CS 577: NATURAL LANGUAGE PROCESSING

Abulhair Saparov

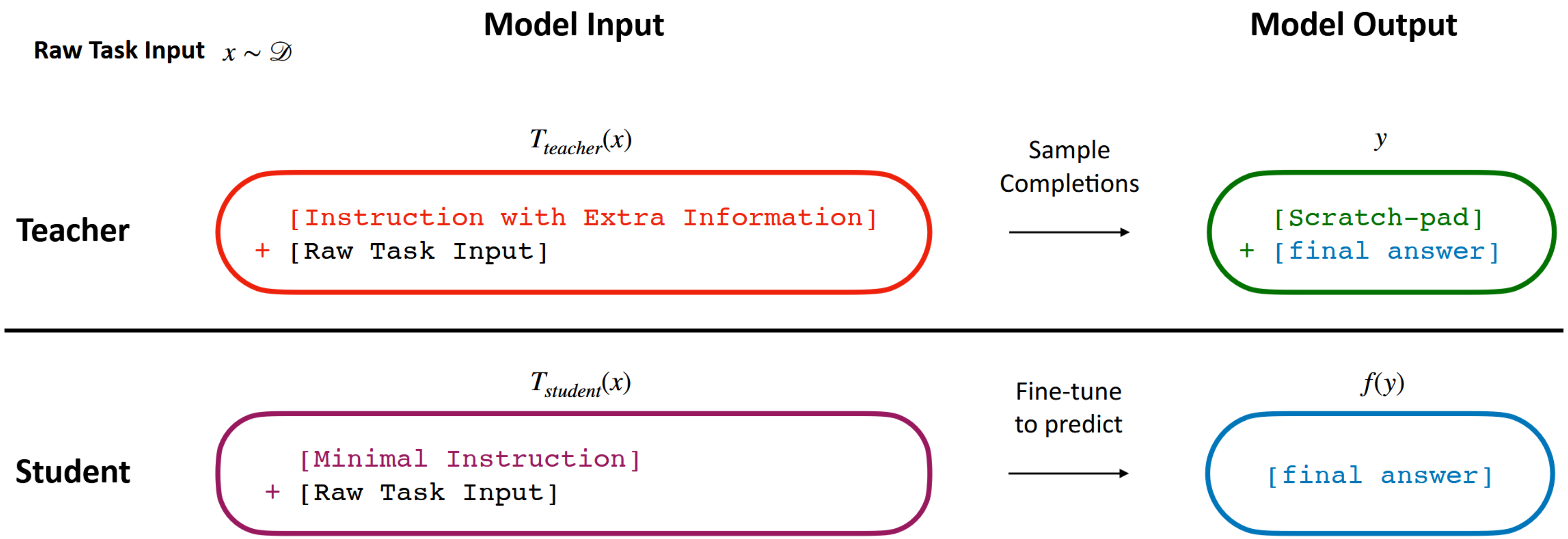
Lecture 18: Mixture of Experts

LAST TIME: DISTILLATION

- Last class, we discussed distillation: how to train a smaller model using the output of a teacher model.
- But "vanilla" distillation imposes an upper bound on the student model:
  - It can never learn to be more capable than the teacher model.
  - But we can augment the outputs of the teacher model before distillation.
  - Careful augmentation can lead to a more capable student model.
- For example, in Self-Instruct, the teacher and student models are the same.
  - But we used a small set of human-labeled data to synthesize a much larger instruction tuning dataset.
  - Then the distilled model has significantly improved instruction following capability.

CONTEXT DISTILLATION

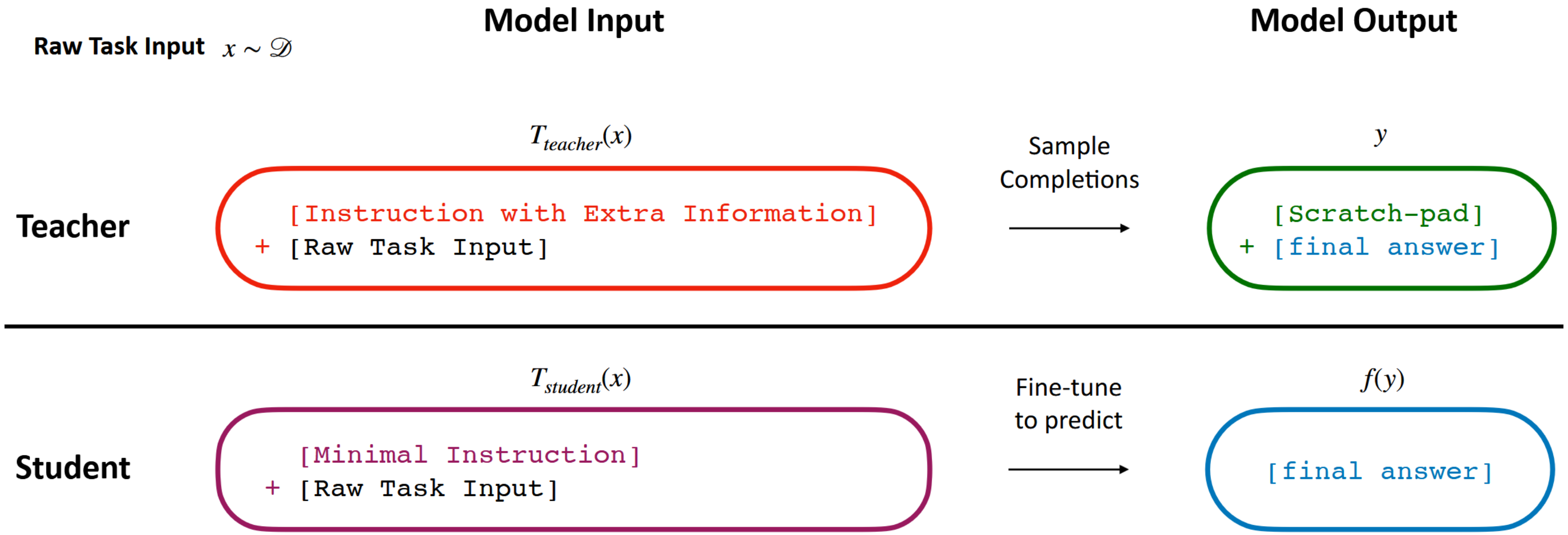
- For many tasks, language models benefit from long detailed instructions, in-context examples, and chain-of-thought.
- Can we teach a model “internalize” this extra information,
  - So that it can perform as well without it?



[Snell et al., 2022]

CONTEXT DISTILLATION

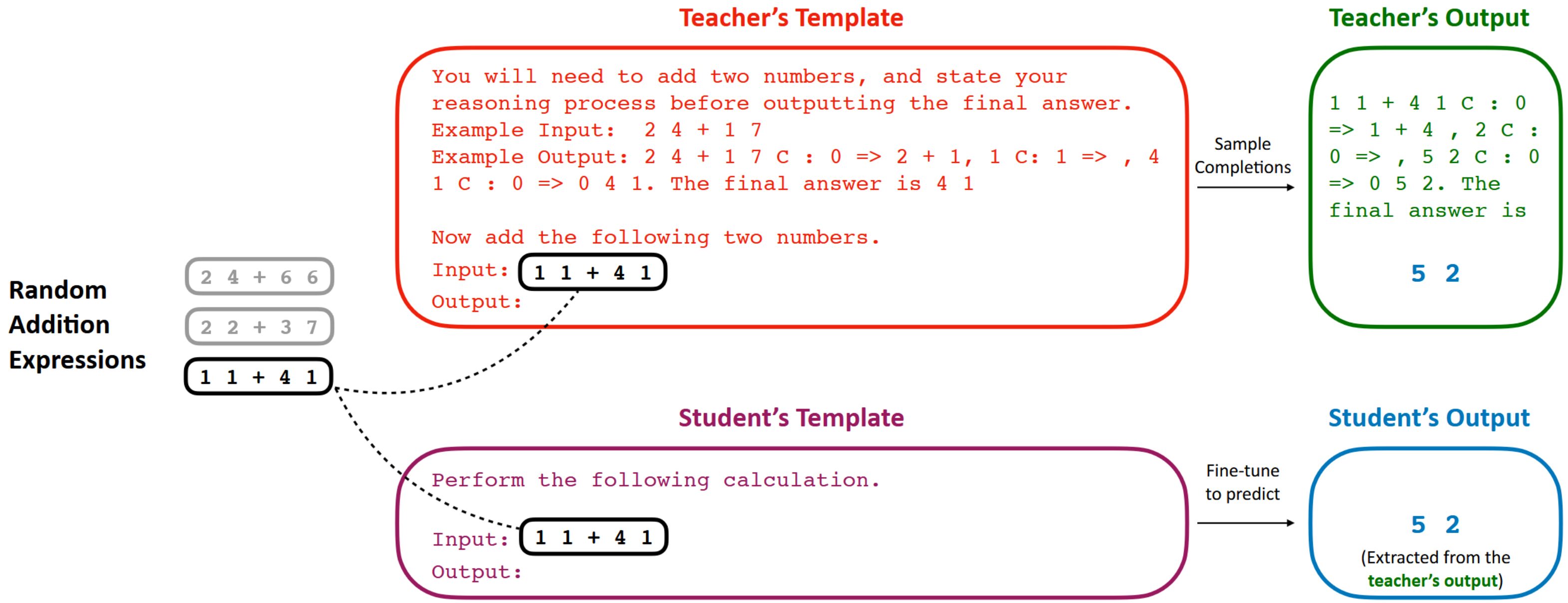
- Snell et al. (2022) call this approach context distillation.
- A teacher model is prompted with extra information (such as few-shot examples, CoT, additional instructions) to produce an output.
- The student model is trained on this output without the extra information.



[Snell et al., 2022]

CONTEXT DISTILLATION

- An example where the student is taught to internalize scratchpad (analogous to CoT):
- By removing the CoT, we are effectively augmenting the teacher model to be able to compute the answer without CoT.



[Snell et al., 2022]

CONTEXT DISTILLATION

- Snell et al. (2022) avoid using soft targets for distillation, since the vocabulary of LLMs is very large (50k-100k).
- Instead, they approximate soft target training by empirically sampling 100 tokens from each logit vector.
    - Then each training example for the student model consists of an "approximate" soft target

      (i.e., the histogram of tokens from the 100 token examples).

