The accuracy of each expert aligns well with the routing policy, especially for error types frequently encountered in the training data, with the median Pearson correlation coefficient being 0.66 for the top seven error types.

We evaluate the accuracy of error type prediction of the router on the BEA-2019 development set and find that the overall accuracy of error type

[5]We use the score from the updated paper on arXiv.

Router	LB	ET	CoNLL-2014	BEA-2019	CWEB-G	CWEB-S	Avg
GShard	✓	✓	66.31	71.71	**39.22**	27.77	**51.25**
GShard	✓	✗	**66.34**	**71.80**	38.84	**27.88**	51.22
SwitchTransformer	✓	✓	**66.43**	**71.24**	**39.37**	27.90	**51.24**
SwitchTransformer	✓	✗	66.31	69.72	37.97	**27.97**	50.49
GShard	✗	✓	**66.23**	**71.54**	**39.39**	**28.55**	**51.43**

Table 5: The effect of the load balancing loss (LB) and error type loss (ET) when training MoECE-Base. The evaluation is based on the $F_{0.5}$ scores on the CoNLL-2014 test set, BEA-2019 test set, CWEB-G test set, CWEB-S test set, and the average of these four test sets.

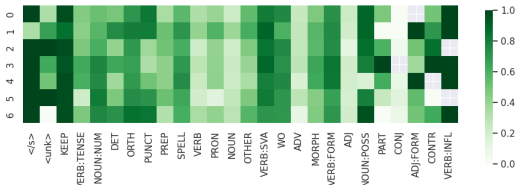


Figure 5: Experts' correction accuracy for each token based on the error type.

prediction by the routers is 92.5%. This provides evidence that the model is capable of learning the error type using the classification head in the router and suggests that the model utilizes this information to route the tokens to the appropriate experts.

6.2 Impact of Error Type Loss and Load Balancing Loss

Despite having a more comprehensible routing decision, models trained with the error type loss do not strictly perform better than models trained without it (Table 5). This shows that the routing can learn a different routing policy by itself that performs comparably well, even though the policy is not based on the error type. However, the routing policy may be hard to interpret and subsequently makes it harder to understand if the model makes weird mistakes. On the other hand, the model trained with the error type loss will produce both the corrected text and the error type at the same time during inference. This will help language learners understand the reason for the correction and can help model developers look for the cause of the issue if the model makes mistakes. On top of that, the expert allocation for each error type is quite clear when the model is trained with error type loss (Figure 4). This allows some degree of modularity to add, modify, or remove the experts during deployment.

We also run an experiment of training the model with the error type loss but without the load balancing loss. In this experimental setting, we use the same routing criteria as GShard but set the load balancing loss multiplier β (Equation 6) to zero. We also obtain a comparable model to the other experimental settings. This shows that the error type loss can serve as an alternative to the existing load balancing loss.

6.3 Shared Feed-Forward Layer

As explained in Section 3.1, we design our MoE to be an addition to the main transformer feed-forward layer, rather than a replacement, during training. We hypothesize that for a neural network to correct errors in text, it requires general skills and specific skills according to the error types. By adding the output of the MoE with the main feed-forward layer, the general skills can be learned by the feed-forward layer that is shared across all experts. This idea is similar to DeepSeekMoE (Dai et al., 2024), in which some experts are always chosen for any input token so that those experts learn the necessary general skills.

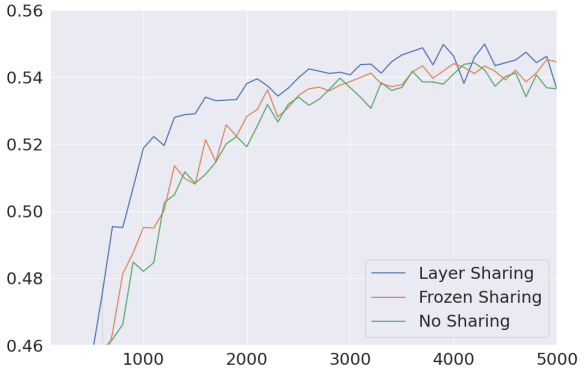


Figure 6: $F_{0.5}$ scores on the BEA-2019 development set of three architecture types: MoECE (LAYER SHARING), MoECE without parameter updates to the shared layer (FROZEN SHARING), and standard MoE architecture where the feed-forward layer is entirely replaced by the MoE layer (NO SHARING).

