Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning

Authors: Haokun Liu*, Derek Tam*, Mohammed Muqeeth*, Jay Mohta, Tenghao Huang, Mohit Bansal, Colin Raffel
ArXiv preprint, May 2022

Setting

- Few-shot classification tasks
- Minimal in-task data -> can’t just train/finetune model from scratch
- Other approaches:
  - In-context Learning with massive LLMs
  - Prompt Tuning and other parameter-efficient methods
- Drawbacks:
  - Massive LLMs are costly to run inference on
  - Context length limits # of few-shot examples that fit in context, and more shots -> higher cost

Goals

- Provide a method that outcompetes prompting a massive LLM
- What features do we need to be competitive?
  - ICL's biggest advantage: single model for all tasks!
  - ICL supports mixed batches: a single batched API call could have each request doing a different task
- What disadvantages of ICL can be removed?
  - Don't put labeled examples in context, to save on cost!
  - Or: do just as well with a smaller model -> faster inference and less expensive
  - ICL is very sensitive! More consistent performance would be key

More Failures of Existing Approaches

- Existing approaches on smaller models:
  - Naive few-shot finetuning
  - Instruction Tuning -> task transfer
  - Parameter-efficient finetuning (PEFT)
- Where do these fail? Succeed?
  - Finetuning a whole model is costly + requires storage, and it may not be reusable
  - PEFT methods do as well as whole model FT for less cost!
    - But inserting new adapter layers require extra compute per input, and this prevents multi-task batches
  - Prompt Tuning works well and allows for multi-task batches!

Contributions

- Contributions + Key Terms:
  - A new parameter-efficient finetuning (“PEFT”) method, “(IA)^3”
  - A “recipe” built on it and T0, called “T-Few”
  - A comparison between PEFT and In-Context Learning (“ICL”)
  - Conclusion:
    - “Few-shot PEFT is better and cheaper than ICL”!

(IA)^3

- (IA)^3: “Infused Adapter by Inhibiting and Amplifying Inner Activations”
  - 3 learned vectors *l_k*, *l_v*, *l_ff*
- Modifies attention:

$$\text{softmax}\left(\frac{Q(l_k \odot K^T)}{\sqrt{d_k}}\right)(l_v \odot V)$$

- And feedforward:

$$(l_{ff} \odot \gamma(W_1 x))W_2,$$

- Pointwise multiplication: cheap component-wise rescaling
  - Very few parameters added (and can be **absorbed**) -> easy to store separately
  - Each input in a batch can take a different task-specific set of learned vectors!

Other Tricks

- Auxiliary losses

For evaluation, we use rank classification (described in section 3.1) which depends on both the probability that the model assigns to the correct choice as well as the probabilities assigned by the model to the incorrect choices. To account for this during training, we consider adding an unlikelihood loss [16, 17]:

$$L_{\text{UL}} = -\frac{\sum_{n=1}^{N}\sum_{t=1}^{T^{(n)}} \log(1 - p(\hat{y}_i^{(n)}|\mathbf{x}, \hat{y}_{<t}^{(n)}))}{\sum_{n=1}^{N} T^{(n)}} \quad (1)$$

which discourages the model from predicting tokens from incorrect target sequences, where $\hat{\mathbf{y}}^{(n)} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{T^{(n)}})$ is the n-th of N incorrect target sequences. We hypothesize that adding L_{UL} will improve results on rank classification because the model will be trained to assign lower probabilities to incorrect choices, thereby improving the chance that the correct choice is ranked highest.

length-normalized log probability of a given output sequence $\beta(\mathbf{x}, \mathbf{y}) = \frac{1}{T}\sum_{t=1}^{T} \log p(y_t|\mathbf{x}, y_{<t})$. Then, we maximize the length-normalized log probability of the correct answer choice by minimizing the softmax cross-entropy loss:

$$L_{\text{LN}} = -\log \frac{\exp(\beta(\mathbf{x}, \mathbf{y}))}{\exp(\beta(\mathbf{x}, \mathbf{y})) + \sum_{n=1}^{N} \exp(\beta(\mathbf{x}, \hat{\mathbf{y}}^{(n)}))} \quad (2)$$

When training a model with L_{LM}, L_{UL}, and L_{LN}, we simply sum them. This avoids introducing any hyperparameters that would be problematic to tune in the few-shot setting (where realistically-sized validation sets are tiny by necessity [31, 32]).