CONTEXT DISTILLATION

- In one of their experiments, they use `Incoder-6.7B` fine-tuned on text-to-SQL code generation as the teacher model.
  - The student model is the same as the teacher, except without in-context examples.
- This approach could be used to distill models with large context sizes into models with smaller sizes.

Model	4 Examples	8 Examples
Teacher	27.7	28.2
Pre-distill Student	0.3	0.3
Post-distill Student	**22.1**	**27.9**
Direct Gradient Descent	13.4	18.9

[Snell et al., 2022]

CONTEXT DISTILLATION

- In another experiment, they use `T5-small` (60M parameters) as the teacher model, which has been fine-tuned on the addition task with scratchpad.
  - The student model is the same model without scratchpad.

	Teach	Pre-Dist	Post-Dist
8 Digit Addition Accuracy %	93	0	**95**

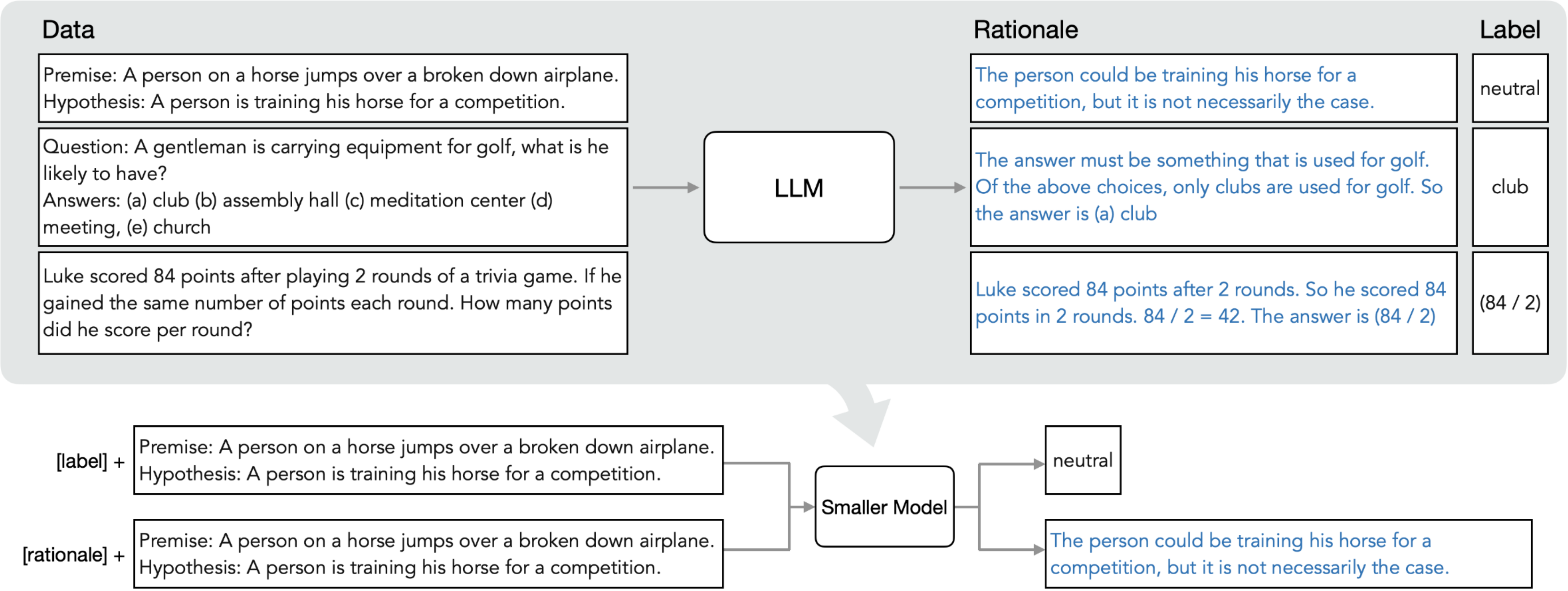
- They did not experiment with larger models, or test whether their approach could be used in a non-task-specific setting.
- This is an example application of distillation where the goal is not model compression.

[Snell et al., 2022]

DISTILL STEP-BY-STEP

- Hsieh et al. (2023) proposed a similar approach which they called "distilling step-by-step."
- In contrast with Snell et al. (2022), they used a much larger teacher model (`PaLM-540B`; Chaudury et al., 2022) to train a small student model (`T5-770M`; Raffel et al., 2020).
- They use the teacher model to generate for each input example:
  - A CoT rationale, as well as the output label.
- Then they train the student model in a *multi-task setting*:
  - If the input example has the word "`[label]`" prepended to it, the model is trained to predict the output label.
  - If the input example has the word "`[rationale]`" prepended to it, the model is trained to predict the rationale.

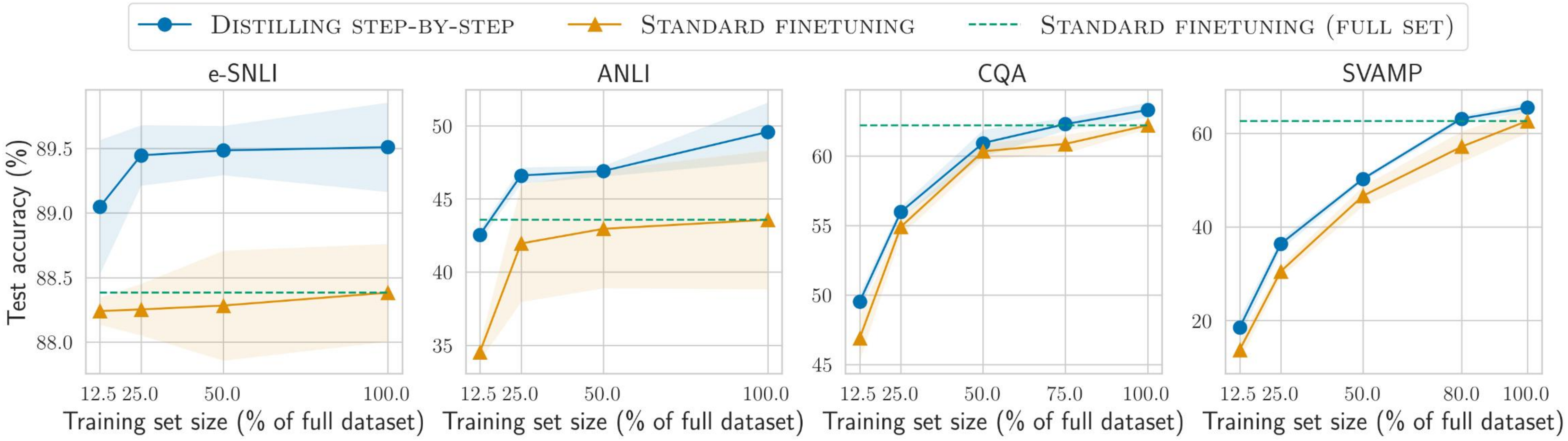
DISTILL STEP-BY-STEP



[Hsieh et al., 2023]

DISTILL STEP-BY-STEP

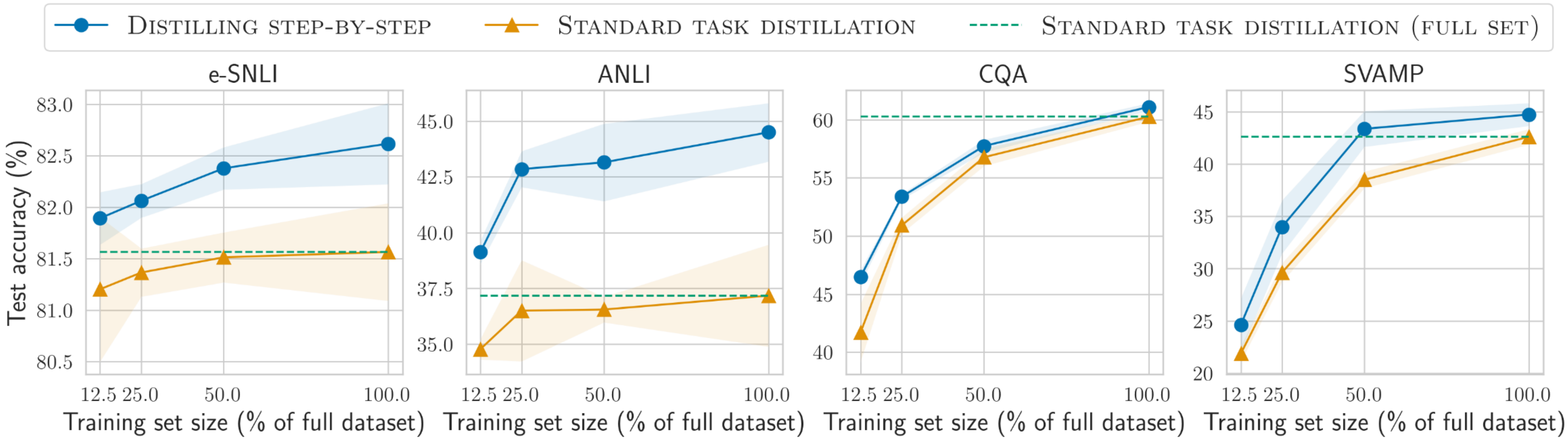
- They trained each student model using 12.5% of a task-specific dataset.
- They compared against full fine-tuning on the same amount of data.



