ReFT: Representation Finetuning for Language Models

Zhengxuan Wu[*†] **Aryaman Arora**[*†] **Zheng Wang**[†] **Atticus Geiger**[‡]
Dan Jurafsky[†] **Christopher D. Manning**[†] **Christopher Potts**[†]
[†]Stanford University [‡]Pr(Ai)^{2}R Group
{wuzhengx,aryamana,peterwz,atticusg,jurafsky,manning,cgpotts}@stanford.edu

Abstract

Parameter-efficient fine-tuning (PEFT) methods seek to adapt large models via updates to a small number of *weights*. However, much prior interpretability work has shown that  encode rich semantic information, suggesting that editing representations might be a more powerful alternative. Here, we pursue this hypothesis by developing a family of **Representation Finetuning (ReFT)** methods. ReFT methods operate on a frozen base model and learn task-specific interventions on hidden representations. We define a strong instance of the ReFT family, Low-rank Linear Subspace ReFT (LoReFT). LoReFT is a drop-in replacement for existing PEFTs and learns interventions that are 10×–50× more parameter-efficient than prior state-of-the-art PEFTs. We showcase LoReFT on eight commonsense reasoning tasks, four arithmetic reasoning tasks, Alpaca-Eval v1.0, and GLUE. In all these evaluations, LoReFT delivers the best balance of efficiency and performance, and almost always outperforms state-of-the-art PEFTs. We release a generic ReFT training library publicly at https://github.com/stanfordnlp/pyreft.

1 Introduction

Pretrained LMs are frequently finetuned to adapt them to new domains or tasks [Dai and Le, 2015]. With finetuning, a single base model can be adapted to a variety of tasks given only small amounts of in-domain data. However, finetuning the entire model is expensive, especially for very large LMs.

Parameter-efficient finetuning (PEFT) methods propose to address the high costs of full finetuning by updating only a small fraction of weights [Han et al., 2024]. This reduces memory usage and training time, and PEFTs have been shown to achieve similar performance to full finetuning in many practical settings [Hu et al., 2023]. Adapters, which are a common family of PEFTs, learn an edit that can be added to a subset of model weights, or an additional set of weights that operate alongside the frozen base model. Recent adapters such as LoRA [Hu et al., 2022] (and variants such as DoRA; Liu et al., 2024b) reduce the number of trainable parameters in learned weight updates by using low-rank approximations in place of full weight matrices during adapter training. QLoRA [Dettmers et al., 2023] further shows that full-precision adapters can be trained on top of reduced-precision models without sacrificing performance. Adapters are generally more efficient and effective than methods that introduce new model components, like prefix-tuning [Li and Liang, 2021].

A hallmark of current state-of-the-art PEFTs is that they modify *weights* rather than *representations*. However, much prior interpretability work has shown that representations encode rich semantic information, suggesting that editing representations might be a more powerful alternative to weight updates. In this paper, we pursue this hypothesis by developing and motivating **Representation Finetuning (ReFT)**. Instead of adapting model weights, ReFT methods train interventions that

[*]Equal contribution.

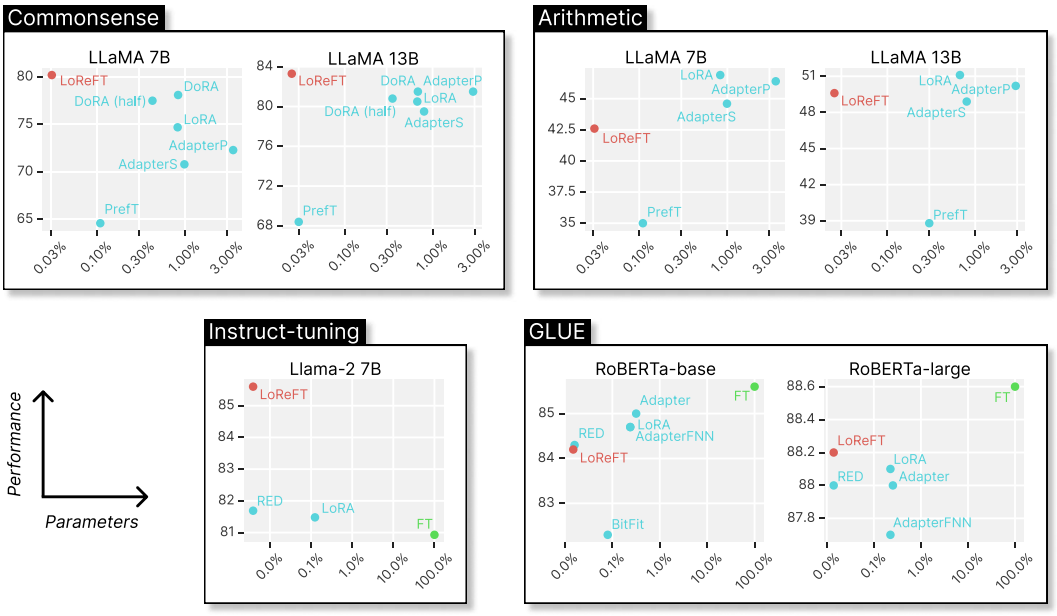

Figure 1: Parameter count vs. performance for LoReFT and other PEFTs across four benchmarks when applied to LLaMA, Llama-2, and RoBERTa models. Despite training much fewer parameters than existing PEFTs, LoReFT achieves competitive or even state-of-the-art performance on all tasks. Its value is most apparent for the largest models in our evaluations. **Note**: FT refers to full-parameter finetuning, which is not a PEFT or ReFT method.

manipulate a small fraction of model representations in order to steer model behaviors to solve downstream tasks at inference time. ReFT methods are drop-in replacements for weight-based PEFTs. This approach is inspired by recent work in LM interpretability that intervenes on representations to find faithful causal mechanisms [Geiger et al., 2023b] and to steer model behaviours at inference time [Turner et al., 2023, Li et al., 2024], and it can be seen as a generalization of the representation-editing work of Wu et al. [2024a], Turner et al. [2023], and Zou et al. [2023] (see Appendix A for formal details on the relationships).

We focus on a strong and highly efficient instance of the ReFT family that we call **Low-rank Linear Subspace ReFT (LoReFT)**. This is a parametrisation of ReFT that intervenes on hidden representations in the linear subspace spanned by a low-rank projection matrix, building directly on the distributed alignment search (DAS) method of Geiger et al. [2023b] and Wu et al. [2023]. We evaluate LoReFT on LLaMA-family models and small-scale LMs against existing PEFTs on standard benchmarks from four domains: commonsense reasoning, arithmetic reasoning, instruction-following, and natural language understanding. Compared to LoRA, we find that LoReFT uses 10×–50× times fewer parameters while achieving state-of-the-art performance for x out of y number datasets (fig. 1). These findings indicate that ReFT methods are worthy of further exploration, as they may emerge as more efficient and effective alternatives to weight-based PEFTs.

2 Related work

Parameter-efficient finetuning methods (PEFTs). In contrast to full finetuning, PEFTs train only a fraction of the model's parameters to adapt it to downstream tasks. PEFTs can be classified into three main categories:

1. **Adapter-based methods** train additional modules (e.g. fully-connected layers) on top of the frozen pretrained model. *Series adapters* insert components between LM attention or MLP layers [Houlsby et al., 2019, Pfeiffer et al., 2020, Wang et al., 2022, He et al., 2022b, Fu et al., 2021], while *parallel adapters* add modules alongside existing components [He et al., 2022a].

Since adapters add new components that cannot be easily folded into existing model weights, they impose additional burden at inference time.[1]

2. **LoRA** [Hu et al., 2022] and its recent variant DoRA [Liu et al., 2024b] use low-rank matrices to approximate additive weight updates during training, and require no additional overhead during inference since the weight updates can be merged into model. These are the strongest PEFTs currently.[2]
3. **Prompt-based methods** add randomly-initialised soft tokens to the input (usually as a prefix) and train their embeddings while keeping the LM weights frozen [Li and Liang, 2021]. The performance of these methods are often far from optimal compared to other PEFTs, and come at the cost of significant inference overhead. A variant of this method where hidden-layer activations are also tuned was introduced as a baseline in Hu et al. [2022], with better performance.

Instead of weight updates, ReFT methods learn interventions to modify a small fraction of model representations.

Representation editing. Recent work on *activation steering* and *representation engineering* shows that adding fixed steering vectors to the residual stream can enable a degree of control over pretrained LM generations without the need for resource-intensive finetuning [Subramani et al., 2022, Turner et al., 2023, Zou et al., 2023, Vogel, 2024]. For example, steering attention-head outputs can increase performance on TruthfulQA [Li et al., 2024]. Similarly, Wu et al. [2024a] show that editing representations with a learned scaling and translation operation can approach (but not surpass) the performance of LoRA on a variety of models and tasks with far fewer learned parameters. The success of these methods affirms that representations induced by pretrained LMs carry rich semantics. However, exploration of such methods has been sporadic and performance is sub-optimal; adapter-based PEFTs continue to be the state-of-the-art and impose no additional inference burden.

Interventional interpretability. Recent work in interpretability has used interventions under the framework of causal abstraction [Geiger et al., 2021] to test hypotheses about how LMs implement various behaviours. In particular, interventions on linear subspaces of representations have provided increasing evidence that human-interpretable concepts are encoded linearly; this includes linguistic features such as gender and number [Lasri et al., 2022, Wang et al., 2023, Hanna et al., 2023, Chintam et al., 2023, Yamakoshi et al., 2023, Hao and Linzen, 2023, Chen et al., 2023, Amini et al., 2023, Guerner et al., 2023, Arora et al., 2024], logical and mathematical reasoning [Wu et al., 2023], and entity attributes [Huang et al., 2024].

3 ReFT

We now define the ReFT family of methods. To do this, we first summarize the core motivation, which emerges from work on intervention-based model interpretability. We then show how this leads directly to Low-rank Linear Subspace ReFT (LoReFT). Finally, we generalize this to a family of ReFT methods.

To keep the presentation simple, we assume throughout that our target model is a Transformer-based [Vaswani et al., 2017] LM that produces contextualised representations of sequences of tokens. Given a sequence of n input tokens $\mathbf{x} = (x_1, \ldots, x_n)$, the model first embeds these into a list of representations $\mathbf{h}^{(0)} = (\mathbf{h}_1^{(0)}, \ldots, \mathbf{h}_n^{(0)})$. Then, m layers successively compute the j-th list of hidden representations $\mathbf{h}^{(j)}$ as a function of the previous list of hidden representations $\mathbf{h}^{(j-1)}$. Each hidden representation is a vector $\mathbf{h} \in \mathbb{R}^d$. The LM uses the final hidden representations $\mathbf{h}^{(m)}$ to produce its predictions. In our experiments, we consider both autoregressive LMs and masked LMs [Devlin et al., 2019]. An autoregressive LM predicts $p(x_{n+1} \mid x_1, \ldots, x_n) = \mathrm{softmax}\left(\mathbf{W}\mathbf{h}_n^{(m)}\right)$, while a masked LM predicts $p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \mathrm{softmax}\left(\mathbf{W}\mathbf{h}_i^{(m)}\right)$, where $\mathbf{W}$ is a learned matrix mapping from representations to logits over the vocabulary space.

[1] Several recent papers introduce new adapter architectures but do not benchmark them on the tasks we consider, or perform hyperparameter-tuning in a different setup than done in this work. These include: LLaMA-Adapter [Zhang et al., 2024b], LLaMA-Adapter v2 [Gao et al., 2023], Aligner [Ziheng et al., 2023].

[2] Additional methods not studied in this work: AutoLoRA [Zhang et al., 2024c], ResLoRA [Shi et al., 2024], SiRA [Zhu et al., 2023].

3.1 Motivation

The **linear representation hypothesis** claims that concepts are encoded in linear subspaces of representations in neural networks. Early connectionist work on distributed neural representations was the first to propose this [Smolensky, 1986, Rumelhart et al., 1986, McClelland et al., 1986], and recent empirical work has found evidence for this claim in neural models trained on natural language as well as other input distributions [Mikolov et al., 2013, Elhage et al., 2022, Park et al., 2023, Nanda et al., 2023, Guerner et al., 2023].

In interpretability research, the framework of causal abstraction [Geiger et al., 2021] uses **interchange interventions** to causally establish the role of neural network components in implementing particular behaviours. The logic of the interchange intervention is as follows: if one fixes a representation to what it would have been given a counterfactual input, and this intervention consistently affects model output in the way predicted by our claims about the component producing that representation, then that component plays a causal role in the behaviour being studied.

To test whether a concept is encoded in a linear subspace of a represention, as claimed by the linear representation hypothesis, one may use a **distributed interchange intervention** [Geiger et al., 2023b].[3] Let $\mathbf{b}$ be the hidden representation created at row i and column k when our model processes input b, and let $\mathbf{s}$ be the corresponding representation when that same model processes input s. A distributed interchange intervention on $\mathbf{b}$ given a counterfactual source representation $\mathbf{s}$ is then defined as

$$\mathsf{DII}(\mathbf{b}, \mathbf{s}, \mathbf{R}) = \mathbf{b} + \mathbf{R}^\top(\mathbf{R}\mathbf{s} - \mathbf{R}\mathbf{b}) \tag{1}$$

where $\mathbf{R} \in \mathbb{R}^{r \times d}$ is a low-rank projection matrix with orthonormal rows, d is the representation dimensionality, and r is the dimensionality of the subspace we are intervening on. We learn the subspace $\mathbf{R}$ using distributed alignment search (DAS), which finds the subspace that maximises the probability of the expected counterfactual output after intervention.

