SUPPLEMENTARY MATERIAL ABOUT SOLVING TOKEN GRADIENT CONFLICT IN MIXTURE-OF-EXPERTS FOR LARGE VISION-LANGUAGE MODEL

Longrong Yang[1], **Dong Shen**[2], **Chaoxiang Cai**[3], **Fan Yang**[2], **Tingting Gao**[2], **Di Zhang**[2], **Xi Li**[1,†]
[1]College of Computer Science and Technology, Zhejiang University
[2]Kuaishou Technology
[3]School of Software Technology, Zhejiang University

In our supplementary material, we provide the following details and experiments:

- Sec. A: We provide more engineering implementation details about training.
- Sec. B: We provide more implementation details about sample-level routing.
- Sec. C: We provide more experimental results about expert loading, loss design, token gradient statistic, and different routing mechanisms.
- Sec. D: We provide a analysis about computational overhead.
- Sec. E: We provide experimental results about language tasks.
- Sec. F: We provide a brief theoretical analysis.
- Sec. G: We provide a deeper analysis about gradient conflicting phenomenon.

A IMPLEMENTATION DETAILS ABOUT TRAINING

Token-level gradient: Each expert is an FFN containing multiple linear layers. For instance, the FFN in Phi-2 (Microsoft, 2023) includes two linear layers, fc^1 and fc^2. Assuming there are $E \times L$ experts, $[e_1, e_2, \cdots, e_{E\times L}]$, where E is the number of experts in each MoE and L is the MoE layer number of LLM. In expert e_i, the weights corresponding to the two linear layers are $w_i^1 \in \mathbb{R}^{D\times D'}$ and $w_i^2 \in \mathbb{R}^{D'\times D}$, and the biases are $b_i^1 \in \mathbb{R}^{D'}$ and $b_i^2 \in \mathbb{R}^{D}$ respectively, where D is the hidden size of LLM and D' is the intermediate size.

First, during the forward pass of a batch, we freeze the parameters except for the biases and compute the main loss. Then, we perform a backward pass, using the Operator "call_for_per_sample_grads" provided by PyTorch to capture the gradients $\mathbf{g}_n^1 \in \mathbb{R}^D$ and $\mathbf{g}_n^2 \in \mathbb{R}^{D'}$ of the token t_n on the biases. Next, we calculate the *average gradients* $\mathbf{g}_{mean}^1$ and $\mathbf{g}_{mean}^2$. Let the tokens processed by the expert e_i be denoted as $\{t_1, \cdots, t_{N_{e_i}}\}$, the *average gradients* are represented as:

$$\begin{aligned} \mathbf{g}_{mean}^1 &= \frac{\sum_{n=1}^{N_{e_i}} \mathbf{g}_n^1}{N_{e_i}}, \\ \mathbf{g}_{mean}^2 &= \frac{\sum_{n=1}^{N_{e_i}} \mathbf{g}_n^2}{N_{e_i}} \end{aligned} \tag{1}$$

where $\mathbf{g}_{mean}^1 \in \mathbb{R}^D$ and $\mathbf{g}_{mean}^2 \in \mathbb{R}^{D'}$. Following that, we compute the cosine similarity between $\mathbf{g}_n^1$ and $\mathbf{g}_{mean}^1$ as s_n^1 and the cosine similarity between $\mathbf{g}_n^2$ and $\mathbf{g}_{mean}^2$ as s_n^2. Thus, for the token t_n, the similarity metric s_n is defined as:

$$s_n = \frac{s_n^1 + s_n^2}{2} \tag{2}$$

Finally, we identify the *conflicting token*: *when s_n is lower than the threshold τ, the token t_n is a conflicting token.*

†Corresponding author is Xi Li.

Table A: **Hyper-parameters in training.**

Instruction Tuning	Epoch 1	Learning rate 2e-5	Learning rate schedule Cosine	Weight decay 0.0
Instruction Tuning	Text max length 2048	Batch size per GPU 16	GPU 8×A800-80G	Precision Bf16

Data	Size	Response formatting prompts
LLaVA	158K	–
ShareGPT	40K	–
VQAv2	83K	Answer the question using a single word or phrase.
GQA	72K	
OKVQA	9K	
OCRVQA	80K	
A-OKVQA	66K	Answer with the option's letter from the given choices directly.
TextCaps	22K	Provide a one-sentence caption for the provided image.
RefCOCO	48K	*Note: randomly choose between the two formats* Provide a short description for this region.
VG	86K	Provide the bounding box coordinate of the region this sentence describes.
Total	665K	

Table B: Instruction-following data mixture. The data is from LLaVA-1.5 (Liu et al., 2023a).

Why to use the gradients on the biases? This engineering trick brings two advantages. (i) Assume the gradients of the token t_n on the weights be $\mathbf{g}_n^{1,w}$ and $\mathbf{g}_n^{2,w}$ respectively. $\mathbf{g}_n^{1,w} \in \mathbb{R}^{D \times D'}$ and $\mathbf{g}_n^{2,w} \in \mathbb{R}^{D' \times D}$. Thus, we need a significant GPU memory overhead to store gradients. If storing only the gradients on the biases, the GPU memory overhead significantly reduces. (ii) When the parameter size in the computational graph is larger, the backward pass is longer. We compute only the gradients on the biases, so it is very fast to capture gradients. *Whether does this operation work?* Similar to the computation of s_n, we use $\mathbf{g}_n^{1,w}$ and $\mathbf{g}_n^{2,w}$ to compute the similarity metric s_n^w of the token t_n. We then compute the Pearson correlation coefficient between s_n^w and s_n ($n \in \{1, \cdots\}$) and find the value is usually larger than 0.9. Thus, we believe that s_n reveals the relationship between the token gradient and the *average gradient* in the expert well. Our experiments also verify that the proposed STGC can increase performance.

After identifying *conflict tokens*, we add the parameters that need to be updated into the optimizer (we follow the training configure of MoE-LLaVA). We add the conflict elimination loss to optimize the router based on the identification of *conflict tokens*.

Training Scheme: Our training scheme follows MoE-LLaVA (Lin et al., 2024). The details are presented in Table A. During instruction fine-tuning, we use a batch size of 128 and a learning rate of 2e-5. We directly use the pre-trained models from MoE-LLaVA (Lin et al., 2024) to conduct instruction tuning.