We investigate the efficacy of our architecture design by comparing three architecture types:

#	Model	Exp Dim	EPC	BEA-2019 Dev ($F_{0.5}$)	CoNLL-2014 Test P	CoNLL-2014 Test R	CoNLL-2014 Test $F_{0.5}$	BEA-2019 Test P	BEA-2019 Test R	BEA-2019 Test $F_{0.5}$
1	T5-v1.1-Base	-	248M	53.97	72.43	48.38	65.88	72.82	62.11	70.39
2	MoECE-ST	2048	248M	54.84	72.75	**49.31**	**66.43**	73.99	62.00	71.24
3	MoECE-ST	128	250M	54.86	72.96	48.51	66.28	74.56	61.37	71.49
4	MoECE-GS	128	252M	54.86	72.18	49.22	66.02	73.74	**62.42**	71.16
5	MoECE-GS	2048	282M	**55.28**	**73.00**	48.53	66.31	**74.59**	62.11	**71.71**

Table 6: Scores of MoECE-GS(-Base) and MoECE-ST(-Base) with original and low-rank expert dimensions (Exp Dim) on the BEA-2019 development set (Dev), CoNLL-2014 test set, and BEA-2019 test set. The rows are sorted from the models with the lowest effective parameter count (EPC) to the highest. The low-rank MoECE-ST-Base has a higher EPC than the original because the original merges the shared feed-forward layer with the feed-forward layers in each expert after training.

our MoECE architecture (LAYER SHARING), our MoECE architecture without parameter updates to the shared layer (FROZEN SHARING), and the common mixture-of-experts architecture where the feed-forward layer is replaced by the MoE layer (NO SHARING). With no gradient updates to the shared layer, the shared skill between experts is limited to what was previously learned during the pre-training of the language model. On the other hand, when there is no layer sharing, each expert must learn the general skills independently without any information sharing.

In the layer sharing and frozen sharing experimental settings, the feed-forward layer is initialized with the pre-trained weights from the language model while the experts are initialized randomly. In the experiment without layer sharing, the feed-forward layer is removed (replaced entirely by the MoE layer), so the weights of the feed-forward layer are used to initialize the experts in the MoE layer. Small random noise is added to the expert weights to encourage the experts to specialize in different aspects rather than becoming copies of the same network.

We observe that the model with frozen layer sharing produces a similar $F_{0.5}$ score on the BEA-2019 development set after convergence, but it takes much longer to train. As seen in Figure 6, the model with a shared layer reaches an $F_{0.5}$ score of 54% after 2,100 parameter updates, while the model with a frozen feed-forward layer requires 3,100 parameter updates. On the other hand, removing the shared layer entirely results in a lower score after convergence. Having a shared feed-forward layer in the transformer block adds insignificant computational costs during training, and the costs disappear after merging the layer with each expert during inference.

6.4 Low-Rank Experts

The MoECE architecture bears some resemblance to the Low-Rank Adaptation (LoRA) network (Hu et al., 2022). In MoECE's architecture, the dimensions of the feed-forward layers in the experts are the same as those of the main transformer feed-forward layer. We investigate whether similar performance can be achieved by using experts with feed-forward layers that project the hidden representation into much smaller dimensions and then project back to the original dimensions, similar to LoRA. Using this type of expert network reduces memory usage and disk space, but the main transformer feed-forward layer can no longer be merged into the expert layers due to the difference in dimensions.

In Table 6, we observe that while MoECE-GS with low-rank experts (row #4) does not perform as well, MoECE-ST with low-rank experts (row #3) performs comparably to the original MoECE-ST (row #2), with slightly higher $F_{0.5}$ score on the BEA-2019 test set but lower $F_{0.5}$ score on the CoNLL-2014 test set. This indicates that low-rank experts are a good alternative when memory consumption is a concern.

7 Related Work

In this section, we briefly outline previous work on interpretable grammatical error correction and recent work on mixture of experts for transformer models.

7.1 Interpretable GEC

An interpretable GEC model is desirable as it can help the user understand the rationale behind the correction produced by a GEC model. Since many GEC users are language learners, understanding

the rationale behind the corrections can help them avoid making the same mistakes in the future. Stahlberg and Kumar (2020) propose a model that generates error types during inference, but the main output of the model is a sequence of edits instead of the corrected text. Sequence-tagging models such as GECToR (Omelianchuk et al., 2020) also provide interpretability from the general transformation tags that it outputs, but those are only a small subset of the output tags. Showing the error types may not be considered complete feedback, but error types can still serve as useful feedback to the user (Qorib et al., 2023).

Kaneko et al. (2022) propose a GEC system that produces a corrected text with an example of a similar sentence from the training data to make their model interpretable. Fei et al. (2023) propose a GEC system that accompanies the corrections with the error types and the evidence words. Even so, these approaches still have their own weaknesses. More research is needed on interpretable and explainable grammatical error correction to provide more useful feedback to language learners.

