A. Scaling up SMoLA vs LoRA

A.1. Effect of scaling up rank for LoRA

Recall that in Section 4.3.3, we studied the effect of scaling up the expert counts for SMoLA$^{\mathtt{E}}_{\mathtt{O}}$-PaLI-3$_{\mathtt{FT}}$. Specifically, with $\mathtt{E}$= 4, 16, 48, and 144 experts per modality, SMoLA$^{\mathtt{E}}_{\mathtt{O}}$-PaLI-3$_{\mathtt{FT}}$ models achieve 1.1, 1.14, 1.2, 1.27 improvements over the PaLI-3$_{\mathtt{FT}}$ baseline, where the metrics are averaged across the tasks presented in Table 3 in the main paper on the validation splits[6] except for InfoVQA. Can we achieve similar performance improvement by using a higher rank of LoRA on top of PaLI-3$_{\mathtt{FT}}$?

We experiment with LoRA tuning on PaLI-3$_{\mathtt{FT}}$ with rank $\mathtt{r}$ of 32, 128, 384, 1536. Note that LoRA tuning with rank $\mathtt{r}$ has the same extra model parameters as SMoLA$^{\mathtt{r}/4}_{\mathtt{MM}}$-PaLI-3$_{\mathtt{FT}}$, and shares the same extra compute as SMoLA$^{\mathtt{r}/8}_{\mathtt{O}}$-PaLI-3$_{\mathtt{FT}}$. These LoRA tuning runs brought 0.58, 0.86, 0.79 and 0.59 gain over the PaLI-3$_{\mathtt{FT}}$ baseline. At lower ranks ($\mathtt{r} = 32, 128$), LoRA tuning led to smaller gains than SMoLA$^{\mathtt{r}/8}_{\mathtt{O}}$-PaLI-3$_{\mathtt{FT}}$, and further scaling up the rank resulted in worse performance in the case of LoRA tuning.

A.2. Effective rank for LoRA

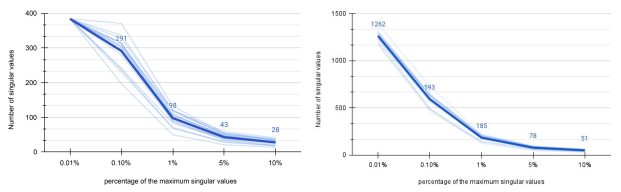

Figure 3. Number of singular values that are greater than different thresholds for LoRA weights. The left is for LoRA with rank 384 and the right is for LoRA with rank 1536.

To help understand why higher ranks do not improve LoRA tuning, we investigate how many ranks have the potential to significantly impact the outputs. Specifically, we consider LoRA with rank 384 and 1536 applied to PaLI-3$_{\mathtt{FT}}$. We compute the singular values for the combined weights of $\mathtt{W}^{\mathtt{out}}\mathtt{W}^{\mathtt{in}}$ for the output layer of each attention module in the encoder blocks, and present statistics over singular values in these two settings in Figure 3. In particular, we plot the number of singular values that are greater than 0.01%, 0.1%, 1%, 5%, 10% of the largest singular value – the light colored lines plot this for each encoder block and the bold line plots the counts averaged across all encoder blocks. For LoRA with rank 1536, there are only about 78 (5%) and 185 (12%) singular values greater than 5% and 1% of the largest singular value, the remaining dimensions would make little contribution to the output with normalized inputs.

[6]We use test split for AI2D as there are only 120 examples in val split.

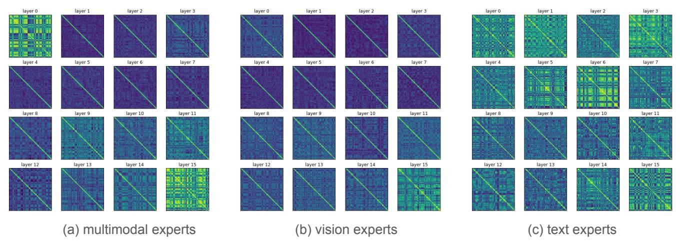

Figure 4. Heat maps for $\Phi\Phi^T$ for different modalities in SMoLA$^{48}_{\mathtt{O}}$-PaLI-3$_{\mathtt{FT}}$

B. Visualization on the routing matrix Φ

We present a heat map visualization of $\Phi\Phi^T$ in Figure 4. If $\Phi\Phi^T$ closely resembles an identity matrix, the routing matrix Φ is more likely to route the input to different experts. We have the following observations: (a) $\Phi\Phi^T$ for text experts are less similar to the identity matrix, indicating less need for specialization. One possible explanation is that input texts in our tasks tend to be easy to process — we trained on VQA tasks with short questions or image captioning tasks using identical prompt. (b) $\Phi\Phi^T$ for early layers in vision and multimodal experts are much closer to the identity matrix, indicating the need for different experts to handle more diverse shallow representations.

C. Pseudo Code

While we largely inherit the Soft MoE [43] design, we provide the Pseudo Code for running SMoLA blocks in Figure 5.

```
class SMoLA:
  def __init__(self, d_in, d_out, E, r=4)

    """ Args:
        E: number of experts.
        r: rank per expert.
        d_in: input dimension
        d_out: output dimension
    """
    self.routing_mat = params([d_in, E])
    self.routing_scale = params([1])
    self.smola_left = params([d_in, E, r])
    self.smola_right = params([E, r, d_out])

  def __call__(self, input_x, input_mask):
    """ Args:
        input_x: input array, shape: [B L d_in]
        input_mask: input array, shape: [B L]
    """
    routing_mat = normalize(self.routing_mat) * self.routing_scale
    logits = Einsum('BLd,dE->BLE')(normalize(input_x), routing_mat)
    dispatcher = softmax(logits, axis = 1)
    combiner = softmax(logits, axis = 2)

    routed_input = Einsum('BLd,BLE->BEd')(input_x, dispatcher)
    lowrank = Einsum('BEd,dEr->BEr')(routed_input, self.smola_left)
    routed_output = Einsum('BEr,Erd->BEd')(lowrank, self.smola_right)
    output = Einsum('BEd,BLE->BLd')(routed_output, combiner)

    return output
```

Figure 5. Pseudo Code for a SMoLA block