Training Datasets: We use LLaVA-mix-665k (Liu et al., 2023a) as instruction tuning training data to conduct most experiments. The data structure is presented in Table B. To verify the scalability of the STGC model, we conducted experiments using 1021K data from the Open-LLaVA-NeXT dataset (Lin & Long, 2024). In Table 1 of the main paper, we report the best performance that we achieve when activating only 3.6B parameters during inference, using 1021K data for training.

B Implementation Details about Sample-level Routing Schemes

Embedding-based: Similar to MoCLE (Gou et al., 2023), we encode all the instructions of different datasets using the all-MiniLM-L6-v2 variant of the Sentence Transformer model (Reimers, 2019)

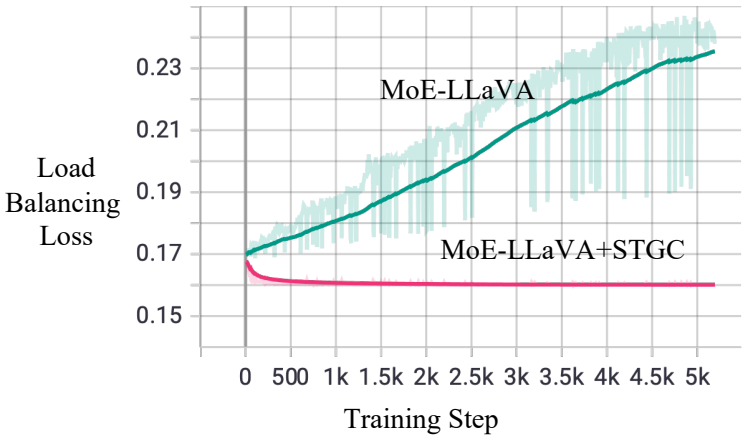

Figure A: **Load balance loss**. "Baseline" indicates MoE-LLaVA. "Baseline+STGC" indicates our method. We present the load balancing loss curve before and after adding STGC. The results are obtained from the regular training. The total training step count is 5194 for an epoch. When the load balancing loss is lower, the expert load is more balanced.

and cluster their embeddings via K-means clustering algorithm. After clustering, following the practice of MoCLE, we initialize K learnable embeddings, and each embedding corresponds to a cluster center. When a sample belongs to the k-th cluster center, the k-th learnable embedding is extracted and fed into the router to predict routing scores. In our experiments, we set K=128. Following the practice of MoCLE, we do not add the load balance loss.

Feature-based: Similar to LoRA-MoE (Chen et al., 2023), we take the average of instruction token representations of each instance as input to predict its routing scores in each expert. Then, the Top-k experts are selected based on routing scores for each sample to generate the prediction. Following the practice of LoRA-MoE, we do not add the load balance loss.

Task-based: Similar to MoLA (Zhou et al., 2024), we promote similarity in routing between data from the same task while emphasizing distinctiveness in routing between data from different tasks. We employ LLaVA-mix-665k (Liu et al., 2023a) to conduct experiments, significantly different from the used data in MoLA (Zhou et al., 2024). Therefore, we empirically divide the data into four types of tasks.: (i) *Caption*. For example, the instruction is "Provide a one-sentence caption for the provided image.". (ii) *VQA*. For example, the instruction is "Answer the question using a single word or phrase.". (iii) *OCR*, including all data in OCRVQA. (iv) *Region-aware*. For example, the instruction is "Provide a short description for this region.". The expert label of *Caption*, *VQA*, *OCR*, or *Region-aware* data is 0, 1, 2, or 3, respectively. The ratio of *Caption*, *VQA*, *OCR*, or *Region-aware* data is 3.5%, 61.6%, 12.8%, and 22.1%, respectively.

C More Experimental Results

C.1 Expert Loading

Loss Curve: We present the the load balancing loss curve in Figure A. As shown in Figure A, the proposed STGC benefits the decrease of the load balancing loss. A possible reason is that the expert load imbalance means that many tokens are routed to an expert, significantly increasing the possibility that tokens have gradient conflicts. After adding the STGC, some tokens are moved from the "crowded" expert (many tokens) to the "empty" expert (few tokens). This may be also a new perspective on why the load balance is important to the MoE system.

Additional Ablations on α: α is the weighting of the load balance loss. We discuss different loss weightings $\alpha \in \{0.01, 0.1\}$ (0.01 is the standard value set in MoE-LLaVA (Lin et al., 2024)). As shown in Table C, we find that increasing the weighting of the load balance loss degenerates the model performance.

Table C: **Study about the load balance loss weighting α.** The configure MoE-LLaVA-4Top2 with StableLM-1.6B is set for experiments.

	α	VQA^{v2}	GQA	VisWiz	SQA^{I}	VQA^{T}	POPE	MME	MMB	MM-Vet	Avg
MoE-LLaVA	0.01	76.7*	60.3*	36.2	62.6	50.1	85.7	1318.2	60.2	26.9	57.3
MoE-LLaVA	0.1	75.7*	59.7*	37.7	61.3	49.9	85.6	1338.0	60.4	27.2	57.2