DAS has been used to find linear representations in LMs of entity attributes [Huang et al., 2024], linguistic features [Arora et al., 2024], sentiment [Tigges et al., 2023], and mathematical reasoning [Wu et al., 2023, Lepori et al., 2023]. However, experiments reveal that DAS is highly expressive and capable of finding causally efficacious subspaces even when a Transformer LM has been **randomly initialised** and thus has not yet learned any task-specific representations [Wu et al., 2023, Arora et al., 2024]. While this has led to debate on whether DAS is faithful enough for interpretability purposes [Makelov et al., 2024, Wu et al., 2024c], DAS's expressivity suggests that it could serve as a powerful tool for *controlling* LM behaviours, in line with work on representation editing and controllable generation [Li et al., 2024, Zou et al., 2023]. Therefore, we sought to use the distributed interchange intervention operation to make a new parameter-efficient method for adapting language models for downstream tasks.

3.2 Low-rank Linear Subspace ReFT (LoReFT)

The formulation of DII in eq. (1) immediately suggests a way to control model generations via interventions. The guiding intuition is that we can learn how to perform interventions that lead the model to accurately predict our task labels. The resulting method, Low-rank Linear Subspace ReFT (LoReFT), is defined by the following variant of eq. (1):

"h" is a "d" dimensional output that you usually get from any Transformer Layer

$$\Phi_{\text{LoReFT}}(\mathbf{h}) = \mathbf{h} + \mathbf{R}^\top(\mathbf{W}\mathbf{h} + \mathbf{b} - \mathbf{R}\mathbf{h}) \tag{2}$$

This is identical to eq. (1), except we use a *learned projected source* $\mathbf{R}\mathbf{s} = \mathbf{W}\mathbf{h} + \mathbf{b}$. Intuitively, LoReFT edits the representation in the r-dimensional subspace spanned by the columns of $\mathbf{R}$ to take on the values obtained from our linear projection $\mathbf{W}\mathbf{h} + \mathbf{b}$.

These are 3 different "trainable" weights for finetuning

The learned parameters are $\phi = \{\mathbf{R}, \mathbf{W}, \mathbf{b}\}$. As with DII, $\mathbf{R} \in \mathbb{R}^{r \times d}$ is a low-rank matrix with orthonormal rows where d is the hidden-state dimensionality and $r \leq d$ is the rank of the subspace. We further define a linear projection $\mathbf{W} \in \mathbb{R}^{r \times d}$ and bias vector $\mathbf{b} \in \mathbb{R}^r$. The parameters of the LM are frozen.

We consider both generation tasks using decoder-only or encoder–decoder LMs and classification tasks using encoder-only models. The pretrained language model is a distribution over token sequences $p(\cdot)$. We denote the model that results from the LoReFT intervention on $p(\cdot)$ as $p_\Phi(\cdot)$

[3] This notion of subspace intervention was also independently discovered by Guerner et al. [2023].

with trainable parameters ϕ. To simplify notation, we refer to the hidden representations produced by the LM on input $\mathbf{x}$ as $\mathbf{h}(\mathbf{x})$, and those by the intervened LM as $\mathbf{h}_\Phi(\mathbf{x})$.

For generation tasks, our training objective is language modelling. Given an input sequence $\mathbf{x} = (x_1, \ldots, x_n)$ with n input tokens as the prompt, the goal is to predict the output sequence $\mathbf{y} = (y_1, \ldots, y_m)$ with m input tokens. We minimise the cross-entropy loss with teacher-forcing over all output positions.

$$\min_{\phi} \left\{ -\sum_{i=1}^{m} \log p_\Phi \left(y_i \mid \mathbf{x}\mathbf{y}_{<i} \right) \right\} \tag{3}$$

For single-label classification tasks, we add a classification head $H_\theta(\cdot)$ with parameters θ that takes the final-layer representation at the first token (`CLS`) as input and outputs a distribution over classes. H has the learned parameters $\theta = \{\mathbf{W}_o, \mathbf{b}_o, \mathbf{W}_d, \mathbf{b}_d\}$.

$$H_\theta(\cdot \mid \mathbf{h}) = \text{softmax}\left(\mathbf{W}_o(\tanh(\mathbf{W}_d\mathbf{h}_1^{(m)} + \mathbf{b}_d)) + \mathbf{b}_o\right) \tag{4}$$

We learn the parameters of the head and those of the intervention function Φ. We minimise the cross-entropy loss of the target class y given input $\mathbf{x}$.

$$\min_{\phi,\theta} \{-\log H_\theta(y \mid \mathbf{h}_\Phi(\mathbf{x}))\} \tag{5}$$

3.3 The ReFT family of methods

It is straightforward to generalize the above to define a family of intervention-based representation finetuning methods. We first define a general notion of *intervention*, i.e. the modification of hidden representations during the model forward pass:

Definition 3.1. An **intervention** I is a tuple $\langle \Phi, P, L \rangle$ that encapsulates a single inference-time modification of the representations computed by a Transformer-based LM. The three components of an intervention are

- The intervention function $\Phi : \mathbb{R}^d \to \mathbb{R}^d$ with learned parameters ϕ.
- A set of input positions $P \subseteq \{1, \ldots, n\}$ that the intervention is applied to.
- The layer $L \in \{1, \ldots, m\}$ at which the intervention is applied. It can be applied to selected Layers only

We implement the intervention I as the following operation that overwrites some representations $\mathbf{h}$:

$$\mathbf{h}^{(L)} \leftarrow \left(\Phi\left(\mathbf{h}_p^{(L)}\right) \text{ if } p \in P \text{ else } \mathbf{h}_p^{(L)}\right)_{p \in 1,\ldots,n} \tag{6}$$

The intervention is applied immediately after the computation of $\mathbf{h}^{(L)}$ and thus affects the representations computed in later layers $\mathbf{h}^{(L+1)}, \ldots, \mathbf{h}^{(m)}$.

Figure 2 provides a schematic overview of an intervention. A ReFT is then defined as a constrained set of non-overlapping interventions:

Definition 3.2. A **ReFT method** is a set of f interventions $\mathbf{I} = \{I_1, \ldots, I_f\}$. We enforce that for any two interventions $I_j, I_k \in \mathbf{I}$ such that they operate on the same layer $L_j = L_k$, their intervention positions must be disjoint, i.e. $P_j \cap P_k = \varnothing$. The parameters $(\phi_1, \ldots, \phi_f)$ of all of the intervention functions are independent.

ReFT is thus a generic framework encompassing interventions on hidden representations during the model forward pass. In appendix A, we show how a variety of existing inference-time intervention methods can be described within this framework.

4 Experiments

To evaluate LoReFT against existing PEFTs, we conduct experiments across four diverse NLP benchmarks covering more than 20 datasets. Our goal is to provide a rich picture of how LoReFT performs in different scenarios. Here is a brief overview of our benchmarks:

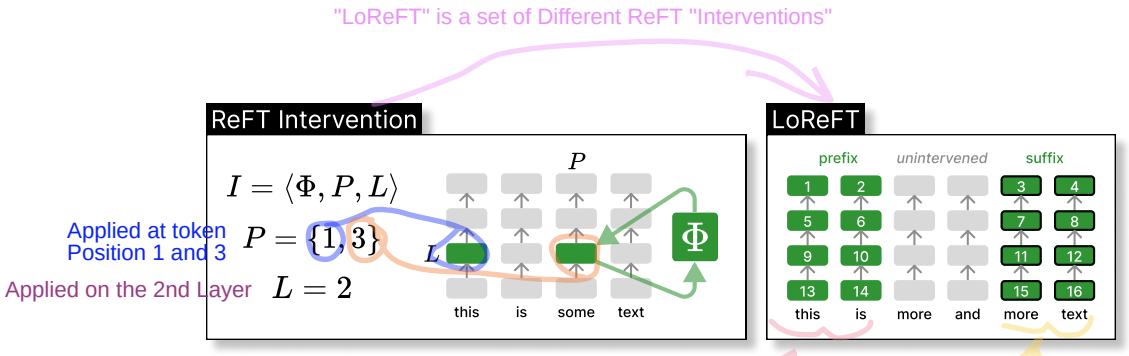

Figure 2: **Illustration of ReFT**. **(1)** The left panel depicts an intervention I: the intervention function Φ is applied to hidden representations at positions P in layer L. **(2)** The right panel depicts the hyperparameters we tune when experimenting with LoReFT. Specifically, the figure depicts application of LoReFT at all layers with prefix length $p = 2$ and suffix length $s = 2$. When not tying layer weights, we train separate intervention parameters at each position and layer, resulting in 16 interventions with unique parameters in this example.

- **Commonsense reasoning** which contains eight commonsense reasoning datasets, including BoolQ [Clark et al., 2019], PIQA [Bisk et al., 2020], SIQA [Sap et al., 2019], HellaSwag [Zellers et al., 2019], WinoGrande [Sakaguchi et al., 2021], ARC-e, ARC-c [Clark et al., 2018], and OBQA [Mihaylov et al., 2018]. The task is formulated as a multiple-choice problem.
- **Arithmetic reasoning** which contains four datasets for math world problems, including AQuA [Ling et al., 2017], GSM8K [Cobbe et al., 2021], MAWPS [Koncel-Kedziorski et al., 2016], and SVAMP [Patel et al., 2021]. Models need to generate chain-of-thought [Wei et al., 2022] before the final answer.
- **Instruction-following** which evaluates whether models can follow human instructions. We use Ultrafeedback [Cui et al., 2023] as our training data, and Alpaca-Eval v1.0 [Li et al., 2023] as our evaluation dataset.
- **Natural language understanding** which contains eight datasets from the GLUE benchmark [Wang et al., 2018] such as sentiment analysis and natural language inference.

We experiment with LMs at different scales ranging from RoBERTa-base [Liu et al., 2019] with 125M to LLaMA-1 [Touvron et al., 2023] with 13B parameters, and both masked and autoregressive transformer language models. We benchmark against existing PEFTs such as prefix-tuning [Li and Liang, 2021], adapter-tuning with both Series Adapters and Parallel Adapters, LoRA [Hu et al., 2022], and DoRA [Liu et al., 2024b]. Our comparisons focus on both performance and parameter efficiency. In our comparisons, we use hyperparameter-tuned scores from previous works when possible. We load our base LMs in `torch.bfloat16` to save memory. **All of our experiments are run with a single GPU: NVIDIA A100 40G/80G or RTX 6000**. Examples of raw model generations are in appendix I.

4.1 Hyperparameter configuration

When using LoReFT in practice, we must decide how many interventions to learn and which layers and input positions to apply each one on. We propose learning interventions on a fixed number of p prefix and s suffix positions in the prompt. Specifically, we tune four hyperparameters:

1. The number of prefix positions p to intervene on, i.e. positions $\{1, \ldots, p\}$.
2. The number of suffix positions s to intervene on, i.e. positions $\{n - s + 1, \ldots, n\}$.
3. Which set of layers L to intervene on.
4. Whether or not to tie intervention parameters ϕ across different positions in the same layer.

This simplifies the hyperparameter search space and ensures only a fixed additional inference cost that does not scale with prompt length. Compared to LoRA, we only have the additional consideration of which positions to intervene on.

Following the definition of ReFT, we thus define the untied and tied variants of our LoReFT intervention as:

$$P = \{1, \ldots, p\} \cup \{n - s + 1, \ldots, n\} \tag{7}$$

$$\mathbf{I}_{\text{untied}} = \{\langle \Phi_{\text{LoReFT}}, \{p\}, l \rangle \mid p \in P, l \in L\} \tag{8}$$

$$\mathbf{I}_{\text{tied}} = \{\langle \Phi_{\text{LoReFT}}, P, l \rangle \mid l \in L\} \tag{9}$$

Additionally, when applying untied LoReFT to a prompt with length n where $n < p + s$, we set $p \leftarrow \min(p, \lfloor n/2 \rfloor)$ and $s \leftarrow \min(s, \lceil n/2 \rceil)$ and do not apply the truncated interventions in $\mathbf{I}_{\text{untied}}$. We also tune neural-network training hyperparameters. Further details are provided in appendix C. We graphically depict how an untied LoReFT intervenes on an example prompt in fig. 2.

Unlike previous works [Hu et al., 2022, 2023, Liu et al., 2024b] where hyperparameter tuning may involve optimising performance directly on test sets, we only tune our hyperparameters on development sets which do not contain any overlapping examples with the test sets of our tasks. We further describe hyperparameter tuning for each benchmark in appendix C.

4.2 Commonsense reasoning

Model	PEFT	Params (%)	Accuracy (↑)								
			BoolQ	PIQA	SIQA	HellaS.	WinoG.	ARC-e	ARC-c	OBQA	Avg.
ChatGPT*	—	—	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-7B	PrefT*	0.110%	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	Adapter^{S}*	0.990%	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Adapter^{P}*	3.540%	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.3
	LoRA*	0.830%	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	DoRA (half)*	0.430%	**70.0**	82.6	79.7	83.2	80.6	80.6	65.4	77.6	77.5
	DoRA*	0.840%	68.5	82.9	79.6	84.8	80.8	81.4	65.8	**81.0**	78.1
	LoReFT (ours)	0.031%	69.3	**84.4**	**80.3**	**93.1**	**84.2**	**83.2**	**68.2**	78.9	**80.2**
LLaMA-13B	PrefT*	0.030%	65.3	75.4	72.1	55.2	68.6	79.5	62.9	68.0	68.4
	Adapter^{S}*	0.800%	71.8	83.0	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Adapter^{P}*	2.890%	**72.5**	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.5
	LoRA*	0.670%	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	DoRA (half)*	0.350%	**72.5**	85.3	79.9	90.1	82.9	82.7	69.7	83.6	80.8
	DoRA*	0.680%	72.4	84.9	81.5	92.4	84.2	84.2	69.6	82.8	81.5
	LoReFT (ours)	0.025%	72.1	**86.3**	**81.8**	**95.1**	**87.2**	**86.2**	**73.7**	**84.2**	**83.3**

Table 1: Accuracy comparison of LLaMA-7B and LLaMA-13B against existing PEFT methods on eight commonsense reasoning datasets. *Performance results of all baseline methods are taken from Liu et al. [2024b]. We report averaged performance of three runs with distinct random seeds for our method. For LoReFT, # Param. (%) is calculated by dividing the number of trainable parameters by the number of parameters of the base LM.