[Hsieh et al., 2023]

DISTILL STEP-BY-STEP

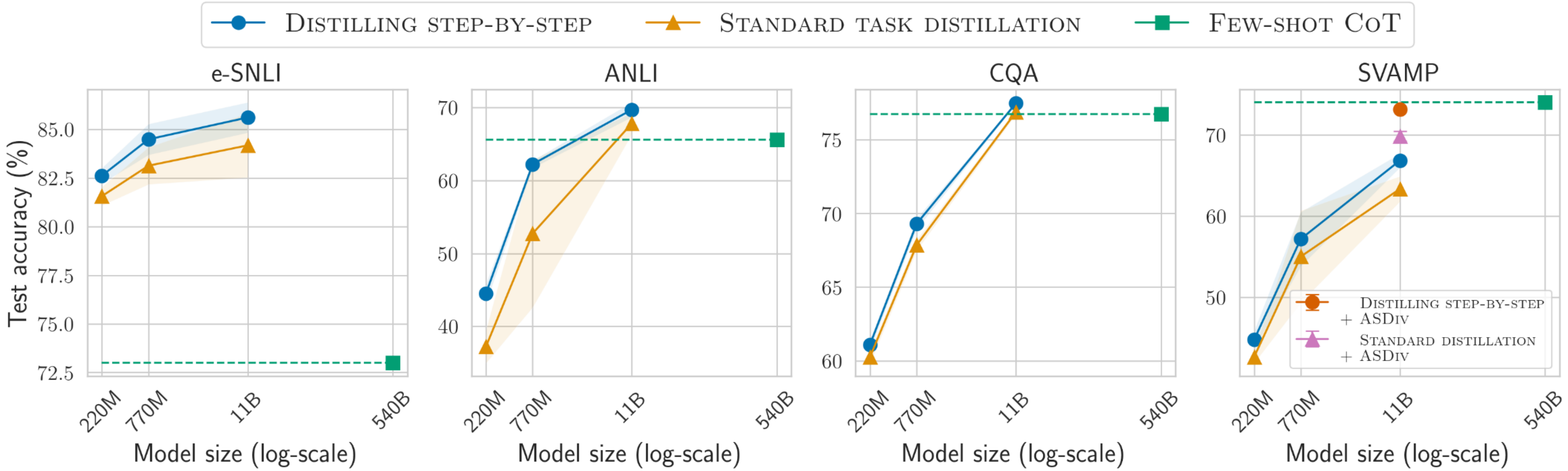
- They also compared against standard knowledge distillation (i.e., without rationales).



[Hsieh et al., 2023]

DISTILL STEP-BY-STEP

- They also compared with the teacher model using few-shot CoT prompting,
  - While varying the size of the student model.
  - (the teacher model was not fine-tuned on any of these datasets)



[Hsieh et al., 2023]

MODEL COMPRESSION SUMMARY

- We have concluded our discussion of model compression, including the three high-level approaches: quantization, pruning, and distillation.
- All three are able to produce smaller models that require less memory and computation time.
- Model compression do come at a cost to accuracy,
  - And further research is needed to better study their differences in behavior (as compared to larger models).
  - E.g., out-of-distribution performance, hallucinations, alignment, etc.

MIXTURE OF EXPERTS

MIXTURE OF EXPERTS

- Mixture of experts (MoE; Jacobs et al., 1991) is an example of an ensemble method,

  Where multiple models are combined,

  With the goal of the ensemble model outperforming any individual model.
- Suppose we have n probabilistic models (i.e., experts).
  - For example, suppose they are trained to perform spam detection.
  - Given an input email x, each model predicts the probability of the input being spam or not $y \in \{\mathtt{SPAM}, \mathtt{NOT\ SPAM}\}$.
  - So the i^{th} model, $f_i(x)$ estimates $p(y/x)$.
- How do we combine the predictions of these models to produce a more accurate prediction?

MIXTURE OF EXPERTS

- Suppose we introduce another random variable z, which depends on the input x, and "selects" one expert.
  - $z \in \{1, 2, \ldots, n\}$.
  - The selected expert's prediction is taken as the final prediction.
- We can write the probability of the full ensemble's output:

$$p(y|x) = \sum_{i=1}^{n} p(z = i|x)\, p(y|x, z = i),$$
$$= \sum_{i=1}^{n} p(z = i|x)\, f_i(x).$$

- The probability $p(z = i|x)$ can be written as a function of the input x:
  - $p(z = i|x) = g(x)_i$.
- This function $g(x)$ or $p(z|x)$ is called the gating model or gating function.

MIXTURE OF EXPERTS

- The gating model can consider information about the input x, and determine which expert is most likely to provide the best prediction.

$$p(y|x) = \sum_{i=1}^{n} g(x)_i f_i(x).$$

- In the simplest case, we can choose to set $g(x)_i$ to be independent of x.
  - Therefore, it is a constant, which we often write as θ_i.

$$p(y|x) = \sum_{i=1}^{n} \theta_i f_i(x).$$

- This approach has been used in a number of older applications.
  - Classifying phonemes from speech (Hampshire and Waibel, 1992),
  - Multi-speaker vowel recognition (Jacobs et al., 1991).

MIXTURE OF EXPERTS

- This approach is also called linear interpolation, since we are taking a weighted average of the n model probabilities.
- Notice that in this formulation, when we want to compute $p(y|x)$ (i.e., a forward pass), we have to compute all $f_i(x)$.
  - That is, we must perform the forward pass for *all* experts.
- *Unless*, the gating model outputs zero probability for some experts.
- Then we can avoid having to compute those experts with zero weight.

SPARSELY-GATED MIXTURE OF EXPERTS

- Shazeer et al. (2017) proposed using a “sparse” gating model:

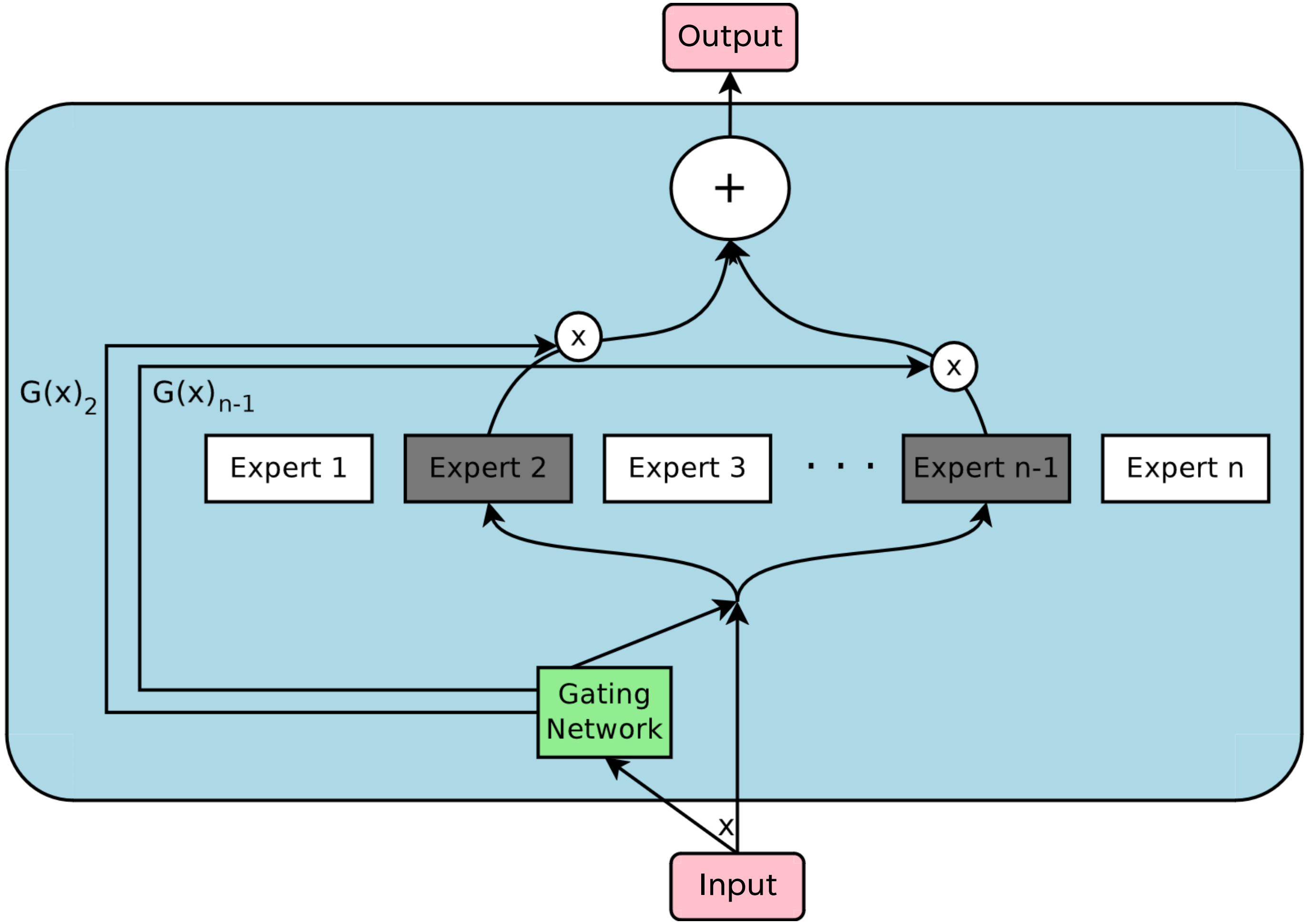
  $$g(x)_i = softmax(top\_k(h(x), k)),$$

  Where $top\_k(v, k)_i = v_i$ if v_i is in the top k elements of v,

  or $top\_k(v, k)_i = -\infty$ otherwise.

- Only the k “best” experts will have non-zero probability, and we can avoid forward pass for all other experts.
- The function $h(x)$ can be a simple linear transformation: $h(x) = x \cdot W_g^{\mathrm{T}} + b_g$,
  where W_g is a learnable weight matrix
  and b_g is a learnable bias vector.