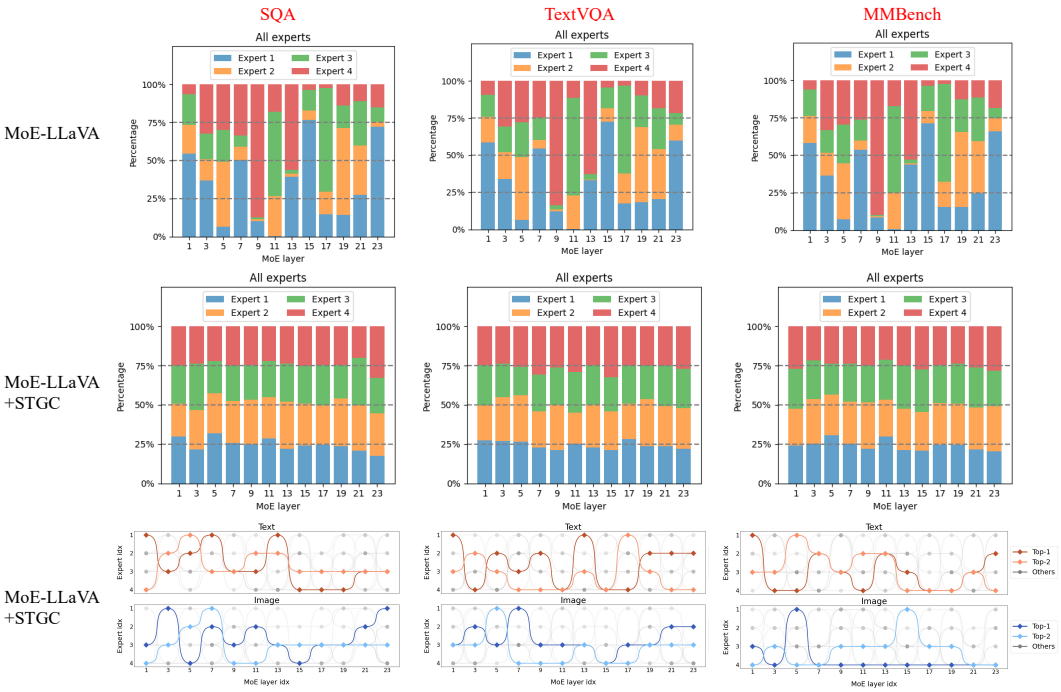

Figure B: **Expert Loading and activated pathways**. The configure MoE-LLaVA-4Top2 with StableLM-1.6B is set for experiments We select three validation datasets, *i.e.*, SQA (Lu et al., 2022), TextVQA (Singh et al., 2019), and MMBench (Liu et al., 2023b), to analyze expert loading and activated pathways. In activated pathways, the colorful paths represent the top-2 paths for text and image, respectively, while the gray paths represent the remaining 8 paths.

Visualization of Expert Loading: we follow MoE-LLaVA (Lin et al., 2024) to obtain the distribution of expert loading and the visualization of the activated pathways. The distribution of expert loading examines the expert use frequency for all tokens (Lin et al., 2024). Activated pathways examine the behavior of experts at the token level (Lin et al., 2024): this visualization tool tracts the activated pathways of all tokens on validation datasets; given all activated pathways, the visualization tool employs PCA to obtain the top-10 pathways. As shown in Figure B, we find that (i) STGC benefits the expert load balance. A possible reason is that the expert load imbalance means that many tokens are routed to an expert, significantly increasing the possibility that tokens have gradient conflicts. After adding the STGC, some tokens are moved from the "crowded" expert (many tokens) to the "empty" expert (few tokens). This further validates that STGC would effectively utilize each expert, instead of collapsing into only using one expert, which is also a new perspective on why the load balance is important to the MoE system. (ii) The activated pathways are significantly different for SQA (Lu et al., 2022), TextVQA (Singh et al., 2019), and MMBench (Liu et al., 2023b). This implies that although the distribution of expert load across different datasets is similar, the token routing behavior is still significantly different among datasets, *i.e.*, different tokens have been assigned to various experts.

Table D: **Study about different conflict elimination loss designs.** The configure MoE-LLaVA-4Top2 with StableLM-1.6B is set for experiments.

	VQA^{v2}	GQA	VisWiz	SQA^{I}	VQA^{T}	POPE	MME	MMB	MM-Vet	Avg
MSE-like	76.7*	60.7*	37.0	62.8	50.6	85.7	1346.5	60.6	27.8	57.7
CE-like	76.9*	60.9*	37.7	62.6	50.7	85.9	1355.1	60.7	28.2	58.0

C.2 Loss Design

The goal of the conflict elimination loss is to reduce the routing score $p_{\text{moe}}(t_n)$ of the *conflicting token* t_n on its current expert. We discuss different designs for the conflict elimination loss: (i) MSE-like: Simply setting the routing score $p_{\text{moe}}(t_n)$ to the minimum.

$$
\begin{aligned}
p_{\text{moe}}(t_n)_i &= \frac{e^{z_{\text{moe}}(t_n)_i}}{\sum_{j=1}^{E} e^{z_{\text{moe}}(t_n)_j}}, \\
\mathcal{L}_{\text{CEL}}^{\text{MSE}} &= \frac{1}{N_{all} \cdot E} \sum_{n=1}^{N_{all}} \sum_{i=1}^{E} p_{\text{moe}}(t_n)_{id_{\text{moe},n}},
\end{aligned}
\tag{3}
$$

where N_{all} is the count of all *conflicting tokens*, E is the number of experts, and $p_{\text{moe}}(t_n)$ represents the routing score for the *conflicting token* t_n. $id_{\text{moe},n}$ is the current expert ID of the *conflicting token* t_n. (ii) CE-like: Utilizing the inverted routing score along with cross-entropy loss. Our motivation for taking the inverted routing score is to minimize the routing score of token on its current expert.

$$
\begin{aligned}
z'_{\text{moe}}(t_n) &= -z_{\text{moe}}(t_n), \\
p'_{\text{moe}}(t_n)_i &= \frac{e^{z'_{\text{moe}}(t_n)_i}}{\sum_{j=1}^{E} e^{z'_{\text{moe}}(t_n)_j}}, \\
\mathcal{L}_{\text{CEL}} &= \frac{1}{N_{all} \cdot E} \sum_{n=1}^{N_{all}} \sum_{i=1}^{E} \log(p'_{\text{moe}}(t_n)_i) \cdot q_{\text{moe}}(t_n)_i.
\end{aligned}
\tag{4}
$$

As shown in Table D, the CE-like loss performs better. The reason may be that, although the optimization direction of the MSE-like loss is consistent with the CE-like loss, the optimization speed of the CE loss is superior to the MSE loss (Zhang & Sabuncu, 2018; Wang et al., 2019; Yang et al., 2020). An analysis of the gradient of CE (Yang et al., 2020) reveals that when the probability of the sample on the ground-truth class is small, CE will produce a significantly larger gradient than MSE.

Loss Curve: Different from the statistical verification in Figure **??** that only uses conflict elimination loss $\mathcal{L}_{\text{moe}}$, $\mathcal{L}_{\text{CEL}}$, $\mathcal{L}_{\text{moe}}$, and $\mathcal{L}_{\text{aux}}$ are used during the normal training process. We present the loss curve of the conflict elimination loss during training. As shown in Figure C, the loss is convergent.