We replicate the experimental setup in Hu et al. [2023] and finetune LLaMA-1 7B and 13B on a combined dataset of eight commonsense reasoning tasks (COMMONSENSE170K). We report scores on each task's test set individually. We compare with PEFTs benchmarked in Hu et al. [2023] as well as the identical experiment reported in Liu et al. [2024b] for DoRA.

Datasets. Examples from these datasets are formulated as multiple-choice problems where the model needs to directly generate the correct choice without rationales. We use the same prompt template as in Hu et al. [2023] with additional string normalization (removing leading and trailing whitespace). Further details on the eight tasks are given in appendix B.1.

Hyperparameter tuning. We do not want to do hyperparameter selection based on test set results. To avoid this, we use the hyperparameter settings of the model that performs best on a development set created from the GSM8K training set, except we use a lower number of epochs (6 instead of 12) because the COMMONSENSE170K training set is more than 20 times larger than GSM8K. This allows us to tune relevant hyperparamters, and also serves to test the robustness of these settings across different domains.

Results. We report results in table 1. LoReFT sets state-of-the-art performance on the commonsense reasoning tasks, outperforming all other methods by a considerable margin.

4.3 Arithmetic reasoning

Model	PEFT	Params (%)	Accuracy (↑)				
			AQuA	GSM8K	MAWPS	SVAMP	Avg.
LLaMA-7B	PrefT*	0.110%	14.2	24.4	63.4	38.1	35.0
	Adapter^{S}*	0.990%	15.0	33.3	77.7	**52.3**	44.6
	Adapter^{P}*	3.540%	18.1	35.3	**82.4**	49.6	46.4
	LoRA*	0.830%	18.9	**37.5**	79.0	52.1	**46.9**
	LoReFT (ours)	0.031%	**21.4**	26.0	76.2	46.8	42.6
LLaMA-13B	PrefT*	0.300%	15.7	31.1	66.8	41.4	38.8
	Adapter^{S}*	0.800%	22.0	44.0	78.6	50.8	48.9
	Adapter^{P}*	2.890%	20.5	43.3	81.1	**55.7**	50.2
	LoRA*	0.670%	18.5	**47.5**	**83.6**	54.6	**51.1**
	LoReFT (ours)	0.025%	**23.6**	38.1	82.4	54.2	49.6

Table 2: Accuracy comparison of LLaMA-7B and LLaMA-13B against existing PEFT methods on four arithmetic reasoning datasets. *Performance results of all baseline methods are taken from Hu et al. [2023]. We report averaged performance of three runs with distinct random seeds for our method.

Similar to the previous experiment, we follow the experimental setup in Hu et al. [2023] and finetune LLaMA-1 7B and 13B on a combined dataset of seven arithmetic reasoning tasks with LM-generated chain-of-thought steps (MATH10K) and report scores on four of the tasks' test sets. We only evaluate correctness on the final numeric or multiple-choice answer.

Hyperparameter tuning. We use the same hyperparameter settings as for the Commonsense Reasoning benchmark with 12 epochs for training.

Datasets. For more details on the datasets, see appendix B.2. We use the same prompt template and hyperparameter settings as in the previous experiment.

Results. We report results in table 2. We find that LoReFT does not perform as well at arithmetic reasoning tasks compared to LoRA and adapters, but does outperform prefix-tuning. Our results suggest that LoReFT may have more trouble on chain-of-thought reasoning than the single-step commonsense reasoning tasks due to the length of generations (and greater length necessarily reduces the effect of the intervention) and overall greater difficulty of the task. Our results show that LoReFT performs better with 13B model than 7B model, which suggests that LoReFT scales with model size. Overall, we note that the arithmetic reasoning results show a lot of variation, with no single method emerging as a clear winner across all of them.

4.4 Instruction-following

Base LMs require instruction finetuning to follow human prompts [Ouyang et al., 2022]. We follow the experimental setup in Wu et al. [2024a] and finetune Llama-2 7B with Ultrafeedback [Cui et al., 2023]. We compare against full parameter fine-tuning, LoRA ,and RED. We use Alpaca-Eval v1.0 [Li et al., 2023] for evaluation which computes win-rate against `text-davinci-003` using GPT-4 as the annotator. We use the same prompt template as in Taori et al. [2023].

Datasets. We finetune Llama-2 7B with Ultrafeedback. Ultrafeedback is high-quality instruction dataset where responses are generated via scoring a diverse set of model responses from a list of candidates (e.g. ChatGPT and Bard). The score is calculated as weighted score of instruction-following, truthfulness, honesty and helpfulness. Some of the best 7B and 13B chat-models (e.g. UltraLM-13B [Ding et al., 2023]) are finetuned with Ultrafeedback.

Hyperparameter tuning. We do hyperparameter-tuning on the unseen instruction-following dataset Alpaca-52K [Taori et al., 2023] with only LLaMA-7B to prevent test-set hill-climbing. We then use the hyperparameter settings of our best performing model to finetune on Ultrafeedback. For hyperparameter tuning, we use Alpaca-Eval v1.0 with GPT-4 turbo as the annotator for fast turnaround, which also prevents overfitting with GPT-4 as a judge.

Model & PEFT	Params (%)	Win-rate (↑)
GPT-3.5 Turbo 1106†	—	86.30
Llama-2 Chat 13B†	—	81.10
Llama-2 Chat 7B†	—	71.40
Llama-2 7B & FT*	100%	80.93
Llama-2 7B & LoRA*	0.1245%	81.48
Llama-2 7B & RED*	0.0039%	81.69
Llama-2 7B & **LoReFT (ours)**	0.0039%	**85.60**
Llama-2 7B & **LoReFT (ours, *half*)**	0.0019%	84.12
Llama-2 7B & **LoReFT (ours, *1K*)**†	0.0039%	81.91

Table 3: Instruction tuning evaluation results for instruction-tuned Llama-2 7B with Alpaca-Eval v1.0. We report averaged performance of two runs with distinct random seeds for our method. *half* denotes our runs with half of the rank; *1K* denotes our runs with a low-resource setting where there is only 1K training examples. †Performance results of baseline methods are taken from Li et al. [2023]. *Performance results of baseline methods are taken from Wu et al. [2024a]. †**It takes ≈18 minutes to train our Llama-2 Chat 7B on a single A100 40G GPU with ≈1MB parameters on disk.**

Results. We report results in table 3. When matched in parameter count to the previous most parameter-efficient PEFT (RED) and trained on Llama-2 7B, LoReFT outperforms all reported finetuning methods (including full finetuning) and achieves a win-rate within 1% of GPT-3.5 Turbo 1106. Furthermore, after halving the parameter count or using only 1/64-th of the data, LoReFT still outperforms other finetuning methods. This result shows that LoReFT can succeed at long-form text generation (despite some potential issues with this in the arithmetic reasoning evaluations). Our results also suggest that LoReFT could be a viable way to evaluate instruction-tuning dataset quality with a quick turnaround time.[4]

Model	PEFT	Params (%)	Accuracy (↑)								
			MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
base	FT	100%	87.3	94.4	87.9	62.4	92.5	91.7	78.3	90.6	85.6
	Adapter*	0.318%	87.0	93.3	88.4	60.9	92.5	**90.5**	76.5	**90.5**	**85.0**
	LoRA*	0.239%	86.6	93.9	88.7	59.7	**92.6**	90.4	75.3	90.3	84.7
	Adapter^{FNN}*	0.239%	**87.1**	93.0	88.8	58.5	92.0	90.2	77.7	90.4	84.7
	BitFit*	0.080%	84.7	**94.0**	88.0	54.0	91.0	87.3	69.8	89.5	82.3
	RED*	0.016%	83.9	93.9	**89.2**	**61.0**	90.7	87.2	78.0	90.4	84.3
	LoReFT (ours)	0.015%	83.1	93.4	**89.2**	60.4	91.2	87.4	**79.0**	90.0	84.2
large	FT	100%	88.8	96.0	91.7	68.2	93.8	91.5	85.8	92.6	88.6
	Adapter*	0.254%	90.1	95.2	90.5	65.4	94.6	**91.4**	85.3	91.5	88.0
	LoRA*	0.225%	90.2	96.0	89.8	65.5	**94.7**	90.7	86.3	**91.7**	88.1
	Adapter^{FNN}*	0.225%	**90.3**	96.1	**90.5**	64.4	94.3	91.3	84.8	90.2	87.7
	RED*	0.014%	89.5	96.0	90.3	**68.1**	93.5	88.8	86.2	91.3	88.0
	LoReFT (ours)	0.014%	89.2	**96.2**	90.1	68.0	94.1	88.5	**87.5**	91.6	**88.2**

Table 4: Accuracy comparison of RoBERTa-base and RoBERTa-large against existing PEFT methods on the GLUE benchmark. *Performance results of all baseline methods are taken from Wu et al. [2024a]. We report averaged performance of five runs with distinct random seeds for our method. # Param. (%) is calculated by dividing the number of trainable parameters (excluding the number of parameters of the classification head) with the number of parameter of the base LM.

4.5 Natural language understanding

We evaluate LoReFT on the GLUE benchmark [Wang et al., 2018] against existing PEFTs. We use this set of experiments to show LoReFT works well even with small-scale LMs, and can improve representations for classification tasks and not just text generation. We finetune RoBERTa-base (125M) as well as RoBERTa-large (350M) on GLUE, a sequence classification benchmark for natural

[4]We release our ReFT weights (<1MB) of our instruction-tuned model through HuggingFace and provide a tutorial at `https://github.com/stanfordnlp/pyreft/blob/main/examples/chat`.

language understanding (NLU) which covers domains such as sentiment classification and natural language inference. Details about the GLUE benchmark can be found in its original paper. We follow Wu et al. [2024a] for proper evaluation on GLUE validation set: we split the validation set into two sets guarded by a random seed, and we pick the best model with highest in-training validation accuracy to evaluate on the other held-out half for testing accuracy.

Hyperparameter tuning. We tune our hyperparameters for each task separately. which is standard for PEFTs. To avoid overfitting to random seeds, we hyperparameter-tune our models with a constant seed, and report averaged results over that and four additional unseen seeds. We describe hyperparameter tuning experiments in Appendix C.

Results. We report results in table 4. LoReFT obtains comparable performance with PEFT methods on both model sizes when parameter matched with RED, the previous most parameter-efficient PEFT for this task. Full results with standard deviation is in table 10.

5 `pyreft`: A ReFT-native Python Library

To lower the cost of switching from PEFTs to ReFT, we release `pyreft`, a Python library made for training and sharing ReFTs. Our library is built on top of `pyvene` [Wu et al., 2024b], a library for performing and training activation interventions on arbitrary PyTorch models. We publish our library on PyPI.[5] Any pretrained LM available on HuggingFace is supported through `pyreft` for finetuning with ReFT methods, and finetuned models can be easily uploaded to HuggingFace. The following example shows steps to wrap a Llama-2 7B model with a single intervention on the residual stream output of the 19-th layer:

```
import torch
import transformers

from pyreft import (
    get_reft_model,
    ReftConfig,
    LoreftIntervention,
    ReftTrainerForCausalLM
)

# loading huggingface model
model_name_or_path = "yahma/llama-7b-hf"
model = transformers.AutoModelForCausalLM.from_pretrained(
    model_name_or_path, torch_dtype=torch.bfloat16, device_map="cuda")

# wrap the model with rank-1 constant reft
reft_config = ReftConfig(representations={
    "layer": 19, "component": "block_output",
    "intervention": LoreftIntervention(
    embed_dim=model.config.hidden_size, low_rank_dimension=1)})
reft_model = get_reft_model(model, reft_config)
reft_model.print_trainable_parameters()
```

The wrapped model can be trained for downstream tasks. We also provide data loading helpers to construct training data that is compatible with HuggingFace trainers:

```
tokenizer = transformers.AutoTokenizer.from_pretrained(model_name_or_path)

# get training data with customized dataloaders
data_module = make_supervised_data_module(
    tokenizer=tokenizer, model=model, layers=[19],
    training_args=training_args, data_args=data_args)

# train
trainer = reft.ReftTrainerForCausalLM(
    model=reft_model, tokenizer=tokenizer, args=training_args, **data_module)
trainer.train()
trainer.save_model(output_dir=training_args.output_dir)
```

[5] `pip install pyreft`

6 Conclusion

In this paper, we propose a strong alternative to PEFTs, LoReFT. LoReFT achieves strong performance across benchmarks from four domains while being 10×–50× more efficient than prior state-of-the-art PEFTs. Notably, LoReFT establishes new state-of-the-art performance on commonsense reasoning, instruction-following, and natural language understanding against the strongest PEFTs. We also show how our method can be described under a generic framework — ReFT. ReFT is a new approach to finetuning that is more powerful, more parameter-efficient, and more interpretable than any existing PEFTs. We hope our work serves as an initial call for the community to study ReFTs. We also hope to explore why ReFT works, and we provide some of our early explorations in our supplementary materials, focusing on memorisation (appendix E and appendix F) and compositional merging of ReFT weights (appendix G).

7 Limitations and future work

More diverse models. Due to limited time and resources, we mainly explored the LLaMA-family of models. In future work, we hope to explore the effectiveness of ReFT on other model families (e.g. Mistral-7B or GPT-2). It would be also be worth trying ReFT on vision-language models such as LLaVA [Liu et al., 2024a].

Additional design considerations. The capabilities of ReFT have not yet been fully explored due to the large hyperparameter search space (e.g. which position to intervene on), and we are interested in automating this search. In particular, it would be rewarding to investigate more effective interventions for arithmetic reasoning tasks — one idea we have considered is scheduling interventions during generation, which may improve performance at the expense of a larger inference-time computation burden. In addition, we have not fully explored the power of orthogonality of our learned subspace yet. We are currently investigating whether learned orthogonal subspaces can be composed together without adaptation. Some encouraging initial findings are reported in appendix G.