Full Method

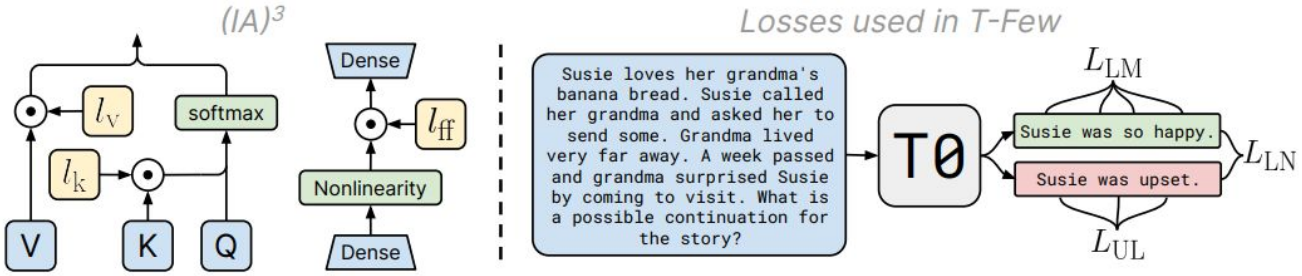


Figure 1: Diagram of (IA)³ and the loss terms used in the T-Few recipe. *Left:* (IA)³ introduces the learned vectors l_k, l_v, and l_{ff} which respectively rescale (via element-wise multiplication, visualized as $\odot$) the keys and values in attention mechanisms and the inner activations in position-wise feed-forward networks. *Right:* In addition to a standard cross-entropy loss L_{LM}, we introduce an unlikelihood loss L_{UL} that lowers the probability of incorrect outputs and a length-normalized loss L_{LN} that applies a standard softmax cross-entropy loss to length-normalized log-probabilities of all output choices.

(IA)^3 vs. Other PEFT Methods

- Outperforms across the board
  - Average over ~11 datasets
- Better than full finetuning (!)
- What makes some methods outperform others?

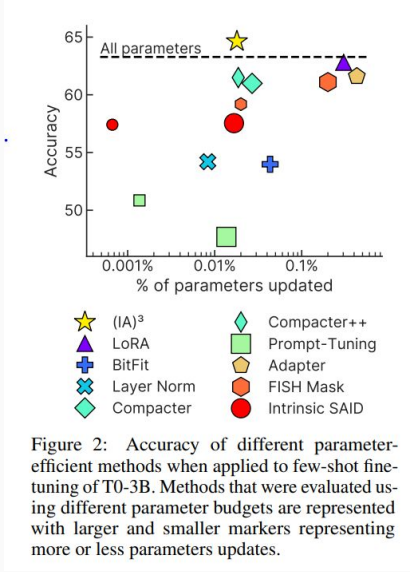


Figure 2: Accuracy of different parameter-efficient methods when applied to few-shot fine-tuning of T0-3B. Methods that were evaluated using different parameter budgets are represented with larger and smaller markers representing more or less parameters updates.

Head-to-head with ICL

- Outperforms GPT-3 175B (`davinci`?)
- Cost of training IA^3 is outweighed by the cost of running GPT-3 on only dozens of examples!

Method	Inference FLOPs	Training FLOPs	Disk space	Accuracy
`T-Few`	1.1e12	2.7e16	4.2 MB	72.4%
T0 [1]	1.1e12	0	0 B	66.9%
T5+LM [14]	4.5e13	0	16 kB	49.6%
GPT-3 6.7B [4]	5.4e13	0	16 kB	57.2%
GPT-3 13B [4]	1.0e14	0	16 kB	60.3%
GPT-3 175B [4]	1.4e15	0	16 kB	66.6%

Table 1: Accuracy on held-out T0 tasks and computational costs for different few-shot learning methods and models. `T-Few` attains the highest accuracy with 1,000× lower computational cost than ICL with GPT-3 175B. Fine-tuning with `T-Few` costs about as much as performing ICL on 20 examples with GPT-3 175B.