C.3 Token Gradient Statistic

Similarity statistic: We compute the distribution of cosine similarity between the gradient of each token and the averaged gradient. Specifically, we randomly sample some data. We extract token-level gradients, calculate the average gradient $\mathbf{g}_{mean}$ for each expert, and compute the cosine similarity between the gradient of each token $\mathbf{g}_n$ and the average gradient of its current expert $\mathbf{g}_{mean}$. We perform the statistics for both initial and fully-trained models. As shown in Figure D, we find that: (i) In the initial model, there is a conflict between token gradients and average gradients; (ii) In the fully-trained model, the conflict between token gradients and average gradients is reduced. This verifies that STGC can effectively increase the cosine similarity between the gradients of tokens within an expert and the average gradient.

***Gradient consistency std* and conflicting ratio**: We further explore the std deviation of *gradient consistency* and the ratio of *conflicting tokens*. Expanding on the statistical verification in the main part, we conduct a statistical verification experiment and define three metrics: The first step is to gather gradients of all tokens within an expert e_i and compute their cosine similarity to form a

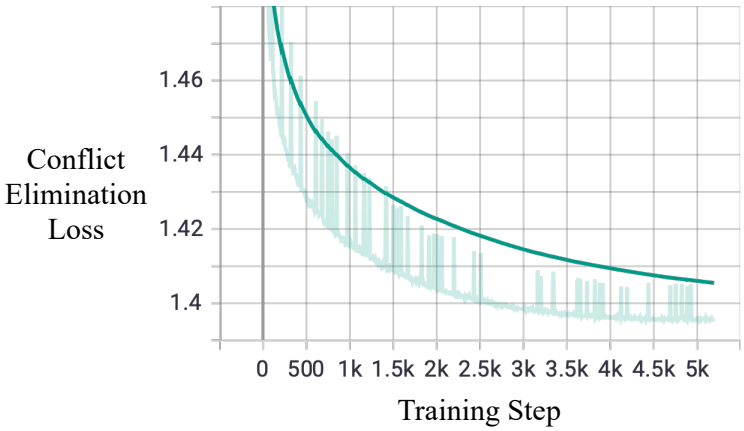

Figure C: **Loss curve of conflict elimination loss**. The above graph shows the loss curve of conflict elimination loss during the normal training process when Phi2-2.7B is used as the LLM. Since the total number of sampling points is limited to 1000 in TensorBoard, the sampling interval is set to 7.

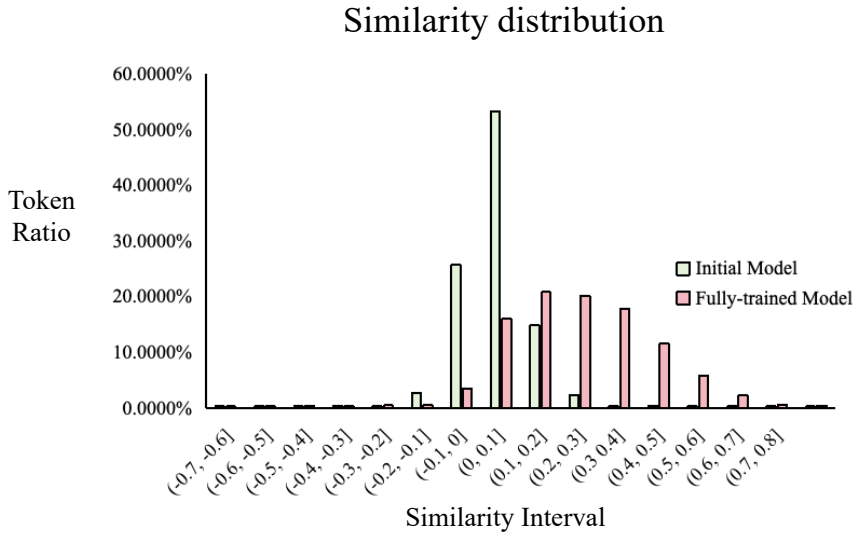

Figure D: **Gradient similarity distribution**. We compute the distribution of cosine similarity between the gradient of each token and the averaged gradient. The configure MoE-LLaVA-4Top2 with StableLM-1.6B is set for experiments. We add the proposed STGC to train models, to generate a fully-trained model from the initial model.

similarity matrix. Then, we define the mean of the similarity matrix as sim_i. We define the mean of sim_i on all experts as *gradient consistency*, serving as a novel metric to evaluate whether the gradient directions are consistent within the expert. We define the std deviation of sim_i on all experts as *gradient consistency std*, serving as a metric to evaluate the degree of discreteness in the *gradient consistency* among different experts. Additionally, we calculate the number of *conflicting tokens* within all experts as N_1, and the total number of tokens as N, defining the conflicting ratio as $\frac{N_1}{N}$. The second step is to update models. We only use the proposed loss to update parameters, to undisturbedly observe its impact on three metrics.

We present the curve graph of the three metrics obtained from the TensorBoard dashboard in Figure E. We can observe: (i) as the conflict elimination loss decreases, the *gradient consistency* increases and the conflicting ratio decreases. This means that STGC effectively reduces gradient conflicts within the expert and reduces the count of *conflicting tokens*. (ii) The *gradient consistency std* increases, meaning that the difference of *gradient consistency* among different experts enlarges.