ReFT, abstraction, and generation. Neural network interpretability research often struggles to contribute directly to improving models. With ReFT, we have shown one way to overcome this challenge. The ReFT framework is rooted in work on causal abstraction [Geiger et al., 2023a] for model interpretability, and LoReFT builds directly on the distributed interchange intervention method of Geiger et al. [2023b] and Wu et al. [2023]. See also the interchange intervention training (IIT) method of Geiger et al. [2022], Wu et al. [2022], Huang et al. [2023c]. In a similar vein, recent work also uses representation-based editing of the Transformer stream to steer model behavior [Li et al., 2024, Zou et al., 2023]. ReFT advances this line of work by showing one way that such steering can be learned, rather than being merely a post hoc analysis step.

The precise ways in which ReFT works deserve deeper exploration. Although these methods intervene on representations, the causal effect of such interventions may only emerge in the model's upstream computations. In other words, the power of ReFT may come from the fact that it creates new causal pathways or modifies the strength of some existing ones. We leave it to future research to track these effects, and perhaps to explore more structured ReFTs to modify complex causal pathways in LMs.

ReFT and model interpretability. ReFT relies on insights from work on interpretability, and it may also be able to contribute insights back to that field. In particular, LoReFT shows that training a set of low-rank interventions on selected residual streams can induce a base LM to follow instructions (section 4.4). In other words, a single set of neurons, when intervened on in a particular way, can achieve generalised control over a vast number of tasks. This is a serious challenge to work seeing to interpret individual neurons in isolation (for related criticisms, see Huang et al. 2023b). The success of ReFT suggests to us a quite different approach to interperetability, one that starts from the assumption that neurons will play different roles in different contexts.

Evaluation practices in PEFT research. In this work, we hyperparameter-tune ReFT on development sets that do not overlap with the test set. Unfortunately, a considerable portion of the literature on PEFTs directly hill-climbs performance on test sets. This results in overfitting to specific tasks, which gives practitioners less certainty about the real-world performance of different methods and impedes fair comparison. We hope that future work can introduce benchmarks for evaluating PEFTs and ReFTs. These should allow for compute- or time-matched hyperparameter-tuning comparisons, and they should disallow any kind of tuning or model selection based on the test set.

Acknowledgements

We thank Jing Huang for helpful discussion in designing our memorisation tests as well as writing. We thank Chenglei Si, Harshit Joshi, Jordan Juravsky, Julie Kallini, Ken Liu, Rohan Pandey, Jiuding Sun, Leonard Tang, Tristan Thrush, Shengguang Wu, Qinan Yu, Yanzhe Zhang, Amir Zur, and Shiqi Chen for helpful discussion about the project and comments on the manuscript.

References

Afra Amini, Tiago Pimentel, Clara Meister, and Ryan Cotterell. Naturalistic causal probing for morpho-syntax. *Transactions of the Association for Computational Linguistics*, 11:384–403, 2023. doi: 10.1162/tacl_a_00554. URL https://aclanthology.org/2023.tacl-1.23.

Aryaman Arora, Dan Jurafsky, and Christopher Potts. CausalGym: Benchmarking causal interpretability methods on linguistic tasks. *arXiv:2402.12560*, 2024. URL https://arxiv.org/abs/2402.12560.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439, 2020. URL https://arxiv.org/abs/1911.11641.

Lewis Carroll. *Alice's Adventures in Wonderland*. Macmillan, London, 1865.

Angelica Chen, Ravid Schwartz-Ziv, Kyunghyun Cho, Matthew L. Leavitt, and Naomi Saphra. Sudden drops in the loss: Syntax acquisition, phase transitions, and simplicity bias in MLMs. *arXiv:2309.07311*, 2023. URL https://arxiv.org/abs/2309.07311v4.

Abhijith Chintam, Rahel Beloch, Willem Zuidema, Michael Hanna, and Oskar van der Wal. Identifying and adapting transformer-components responsible for gender bias in an English language model. In Yonatan Belinkov, Sophie Hao, Jaap Jumelet, Najoung Kim, Arya McCarthy, and Hosein Mohebbi, editors, *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 379–394, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.blackboxnlp-1.29. URL https://aclanthology.org/2023.blackboxnlp-1.29.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. UltraFeedback: Boosting language models with high-quality feedback. *arXiv:2310.01377*, 2023. URL https://arxiv.org/abs/2310.01377.

Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/hash/7137debd45ae4d0ab9aa953017286b20-Abstract.html.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. 36:10088–10115, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1feb87871436031bdc0f2beaa62a049b-Abstract-Conference.html.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.183. URL https://aclanthology.org/2023.emnlp-main.183.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/toy_model/index.html.

Stanislav Fort. Scaling laws for adversarial attacks on language model activations, 2023. URL http://arxiv.org/abs/2312.02780.

Cheng Fu, Hanxian Huang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. Learn-to-Share: A hardware-friendly transfer learning framework exploiting computation and parameter sharing. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 3469–3479. PMLR, 2021. URL http://proceedings.mlr.press/v139/fu21a.html.

Peng Gao, Jiaming Han, Renrui Zhang, Ziyi Lin, Shijie Geng, Aojun Zhou, Wei Zhang, Pan Lu, Conghui He, Xiangyu Yue, et al. LLaMA-Adapter v2: Parameter-efficient visual instruction model. *arXiv:2304.15010*, 2023. URL https://arxiv.org/abs/2304.15010.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9574–9586. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/4f5c422f4d49a5a807eda27434231040-Paper.pdf.

Atticus Geiger, Zhengxuan Wu, Hanson Lu, Josh Rozner, Elisa Kreiss, Thomas Icard, Noah Goodman, and Christopher Potts. Inducing causal structure for interpretable neural networks. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7324–7338. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/geiger22a.html.

Atticus Geiger, Chris Potts, and Thomas Icard. Causal abstraction for faithful model interpretation. *arXiv:2301.04709*, 2023a. URL https://arxiv.org/abs/2301.04709.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah D. Goodman. Finding alignments between interpretable causal variables and distributed neural representations. *arXiv:2303.02536*, 2023b. URL https://arxiv.org/abs/2303.02536.

Clément Guerner, Anej Svete, Tianyu Liu, Alexander Warstadt, and Ryan Cotterell. A geometric notion of causal probing. *arXiv:2307.15054*, 2023. URL https://arxiv.org/abs/2307.15054.

Zeyu Han, Chao Gao, Jinyang Liu, Sai Qian Zhang, et al. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv:2403.14608*, 2024. URL https://arxiv.org/abs/2403.14608.

Michael Hanna, Yonatan Belinkov, and Sandro Pezzelle. When language models fall in love: Animacy processing in transformer language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12120–12135, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.744. URL https://aclanthology.org/2023.emnlp-main.744.

Sophie Hao and Tal Linzen. Verb conjugation in transformers is determined by linear encodings of subject number. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4531–4539, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.300. URL https://aclanthology.org/2023.findings-emnlp.300.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *The Tenth International Conference on Learning Representations, ICLR 2022*, Virtual Event, 2022a. URL https://openreview.net/forum?id=0RDcd5Axok.

Shwai He, Liang Ding, Daize Dong, Jeremy Zhang, and Dacheng Tao. SparseAdapter: An easy approach for improving the parameter-efficiency of adapters. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2184–2190, Abu Dhabi, United Arab Emirates, December 2022b. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.160. URL https://aclanthology.org/2022.findings-emnlp.160.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1058. URL https://aclanthology.org/D14-1058.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 2019. URL http://proceedings.mlr.press/v97/houlsby19a.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022*, Virtual Event, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. LLM-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5254–5276, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.319. URL https://aclanthology.org/2023.emnlp-main.319.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. Lorahub: Efficient cross-task generalization via dynamic lora composition. *arXiv preprint arXiv:2307.13269*, 2023a.

Jing Huang, Atticus Geiger, Karel D'Oosterlinck, Zhengxuan Wu, and Christopher Potts. Rigorously assessing natural language explanations of neurons. In Yonatan Belinkov, Sophie Hao, Jaap Jumelet, Najoung Kim, Arya McCarthy, and Hosein Mohebbi, editors, *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 317–331, Singapore, December 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.blackboxnlp-1.24. URL https://aclanthology.org/2023.blackboxnlp-1.24.

Jing Huang, Zhengxuan Wu, Kyle Mahowald, and Christopher Potts. Inducing character-level structure in subword-based language models with type-level interchange intervention training. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 12163–12180, Toronto, Canada, July 2023c. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.770. URL https://aclanthology.org/2023.findings-acl.770.

Jing Huang, Christopher Potts Zhengxuan Wu, Mor Geva, and Atticus Geiger. RAVEL: Evaluating interpretability methods on disentangling language model representations. *arXiv:2402.17700*, 2024. URL https://arxiv.org/abs/2402.17700.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597, 2015. doi: 10.1162/tacl_a_00160. URL https://aclanthology.org/Q15-1042.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. MAWPS: A math word problem repository. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1136. URL https://aclanthology.org/N16-1136.

Karim Lasri, Tiago Pimentel, Alessandro Lenci, Thierry Poibeau, and Ryan Cotterell. Probing for the usage of grammatical number. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8818–8831, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.603. URL https://aclanthology.org/2022.acl-long.603.

Michael A. Lepori, Thomas Serre, and Ellie Pavlick. Uncovering causal variables in transformers using circuit probing. *arXiv:2311.04354*, 2023. URL https://arxiv.org/abs/2311.04354.

Hector Levesque, Ernest Davis, and Leora Morgenstern. The Winograd Schema Challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, 2012. URL https://cdn.aaai.org/ocs/4492/4492-21843-1-PB.pdf.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. 36, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/81b8390039b7302c909cb769f8b6cd93-Abstract-Conference.html.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaEval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 2023.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv:1705.04146*, 2017. URL https://arxiv.org/abs/1705.04146.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in Neural Information Processing Systems*, 36, 2024a. URL https://arxiv.org/abs/2304.08485.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024b. URL https://arxiv.org/abs/2402.09353.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv:1907.11692*, 2019. URL https://arxiv.org/abs/1907.11692.

Aleksandar Makelov, Georg Lange, Atticus Geiger, and Neel Nanda. Is this the subspace you are looking for? An interpretability illusion for subspace activation patching. In *The Twelfth International Conference on Learning Representations*, Vienna, Austria, 2024. URL https://arxiv.org/abs/2311.17030.

James L. McClelland, David E. Rumelhart, and PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2: Psychological and Biological Models. MIT Press, 1986.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv:1809.02789*, 2018. URL https://arxiv.org/abs/1809.02789.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In Lucy Vanderwende, Hal Daumé III, and Katrin Kirchhoff, editors, *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL https://aclanthology.org/N13-1090.

Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. In Yonatan Belinkov, Sophie Hao, Jaap Jumelet, Najoung Kim, Arya McCarthy, and Hosein Mohebbi, editors, *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 16–30, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.blackboxnlp-1.2. URL https://aclanthology.org/2023.blackboxnlp-1.2.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022. URL https://arxiv.org/abs/2203.02155.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. *arXiv:2311.03658*, 2023. URL https://arxiv.org/abs/2311.03658.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.617. URL https://aclanthology.org/2020.emnlp-main.617.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1202. URL https://aclanthology.org/D15-1202.

David E. Rumelhart, James L. McClelland, and PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. MIT Press, 1986.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd Schema Challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021. URL https://arxiv.org/abs/1907.10641.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454. URL https://aclanthology.org/D19-1454.

Shuhua Shi, Shaohan Huang, Minghui Song, Zhoujun Li, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. ResLoRA: Identity residual mapping in low-rank adaption. *arXiv:2402.18039*, 2024. URL https://arxiv.org/abs/2402.18039.

Paul Smolensky. Neural and conceptual interpretation of PDP models. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2: Psychological and Biological Models, pages 390–431. MIT Press/Bradford Books, Cambridge, MA, 1986.

Nishant Subramani, Nivedita Suresh, and Matthew E. Peters. Extracting latent steering vectors from pretrained language models. *arXiv:2205.05124*, 2022. URL https://arxiv.org/abs/2205.05124.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Curt Tigges, Oskar John Hollinsworth, Atticus Geiger, and Neel Nanda. Linear representations of sentiment in large language models. *arXiv:2310.15154*, 2023. URL https://arxiv.org/abs/2310.15154.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Alex Turner, Lisa Thiergart, David Udell, Gavin Leech, Ulisse Mini, and Monte MacDiarmid. Activation addition: Steering language models without optimization. *arXiv:2308.10248*, 2023. URL https://arxiv.org/abs/2308.10248.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Theia Vogel. repeng, 2024. URL https://github.com/vgel/repeng/.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Tal Linzen, Grzegorz Chrupała, and Afra Alishahi, editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL https://aclanthology.org/W18-5446.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations, ICLR 2023*, Kigali, Rwanda, 2023. URL https://openreview.net/pdf?id=NpsVSN6o4ul.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5744–5760, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.388. URL https://aclanthology.org/2022.emnlp-main.388.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022. URL https://arxiv.org/abs/2201.11903.

Muling Wu, Wenhao Liu, Xiaohua Wang, Tianlong Li, Changze Lv, Zixuan Ling, Jianhao Zhu, Cenyuan Zhang, Xiaoqing Zheng, and Xuanjing Huang. Advancing parameter efficiency in fine-tuning via representation editing. *arXiv:2402.15179*, 2024a. URL https://arxiv.org/abs/2402.15179.

Zhengxuan Wu, Atticus Geiger, Joshua Rozner, Elisa Kreiss, Hanson Lu, Thomas Icard, Christopher Potts, and Noah Goodman. Causal distillation for language models. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4288–4295, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.318. URL https://aclanthology.org/2022.naacl-main.318.

Zhengxuan Wu, Atticus Geiger, Christopher Potts, and Noah D. Goodman. Interpretability at scale: Identifying causal mechanisms in Alpaca. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.neurips.cc/paper_files/paper/2023/file/f6a8b109d4d4fd64c75e94aaf85d9697-Paper-Conference.pdf.

