When Do Prompting and Prefix-Tuning Work? A Theory of Capabilities and Limitations

Aleksandar Petrov, Philip H.S. Torr & Adel Bibi
University of Oxford
Oxford, United Kingdom
{aleksandar.petrov,philip.torr,adel.bibi}@eng.ox.ac.uk

Abstract

Context-based fine-tuning methods like prompting, in-context learning, soft prompting (prompt tuning) and prefix-tuning have gained popularity as they often match the performance of full fine-tuning with a fraction of the parameters. Still, there is little theoretical understanding of how these techniques influence the internal computation of the model and their expressiveness limitations. We show that despite the continuous embedding space being more expressive than the discrete token space, soft-prompting and prefix-tuning are strictly less expressive than full fine-tuning. Concretely, context-based fine-tuning cannot change the relative attention pattern over the content and can only bias the outputs of an attention layer in a fixed direction. While this means that context-based fine-tuning techniques can successfully elicit or combine skills already present in the pretrained model, they cannot learn tasks requiring new attention patterns.

1 Introduction

Language model advancements are largely driven by larger models and amounts of training data (Kaplan et al., 2020; Rae et al., 2021). Training cutting-edge models is out of reach for most academic researchers, small enterprises, and individuals, and it has become common to instead fine-tune open-source pretrained models (Devlin et al., 2019; Min et al., 2021). However, due to escalating computational demands, even fine-tuning is becoming prohibitively expensive (Lialin et al., 2023).

As a result, there is an acute need for more efficient fine-tuning methods, either by sparsely modifying the parameters of the model or modifying its input context. An example of the first type are adapter modules which introduce a few trainable layers to modify the behaviour of the frozen pretrained network (Rebuffi et al., 2017; Houlsby et al., 2019; Hu et al., 2023). One can also use low-rank updates, which also results in a reduced number of trainable parameters (Hu et al., 2021).

Context-based fine-tuning has been motivated by the success of few-shot and zero-shot learning (Wei et al., 2021; Kojima et al., 2022). The most popular context-based approach is prompting, where generation is conditioned on either human-crafted or automatically optimized tokens (Shin et al., 2020; Liu et al., 2023). In-context learning —prompting via providing input-label examples— is another widely used technique (Brown et al., 2020). Given the challenges of discrete optimization over tokens, there is a growing interest in methods that optimize real-valued embeddings (Lester et al., 2021). It is widely believed that these *soft prompts* offer greater expressiveness due to the expansive nature of continuous space. Furthermore, beyond only optimizing input embeddings, one can optimize the inputs of every attention layer (Li and Liang, 2021). This technique, *prefix-tuning*, has proven to be very successful and competitive to full fine-tuning (Liu et al., 2022).

While context-based fine-tuning approaches have witnessed impressive empirical successes and widespread adoption, we have little theoretical understanding of how they work. In this work, we

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

offer a mechanistic explanation of how prompts and prefixes influence the internal computations of a pretrained model and delineate their limitations. Specifically, we address the following questions:

1. We show by construction that, given careful choice of the transformer weights, controlling a single virtual token can generate any of the V^N completions of N tokens, while controlling a single token can produce only V completions, with V being the vocabulary size. Therefore, a transformer can indeed utilize the additional capacity of the embedding space.
2. Despite the expressiveness of continuous space, prefix-tuning has structural limitations. A prefix cannot change the relative attention over the content tokens and can only bias the output of the attention block in a constant direction. In contrast, full fine-tuning can learn new attention patterns and arbitrarily modify attention block outputs, making it strictly more powerful.
3. We show that the prefix-induced bias can steer the model towards a pretraining task and *some* new tasks similar to pretraining tasks. Hence, while context-based fine-tuning is structurally unable to learn all new skills, it can elicit or combine pretrained model skills.

2 Background

Assume vocabulary size V and input sequence $(\mathtt{x}_1, \ldots, \mathtt{x}_p)$, $\mathtt{x}_i \in \{1, \ldots, V\}$. Each token is mapped to a d_e-dimensional vector that is the $\mathtt{x}_i$-th column of an embedding matrix $\boldsymbol{E} \in \mathbb{R}^{d_e \times V}$. For a model with maximum input length N (*context size*), we use a one-hot position encoding $\boldsymbol{e}_N(i)$ concatenated with the embedding, resulting in $\boldsymbol{x}_i = [\boldsymbol{E}_{:,\mathtt{x}_i}^\top, \boldsymbol{e}_N^\top(i)]^\top$. A transformer has attention blocks operating across the whole sequence and Multi-Layer Perceptrons (MLPs) operating on each individual element. An attention block consists of H heads. A head h is parameterized by query, key, and value matrices $\boldsymbol{W}_Q^h, \boldsymbol{W}_K^h \in \mathbb{R}^{k \times d_{\text{in}}}$, $\boldsymbol{W}_V^h \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$.[1] The attention matrix $\boldsymbol{A}^h \in \mathbb{R}^{p \times p}$ for head h then has elements