7.2 Mixture of Experts

Mixture of experts is a classic method that was proposed by Jacobs et al. (1991) and reinterpreted for neural networks by Shazeer et al. (2017). Shazeer et al. (2017) reach the state of the art on language modeling and machine translation by applying MoE convolutionally between stacked LSTM layers.

MoE has recently gained popularity thanks to its effectiveness in scaling up the number of parameters of transformer models while maintaining reasonable computation cost (Fedus et al., 2022a). Much research has been done on improving training stability (Du et al., 2021; Zoph et al., 2022), the router (Zhou et al., 2022; Lewis et al., 2021), and the load balancing loss (Lepikhin et al., 2021; Fedus et al., 2022b), but little has been done on its transferability from dense models. Most MoE models go through a pre-training process instead of transferring the weights from a dense model and transforming it into an MoE during fine-tuning, which is what we have done in this work. Gao et al. (2022) propose to expand a pre-trained language model into a mixture of experts, but they use the parameter matrix of the matrix product operator (a tensor decomposition from quantum many-body physics) to be the expert. Their architecture design is significantly different from ours which uses the MoE layer as an addition instead of a replacement of the transformer feed-forward layer.

8 Conclusion and Future Work

In this paper, we present a new grammatical error correction model that is more efficient by utilizing a mixture of experts, called MoECE. Our experiments show that our model can improve the $F_{0.5}$ scores of the comparable dense model by up to 0.55 points on the CoNLL-2014 test set and 1.32 points on the BEA-2019 test set. With the same model and hyper-parameters, the model can improve the $F_{0.5}$ score on out-of-domain test sets by up to 2.59 points on CWEB-G and 1.07 points on CWEB-S. The larger variant of our model, MoECE-GS-Large, successfully reaches performance slightly better than a model, T5-XL, that has three times its effective parameter count and almost double its total parameter count.

Our proposed error type loss makes our model interpretable by producing corrections with error types. Our analysis shows that the error type loss helps in routing the input token to the appropriate expert based on its error type. In addition, we find that our error type loss can be an alternative to existing load balancing loss. We believe that interpretable and explainable grammatical error correction models are needed to help language learners with their study and we hope more research explores this direction.

Limitations

In this work, we only investigate grammatical error correction for English. Our method is applicable to grammatical error correction for other languages when sufficient training data is available. We have not run experiments on larger models due to limitation of our compute budget, but we believe our current experimental configurations are sufficient to empirically demonstrate the effectiveness of our method. We believe our work does not bring any direct harm to individuals or society.

References

Andrey Bout, Alexander Podolskiy, Sergey Nikolenko, and Irina Piontkovskaya. 2023. Efficient grammatical error correction via multi-task training and optimized training schedule. In *Proceedings of EMNLP*, pages 5800–5816.

Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. 2019. The BEA-2019 shared

task on grammatical error correction. In *Proceedings of BEA*, pages 52–75.

Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. Automatic annotation and evaluation of error types for grammatical error correction. In *Proceedings of ACL*, pages 793–805.

Christopher Bryant, Zheng Yuan, Muhammad Reza Qorib, Hannan Cao, Hwee Tou Ng, and Ted Briscoe. 2023. Grammatical error correction: A survey of the state of the art. *Computational Linguistics*, 49(3):643–701.

Shamil Chollampatt and Hwee Tou Ng. 2018. A multilayer convolutional encoder-decoder neural network for grammatical error correction. In *Proceedings of AAAI*, pages 5755–5762.

Shamil Chollampatt, Kaveh Taghipour, and Hwee Tou Ng. 2016. Neural network translation models for grammatical error correction. In *Proceedings of IJCAI*, pages 2768–2774.

Daniel Dahlmeier and Hwee Tou Ng. 2012. Better evaluation for grammatical error correction. In *Proceedings of NAACL*, pages 568–572.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. *ArXiv*, abs/2401.06066.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen S. Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Z. Chen, and Claire Cui. 2021. Glam: Efficient scaling of language models with mixture-of-experts. In *Proceedings of ICML*, pages 5547–5569.

William Fedus, Jeff Dean, and Barret Zoph. 2022a. A review of sparse expert models in deep learning. *ArXiv*, abs/2209.01667.

William Fedus, Barret Zoph, and Noam Shazeer. 2022b. Switch Transformers: scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 23(1):1–39.