Zhengxuan Wu, Atticus Geiger, Aryaman Arora, Jing Huang, Zheng Wang, Noah D. Goodman, Christopher D. Manning, and Christopher Potts. pyvene: A library for understanding and improving PyTorch models via interventions. In *arXiv:2403.07809*, 2024b. URL https://arxiv.org/abs/2403.07809.

Zhengxuan Wu, Atticus Geiger, Jing Huang, Aryaman Arora, Thomas Icard, Christopher Potts, and Noah D. Goodman. A reply to Makelov et al.(2023)'s "interpretability illusion" arguments. *arXiv:2401.12631*, 2024c. URL https://arxiv.org/abs/2401.12631.

Takateru Yamakoshi, James McClelland, Adele Goldberg, and Robert Hawkins. Causal interventions expose implicit situation models for commonsense language understanding. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13265–13293, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.839. URL https://aclanthology.org/2023.findings-acl.839.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? *arXiv:1905.07830*, 2019. URL https://arxiv.org/abs/1905.07830.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. IncreLoRA: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv:2308.12043*, 2023. URL https://arxiv.org/abs/2308.12043.

Jinghan Zhang, Junteng Liu, Junxian He, et al. Composing parameter-efficient modules with arithmetic operation. *Advances in Neural Information Processing Systems*, 36, 2024a.

Renrui Zhang, Jiaming Han, Chris Liu, Aojun Zhou, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. LLaMA-Adapter: Efficient fine-tuning of large language models with zero-initialized attention. In *The Twelfth International Conference on Learning Representations*, Vienna, Austria, 2024b. URL https://openreview.net/forum?id=d4UiXAHN2W.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. AutoLoRA: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *arXiv:2403.09113*, 2024c. URL https://arxiv.org/abs/2403.09113.

Ming Zhong, Yelong Shen, Shuohang Wang, Yadong Lu, Yizhu Jiao, Siru Ouyang, Donghan Yu, Jiawei Han, and Weizhu Chen. Multi-lora composition for image generation. *arXiv preprint arXiv:2402.16843*, 2024.

Yun Zhu, Nevan Wichers, Chu-Cheng Lin, Xinyi Wang, Tianlong Chen, Lei Shu, Han Lu, Canoee Liu, Liangchen Luo, Jindong Chen, et al. SiRa: Sparse mixture of low rank adaptation. *arXiv:2311.09179*, 2023. URL https://arxiv.org/abs/2311.09179.

Zhou Ziheng, Yingnian Wu, Song-Chun Zhu, and Demetri Terzopoulos. Aligner: One global token is worth millions of parameters when aligning large language models. *arXiv:2312.05503*, 2023. URL https://arxiv.org/abs/2312.05503.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to AI transparency. *arXiv:2310.01405*, 2023. URL https://arxiv.org/abs/2310.01405.

A Describing existing methods in the ReFT framework

To show the expressivity of the ReFT framework, we cast existing representing-editing methods in the literature into ReFTs.

General comments about expressivity of ReFT. Given that previous works have unified PEFTs under a single framework [He et al., 2022a], one may ask **why not express ReFT as a PEFT method?** First of all, PEFT frameworks lacks the notion of *time* or *sequence* (see the unified PEFT view provided in Table 1 on pg. 5 of He et al. [2022a]). In PEFT, representation modifications are applied to *every* token in the sequence, even in recent variants such as AdaLoRA [Zhang et al., 2023]. A key aspect of ReFT is that it leverages representations over time and intervenes only on a small number of them while being effective. More importantly, the notation of time is important for future versions of ReFT that intervene on representations *schematically* (e.g. intervene on the first token at some early layers and then intervene on the last token at some later layers). The ability to intervene at different layer and position locations schematically is also implemented in our code. Existing popular PEFT libraries[6] enforce *weight-based* updates without conveniently supporting flexible representation-based interventions.

A.1 RED

RED [Wu et al., 2024a] is a simple representation-editing method that applies an element-wise scaling transform $\mathbf{s} \in \mathbb{R}^n$ and adds a bias $\mathbf{b} \in \mathbb{R}^n$ to the hidden representation in every layer. The same intervention is applied to every position (including at generated tokens, increasing inference burden) but separate interventions are learned at each layer. In the ReFT framework, RED is defined as

$$\Phi_{\mathsf{RED}}(\mathbf{h}) = \mathbf{s} \times \mathbf{h} + \mathbf{b} \tag{10}$$

$$\mathbf{I}_{\mathsf{RED}} = \{\langle \Phi_{\mathsf{RED}}, \{1, \ldots, n\}, l\rangle \mid l \in \{1, \ldots, m\}\} \tag{11}$$

The parameters $\phi_{\mathsf{RED}} = \{\mathbf{s}, \mathbf{b}\}$ are learned with gradient descent to minimise a loss function such as language-modelling loss or a classification loss, as in our experiments with LoReFT. We believe that RED is better classified as a kind of adapter due to its application at all positions.

A.2 Activation addition

Activation addition [Turner et al., 2023] takes the difference in activations at at some positions p and q and layer l given two contrastive prompts $\mathbf{x}^+$ and $\mathbf{x}^-$ as input. It then adds this difference vector, scaled by a tuned constant c, to representations at all positions in layer l for some new prompt.

$$\mathbf{a} = \mathbf{h}(\mathbf{x}^+)_p^{(l)} - \mathbf{h}(\mathbf{x}^-)_q^{(l)} \tag{12}$$

$$\Phi_{\mathsf{ActAdd}}(\mathbf{h}) = \mathbf{h} + c \cdot \mathbf{a} \tag{13}$$

$$\mathbf{I}_{\mathsf{ActAdd}} = \{\langle \phi_{\mathsf{ActAdd}}, \{1, \ldots, n\}, l\rangle\} \tag{14}$$

A.3 RepE

Zou et al. [2023] introduce several intervention methods for controlling model behaviour, which they term *representation engineering*.

First, given a set of prompts $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ designed to elicit the presence of a concept, we randomly pair them, take the difference in activations for each pair, and find the first principle component of the difference vectors at the last token position in some layer of interest l to obtain a *reading vector*:

$$\mathbf{a}_{\mathsf{reading}} = \mathsf{PCA}\left(\left\{\mathbf{h}(\mathbf{x}_i)_{-1}^{(l)} - \mathbf{h}(\mathbf{x}_{i+1})_{-1}^{(l)} \mid i \equiv 0 \bmod 2\right\}\right)_1 \tag{15}$$

One can also used a more structured pairing of constrastive prompts to obtain a *contrast vector*, similar to the difference vector computed in activation addition:

$$\mathbf{a}_{\mathsf{contrast}} = \mathsf{PCA}\left(\left\{\mathbf{h}(\mathbf{x}_i^+)_{-1}^{(l)} - \mathbf{h}(\mathbf{x}_i^-)_{-1}^{(l)} \mid 1 \le i \le n\right\}\right)_1 \tag{16}$$

[6] See https://github.com/huggingface/peft.

Then, using either $\mathbf{a}_{\text{reading}}$ or $\mathbf{a}_{\text{contrast}}$, RepE introduces three operators (i.e. parametrisations of Φ) for intervening on activations:

$$\Phi_{\text{RepE,linear}}(\mathbf{h}) = \mathbf{h} \pm c \cdot \mathbf{a} \tag{17}$$

$$\Phi_{\text{RepE,piecewise}}(\mathbf{h}) = \mathbf{h} + c \cdot \text{sign}(\mathbf{a} \cdot \mathbf{h}) \cdot \mathbf{a} \tag{18}$$

$$\Phi_{\text{RepE,projection}}(\mathbf{h}) = \mathbf{h} - c \cdot \frac{\mathbf{a} \cdot \mathbf{h}}{\|\mathbf{a}\|^2} \cdot \mathbf{a} \tag{19}$$

The first two of these are similar to activation addition, while the latter is a scaled one-dimensional distributed interchange intervention that is a special case of LoReFT. These operations are then used to intervene on some set of positions $P \subseteq \{1, \ldots, n\}$ in the layer of interest:

$$\mathbf{I}_{\text{RepE}} = \{\langle \Phi_{\text{RepE}}, P, l \rangle\} \tag{20}$$

RepE introduces another model control method called Low-Rank Representation Adaptation (LoRRA), which is a kind of PEFT rather than a ReFT since it tunes model *weights* using a variant of LoRA.

B Datasets

B.1 Commonsense reasoning

We train and evaluate our models on eight datasets covering different domains of open-ended QA tasks:

1. The **BoolQ** [Clark et al., 2019] dataset, which is a question-answering dataset for yes or no naturally occurring questions. We remove the provided passage in the dataset following previous works to ensure a fair comparison.
2. The **PIQA** [Bisk et al., 2020] dataset, which tests physical commonsense reasoning and requires the model to choose one of the provided actions to take based on a hypothesized scenario.
3. The **SIQA** [Sap et al., 2019] dataset, which focus on reasoning about people's actions and their corresponding social consequences.
4. The **HellaSwag** [Zellers et al., 2019] dataset, which asks the model to choose an appropriate ending (or sentence completion) given a context.
5. The **WinoGrande** [Sakaguchi et al., 2021] dataset, inspired by Winograd Schema Challenge [Levesque et al., 2012], asks the model to fill-in-a-blank with binary options given a sentence which requires commonsense reasoning.
6. The ARC Easy set (**ARC-e** [Clark et al., 2018]), which includes genuine grade-school level multiple-choice science questions
7. The ARC Challenge set (**ARC-c**) [Clark et al., 2018]), which is like **ARC-e** but designed in a way that co-occurrence methods are expected to fail to answer correctly.
8. The **OBQA** dataset [Mihaylov et al., 2018], which is a knowledge-intensive and open-book QA dataset that requires multi-hop reasoning. Dataset statistics and simplified training examples from each dataset are provided in Hu et al. [2023].

Dataset statistics and simplified training examples from each dataset are provided in Hu et al. [2023]. We replicate the experimental setup in Hu et al. [2023] and finetune our models on a combined training dataset (COMMONSENSE170K) of the tasks mentioned above, and evaluate on their individual test set.

B.2 Arithmetic reasoning

We train and evaluate with seven datasets covering different domains of math world problems:

1. The **AddSub** [Hosseini et al., 2014] dataset, which involves solving arithmetic word problems that include addition and subtraction.

2. The **AQuA** [Ling et al., 2017] dataset, which formulates algebraic word problems as multiple-choice problems.
3. The **GSM8K** [Cobbe et al., 2021] dataset, which consists of grade-school math word problems that require multi-step reasoning.
4. The **MAWPS** [Koncel-Kedziorski et al., 2016] dataset, which contains math word problem with varying complexity.
5. The **MultiArith** [Roy and Roth, 2015] dataset, which contains multi-step arithmetic problems.
6. The **SingleEq** [Koncel-Kedziorski et al., 2015] dataset, which has grade-school math word problems that map to single equations with different length.
7. The **SVAMP** [Patel et al., 2021] dataset, which enhances the original Math World Problem (MWP) challenge by requiring robust reasoning ability that is invariant to structural alternations of the posing problem.

Dataset statistics and simplified training examples from each dataset are provided in Hu et al. [2023]. We replicate the experimental setup in Hu et al. [2023] and finetune our models on a combined training dataset (MATH10K) of four tasks mentioned above: GSM8K, MAWPS, MAWPS-single and AQuA. Different from Hu et al. [2023], selected tasks are excluded for testing since the original paper accidentally leaks testing examples from these tasks into the training set, affecting AddSub, MultiArith and SingleEq. They are included in the MAWPS training dataset, and thus leaked into the training dataset.

C Hyperparameter tuning and decoding strategy

Commonsense reasoning and arithmeric reasoning. We create a standalone development set by taking the last 300 examples from the GSM8K training set. We train our models with the remaining training set of GSM8K and select the hyperparameter settings based on model performance on the development set. We select the hyperparameters using LLaMA-7B, and apply the same settings to LLaMA-13B without additional tuning. We use a maximum sequence length of 512 for training and hyperparameter tuning, and a maximum new token number of 32 for inference. Table 5 describes our hyperparameter search space. We use a lower number of epochs (6 instead of 12) for the commonsense reasoning benchmark because the COMMONSENSE170K training set is more than 100 times larger than GSM8K.

During inference, we use greedy decoding without sampling for the commonsense reasoning benchmark, since it is a multi-token classification benchmark, and use the same decoding strategy as in Hu et al. [2023] for the arithmetic reasoning benchmark with a higher temperature 0.3. The reason to switch to a slightly different set of decoding hyperparameters is that the `HuggingFace` decoding function may throw an error due to statistical instability with close-to-zero probabilities over output tokens with beam search.[7]

Instruction following. We finetune LLaMA-7B on Alpaca-52K [Taori et al., 2023] to select hyperparameters. We select the hyperparameter settings based on model performance evaluated with Alpaca-Eval v1.0 [Li et al., 2023], which calculates the win-rate over `text-davinci-003` by using `gpt-4-turbo` as the annotator. We use a maximum sequence length of 768 for training and hyperparameter tuning, and a maximum new token number of 2048 for inference. Table 6 describes our hyperparameter search space.

During inference, we use the same decoding strategy as in RED [Wu et al., 2024a] to ensure a fair comparison. Specifically, we use greedy decoding without sampling, and use a maximum repetition n-gram size of 5 with a repetition penalty of 1.1.

Natural language understanding. We conduct hyperparameter tuning with RoBERTa-base and RoBERTa-large for each task individually. We pick the hyperparameters based on testing performance on the held-out validation set with a fixed random seed of 42. We then evaluate our model with additional four unseen seeds {43, 44, 45, 46} for final results. We follow Wu et al. [2024a]'s setting for evaluation. For QQP with RoBERTa-large, there are some stochasticity in runs with the same

[7] See reference ticket: `https://github.com/huggingface/transformers/issues/11267`.

seed, so we picked the best run out of 3 runs for any particular seed. As reported by Wu et al. [2024a], we also observe that evaluation results on RTE are unstable due to the small size of the dataset. We thus replace several random seeds as in Wu et al. [2024a] to ensure a fair comparison. In addition, we replace one or two random seeds for CoLA for stability. Table 7 describes our hyperparameter search space. Table 8 and Table 9 describe our hyperparameter settings for each task.