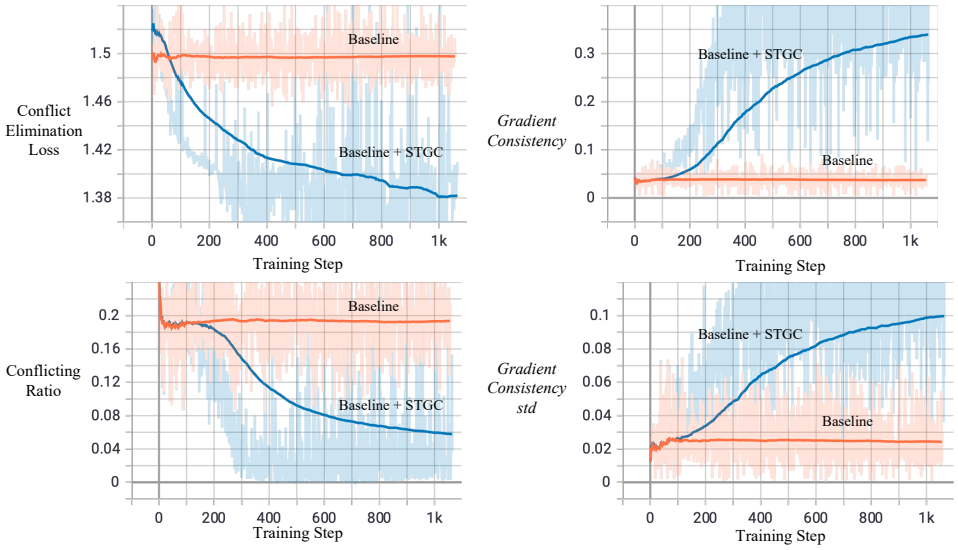

Figure E: ***Gradient consistency* and conflicting ratio analysis**. The configure MoE-LLaVA-4Top2 with StableLM-1.6B is set for experiments. We finish the statistic on one GPU, so the total step number is 41581. The sampling interval is set to 1 in TensorBoard for the above graph and the sampling interval is 7 in TensorBoard for Figure **??**, so the above graph appears to have a slight difference with Figure **??**.

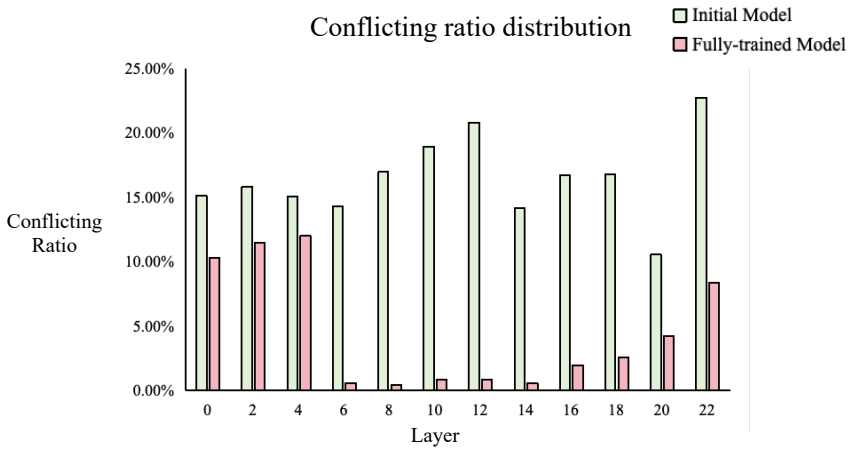

Figure F: **Conflicting ratio distribution** on different MoE layers after learning.

We speculate that this is due to the different rates at which *gradient consistency* increases across various layers. For example, Figure **??** (b) indicates that deeper layers have faster *gradient consistency* increase rate after adding STGC. Figure F also shows that STGC mainly reduces *conflicting tokens* at deep layers, and most *conflicting tokens* emerge in the shallow layers after learning.

Conflicting token after learning: In Figure D, although we observe a significant reduction in *conflicting tokens* for the fully-trained model, we find that there are still some *conflicting tokens*. As shown in Figure F, we analyze the layers in which they appear and find that most *conflicting tokens* emerge in the shallow layers after learning.

Table E: **Study about Batch Prioritized Routing (BPR).** The configure MoE-LLaVA-4Top2 with StableLM-1.6B is set for experiments. BPR refers to Batch Prioritized Routing.

	eval capacity	BPR	GQA	SQA^I	VQA^T	POPE	MME	MMB	MM-Vet	Avg
MoE-LLaVA	2.0		60.3*	62.6	50.1	85.7	1318.2	60.2	26.9	57.6
+STGC	2.0		60.9*	62.6	50.7	85.9	1355.1	60.7	28.2	58.2
MoE-LLaVA	0.5		10.2*	fail	15.2	69.8	fail	6.0	16.4	-
+STGC	0.5		21.1*	18.0	13.6	71.9	fail	7.0	20.7	-
MoE-LLaVA	0.5	✓	51.0*	fail	33.4	83.8	1032.4	26.8	22.0	-
+STGC	0.5	✓	58.0*	57.7	44.3	85.3	1234.3	48.5	22.8	52.8

C.4 Routing

SMoE (Jiang et al., 2024) claims that "Surprisingly, we do not observe obvious patterns in the assignment of experts based on the topic." As shown in Figure B, we also observe that the distribution of expert loading across some different datasets is similar, but we notice diversity in token-level activated pathways for different datasets. We suspect that the distribution of expert loading may not be sufficiently accurate to reflect the routing of diverse data. Then, V-MoE (Riquelme et al., 2021) and MoNE (Jain et al., 2024) focus on leveraging the token importance difference to further accelerate MoE. V-MoE proposes Batch Prioritized Routing to discard unimportant tokens and MoNE proposes Expert Preferred Router to allocate more tokens to experts with a larger volume. We focus on avoiding token interference during training to enhance performance, which is parallel to the focus of V-MoE or MoNE. Thus, theoretically, STGC could be integrated with V-MoE or MoNE. Since MoNE does not have the official open-source code, we attempt to use Batch Prioritized Routing from V-MoE for further inference acceleration.

During inference, the eval capacity of MoE is set to 2.0 Lin et al. (2024). As shown in Table E, when we reduce the capacity from 2.0 to 0.5, the model performance of both MoE-LLaVA and MoE-LLaVA+STGC declines significantly because many tokens are discarded. When Batch Prioritized Routing is added, there is a noticeable performance improvement. We find that when the capacity is reduced to 0.5, regardless of whether Batch Prioritized Routing is added or not, MoE-LLaVA+STGC shows a significant performance improvement compared to MoE-LLaVA. A possible reason is that STGC can prevent a single expert from handling too many tokens, thereby reducing the number of discarded tokens.

D Computational Overhead

STGC does not increase the inference overhead, while it may need the memory and time overhead during training. The memory and time overhead mainly results from storing and computing gradients respectively. We analyze the additional overhead during training from these two aspects.