SPARSELY-GATED MIXTURE OF EXPERTS



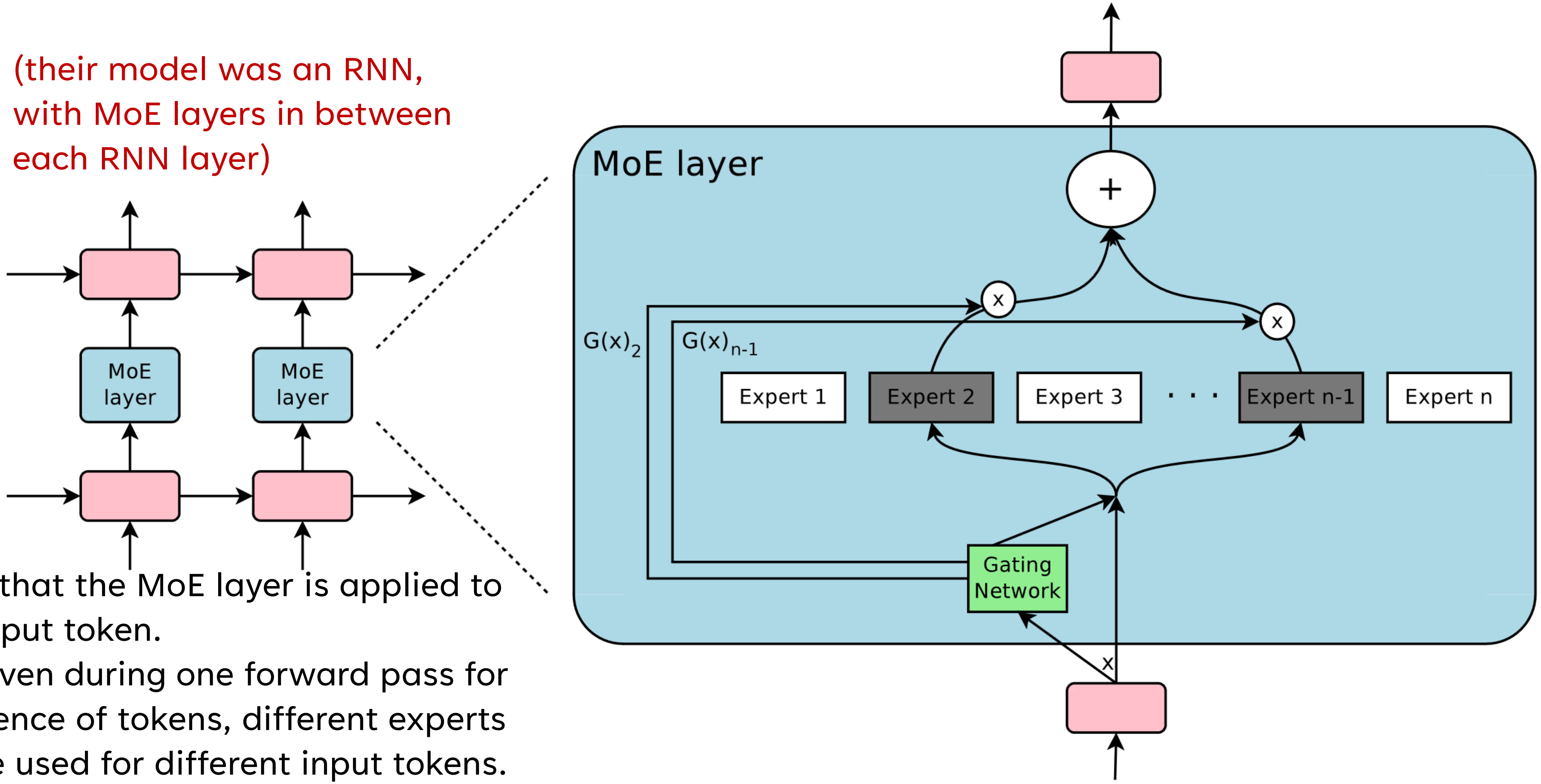
[Shazeer et al., 2017]

SPARSELY-GATED MIXTURE OF EXPERTS LAYER

- In the approach of Shazeer et al. (2017), each expert was not a full model.
- Instead, they applied the mixture-of-experts concept to the feedforward layers *within* the model.
- Each expert is an FF layer with smaller dimension d_{ff}.
- This idea has become much more popular lately because FF layers are the most computationally expensive components of large-scale transformer models.
  - In `PaLM-540B` (Chowdhery et al., 2023), for example, 90% of its parameters are in the FF layers.

SPARSELY-GATED MIXTURE OF EXPERTS LAYER

(their model was an RNN, with MoE layers in between each RNN layer)

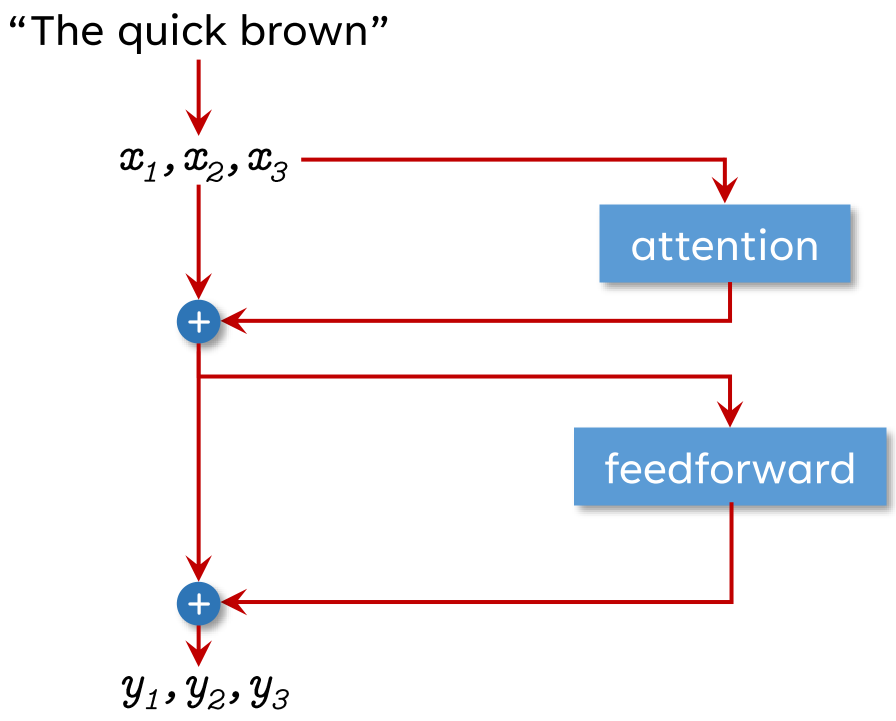


- Notice that the MoE layer is applied to each input token.
- Thus, even during one forward pass for a sequence of tokens, different experts may be used for different input tokens.

[Shazeer et al., 2017]

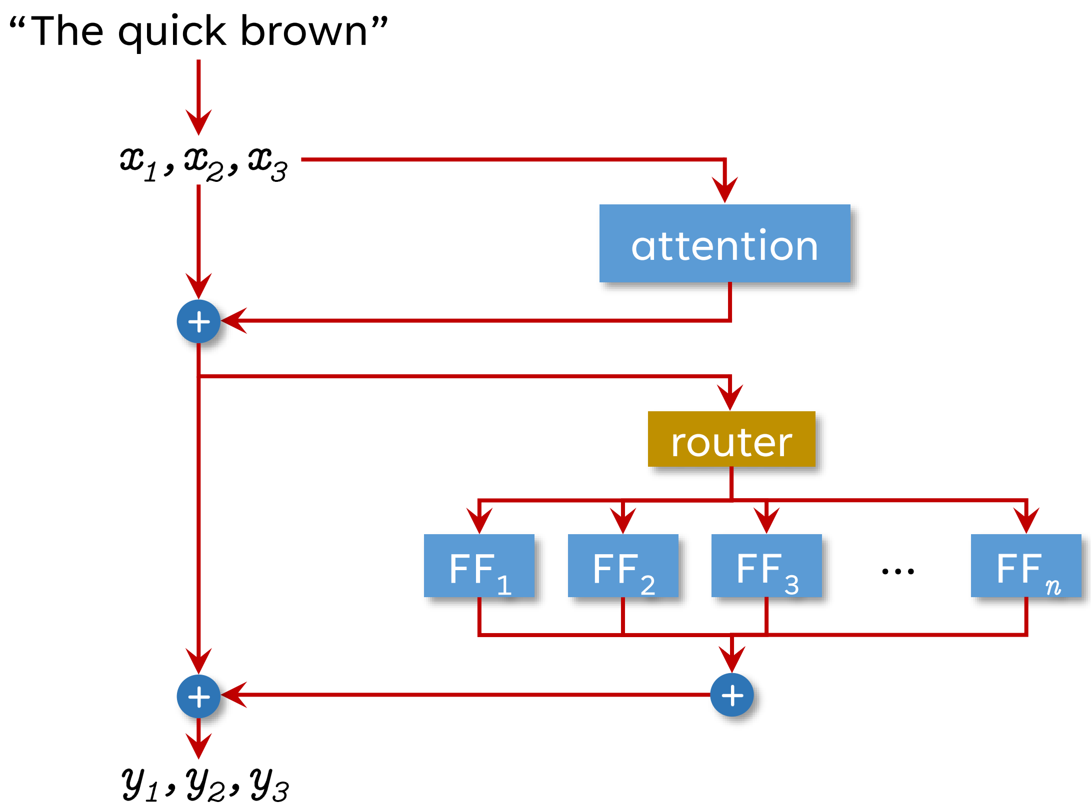
SPARSELY-GATED MIXTURE OF EXPERTS LAYER

- The same idea can be applied to the FF layers in transformers:



SPARSELY-GATED MIXTURE OF EXPERTS LAYER

- The same idea can be applied to the FF layers in transformers:
  - The gating model is also called the router.
  - Again note that different experts can be selected for different tokens.
  - E.g., FF_1 and FF_3 may be selected for "The",
  - Whereas FF_2 and FF_3 may be selected for "quick".
  - etc...



TRAINING LARGE MOE MODELS

- If we try training MoE models as described in the previous slides, two major problems arise:
  - Instability: Training is much more likely to diverge.
  - Load imbalance: The router learns to consistently select a small number of experts for almost all inputs.
- Incidentally, one possible cause of training instability is load imbalance.
  - If the model suddenly begins relying on a single expert for all of its predictions, then its loss may suddenly increase.
  - Always relying on one expert is equivalent to drastically reducing the number of parameters.
  - This is called routing collapse.

LOAD BALANCING

- How can we encourage the router to more evenly select experts?
- One method is to add a regularization term to the loss function that penalizes the router for non-uniform routing.