Yuejiao Fei, Leyang Cui, Sen Yang, Wai Lam, Zhenzhong Lan, and Shuming Shi. 2023. Enhancing grammatical error correction systems with explanations. In *Proceedings of ACL*, pages 7489–7501.

Simon Flachs, Ophélie Lacroix, Helen Yannakoudakis, Marek Rei, and Anders Søgaard. 2020. Grammatical error correction in low error density domains: A new benchmark and analyses. In *Proceedings of EMNLP*, pages 8467–8478.

Ze-Feng Gao, Peiyu Liu, Wayne Xin Zhao, Zhong-Yi Lu, and Ji-Rong Wen. 2022. Parameter-efficient mixture-of-experts architecture for pre-trained language models. In *Proceedings of COLING*, pages 3263–3273.

Wenjuan Han and Hwee Tou Ng. 2021. Diversity-driven combination for grammatical error correction. In *Proceedings of ICTAI*, pages 972–979.

Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. 2021. FastMoE: A fast mixture-of-expert training system. *ArXiv*, abs/2103.13262.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *Proceedings of ICLR*.

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

Masahiro Kaneko, Sho Takase, Ayana Niwa, and Naoaki Okazaki. 2022. Interpretability for language learners using example-based grammatical error correction. In *Proceedings of ACL*, pages 7176–7187.

Yoav Kantor, Yoav Katz, Leshem Choshen, Edo Cohen-Karlik, Naftali Liberman, Assaf Toledo, Amir Menczel, and Noam Slonim. 2019. Learning to combine grammatical error corrections. In *Proceedings of BEA*, pages 139–148.

Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. 2021. Beyond distillation: Task-level mixture-of-experts for efficient inference. In *Findings of EMNLP*, pages 3577–3599.

Shaopeng Lai, Qingyu Zhou, Jiali Zeng, Zhongli Li, Chao Li, Yunbo Cao, and Jinsong Su. 2022. Type-driven multi-turn corrections for grammatical error correction. In *Findings of ACL 2022*, pages 3225–3236.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. GShard: Scaling giant models with conditional computation and automatic sharding. In *Proceedings of ICLR*.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. 2021. Base layers: Simplifying training of large, sparse models. In *Proceedings of ICML*, pages 6265–6274.

Dingcheng Li, Xu Li, Jun Wang, and Ping Li. 2020. Video recommendation with multi-gate mixture of experts soft actor critic. In *Proceedings of SIGIR*, pages 1553–1556.

Yinghao Li, Xuebo Liu, Shuo Wang, Peiyuan Gong, Derek F. Wong, Yang Gao, Heyan Huang, and Min Zhang. 2023. TemplateGEC: Improving grammatical error correction with detection template. In *Proceedings of ACL*, pages 6878–6892.

Ruixi Lin and Hwee Tou Ng. 2021. System combination for grammatical error correction based on integer programming. In *Proceedings of RANLP*, pages 829–834.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. GECToR – grammatical error correction: Tag, not rewrite. In *Proceedings of BEA*, pages 163–170.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, pages 48–53.

Muhammad Reza Qorib, Geonsik Moon, and Hwee Tou Ng. 2023. ALLECS: A lightweight language error correction system. In *Proceedings of EACL*, pages 298–306.

Muhammad Reza Qorib, Seung-Hoon Na, and Hwee Tou Ng. 2022. Frustratingly easy system combination for grammatical error correction. In *Proceedings of NAACL*, pages 1964–1974.

Muhammad Reza Qorib and Hwee Tou Ng. 2023. System combination via quality estimation for grammatical error correction. In *Proceedings of EMNLP*, pages 12746–12759.

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

Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. 2021. A simple recipe for multilingual grammatical error correction. In *Proceedings of ACL*, pages 702–707.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of ICLR*.

Alexey Sorokin. 2022. Improved grammatical error correction by ranking elementary edits. In *Proceedings of EMNLP*, pages 11416–11429.

Felix Stahlberg and Shankar Kumar. 2020. Seq2Edits: Sequence transduction using span-level edit operations. In *Proceedings of EMNLP*, pages 5147–5159.

Xin Sun and Houfeng Wang. 2022. Adjusting the precision-recall trade-off with align-and-predict decoding for grammatical error correction. In *Proceedings of ACL*, pages 686–693.

Raymond Hendy Susanto, Peter Phandi, and Hwee Tou Ng. 2014. System combination for grammatical error correction. In *Proceedings of EMNLP*, pages 951–962.

Maksym Tarnavskyi, Artem Chernodub, and Kostiantyn Omelianchuk. 2022. Ensembling and knowledge distilling of large sequence taggers for grammatical error correction. In *Proceedings of ACL*, pages 3842–3852.