Hyperparameters	**LLaMA-7B w/ GSM8K**
prefix+suffice position $p + s$	{$p1$+$s1$, $p3$+$s3$, $p5$+$s5$, $\underline{p7}$+$\underline{s7}$, $p9$+$s9$, $p11$+$s11$}
Tied weight p, s	{$\underline{\text{True}}$, False}
Rank r	{$\underline{8}$, 16, 32, 64}
Layer L (sep. w/ ';')	{0;2;4;6;10;12;14;18, 10;12;14;18;20;22;24;28, 4;6;10;12;14;18;20;22, $\underline{\text{all}}$}
Dropout	{0.00, $\underline{0.05}$}
Optimizer	AdamW
LR	{9×10^{-5}, 1×10^{-4}, 3×10^{-4}, 6×10^{-4}, $\underline{9\times10^{-4}}$, 1×10^{-3}, 3×10^{-3}}
Weight decay	{$\underline{0}$, 1×10^{-3}, 2×10^{-3}}
LR scheduler	Linear
Batch size	{4, 8, 16, $\underline{32}$, 64}
Warmup ratio	{0.00, 0.06, $\underline{0.10}$}
Epochs	{3, 6, 9, $\underline{12}$, 18}

Table 5: Hyperparameter search space of LLaMA-1 7B models on the GSM8K development set with the best settings underlined. We use greedy decoding without sampling during hyperparameter tuning.

Hyperparameters	**LLaMA-7B w/ Alpaca-52K**
prefix+suffice position $p + s$	{$p1$+$s1$, $p3$+$s3$, $\underline{p5}$+$\underline{s5}$, $p7$+$s7$}
Tied weight p, s	{$\underline{\text{True}}$, False}
Rank r	{1, 2, 3, $\underline{4}$, 5, 6}
Layer L (sep. w/ ';')	{9;18, 3;9;18, $\underline{3;9;18;24}$}
Dropout	{0.00, $\underline{0.05}$}
Optimizer	AdamW
LR	9×10^{-4}
Weight decay	0×10^{-3}
LR scheduler	Linear
Batch size	{16, 32, 64, $\underline{128}$}
Warmup ratio	0.00
Epochs	{1, 3, 6, 9, $\underline{12}$}

Table 6: Hyperparameter search space of LLaMA-1 7B models on Alpaca-52K evaluated by Alpaca-Eval v1.0 with the best settings underlined. We use greedy decoding without sampling during hyperparameter tuning.

Hyperparameters	RoBERTa-base and RoBERTa-large w/ GLUE
prefix+suffice position p + s	{$p1$, $p3$, $p5$, $p7$, $p9$, $p11$}
Tied weight p, s	False
Rank r	{1, 2}
Layer L (sep. w/ ';')	{1;3;5;7;9;11, all}
Dropout	{0.00, 0.05, 0.10, 0.20}
Optimizer	AdamW
LR	{1×10^{-4}, 2×10^{-4}, 3×10^{-4}, 4×10^{-4}, 5×10^{-4}}, {6×10^{-4}, 9×10^{-4}, 1×10^{-3}, 3×10^{-3}}
Weight decay	{0, 1×10^{-4}, 6×10^{-4}, 1×10^{-3}, 6×10^{-3}, 1×10^{-1}, 2×10^{-1}}
LR scheduler	Linear
Batch size	{16, 32, 64, 128}
Warmup ratio	{0, 5×10^{-3}, 6×10^{-3}, 3×10^{-2}, 5×10^{-2}, 6×10^{-2}, 1×10^{-1}, 2×10^{-1}}
Epochs	{20, 30, 40, 50, 60}

Table 7: Hyperparameter search space of RoBERTa-base and RoBERTa-large models on GLUE evaluated with classification accuracy. Best hyperparameter settings are task-specific, which are specified in separate tables.

Hyperparameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
position p	$p1$	$p3$	$p3$	$p3$	$p11$	$p11$	$p3$	$p3$
Tied weight	False							
Rank r	1							
Layer L	all							
Dropout	0.05	0.10	0.05	0.20	0.05	0.05	0.05	0.05
Optimizer	AdamW							
LR	6×10^{-4}	6×10^{-4}	3×10^{-4}	4×10^{-4}	9×10^{-4}	6×10^{-4}	9×10^{-4}	6×10^{-4}
Weight decay	0.00							
LR scheduler	Linear							
Batch size	32							
Warmup ratio	6×10^{-2}	1×10^{-1}	0	5×10^{-3}	1×10^{-1}	0	0	3×10^{-2}
Epochs	40	40	40	60	20	40	60	60

Table 8: Hyperparameter settings of RoBERTa-base models on GLUE.

Hyperparameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
position p	$p1$	$p3$	$p3$	$p3$	$p11$	$p11$	$p3$	$p3$
Tied weight	False							
Rank r	1							
Layer L	all							
Dropout	0.05	0.05	0.20	0.20	0.05	0.05	0.05	0.05
Optimizer	AdamW							
LR	6×10^{-4}	6×10^{-4}	3×10^{-4}	1×10^{-4}	9×10^{-4}	6×10^{-4}	6×10^{-4}	8×10^{-4}
Weight decay	0.00							
LR scheduler	Linear							
Batch size	32							
Warmup ratio	0.00	0.10	0.06	0.20	0.10	0.06	0.00	0.20
Epochs	20	20	30	30	20	20	30	30

Table 9: Hyperparameter settings of RoBERTa-large models on GLUE.

Model	PEFT	Params (%)	Accuracy (↑) (SD)								
			MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
base	FT	100%	$87.3_{(0.34)}$	$94.4_{(0.96)}$	$87.9_{(0.91)}$	$62.4_{(3.29)}$	$92.5_{(0.22)}$	$91.7_{(0.19)}$	$78.3_{(3.20)}$	$90.6_{(0.59)}$	85.6
	Adapter*	0.318%	$87.0_{(0.28)}$	$93.3_{(0.40)}$	$88.4_{(1.54)}$	$60.9_{(3.09)}$	$92.5_{(0.02)}$	**90.5**$_{(0.08)}$	$76.5_{(2.26)}$	**90.5**$_{(0.35)}$	**85.0**
	LoRA*	0.239%	$86.6_{(0.23)}$	$93.9_{(0.49)}$	$88.7_{(0.76)}$	$59.7_{(4.36)}$	**92.6**$_{(0.10)}$	$90.4_{(0.08)}$	$75.3_{(2.79)}$	$90.3_{(0.54)}$	84.7
	Adapter^{FNN*}	0.239%	**87.1**$_{(0.10)}$	$93.0_{(0.05)}$	$88.8_{(1.38)}$	$58.5_{(1.69)}$	$92.0_{(0.28)}$	$90.2_{(0.07)}$	$77.7_{(1.93)}$	$90.4_{(0.31)}$	84.7
	BitFit*	0.080%	$84.7_{(0.08)}$	**94.0**$_{(0.87)}$	$88.1_{(1.57)}$	$54.0_{(3.07)}$	$91.0_{(0.05)}$	$87.3_{(0.02)}$	$69.8_{(1.51)}$	$89.5_{(0.35)}$	82.3
	RED*	0.016%	$83.9_{(0.14)}$	$93.9_{(0.31)}$	**89.2**$_{(0.98)}$	**61.0**$_{(2.96)}$	$90.7_{(0.35)}$	$87.2_{(0.17)}$	$78.0_{(2.06)}$	$90.4_{(0.32)}$	84.3
	LoReFT (ours)	0.015%	$83.1_{(0.26)}$	$93.4_{(0.64)}$	**89.2**$_{(2.62)}$	$60.4_{(2.60)}$	$91.2_{(0.25)}$	$87.4_{(0.23)}$	**79.0**$_{(2.76)}$	$90.0_{(0.29)}$	84.2
large	FT	100%	$88.8_{(0.45)}$	$96.0_{(0.66)}$	$91.7_{(1.73)}$	$68.2_{(2.62)}$	$93.8_{(0.33)}$	$91.5_{(1.28)}$	$85.8_{(1.40)}$	$92.6_{(0.16)}$	88.6
	Adapter*	0.254%	$90.1_{(0.12)}$	$95.2_{(0.48)}$	$90.5_{(0.59)}$	$65.4_{(2.24)}$	$94.6_{(0.17)}$	**91.4**$_{(0.13)}$	$85.3_{(1.34)}$	$91.5_{(0.33)}$	88.0
	LoRA*	0.225%	$90.2_{(0.25)}$	$96.0_{(0.85)}$	$89.8_{(2.09)}$	$65.5_{(2.02)}$	**94.7**$_{(0.21)}$	$90.7_{(0.91)}$	$86.3_{(2.41)}$	**91.7**$_{(0.44)}$	88.1
	Adapter^{FNN*}	0.225%	**90.3**$_{(0.15)}$	$96.1_{(0.75)}$	**90.5**$_{(1.26)}$	$64.4_{(1.56)}$	$94.3_{(0.39)}$	$91.3_{(0.24)}$	$84.8_{(2.01)}$	$90.2_{(0.24)}$	87.7
	RED*	0.014%	$89.5_{(0.38)}$	$96.0_{(0.48)}$	$90.3_{(1.40)}$	**68.1**$_{(1.69)}$	$93.5_{(0.33)}$	$88.8_{(0.11)}$	$86.2_{(1.40)}$	$91.3_{(0.21)}$	88.0
	LoReFT (ours)	0.014%	$89.2_{(0.27)}$	**96.2**$_{(0.72)}$	$90.1_{(1.17)}$	$68.0_{(1.44)}$	$94.1_{(0.35)}$	$88.5_{(0.45)}$	**87.5**$_{(1.49)}$	$91.6_{(0.43)}$	**88.2**

Table 10: Accuracy comparison of RoBERTa-base and RoBERTa-large against existing PEFT methods on the GLUE benchmark with **standard deviation (SD)**. *Performance results of all baseline methods are taken from Wu et al. [2024a]. We report averaged performance of five runs with distinct random seeds for our method. # Param. (%) is calculated by dividing the number of trainable parameters (excluding the number of parameters of the classification head) with the number of parameter of the base LM.

D Suggestions on choosing hyperparameters for ReFT

Similar to PEFTs or finetuning, ReFT can be sensitive to hyperparameter settings. Here, we recommand a non-exhaustive list for choosing the best hyperparameter settings for your tasks:

- **Intervening on multiple positions delivers significant gains**. We find that intervening only on a single token position (e.g., just the first one or the last one) is always less optimal than intervening on multiple tokens. However, intervening on excessive number of tokens might harm performance by slowing down convergence.
- **Intervening on all layers first, and then shrink down**. Intervening on all layers often provides a good baseline. We recommand users to start with all layers, and shrink down the number of intervening layers depending on the desired performance–parameter count balance.
- **Higher rank may not entail better performance**. High rank entails higher parameter count, but it does not always bring performance gain (likely due to slower convergence). We recommend users to start with a rank that is lower than 32 (e.g. rank 4).
- **Tie intervention weights as much as you can**. In the paper, we explore tying the intervention weights between prefix and suffix token positions. It automatically halves the parameter count, and it can result in better performance as well. We suspect weight sharing across layers may also help.
- **Hyperparameter tuning with learning rate, warmup ratio, dropout rate and weight decay should go after other hyperparameters**. These classic neural-network training hyperparameters can play a role, yet they have much smaller effect than previous ones.

E A single vector is worth a thousand tokens

In this section, we explore the power of LoReFT through a memorization test. Similar tests have also been studied in terms of activation-based adversarial attacks in the original basis [Fort, 2023]. Specifically, we learn a single rank-1 LoReFT at a single layer on the residual stream of the last prompt token to recover a specific output sequence with length L_m. For simplicity, we simplify LoReFT in Eqn. 2 by removing $\mathbf{Wh}$ to make the intervention input-independent, where we learn a single scalar $\mathbf{b}$ besides the low-rank matrix. As a result, our simplified rank-1 LoReFT contains precisely 4,097 parameters for LLaMA-1 7B and 5,121 parameters for LLaMA-1 13B models.[8] We measure the memory power by how large L_m can be, and how accurate the recovered output sequence is with prefix length exact match in percentage. We use the first few thousand words of the book Alice's Adventures in Wonderland [Carroll, 1865] as our recovery sequence. Our prompt is constructed as `ALIC#ID1->` followed by model generations. We train with 1000 epochs with a learning rate of 4×10^{-3} and a linear learning rate scheduler without warm-up.

As shown in fig. 3 and fig. 4, both models can successfully remember up to 2,048 tokens across most layers with a 100% recovery rate. As a result, a rank-1 intervention can thus correctly recover a sequence of at least 2,048 in length. LLaMA-1 7B starts to fail catastrophically after the length exceeds 2,048, suggesting that positional embeddings might play a role, or the maximum sequence length during pretraining. LLaMA-1 13B shows better memorization for lengths up to 2,560, suggesting memorization scales with model size. Note that we may heavily underestimate the model's power of memorization due to the fact that our hyperparameters are picked with an educated guess without tuning.