Ensures load balancing loss stays fixed with an increasing number of experts

$$L_{\texttt{load-balance}}(\mathcal{X}) = \overbrace{w_{\texttt{load-balance}}}^{\text{Constant scaling factor for the load balancing loss}} \cdot N \cdot \sum_{i=1}^{N} \boxed{f_i}\,\boxed{P_i}$$

$$\boxed{f_i = \frac{1}{T}\sum_{x\in\mathcal{B}} \mathbb{1}\{\operatorname{argmax} p(x) = i\}}$$

Fraction of tokens sent to expert i (not differentiable)

$$\boxed{P_i = \frac{1}{T}\sum_{x\in\mathcal{B}} p_i(x).}$$

Fraction of probability allocated to expert i (is differentiable)

- N is the number of experts,
- T is the batch size,
- f_i is estimated using the training batch.
- The loss is minimized when P_i is uniform. (note the similarity of the expression to negative entropy)

[Fedus and Zoph et al., 2022; Wolfe, nanoMoE: Mixture-of-Experts (MoE) LLMs from Scratch in PyTorch, 2025]

ROUTING PRECISION

- Another potential source of training instability is that the router is more prone to precision instability.
- More specifically, if the router logits (before softmax) are very large or very small, rounding errors could be more significant.
  - Recall that floating-point numbers are more accurate when closer to 0.
- We can add another loss term:

$$L_{\texttt{router-z}}(\mathcal{X}) = \overbrace{w_{\texttt{router-z}}}^{\text{Constant scaling factor for the router-z}} \cdot \frac{1}{C} \sum_{x \in \mathcal{X}} \underbrace{\left(\log \sum_{i=1}^{N} e^{\overbrace{\texttt{top-k}(x \cdot W_g)_i}^{\text{Router logits (before softmax)}}} \right)^2}_{\text{Captures magnitude of router logits for token x}}$$

Total number of tokens in the batch

ROUTING PRECISION

- Thus, our overall loss function looks like:

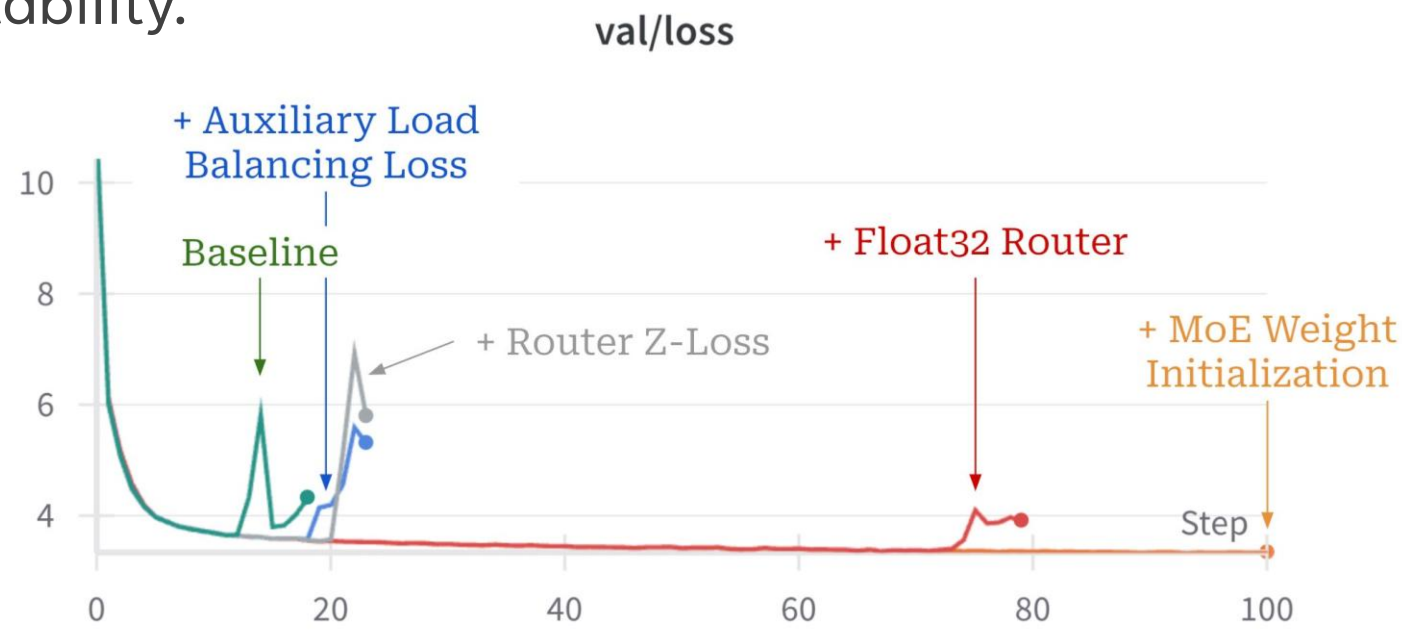
$$L_{\texttt{full}}(\mathcal{X}) = \underbrace{L_{\texttt{LM}}(\mathcal{X})}_{\text{Standard Language Modeling Loss}} + \underbrace{w_{\texttt{load-balance}} \cdot L_{\texttt{load-balance}}(\mathcal{X})}_{\text{Weighted load balancing loss}} + \underbrace{w_{\texttt{router-z}} \cdot L_{\texttt{router-z}}(\mathcal{X})}_{\text{Weighted router-z loss}}$$

- To further reduce the chance of training instability due to loss of precision, we can selectively use higher-precision number formats for the router.
  - E.g., use `bfloat16` for most model parameters, but use `float32` for router.
- Fedus and Zoph et al. (2022) also found that more careful initialization of the router parameters further helps with stability.
  - They use a uniform distribution with mean 0 and small variance.

[Wolfe, nanoMoE: Mixture-of-Experts (MoE) LLMs from Scratch in PyTorch, 2025]

TRAINING LARGE MOE MODELS

- Wolfe (2025) experimented with training a small model with and without these techniques.
- Higher-precision and better-initialized router parameters significantly improved stability.



[Wolfe, nanoMoE: Mixture-of-Experts (MoE) LLMs from Scratch in PyTorch, 2025]

TRAINING LARGE MOE MODELS

- Zoph and Bello et al. (2022) found that `float32` router precision without the router z-loss was insufficient for training stability.

Method	Fraction Stable	Quality (↑)
Baseline	4/6	-1.755 ±0.02
Update clipping (clip = 0.1)	3/3	-4.206 ±0.17
Router Z-Loss	3/3	**-1.741** ±0.02

- The baseline here uses `float32` precision without router z-loss.
- They found that the router z-loss led to no statistically significant difference in model performance/accuracy.

[Zoph and Bello et al., 2022]

NUMBER OF ACTIVE EXPERTS?

- How should we select k, i.e., the number of active experts?
  - In transformer models, k is set as small as possible.
  - Though smaller values of k can lead to greater training instability.
- `OlMoE` (Muennighoff et al., 2024) uses $k = 8, n = 64$.
- `Mixtral` (Jiang et al., 2024) uses $k = 2, n = 8$.
  - The model has 47B parameters total,
  - But only 13B are active for each token.
- Fedus and Zoph et al. (2022) were able to train MoE models with $k = 1$ by utilizing many of the techniques we discussed earlier for improving training stability.
- `DeepSeek v3` and `r1` use $k = 8, n = 256$.

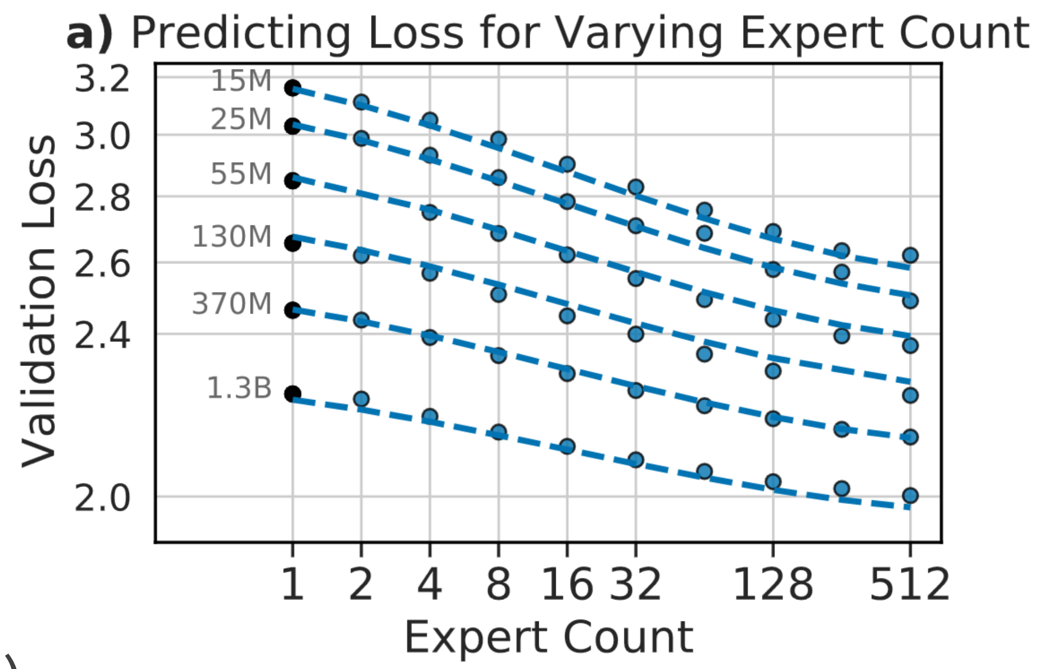
MIXTURE OF EXPERTS REVIEW

- The idea that only certain parts of the network are active for each example is called conditional computation.
- MoE helps reduce the computation time of the model, but does not help with memory usage (additional router parameters are negligible).
- This approach could be used to further increase the model size without increasing the computation per forward pass.
  - For example, we can simply increase the number of experts,
  - But keep constant the size of each expert.
- Question: How does MoE affect scaling laws?
  - If we fix the number of parameters but increase the number of experts, will the model's performance suffer?