Zheng Yuan, Felix Stahlberg, Marek Rei, Bill Byrne, and Helen Yannakoudakis. 2019. Neural and FST-based approaches to grammatical error correction. In *Proceedings of BEA*, pages 228–239.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, zhifeng Chen, Quoc V Le, and James Laudon. 2022. Mixture-of-experts with expert choice routing. In *Proceedings of NeurIPS*, pages 7103–7114.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. 2022. ST-MoE: Designing stable and transferable sparse expert models. *ArXiv*, abs/2202.08906.

A Task Interference in T5-v1.1-Large

We observe a similar indication of task interference that we explain in the introduction on the training of T5-v1.1-Large (Figure 7).

B Compute Budget

We list the total number of parameters of our models in Table 7. The training of the base models took about 33.3 hours for each model on a single NVIDIA H100 GPU, while training the large models took about 16.5 hours on two NVIDIA H100 GPUs. The reason why the training times of the base and large models are similar is that large models converged much earlier.

Model	M	EPC	TPC
MoECE-GS-Base	7	282M	490M
MoECE-ST-Base	7	248M	490M
MoECE-GS-Large	7	917M	1.7B
MoECE-ST-Large	7	784M	1.7B

Table 7: Effective parameter counts (EPC) and total parameter counts (TPC) of our models.

C Experiments

The parameters that we need to set to train the models are given in Table 8. The rest follows the default hyper-parameters of Fairseq[6] (Ott et al., 2019).

[6] https://github.com/facebookresearch/fairseq/

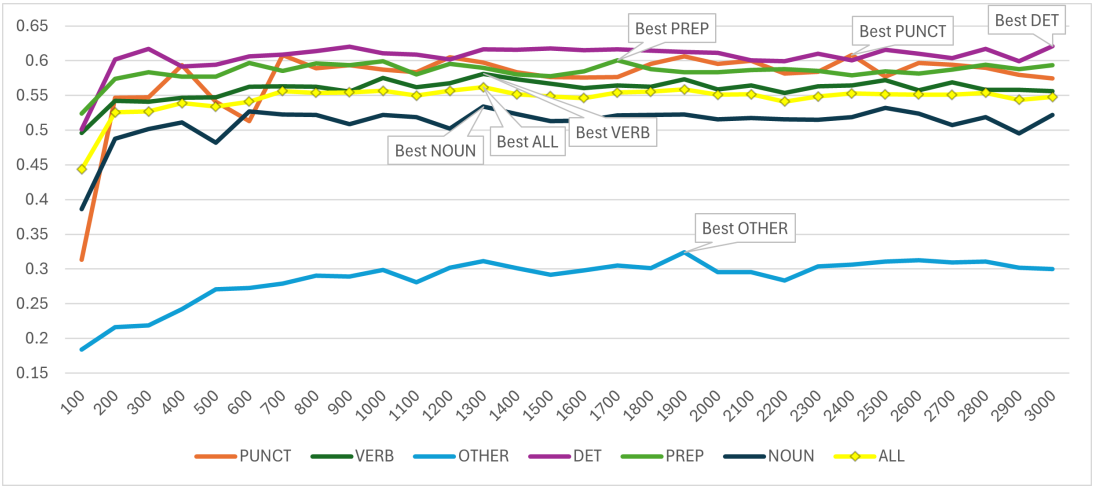


Figure 7: $F_{0.5}$ scores of a T5-v1.1-Large model on different error types in the BEA-2019 development set at different numbers of training steps.

Testing the model does not require specifying the hyper-parameters. We did not conduct an extensive hyper-parameter search, but we specify our parameter bounds in Table 9. Note that we perform hyper-parameter search on the smaller models.

We performed experiments on multiple configurations to verify the effectiveness of our method. Each configuration was only run once, but the results were verified through statistical significance tests that we explain in Section 4.2.

Name	value
# tokens in one gradient update	524,288
Learning rate	0.0002
Optimizer	Adafactor
Max sequence length	128
Expert dropout	0.25
Router hidden dimension	384
α	0.1
β	1.0

Table 8: # tokens in one gradient update is the maximum number of tokens in one batch $\times$ gradient accumulation $\times$ # GPU.

Name	value
# tokens in one gradient update	{524,288, 1,048,576}
Learning rate	{0.0002, 0.0004}
Optimizer	Adafactor
Max sequence length	128
Expert dropout	{0.25, 0.3}
Router hidden dimension	384
α	{0.1, 0.5}
β	{0.1, 0.5, 1.0}

Table 9: The hyper-parameter search bounds.