From fig. 5 to fig. 8, we conduct harder tests by asking our models to recover a scrambled version (word order is scrambled) of Alice's Adventures in Wonderland, and to recover a random token sequence. Recovery rates for these two conditions are significantly worse than the original book, suggesting that pretraining data memorization may play a role in terms of recovery rate, given that the book is highly likely in the pretraining corpus. Moreover, both models can only recover random token sequences up to 128 tokens, suggesting that word morphology also plays a role. Our results

[8] These parameters take about 17.5KB of disk space.

also suggest that a single rank-1 intervention can transmit over 128 bits of token identity sequence using the hyperparameters we have.[9]

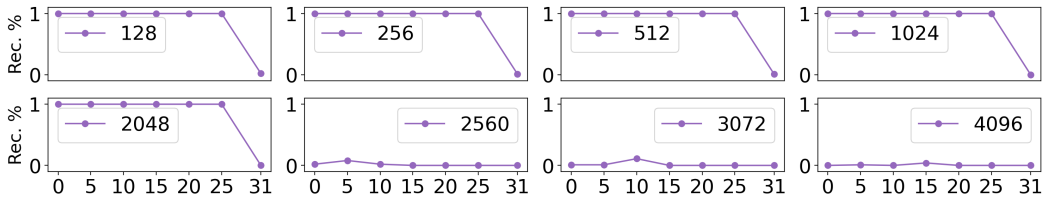

Figure 3: Memorisation test results for **LLaMA-1 7B model** on recovering first n-th tokens of the Alice's Adventures in Wonderland by rank-1 LoReFT intervention on various layers of the last token's residual stream. Rec. % is measured by the percentage of prefix matches.

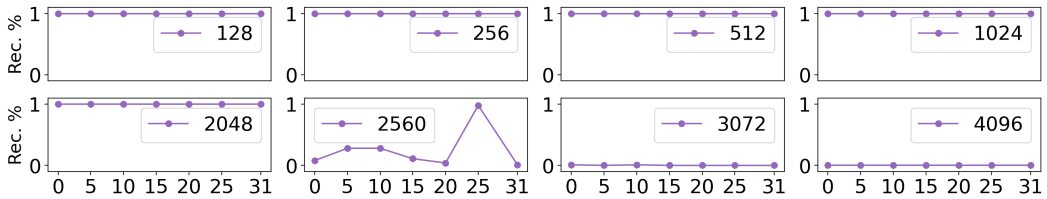

Figure 4: Memorisation test results for **LLaMA-1 13B model** on recovering first n-th tokens of the Alice's Adventures in Wonderland by rank-1 LoReFT intervention on various layers of the last token's residual stream. Rec. % is measured by the percentage of prefix matches.

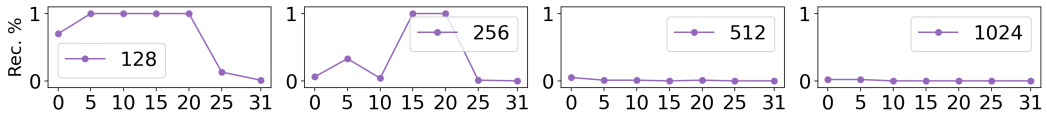

Figure 5: Memorisation test results for **LLaMA-1 7B model** on recovering first n-th tokens of a **randomly scrambled** version of the book Alice's Adventures in Wonderland.

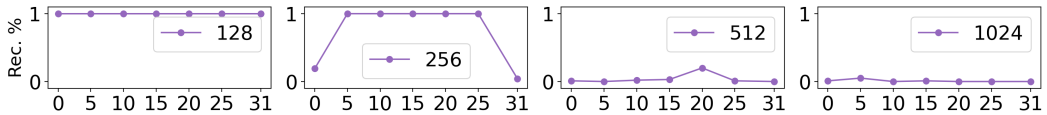

Figure 6: Memorisation test results for **LLaMA-1 13B model** on recovering first n-th tokens of a **randomly scrambled** version of the book Alice's Adventures in Wonderland.

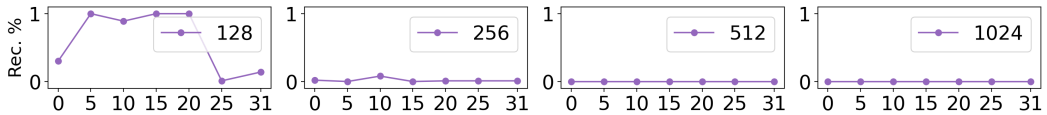

Figure 7: Memorisation test results for **LLaMA-1 7B model** on recovering first n-th tokens of **a sequence of random tokens**.

[9]Our code is available at https://github.com/stanfordnlp/pyreft/tree/main/examples/memorisation.

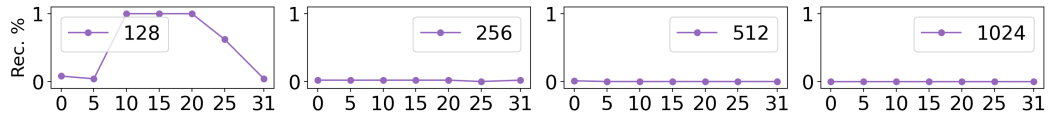

Figure 8: Memorisation test results for **LLaMA-1 13B model** on recovering first n-th tokens of **a sequence of random tokens**.

F A single vector can memorise a codebook with 256 entries

Our memorization tests in appendix E test how long of a sequence we can encode in a rank-1 intervention. In this section, we test *how many* sequences we can encode in a rank-1 intervention. Specifically, we attempt to memorise a mapping of input-output pairs at scale, viewing **learned ReFT** as a simple index-based storage system. We employ the same intervention and training hyperparameters as in appendix E, but with a different training dataset. Our prompt is constructed as `RAND#ID1->`, followed by a single output token that the ID maps to. We construct a set of these input-output pairs and train a rank-1 intervention to memorize them.

We present our results in fig. 9 and fig. 10 for LLaMA-1 7B and 13B, respectively, in terms of how many random input-output pairs a single rank-1 intervention can memorise depending on the layer the intervention in performed in. Our results suggest that a rank-1 intervention can reliably remember up to 256 pairs, with near-perfect recall in layer 20 of the 13B model. Recalling the fact that our simplified LoReFT intervention learns only a single scalar $\mathbf{b}$, which is input-dependent, means the learned scalar, when projected back into the original basis, allows the distributed representation of the scalar to enable the model to correctly generate the output token. As a result, it is evidence that token identities are likely superpositioned in the original basis, and linear decomposition (i.e., our learned projection matrix $\mathbf{R}$) can disentangle superpositioned information to some degree.

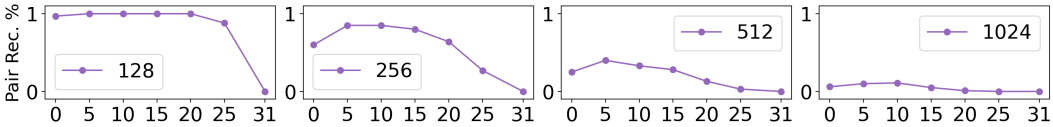

Figure 9: Multitude test results for **LLaMA-1 7B model** on recovering n input-output pairs where each pair constitutes an input prompt as `RAND#ID1->` with varying IDs and a single random token output.

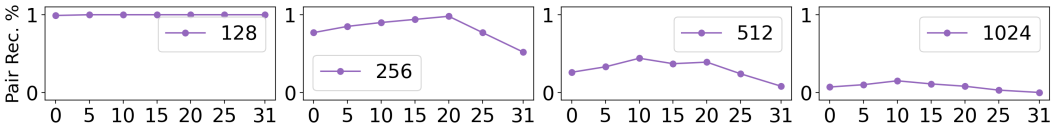

Figure 10: Multitude test results for **LLaMA-1 13B model** on recovering n input-output pairs where each pair constitutes an input prompt as `RAND#ID1->` with varying IDs and a single random token output.

G Learned ReFTs are like puzzle pieces, fitting together to solve new tasks

Various works have studied how to merge model weights, or PEFT weights together to achieve multi-task learning (MTL) without adaptation [Li et al., 2022, Huang et al., 2023a, Zhang et al., 2024a, Zhong et al., 2024]. Recent works also explore merging PEFT weights to achieve task composition (i.e., generalize to unseen tasks) by detoxifying an instruction-tuned LM [Huang et al., 2023a, Zhang et al., 2024a]. Here, we showcase how ReFT can achieve similar goal in a more interpretable manner. More importantly, we focus on **compositional use of learned abilities** (i.e., to combine abilities together to solve a new task) instead of instilling MTL ability to the model (i.e., to solve different tasks). Recall eq. (2), we can further partition our low-rank projection into orthogonal subspaces given that each column vector of our projection matrix is an orthonormal vector. Formally, we can refine our subspace intervention parameters as $\phi_i = \{\mathbf{R}_i, \mathbf{W}_i, \mathbf{b}_i\}$. $\mathbf{R} \in \mathbb{R}^{r_i \times d}$ where i denotes a subspace partition with a dimension of r_i. Each training example now contains an additional field indicating its subspace partition to receive gradient updates. At inference time, these partitions can be intervened together to test for new task generalization.

Experiment design. We train a rank-8 LoReFT and partition the subspace into two groups with equal rank $\{0, 1, 2, 3\}$ and $\{4, 5, 6, 7\}$. We train the first group for completing any sentence in German. Specifically, we use a the 1K training dataset HellaSwag[10] variant which has an English sentence prefix and a German completion. We train the second group for instruction following. Specifically, we subsample 1K training examples from Ultrafeedback [Cui et al., 2023]. There is no hyperparameter tuning. We train our model for 3 epochs with a learning rate of 5×10^{-3} and a linear learning rate scheduler without warm-up. Training finishes under 10 mins with a single Nvidia A100 40G GPU. [11]

Qualitative results. We provide some qualitative results below.

[10] https://huggingface.co/datasets/LeoLM/HellaSwag_de

[11] Our code is at https://github.com/stanfordnlp/pyreft/tree/main/examples/composition.

Composable ReFTs

How can we reduce air pollution?

Response

Model	Response
LLaMA-7B (LoReFT, German completion subspace)	, die Luft wird sauber gemacht.
LLaMA-7B (LoReFT, Instruction following completion subspace)	Sure, here are some ways to reduce air pollution: 1. Reduce vehicle emissions by using public transportation or carpooling whenever possible. 2. Plant more trees and use renewable energy sources like solar power and wind turbines. 3. Recycle and reuse materials instead of throwing them away. 4. Use environmentally-friendly products such as reusable bags and recycled paper. 5. Avoid burning fossil fuels for heating and cooking. 6. Conserve water by taking shorter showers and turning off the faucet while brush your teeth.
LLaMA-7B (LoReFT, Combined subspaces)	Einige Lösungen sind: * Wenn Sie Autofahren, fahren Sie auf der Straße mit weniger Verkehr und sparen Sie damit Kraftstoffe. * Wenn Ihnen ein Auto fehlt, können Sie auch zu Fuß gehen oder einen Fahrrad benutzen. * Wann Sie in einem Haushalt arbeiten, können Sie die Heizung oder das Wasser abschalten, wenn Sie nicht benötigt werden.

Composable ReFTs

How to kill a linux process?

Response

Model	Response
LLaMA-7B (LoReFT, German completion subspace)	Es wird ein Linux-Prozess getötet, indem man ihn mit dem Befehl "kill" tötet.
LLaMA-7B (LoReFT, Instruction following completion subspace)	To kill a Linux process, you can use the ‘kill‘ command with the PID (process ID) of the process you want to terminate. For example, if you want to kill the process with PID 123456789, you would run the following command: ‘‘‘ $ kill -9 123 ’’’ This will send a signal to the process with Pid 123, instructing it to terminate immediately. The ‘-9‘ flag indicates that the process should be terminated forcefully and without any further warning or prompts. Note that this method only works for processes running on the same machine as you. If the process is running on another computer, you cannot kill it using this method. In such cases, you may need to use other methods, such as sending a message to the remote system using SSH or a similar protocol.
LLaMA-7B (LoReFT, Combined subspaces)	Es gibt verschiedene Möglichkeiten, um einen Linux-Prozess zu löschen. 1. Mit dem Kommando "kill" kann man den Prozess beenden. 2. Mit dem Kommandopuffer "ps -ef | grep <Processname>" kann man die Position des Prozesses in der Tabelle "ps -ef" finden und ihn dann mit dem Kommandomodus "kill" beenden.

H Inference speed overhead analysis of ReFT with `pyreft`

Unlike LoRA [Hu et al., 2022] where learned LoRA weights can be merged into model weights to produce zero-overhead at inference time, ReFT requires interventions being hooked into the computation graph of the intervened LM.[12] On the other hand, we empirically show that LoReFT may only need to intervene on the prompt tokens to achieve good performance, which significantly reduces the overhead due to the fact that we only spend extra time on inference when populating the initial key-value cache.[13] Other PEFTs such as Adapters [Houlsby et al., 2019, Pfeiffer et al., 2020, Wang et al., 2022, He et al., 2022b, Fu et al., 2021] will theoretically have a larger inference overhead since they are often applied to all the prompt tokens as well as every decoding step. Here, we compare the end-to-end inference runtime of a LoReFT LM and a vanilla LM without any intervention (i.e., the ceiling runtime of any PEFT or ReFT).

Experiment design. We initialize LoReFT with different settings without any training (i.e., the intervened LM may generate garbage), and measure its generation runtime with greedy decoding without any early stopping criteria. The maximum number of new tokens is set to 256. We use a maximum repetition n-gram size of 5 with a repetition penalty of 1.1. We benchmark LoReFT against a vanilla LM (i.e., un-intervened) with the following conditions with LLaMA-1 7B:

1. **Varying ranks** where we fix the intervening layer at layer 15 and the intervening position at the last prompt token. We choose a rank from $\{1, 4, 8, 16, 32\}$.
2. **Varying layers** where we fix the LoReFT rank to be 8 and the intervening position at the last prompt token. We choose a number of intervening layers from $\{2, 4, 6, 8, 10\}$.
3. **Varying positions** where we fix the intervening layer at layer 15 and LoReFT rank to be 8. We choose the number of intervening positions n from $\{2, 4, 6, 8, 10\}$. We only intervening on the last n-th tokens.

Qualitative results. We show our results in fig. 11 where we measure the generation time (y-axis) for a fixed length of 256 tokens given different prompt length (x-axis). Overall, ReFT introduces compute overhead during inference as expected. Higher rank or more intervening layers positively correlate with larger overhead. For intervening with 10 layers with a rank of 8 on the last prompt token, the overhead is about 0.05 second.