MOE SCALING LAWS

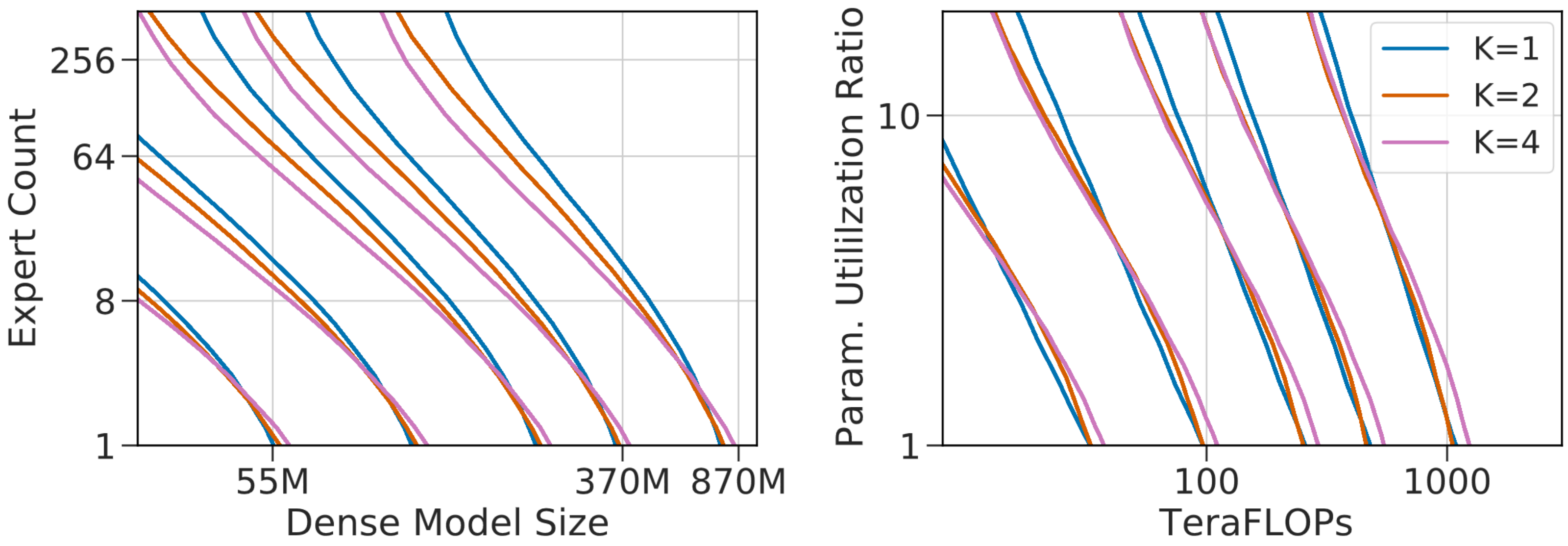
- Clark, de las Casas, Guy, and Mensch et al. (2022) trained transformer language models with various sizes and number of experts.
- Interestingly, they find that increasing experts improves model accuracy.
- Though small models benefit more from increasing the number of experts.
- Larger models do not see as large of a benefit.

(compare the slopes of each of the blue curves)



MOE SCALING LAWS

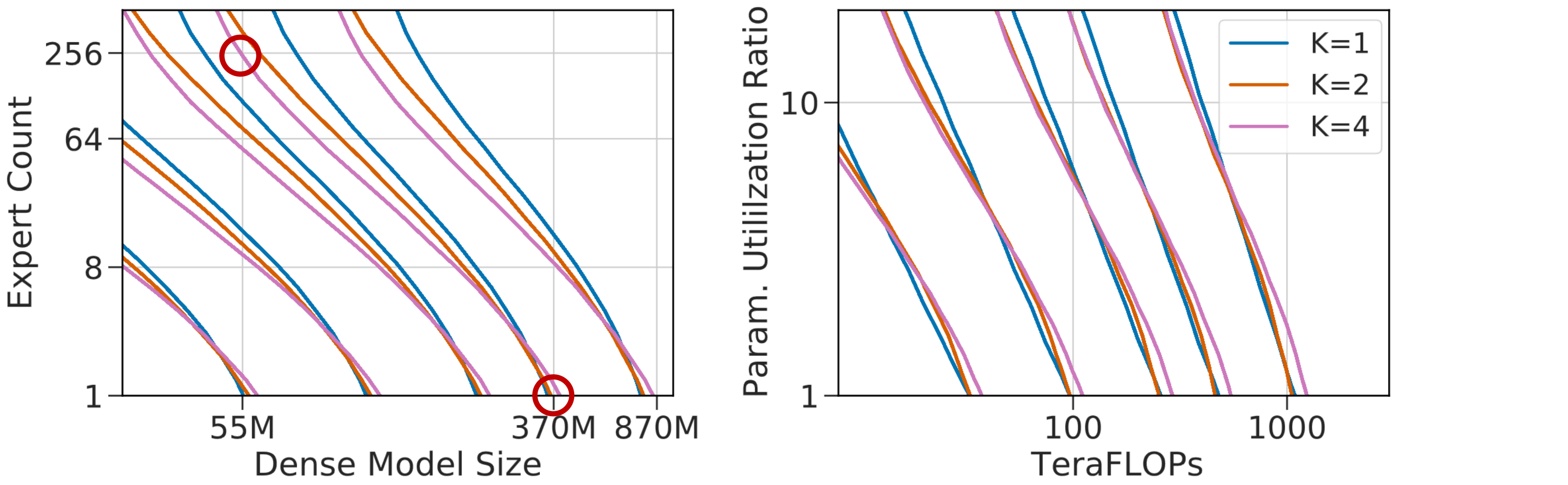
- They also experiment with varying the number of active experts k.
- They create iso-loss curves (i.e., level curves) where each curve represents a set of points that achieves the same loss.



[Clark, de las Casas, Guy, and Mensch et al., 2022]

MOE SCALING LAWS

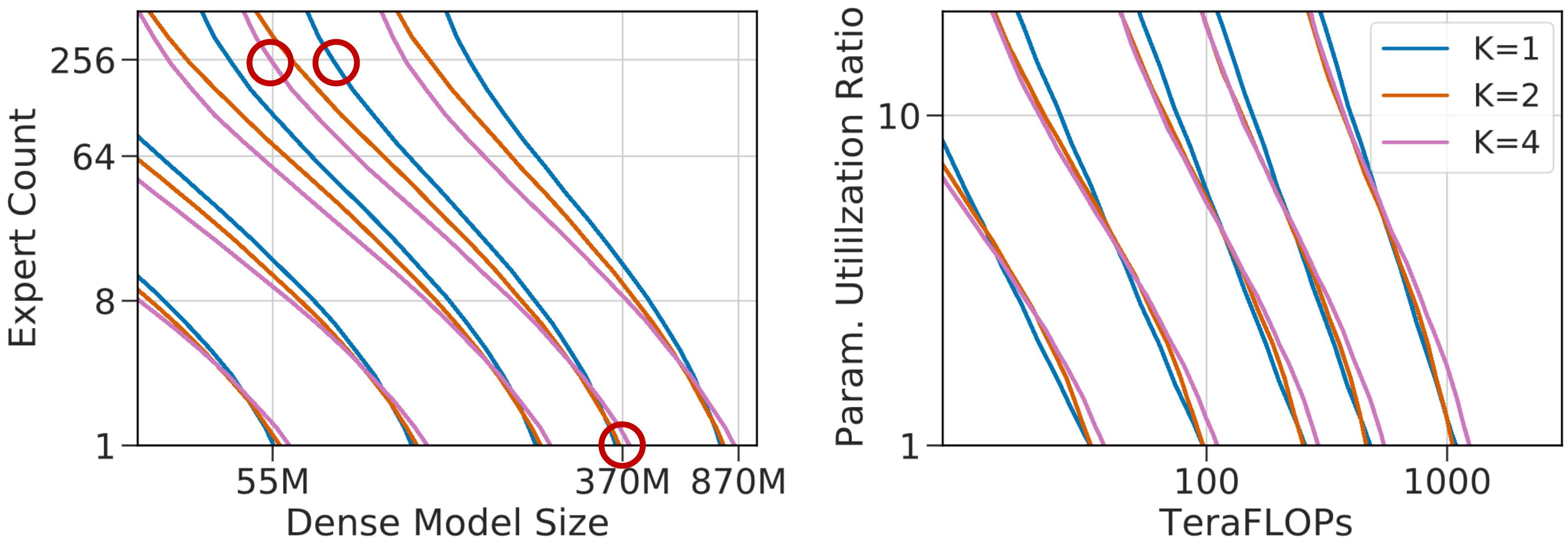
- Loss decreases (model performance increases) as you go further to the right and up in each plot.
- In the left plot, a 370M-parameter 1-expert model (i.e., a “dense” model) has the same loss as a 55M-parameter 256-expert model and $k = 4$.



[Clark, de las Casas, Guy, and Mensch et al., 2022]

MOE SCALING LAWS

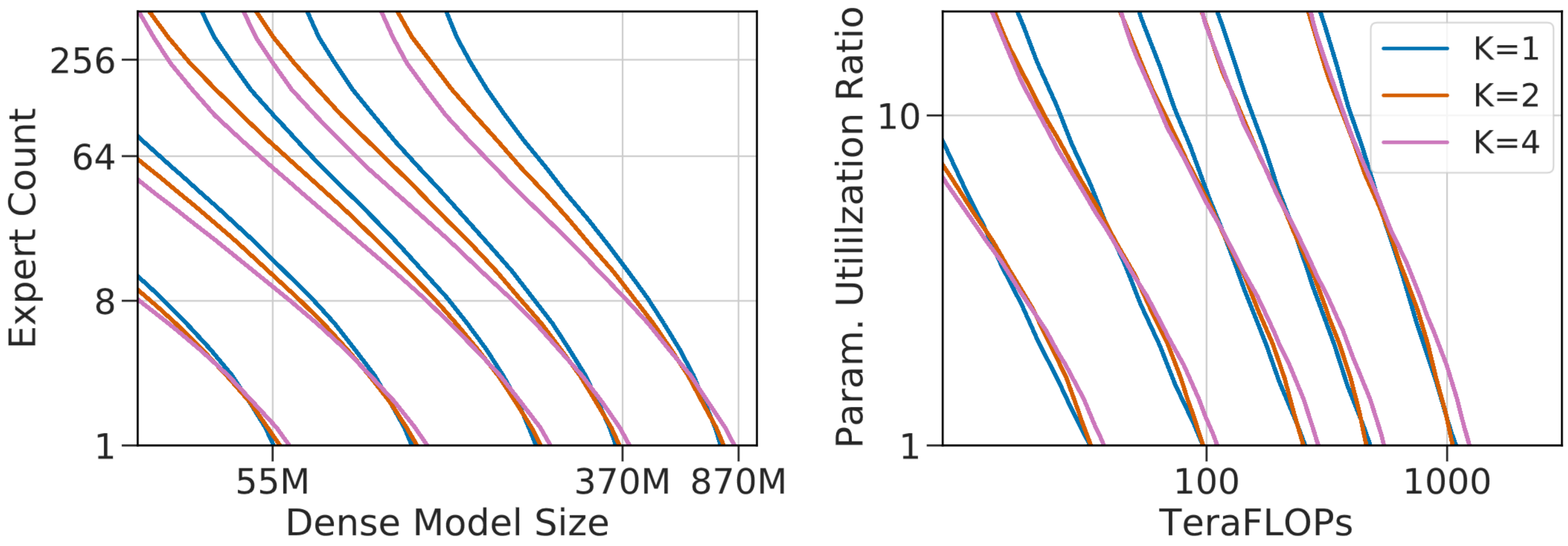
- But note that the blue curve is slightly to the right of the purple, indicating that $k = 1$ models have higher loss than $k = 4$ models of the same size.
- However, the authors note that $k = 4$ models are more computationally expensive than $k = 1$ models.