RAFT Benchmark

- 11 “real-world” tasks
- No val set, hidden test set, 50 train examples
- Train T-Few on all tasks’ train sets combined

Method	Acc.
T-Few	75.8%
Human baseline [2]	73.5%
PET [50]	69.6%
SetFit [51]	66.9%
GPT-3 [4]	62.7%

Table 2: Top-5 best methods on RAFT as of writing. T-Few is the first method to outperform the human baseline and achieves over 6% higher accuracy than the next-best method.

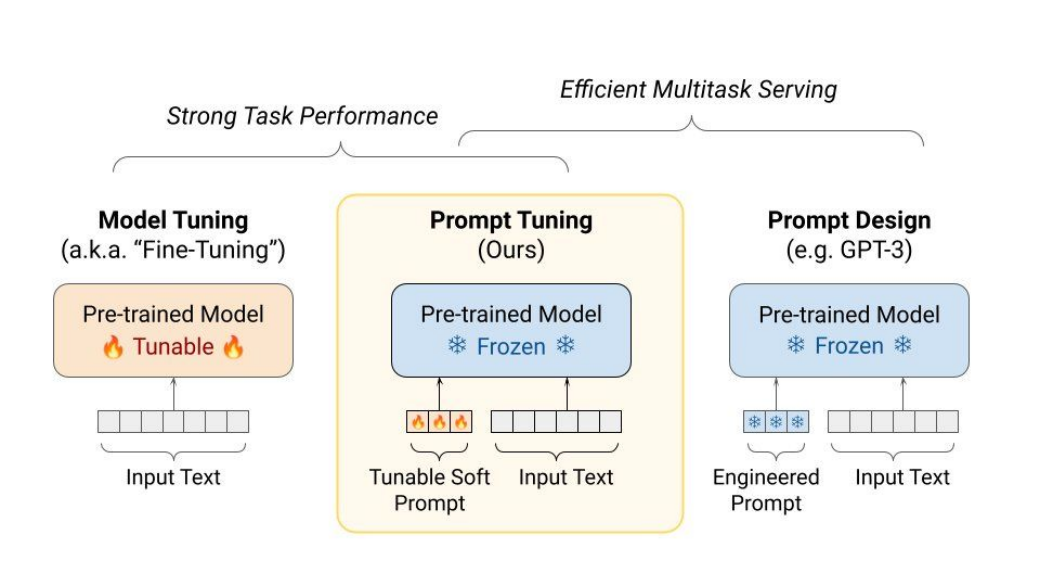
Miscellaneous Discoveries

- T0 does better zero-shot than with few-shot examples!
  - Since only trained with zero-shot data. FLAN 2022 / Flan Collection demonstrate adding few-shot data in instruction tuning mixture makes models more robust to formatting + do few shot better
- For all datasets, authors report median across many prompts
  - Intentionally did not select specific prompt or write these for the paper

Discussion Questions

- Could you understand the explanation of (IA)^3 ?
- What's your intuition for why IA^3 works?
  - (What is your comfort level with the "baseline" PEFT methods described here?)
- Do you expect this method to transfer to decoder-only models? Why or why not?