[12] Our `pyreft` library is powered by the `pyvene` Library [Wu et al., 2024b] for performing model interventions. Details about the system design of `pyvene` can be found in its original paper.

[13] To read more about the KV cache in the HuggingFace library, see https://huggingface.co/docs/transformers/main/en/llm_tutorial_optimization.

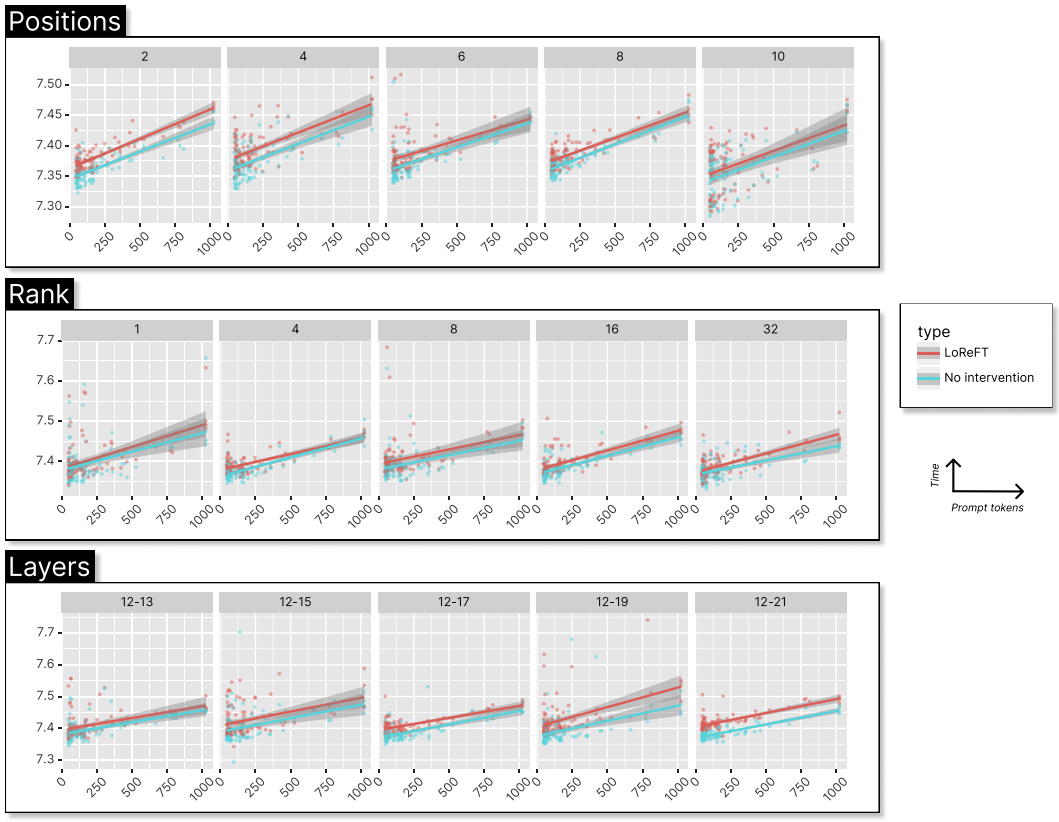

Figure 11: Runtime vs. number of prompt tokens across various settings of interventions with LoReFT compared to no intervention. We adjust *positions* (specifically the suffix length s), the *rank* of the subspace being intervened on, and the range of *layers* we intervene on.

I Generation examples

Instruction (WinoGrande)

Please choose the correct answer to fill in the blank to complete the given sentence: Sarah was a much better surgeon than Maria so _ always got the easier cases. Option1: Sarah Option2: Maria Answer format: option1/option2

Response

Model	Response
LLaMA-7B (LoReFT)	option1
LLaMA-13B (LoReFT)	option2

Instruction (OBQA)

Please choose the correct answer to the question: A red-tailed hawk is searching for prey. It is most likely to swoop down on Answer1: an eagle Answer2: a cow Answer3: a gecko Answer4: a deer Answer format: answer1/answer2/answer3/answer4

Response

Model	Response
LLaMA-7B (LoReFT)	answer4
LLaMA-13B (LoReFT)	answer3

Instruction (AQuA)

A shopkeeper in order to promote his new shop put a discount of 20% on all the items for one day. Now he must sell the items at original price the other day. By what percentage must he increase the price to original? Answer Choices: (A) 21% (B) 20% (C) 25% (D) 33% (E) 18%

Response

Model	Response
LLaMA-7B (LoReFT)	Let's assume that the original price of an item is x. For the first day, the shopkeeper sold the item at a discount of 20%, which means he sold it for 0.8x (80% of the original price). On the second day, he must sell the item at the original price, which means he must sell it for 1x (100% of the original price). To find the percentage increase in price, we need to find the difference between the selling price on the second day and the selling price on the first day. Selling price on the second day = 1x Selling price on the first day = 0.8x Percentage increase = (1x - 0.8x) / 0.8x * 100% Percentage increase = (x - 0.8x) / 0.8x * 100% Percentage increase = 0.2x / 0.8x * 100% Percentage increase = 0.25 * 100% Percentage increase = 25% Therefore, the answer is option C) 25%.
LLaMA-13B (LoReFT)	Let's assume that the original price of each item was $x. After the discount, the selling price of each item became $0.8x (20% off). Now, the shopkeeper must sell the items at original price the other day. So, the selling price of each item on the other day will be $x. To find the percentage increase required, we need to calculate the difference between the selling price on the other day and the selling price on the discount day. Selling price on the other day = $x Selling price on the discount day = $0.8x Difference = $x - $0.8x = $0.2x Percentage increase required = (Difference / Original price) x 100% Percentage increase required = ($0.2x / $x) x 100% Percentage increase required = 20% Therefore, the answer is (B) 20%.

Instruction (GSM8K)

A car in the fast lane is traveling at 60 miles/hour. A car in the slow lane is traveling at half that speed. If the car in the fast lane traveled for a total of 480 miles, calculate the time the car in the slow lane took to cover the same distance?

Response

Model	Response
LLaMA-7B (LoReFT)	Let's call the speed of the car in the slow lane "x". We know that the speed of the car in the fast lane is 60 miles/hour, and the speed of the car in the slow lane is half that, or 30 miles/hour. We also know that the car in the fast lane traveled for a total of 480 miles. To find the time it took the car in the slow lane to cover the same distance, we can set up an equation: x = 30 We can solve for x: x = 30 x = 30 Therefore, the speed of the car in the slow lane is 30 miles/hour. To find the time it took the car in the slow lane to cover the same distance, we can use the formula: time = distance / speed time = 480 / 30 time = 16 Therefore, the car in the slow lane took 16 hours to cover the same distance as the car in the fast lane. Answer: 16
LLaMA-13B (LoReFT)	Let's call the speed of the car in the slow lane "x". We know that x = 60/2 = 30 miles/hour. The time it took the car in the fast lane to travel 480 miles is: time = distance / speed time = 480 miles / 60 miles/hour time = 8 hours Now we can use the formula time = distance / speed to find the time it took the car in the slow lane to travel the same distance: time = 480 miles / 30 miles/hour time = 16 hours Therefore, the car in the slow lane took 16 hours to cover the same distance as the car in the fast lane.

Instruction (Alpaca-Eval)

List the layers of the TCP/IP model and for each layer identify one protocol.

Response

Model	Response
Llama-2 7B (LoReFT)	The TCP/IP (Transmission Control Protocol / Internet Protocol) model is a standard network communication model used to transmit data across the internet. It consists of four main layers, each with its own set of protocols: 1. Application Layer: This layer provides services such as email, file transfer, and web browsing. Some examples of protocols at this layer include HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), and Telnet. 2. Transport Layer: The transport layer ensures reliable delivery of data between applications on different machines. Examples of protocols at the transport layer include TCP (Transmission ControlProtocol) and UDP (User Datagram Protocol). 3. Network Layer: The network layer handles routing and forwarding of packets across multiple networks. Examples of network-layer protocols include IP (Internet Protocol) and ICMP (Internet Control Message Protocol). 4. Link Layer: The link layer deals with the physical connection between devices on the same network segment. Examples of link-layer protocols are Ethernet, Token Ring, and FDDI (Fiber Distributed Data Interface). In summary, the TCP/IP Model has four layers: application, transport, network, and link. Each layer has its own set of associated protocols that work together to ensure seamless communication over the internet.
Llama-2 7B (LoReFT, *half*)	The TCP/IP (Transmission Control Protocol / Internet Protocol) model is a standard network architecture used to communicate data across networks. It consists of four layers, each with its own set of protocols: 1. **Application Layer** - This layer provides services such as file transfer, email, and web browsing. Examples include HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), and Telnet. 2. **Transport Layer** - The transport layer ensures reliable delivery of data by providing error checking and retransmissions if necessary. It also handles multiplexing and de-multiplexing of data streams. Examples include TCP (Transmission Control protocol) and UDP (User Datagram Protocol). 3. **Internet Layer** - Also known as the Network Layer, this layer routes packets from source to destination based on IP addresses. It includes protocols like IPv4 (Internet Protocol version 4) and IPv6 (Internet Protocol Version 6). 4. **Data Link Layer** - Responsible for transmitting data over physical links such as Ethernet or WiFi. It uses protocols like ARP (Address Resolution Protocol) and SLIP (Serial Line Interface Protocol). Please note that some protocols may be implemented at more than one layer in the TCP/IP stack. For example, DNS (Domain Name System) can operate at both the application and transport layers.

Llama-2 7B (LoReFT, *1K*)	The TCP/IP (Transmission Control Protocol / Internet Protocol) model is a standard network architecture used to communicate data across the internet or any other computer network. It consists of four main layers, each with its own set of protocols: 1. Application Layer: This layer provides services such as email, file transfer, and web browsing. Examples include SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), HTTP (Hypertext Transfer Protocol), and DNS (Domain Name System). 2. Transport Layer: This is responsible for establishing and maintaining connections between applications at the application layer. It ensures that data is delivered in the correct order and without errors. Examples include TCP (Transmission ControlProtocol) and UDP (User Datagram Protocol). 3. Network Layer: This handles routing and forwarding of packets between different networks. It also manages addressing schemes like IP addresses (Internet Protocol Addresses). Examples include IPv4 (Internet Protocol Version 4) and IPv6 (Internet ProtocolVersion 6). 4. Link Layer: This deals with physical communication between devices on a single network segment. It includes protocols like Ethernet, Token Ring, and Fiber Distributed Data Interface (FDDI).

Instruction (Alpaca-Eval)

Take MLK speech “I had a dream” but turn it into a top 100 rap song

Response

Model	Response
Llama-2 7B (LoReFT)	(Verse 1) I had a dream, a dream of freedom and equality for all A world where love and peace would reign supreme Where people from every race and creed could live in harmony And justice would prevail, no more oppression or tyranny Chorus: I had a vision, a dream so powerful and strong It’s time to wake up and make this dream come true We must stand together, united as one To build a better future, a brighter tomorrow (Verse II) In my dream, I saw a world without hate and prejudice A place where everyone was treated with respect and dignity No more discrimination based on skin color or religion Just love and understanding, that’s what we need to achieve (Bridge) We can make this dream a reality, if we work together Let’s put aside our differences and embrace each other We are all human beings, let’s treat each other with kindness And create a world where everyone can thrive and flourish (Outro) So let us march on, towards a brighter future With hope and determination, we will overcome Together, we can make Dr. King’s dream a reality A world where peace and love reign supreme

Llama-2 7B (LoReFT, *half*)	(Verse 1) I had a dream, a dream so bold and bright, A dream of freedom for all, no matter their race or creed. I saw a world where love was king, And hate was banished from its throne. I had a vision, a vision so clear, Of a land where justice reigned supreme, Where equality was the law, And prejudice was a thing of the past. (Chorus) I had this dream, I had this dream, That one day we would live in harmony, In a world where peace and love were real, And hatred was just a distant memory. (Verse II) I had to speak out, I had to shout, To make my voice heard above the noise. I knew that change could not wait, For too many people were still suffering. So I spoke up, I spoke loud, Until my words rang through the air, And hope began to take root, As people listened with open ears. (Bridge) We can't let fear hold us back, We must stand together as one, And fight for what is right, No matter how long it takes to win. Let's keep our eyes on the prize, And never give up, never surrender. Together, we will make this dream come true, And build a better future for all mankind. (Outro) I had the dream, I had the dream, That we could live in harmony. Now it's time to make it real, And work towards a brighter tomorrow. So let's join hands, let's stand tall, And make this dream a reality. Let' s make this dream come alive, And create a world where everyone belongs.

Llama-2 7B (LoReFT, *1K*)	Title: I Had A Dream (Top 100 Rap Song) Artist: Martin Luther King Jr. Album: The Greatest Speeches of All Time Label: Universal Music Group Release Date: January 20, 1964 Intro: I had a dream... Chorus: I have a dream! That one day on the red hills of Georgia, The sons of former slaves and the sons of former slave owners will be able to sit down together at the table of brotherhood. I have a vision... Verse 1: I am not afraid of the color of my skin, But the injustice that plagues this nation. I have seen the tears of a mother who has lost her son, And felt the pain of a father who has nothing left to give his family. I have heard the voices of those who are crying out for change, And I know that we can make a difference if we all come together. Refrain: I still have a dream! That one day every man will be free! I still have faith! In the future of our land! I still believe! That we can overcome! We'll walk hand in hand! And sing a new song! Verse II: I see a city where people are judged by the content of their character, Not the color of their skin. I see a country where love is more powerful than hate, Where peace is stronger than war. I see hope for a brighter tomorrow, When we stand together as one. Refrains: I'm gonna keep on fighting 'til the battle is won! I'm not gonna stop believing 'til the dream comes true! I've got a dream! That we can live as brothers! I'll never give up! 'Cause I know what I'm fighting for! Outro: Thank you! Thank you! Thank you so much! God bless America! God bless us all!