[Clark, de las Casas, Guy, and Mensch et al., 2022]

MOE SCALING LAWS

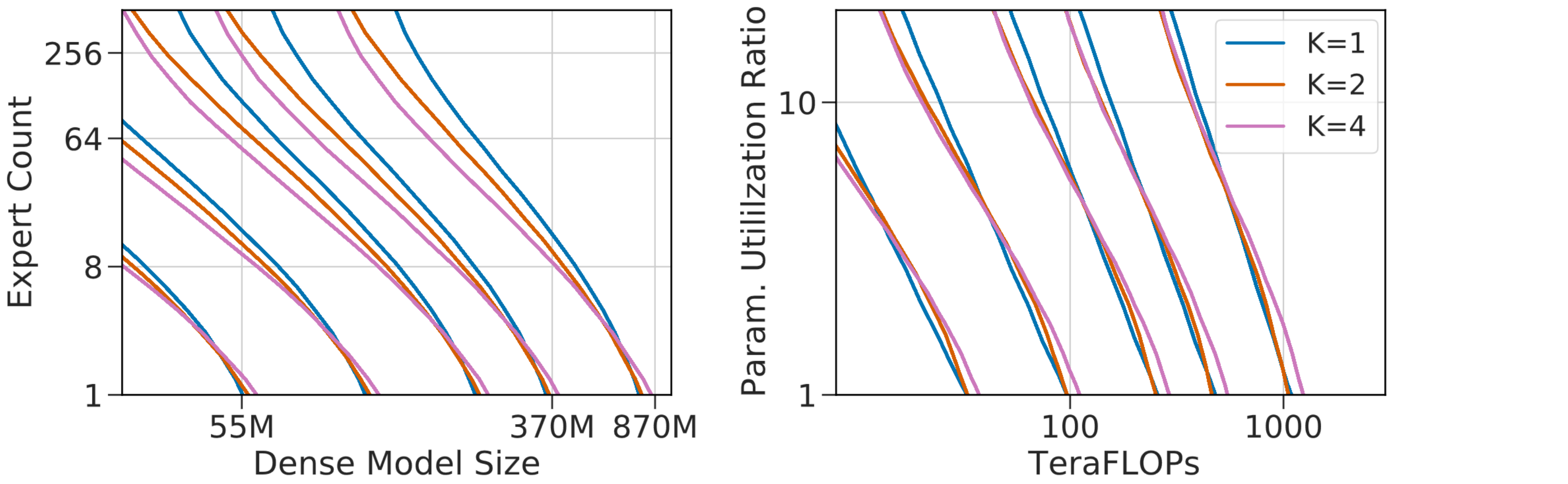
- The right plot accounts for the increasing cost of larger k.
- The “parameter utilization ratio” is the number of parameters divided by the TFLOPs required for each forward pass.
  - So this ratio increases linearly with the number of experts.



[Clark, de las Casas, Guy, and Mensch et al., 2022]

MOE SCALING LAWS

- On the right, we see that the curves almost overlap for different values of k,
- Suggesting that the number of active experts k does not have a significant effect on the scaling behavior of MoE transformer models.
- They did not experiment with more experts than 512.



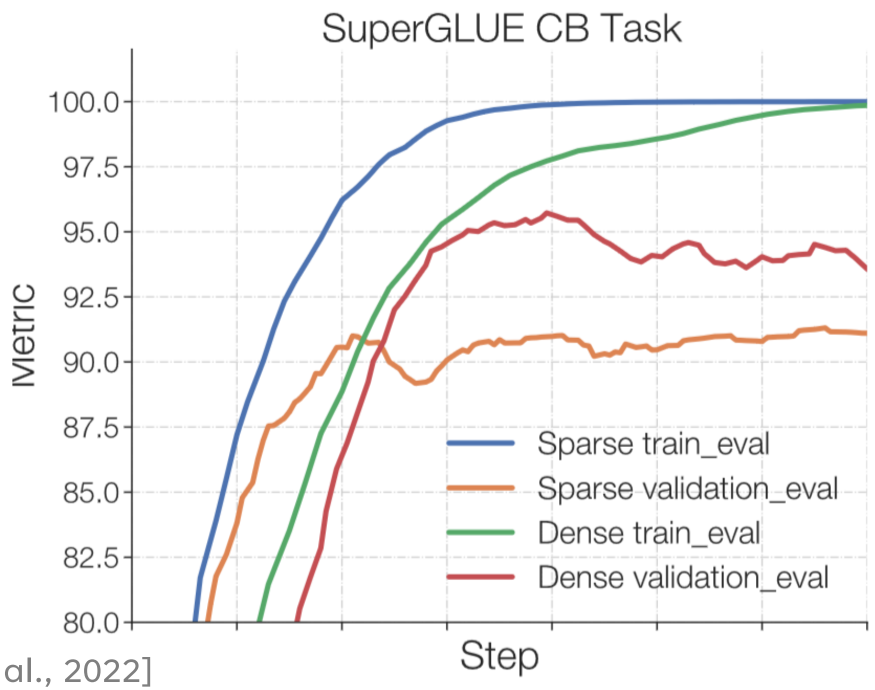
[Clark, de las Casas, Guy, and Mensch et al., 2022]

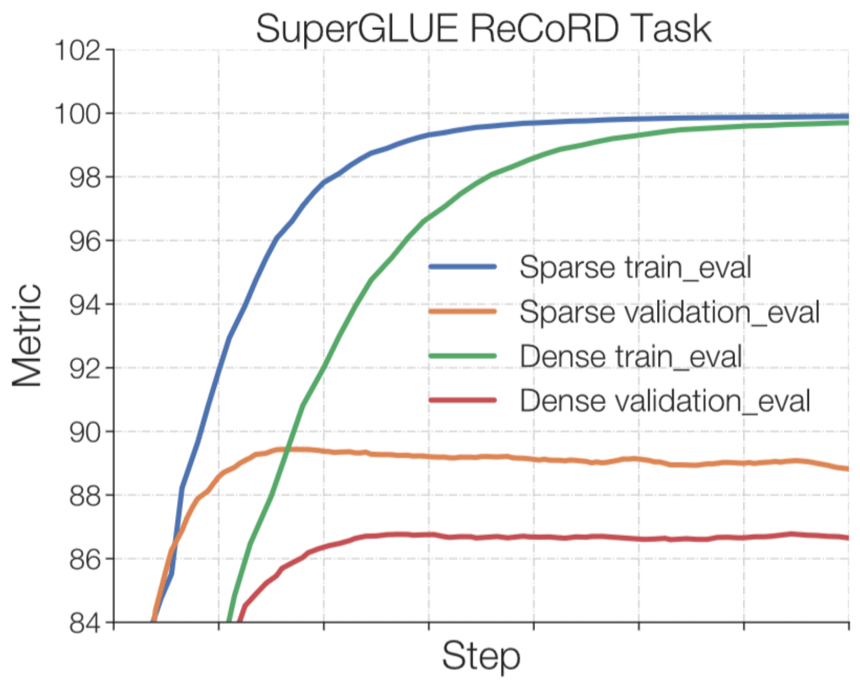
MOE SCALING LAWS

- A later study (Krajewski and Ludziejewski et al., 2024), showed that the compute-optimal amount of training for MoE models is larger than that of dense transformers.
  - I.e., MoE models need to be trained for longer/on more data.
- More experts seems to be better in general.
  - Why?
  - The number of active parameters decreases with more experts.
  - Perhaps each expert is trained to be more specialized,
    - And each FF parameter is being used more efficiently to fit the data.

FINE-TUNING MOE MODELS

- Interestingly, MoE models have been found to overfit more easily (Zoph and Bello et al., 2022).
  - This is apparent in fine-tuning.

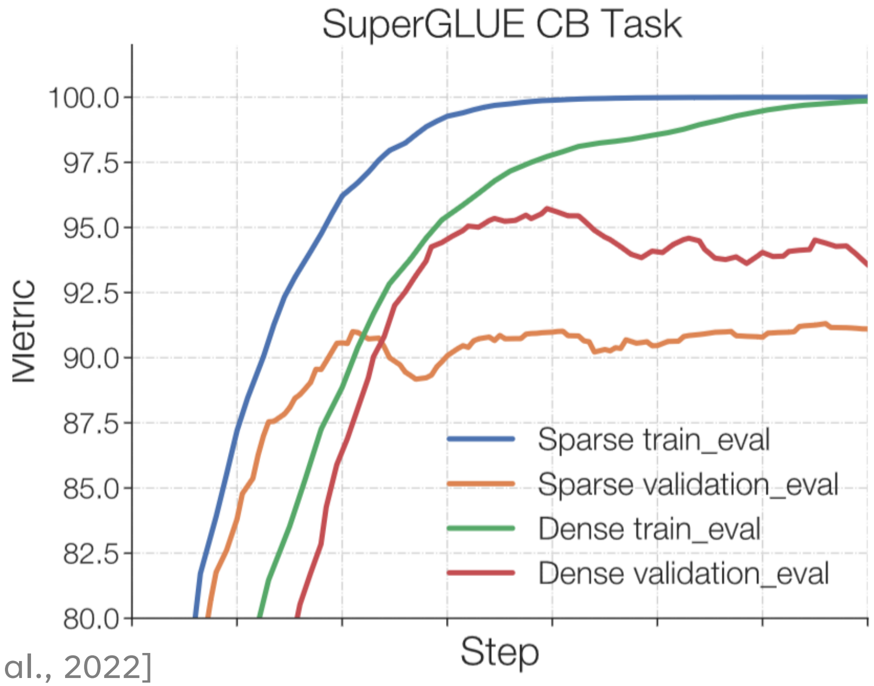


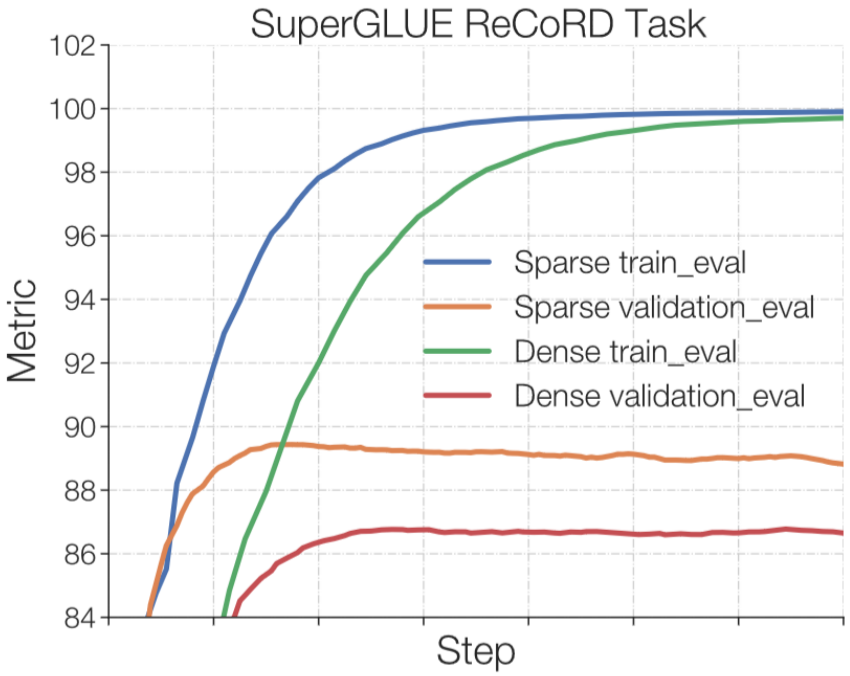


[Zoph and Bello et al., 2022]