Training memory overhead. We begin our analysis from StableLM-1.6B. The FFN layer in StableLM-1.6B contains three linear layers, $fc^{gate} \in R^{2048\times5632}$, $fc^{up} \in R^{2048\times5632}$, and $fc^{down} \in R^{5632\times2048}$. As stated in Sec. A, for each token, we only store the gradient it produces on the bias within the experts. The token count per layer is about 1000, and each token only goes through one expert, so the gradient matrix stored per layer is $\mathbf{G} \in R^{1000\times(5632+5632+2048)}$. We need to store a gradient matrix $\mathbf{G}$ for each MoE layer, and the MoE layer in StableLM-1.6B is $\{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22\}$ (the total layer number is 24). Thus, the theoretical memory overhead is $12\times1000\times(5632+5632+2048)$. The parameter is bfloat16 (2 bytes), so the theoretical memory overhead is about 0.29 GB, which is significantly less than the memory cost of LLM and data (usually larger than 10 GB during training). The FFN layer in Phi2-2.7B contains two linear layers, $fc^1 \in R^{2560\times10240}$ and $fc^2 \in R^{10240\times2560}$, and Phi2-2.7B has 16 MoE layers. Thus, the theoretical memory overhead is $16 \times 1000 \times (2560 + 10240)$, *i.e.*, 0.38 GB.

If storing the token-level gradients on each weight, the required storage overhead is $12 \times 1000 \times (2048\times5632+2048\times5632+5632\times2048)$ (773 GB) for StableLM-1.6B and $16\times1000\times(2560\times$

Table F: **Study about computational overhead.** We set MoE-LLaVA-4Top2 with StableLM-1.6B and Phi2-2.7B to conduct the analysis. One step (s) means the time needed for one step during training. STGC-full means computing token-level gradients on the weight.

LLM		Memory theoretical overhead	Time train_samples_per_second	Time train_steps_per_second	Time one step (s)
StableLM-1.6B	MoE-LLaVA	-	19.796	0.154	6.494
	+STGC	0.29 GB	16.283	0.127	7.874 (+21.3%)
	+STGC-full	773 GB	fail	fail	fail
Phi2-2.7B	MoE-LLaVA	-	10.765	0.084	11.905
	+STGC	0.38 GB	8.851	0.069	14.493 (+21.7%)
	+STGC-full	1562 GB	fail	fail	fail

Table G: **Study about language tasks.** Our study investigates the MoE integrated with STGC on language tasks using the GLUE benchmark (Wang, 2018), with BERT-large as the backbone model. MoE-8Top2 means a traditional MoE, configured with 8 experts, of which the Top-2 are activated, *i.e.*, 8Top2. * means the re-implemented results.

	COLA	MRPC	QNLI	MNLI	RTE	Avg
MoE-8Top2 (Guo et al., 2024)	64.5	90.2	92.4	86.7	74.9	81.7
DYNMOE (Guo et al., 2024)	65.2	90.6	92.6	86.4	73.4	81.6
MoE-8Top2*	64.5	90.0	93.4	86.9	72.9	81.5
+STGC	66.8	91.2	93.8	87.6	74.7	82.8

$10240 + 10240 \times 2560$) (1562 GB) for Phi2-2.7B, which is amazingly large. Thus, it is impossible to use the token-level gradient on each weight to conduct experiments.

Training time overhead. MoE-LLaVA performs a forward pass, followed by a backward pass to update parameters. As Section Sec. A mentioned, MoE-LLaVA+STGC freezes parameters except for the bias within the experts, performs a forward pass, followed by a backward pass to compute token-level gradients; then, MoE-LLaVA+STGC unfreeze parameters and performs a backward pass to update parameters. Thus, the main time overhead results from "the computation of token-level gradients". We directly report the train_samples_per_second and train_steps_per_second recorded in "trainer_state.json" after training. As shown in Table F, we have reduced the additional time overhead to about 20% through some engineering tricks (*e.g.*, only computing the token-level gradient on the bias and freezing parameters that do not require gradient computation).

E Language Tasks

We study the use of STGC on language tasks. Theoretically, the deployment of STGC is not constrained to specific task types. To validate the generalization of STGC, we extend its use to language tasks. Specifically, we follow DYNMOE Guo et al. (2024) to apply the MoE framework for language tasks. Specifically, the language tasks adhere to the same settings as those in MoEfication Zhang et al. (2021) and EMoE (Qiu et al., 2023). The MoE is built upon the BERT-large Devlin (2018) architecture, employing the MoEfication method, and is fine-tuned on GLUE Wang (2018) benchmark, which encompasses COLA Warstadt (2019), QNLI Wang (2018), RTE Bentivogli et al. (2009), MNLI Xu et al. (2020), and MRPC Dolan & Brockett (2005). For MoE configuration, we set the total count of experts to 8, with the Top-2 experts being activated, and we refer to this configuration as 8Top2. We add the proposed STGC to the MoE. As shown in Table I, the experimental results show the significant effectiveness of STGC on language tasks.

F Theoretical Analysis

We conduct a brief theoretical analysis based on the theory of PCGrad (Yu et al., 2020). Suppose there are tokens t_n and t'_n, which pass through the same expert and generate gradients $\mathbf{g_n}$ and $\mathbf{g_{n'}}$

on that expert. Let $\mathbf{g_n} = \nabla\mathcal{L}_n$, $\mathbf{g_{n'}} = \nabla\mathcal{L}_{n'}$, and $\mathbf{g} = \nabla\mathcal{L} = \mathbf{g_n} + \mathbf{g_{n'}}$ ($\nabla\mathcal{L} = \nabla_\theta\mathcal{L}$, where θ is the parameter). $cos(\phi_{nn'})$ is the cosine similarity between gradients $\mathbf{g_n}$ and $\mathbf{g'_n}$. Token gradient conflicts mean the cosine similarity $cos(\phi_{nn'}) < 0$, $cos(\phi_{nn'}) < 0$ potentially leads to an increase in the loss. When $cos(\phi_{nn'}) > 0$, the loss decreases strictly, *i.e.*, $\nabla\mathcal{L} < 0$, which can reach the optimal value. Then, different from PCGrad, STGC does not alter the gradients of the tokens but changes the routing of the tokens to avoid conflicting gradients, with the function of increasing the cosine similarity $cos(\phi_{nn'})$ to satisfy the condition $cos(\phi_{nn'}) > 0$.