Extra: Prompt Tuning



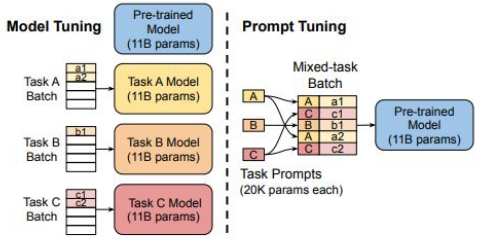
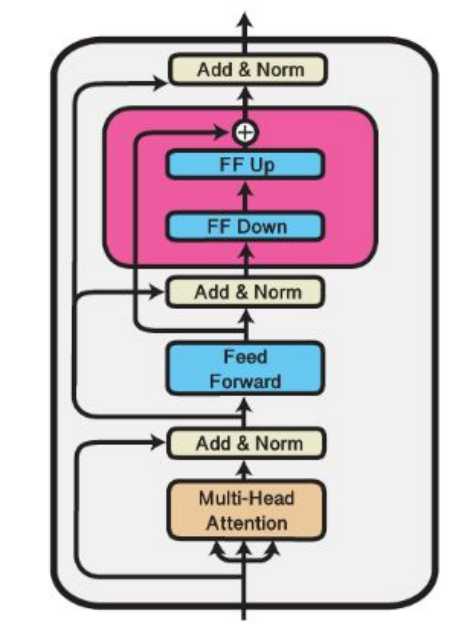
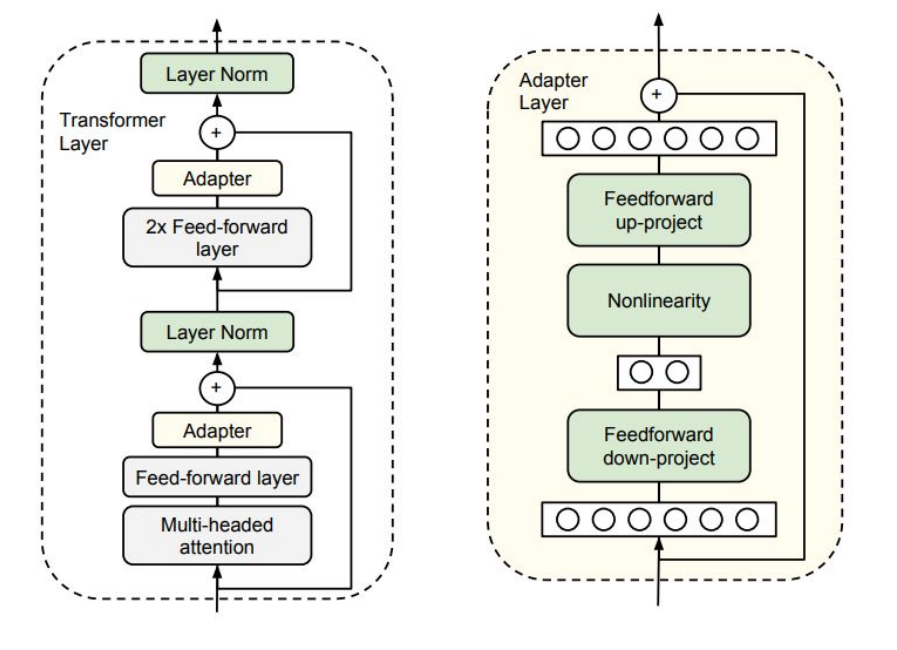


Figure 2: **Model tuning** requires making a task-specific copy of the entire pre-trained model for each downstream task and inference must be performed in separate batches. **Prompt tuning** only requires storing a small task-specific prompt for each task, and enables mixed-task inference using the original pre-trained model. With a T5 "XXL" model, each copy of the tuned model requires 11 billion parameters. By contrast, our tuned prompts would only require 20,480 parameters per task—a reduction of *over five orders of magnitude*—assuming a prompt length of 5 tokens.

Image credits: https://ai.googleblog.com/2022/02/guiding-frozen-language-models-with.html ; Power of Scale for Prompt Tuning (Lester et al. 2021)

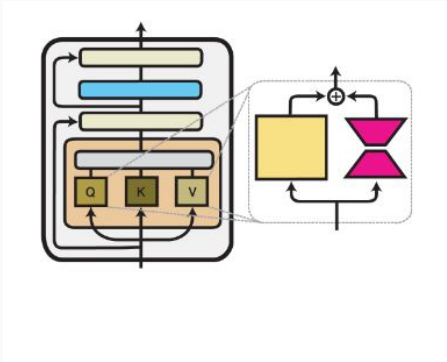
Extra: Adapter layers





Extra: LoRA

- Low-rank factorization
- Applied in *parallel* -> **absorbable** into the model!



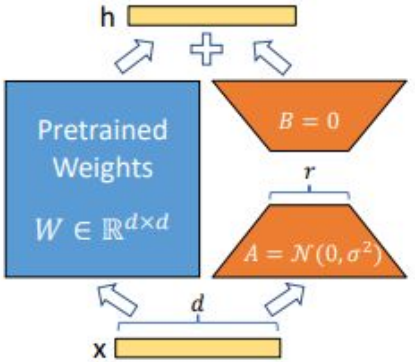


Figure 1: Our reparametrization. We only train A and B.

Extra: Prefix Tuning

- Prompt tuning, but at every layer