FINE-TUNING MOE MODELS

- The SuperGLUE Commitment Bank (CB) task is an entailment task.
- The MoE model learns faster than the dense model, but overfits.

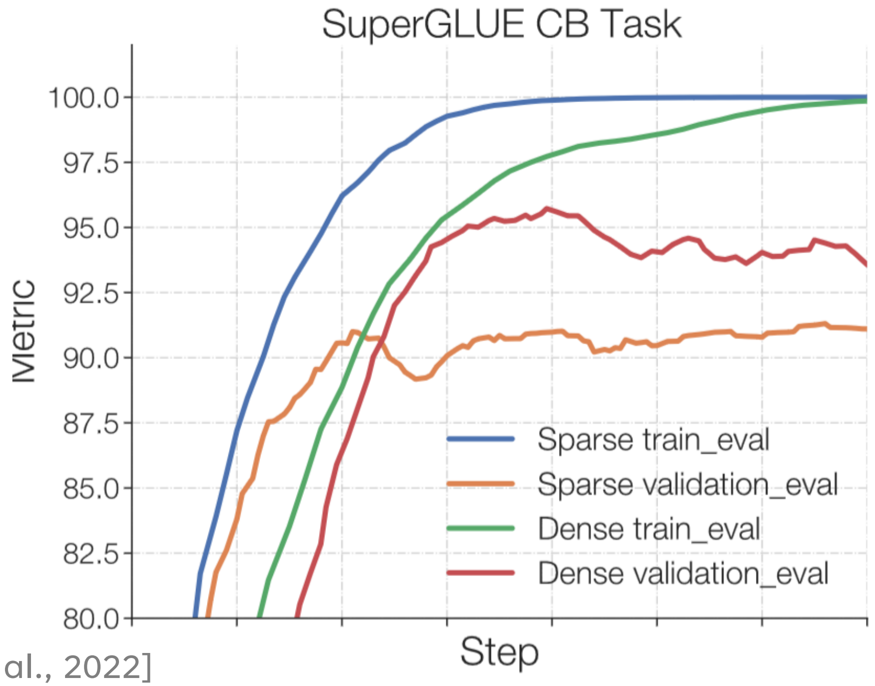


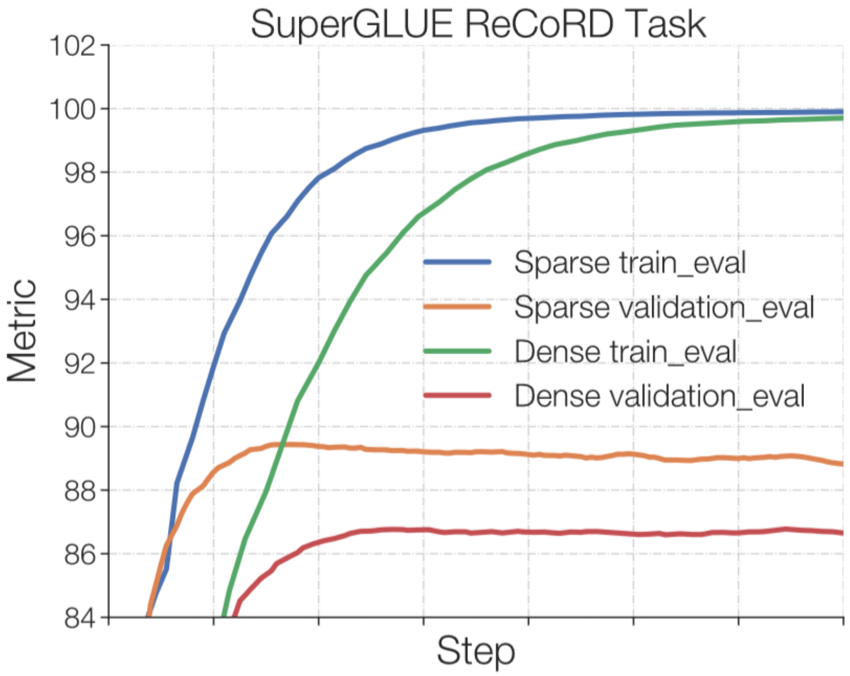


[Zoph and Bello et al., 2022]

FINE-TUNING MOE MODELS

- SuperGLUE ReCoRD is the Reading Comprehension with Commonsense Reasoning task.
- The MoE model learns faster than the dense model and generalizes better.



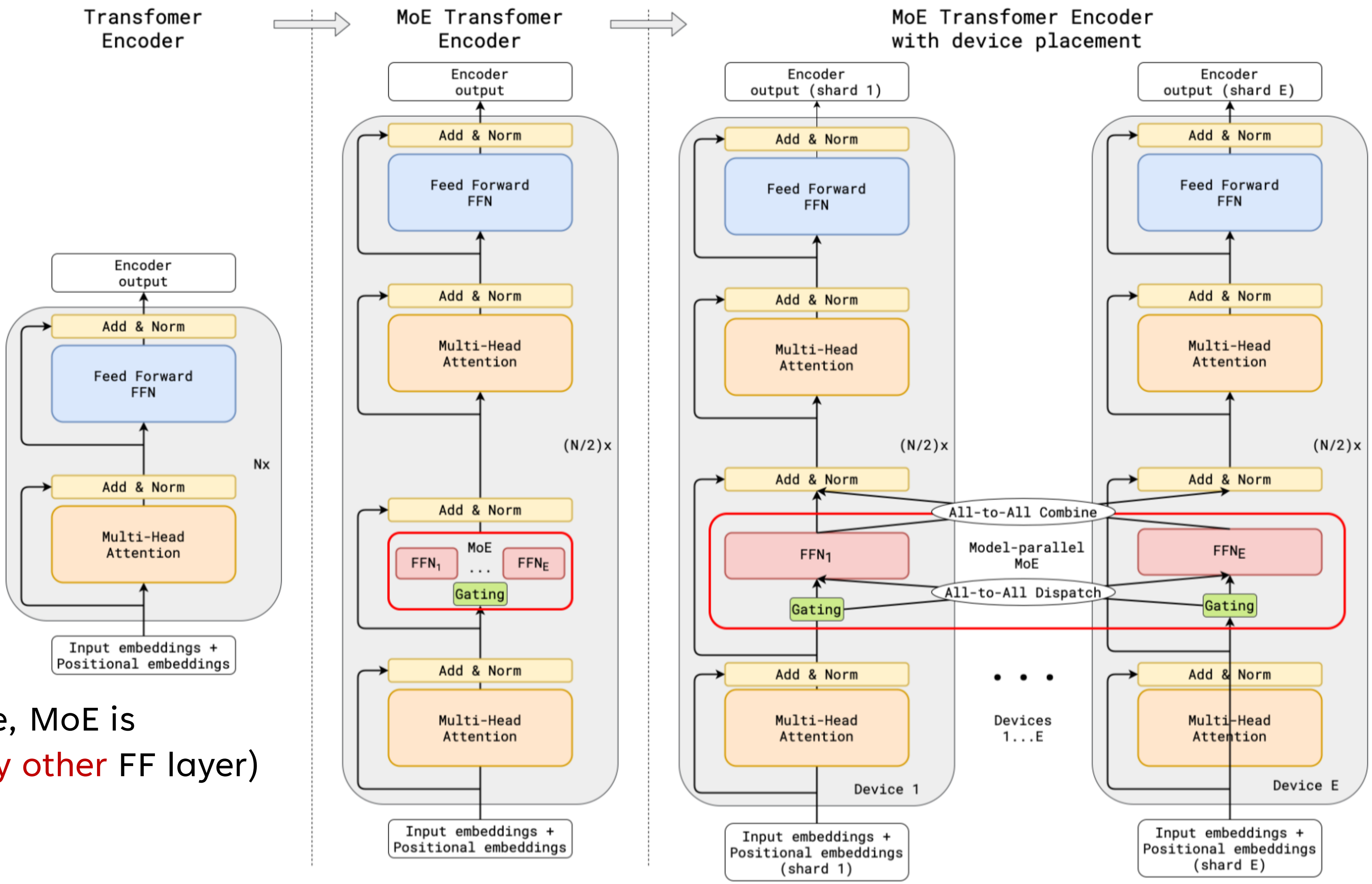


[Zoph and Bello et al., 2022]

PARALLELIZING MIXTURE OF EXPERTS

- One big advantage of MoE is that the experts provide a natural way to parallelize the model.
- Each device (i.e., GPU) can be assigned to one expert.
- Whenever we perform a forward pass with the FF layer,
  - We run the router model and determine which tokens should be sent to which experts.
  - We communicate the routing information to all devices so that each expert can perform the forward pass on their respective assigned tokens.
  - Finally, we perform an all-reduce operation to share the result across devices.
- The other parts of the model are not as memory-intensive, and so they can be replicated on each device.

PARALLELIZING MIXTURE OF EXPERTS



(in this example, MoE is applied to every other FF layer)

[Lepikhin et al., 2020]

QUESTIONS?