The notation $||\cdot||$ represents the L_2-norm. During each iteration, if $\cos(\phi_{nn'}) > 0$, a standard gradient descent step with a learning rate $t \leq \frac{1}{L}$ is employed. This results in a strict reduction in the value of the objective function $\mathcal{L}(\theta)$, given that the function is convex, unless the gradient $\nabla\mathcal{L}(\theta)$ is zero, which happens exclusively when θ equals the optimal value θ^* (Boyd & Vandenberghe, 2004).

We further analyze the loss: Assuming that the gradient of the loss $\nabla\mathcal{L}$ is Lipschitz continuous with a Lipschitz constant L, it implies that the Hessian matrix $\nabla^2\mathcal{L}(\theta) - LI$ is negative semi-definite. Leveraging this property, we can perform a quadratic expansion of $\mathcal{L}$ around $\mathcal{L}(\theta)$, leading to the subsequent inequality:

$$\begin{aligned}\mathcal{L}(\theta^+) &\leq \mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^T(\theta^+ - \theta) + \frac{1}{2}\nabla^2\mathcal{L}(\theta)||\theta^+ - \theta||^2 \\ &\leq \mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^T(\theta^+ - \theta) + \frac{1}{2}L||\theta^+ - \theta||^2\end{aligned}$$

Then, given $\theta^+ = \theta - t\cdot\mathbf{g}$, the inequality can be expressed as:

$$\begin{aligned}&\mathcal{L}(\theta^+) \leq \mathcal{L}(\theta) - t\cdot\mathbf{g}^T\mathbf{g} + \frac{1}{2}Lt^2||\mathbf{g}||^2 \\ &\text{(Expanding, using the identity } \mathbf{g} = \mathbf{g_n} + \mathbf{g_{n'}}) \\ &= \mathcal{L}(\theta) - (t - \frac{1}{2}Lt^2)(||\mathbf{g_n}||^2 + ||\mathbf{g_{n'}}||^2 + 2\cdot\mathbf{g_n}\cdot\mathbf{g_{n'}}) \\ &\text{(Using the identity } \cos(\phi_{nn'}) = \frac{\mathbf{g_n}\cdot\mathbf{g_{n'}}}{||\mathbf{g_n}||||\mathbf{g_{n'}}||}) \\ &= \mathcal{L}(\theta) - (t - \frac{1}{2}Lt^2)\cdot(||\mathbf{g_n}||||\mathbf{g_{n'}}||)(\frac{||\mathbf{g_n}||}{||\mathbf{g_{n'}}||} + \frac{||\mathbf{g_{n'}}||}{||\mathbf{g_n}||} + 2\cdot\cos(\phi_{nn'})).\end{aligned} \tag{5}$$

By setting $t \leq \frac{1}{L}$, we can obtain $-(1 - \frac{1}{2}Lt) = \frac{1}{2}Lt - 1 \leq \frac{1}{2}L(1/L) - 1 = \frac{-1}{2}$ and $Lt^2 \leq t$.

Incorporating the bound into the previous expression, we can deduce the following conclusion:

$$\mathcal{L}(\theta^+) \leq \mathcal{L}(\theta) - \frac{1}{2}t(||\mathbf{g_n}||||\mathbf{g_{n'}}||)(\frac{||\mathbf{g_n}||}{||\mathbf{g_{n'}}||} + \frac{||\mathbf{g_{n'}}||}{||\mathbf{g_n}||} + 2\cdot\cos(\phi_{nn'})).$$

If $\cos(\phi_{nn'}) < -\frac{1}{2}\cdot(\frac{||\mathbf{g_n}||}{||\mathbf{g_{n'}}||} + \frac{||\mathbf{g_{n'}}||}{||\mathbf{g_n}||})$, $\frac{||\mathbf{g_n}||}{||\mathbf{g_{n'}}||} + \frac{||\mathbf{g_{n'}}||}{||\mathbf{g_n}||} + 2\cdot\cos(\phi_{nn'})$ will be negative. The bound of $\mathcal{L}(\theta^+)$ is larger than $\mathcal{L}(\theta)$, so the loss may increase. If $\cos(\phi_{nn'}) > 0$, $\frac{||\mathbf{g_n}||}{||\mathbf{g_{n'}}||} + \frac{||\mathbf{g_{n'}}||}{||\mathbf{g_n}||} + 2\cdot\cos(\phi_{nn'})$ will always be positive. This positivity ensures that the objective function value decreases strictly with each iteration, suggesting that by repeatedly applying this process, we can converge to the optimal value. Note that this result only holds when we select a sufficiently small learning rate, specifically $t \leq \frac{1}{L}$.

The goal of STGC is to increase $cos(\phi_{nn'})$ to satisfy the condition $cos(\phi_{nn'}) > 0$. Figure **??** and Figure E have verified that STGC can increase $cos(\phi_{nn'})$. Consequently, STGC is beneficial to the convergence of the objective function towards its optimal value, thereby improving the overall effectiveness of the model.

Table H: **Study about feature-gradient relationship.** The configure MoE-LLaVA-4Top2 with StableLM-1.6B is set for experiments.

	Pearson correlation coefficient
MoE-LLaVA	0.2654
+ STGC	0.3563

Table I: **Study about gradient conflicting.** z'_{i,y_i} denotes the average logit of tokens for their corresponding labels when calculating the main loss. z'_{i,y_j} means the average logit of tokens on the labels of their conflicting tokens (meaning tokens having similar feature but divergent gradients). $z'_{i,o}$ denotes the average logit of tokens for all other labels, excluding the aforementioned labels.

	z'_{i,y_i}	z'_{i,y_j}	$z'_{i,o}$
Value	23.4885	14.5673	-0.6793

G Gradient Conflicting Phenomenon

G.1 Feature-Gradient Relationship

Similar features do not necessarily imply similar gradients (Mu et al., 2020). When tokens have highly similar features (similar tokens), the router can assign them to one expert. However, they may have dissimilar gradients. STGC can be understood as learning token features based on token-level gradient relationships. After using STGC, the features of tokens become less similar when they have dissimilar (conflicting) gradients, for being routed to different experts. To verify this, we sample some data for analysis. Suppose there are N tokens and the cosine similarity is $S \in R^{N \times N}$ between features of these tokens. We flatten $S \in R^{N \times N}$ to $\bar{S} \in R^{N^2}$. Meanwhile, we calculate the cosine similarity $S_g \in R^{N \times N}$ between gradients of these tokens, flattening it to $\bar{S}_g \in R^{N^2}$. We compute the Pearson correlation coefficient between $\bar{S}$ and $\bar{S}_g$ to measure whether feature relationships reveal gradient relationships. We select MoE-LLaVA and MoE-LLaVA + STGC for our experiments. As shown in Table H, the experimental results indicate that token features and gradients have a higher correlation after using STGC.

G.2 Possible Reason for Token Gradient Conflicting

In theory, similar tokens will be assigned to the same expert by a feature-based router. We focus on token gradient conflicting within an expert, so we want to further conduct a study about the phenomenon "similar tokens have divergent gradients". In specific, we compute the cosine similarities between token features in an expert as S. We then compute the cosine similarities between token gradients in an expert as S_g. We directly compute the difference $S' = S - S_g$. When $S'_{i,j}$ is large, token t_i and token t_j has a high feature similarity but a low gradient similarity, meaning that the phenomenon "similar tokens have divergent gradients" is significant. We find that $S'_{i,j}$ is large when two tokens are confusing. Suppose labels of two tokens t_i on y_i are y_i and y_j respectively. z is the logit of each token for computing the main loss. z_{i,y_i} means the logit of token t_i on y_i. We define that t_i and t_j are confusing when z_{i,y_j} and z_{j,y_i} are high.

For example, in one expert, S' has the maximum at $S'_{78,51}$. $S_{78,51}$ is 0.9316 and $S'_{78,51}$ is -0.6875. t_{78} has the label 13 and t_{51} has the label 11. We find that t_{78} (t_{51}) has a second highest logit on 11 (13). This is somewhat similar to the class confusion phenomenon in traditional visual classification tasks. For example, in visual classification, dog and cat are visually similar and they have similar features, but the learning of cat may lead to the misclassification of dog.

We further sample some pairs of tokens to conduct a statistical verification. Each pair of tokens $\{t_i, t_j\}$ has the significant feature and gradient cosine similarity difference $S'_{i,j}$. z_{i,y_i} means the logit of token t_i on y_i. In addition, removing z_{i,y_i} and z_{i,y_j}, we record the mean logit in z_i as $z_{i,o}$. z'_{i,y_j} is the average of z_{i,y_i}. As shown in the below table, z'_{i,y_i} is significantly higher than $z'_{i,o}$, and close to z'_{i,y_i}, which validates the above hypothesis.

REFERENCES

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. The fifth pascal recognizing textual entailment challenge. *TAC*, 7(8):1, 2009.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Zeren Chen, Ziqin Wang, Zhen Wang, Huayang Liu, Zhenfei Yin, Si Liu, Lu Sheng, Wanli Ouyang, Yu Qiao, and Jing Shao. Octavius: Mitigating task interference in mllms via moe. *arXiv preprint arXiv:2311.02684*, 2023.

Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Bill Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Third international workshop on paraphrasing (IWP2005)*, 2005.

Yunhao Gou, Zhili Liu, Kai Chen, Lanqing Hong, Hang Xu, Aoxue Li, Dit-Yan Yeung, James T Kwok, and Yu Zhang. Mixture of cluster-conditional lora experts for vision-language instruction tuning. *arXiv preprint arXiv:2312.12379*, 2023.

Yongxin Guo, Zhenglin Cheng, Xiaoying Tang, and Tao Lin. Dynamic mixture of experts: An auto-tuning approach for efficient transformer models. *arXiv preprint arXiv:2405.14297*, 2024.

Gagan Jain, Nidhi Hegde, Aditya Kusupati, Arsha Nagrani, Shyamal Buch, Prateek Jain, Anurag Arnab, and Sujoy Paul. Mixture of nested experts: Adaptive processing of visual tokens. *arXiv preprint arXiv:2407.19985*, 2024.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Bin Lin, Zhenyu Tang, Yang Ye, Jiaxi Cui, Bin Zhu, Peng Jin, Junwu Zhang, Munan Ning, and Li Yuan. Moe-llava: Mixture of experts for large vision-language models. *arXiv preprint arXiv:2401.15947*, 2024.

Chen Lin and Xing Long. Open-llava-next: An open-source implementation of llava-next series for facilitating the large multi-modal model community. https://github.com/xiaoachen98/Open-LLaVA-NeXT, 2024.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. *arXiv preprint arXiv:2310.03744*, 2023a.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? *arXiv preprint arXiv:2307.06281*, 2023b.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, 2022.

Microsoft. Phi-2: The surprising power of small language models. https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models, 2023.

Fangzhou Mu, Yingyu Liang, and Yin Li. Gradients as features for deep representation learning. *arXiv preprint arXiv:2004.05529*, 2020.

Zihan Qiu, Zeyu Huang, and Jie Fu. Emergent mixture-of-experts: Can dense pre-trained transformers benefit from emergent modular structures? *arXiv preprint arXiv:2310.10908*, 2023.

N Reimers. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34:8583–8595, 2021.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards vqa models that can read. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8317–8326, 2019.

Alex Wang. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Yisen Wang, Xingjun Ma, Zaiyi Chen, Yuan Luo, Jinfeng Yi, and James Bailey. Symmetric cross entropy for robust learning with noisy labels. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 322–330, 2019.

A Warstadt. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2019.

Liang Xu, Hai Hu, Xuanwei Zhang, Lu Li, Chenjie Cao, Yudong Li, Yechen Xu, Kai Sun, Dian Yu, Cong Yu, et al. Clue: A chinese language understanding evaluation benchmark. *arXiv preprint arXiv:2004.05986*, 2020.

Longrong Yang, Fanman Meng, Hongliang Li, Qingbo Wu, and Qishang Cheng. Learning with noisy class labels for instance segmentation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*, pp. 38–53. Springer, 2020.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer feed-forward layers are mixtures of experts. *arXiv preprint arXiv:2110.01786*, 2021.

Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018.

Yuhang Zhou, Zihua Zhao, Haolin Li, Siyuan Du, Jiangchao Yao, Ya Zhang, and Yanfeng Wang. Exploring training on heterogeneous data with mixture of low-rank adapters. *arXiv preprint arXiv:2406.09679*, 2024.