$$\boldsymbol{A}_{ij}^h = \frac{\exp\left(T/\sqrt{k}(\boldsymbol{W}_Q^h \boldsymbol{x}_i)^\top (\boldsymbol{W}_K^h \boldsymbol{x}_j)\right)}{\sum_{r=1}^p \exp\left(T/\sqrt{k}(\boldsymbol{W}_Q^h \boldsymbol{x}_i)^\top (\boldsymbol{W}_K^h \boldsymbol{x}_r)\right)}, \tag{1}$$

where $p \leq N$ is the current length of the input and $T > 0$ is an inverse temperature parameter.[2] The output of the attention block $\mathcal{A}$ with H heads is the sum of the attention-weighted values:

$$\mathcal{A}[(\boldsymbol{W}_Q^1, \ldots, \boldsymbol{W}_Q^H), (\boldsymbol{W}_K^1, \ldots, \boldsymbol{W}_K^H), (\boldsymbol{W}_V^1, \ldots, \boldsymbol{W}_V^H)](\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p) = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_p),$$
$$\boldsymbol{t}_i = \textstyle\sum_{h=1}^H \sum_{j=1}^p \boldsymbol{A}_{ij}^h \boldsymbol{W}_V^h \boldsymbol{x}_j. \tag{2}$$

A transformer then applies an MLP to each output of an attention block before passing them to next attention block. We will consider linear layers $\mathcal{L}[\boldsymbol{M}, \boldsymbol{b}](\boldsymbol{x}) = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{b}$ and ReLU-activated linear layers $\hat{\mathcal{L}}[\boldsymbol{M}, \boldsymbol{b}](\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{M}\boldsymbol{x} + \boldsymbol{b})$. We use the *then* operator $\fatsemi$ for left-to-right function composition. Therefore, a transformer model predicting confidences over the vocabulary can be represented as:

$$(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p) = \left(\mathcal{A}_1 \fatsemi \hat{\mathcal{L}}_{1,1} \fatsemi \mathcal{L}_{1,2} \fatsemi \mathcal{A}_2 \fatsemi \hat{\mathcal{L}}_{2,1} \fatsemi \mathcal{L}_{2,2} \fatsemi \mathrm{softmax}\right) \left(\begin{bmatrix} \boldsymbol{E}_{:,\mathtt{x}_1} \\ \boldsymbol{e}_N(1) \end{bmatrix}, \ldots, \begin{bmatrix} \boldsymbol{E}_{:,\mathtt{x}_p} \\ \boldsymbol{e}_N(p) \end{bmatrix} \right), \tag{3}$$

with the dimension of the last layer being V. The next token is $\mathtt{x}_{p+1} = \arg\max_{u \in 1,\ldots,V} \boldsymbol{y}_{p,u}$. Thus, a sequence conditional on user input is denoted as $(\mathtt{S}_1, ..., \mathtt{S}_{n_S}, \mathtt{X}_1, ..., \mathtt{X}_{n_X}, \mathtt{Y}_1, ..., \mathtt{Y}_{n_Y})$, with a system prompt $\mathtt{S}$, user provided input $\mathtt{X}$ and autoregressively generated response $\mathtt{Y}$.

Prompting. The most frequently used content-based fine-tuning approach is *prompting*: prefixing the input $(\mathtt{X}_1, ..., \mathtt{X}_{n_X})$ with a token sequence $S \in \{1, ..., V\}^{n_S}$ to guide the model response: $(\mathtt{S}_1, ..., \mathtt{S}_{n_S}, \mathtt{X}_1, ..., \mathtt{X}_{n_X})$. This is how most people interact with language models such as ChatGPT.

Soft prompting. Soft prompting replaces the embeddings of the system input $\boldsymbol{E}_{:,\mathtt{S}_i}$ with learned vectors $\boldsymbol{s}_i \in \mathbb{R}^{d_e}$ called *virtual tokens* (Hambardzumyan et al., 2021; Lester et al., 2021; Qin and Eisner, 2021). Hence, the input in Equation (3) is modified to be:

$$\left(\begin{bmatrix} \boldsymbol{s}_1 \\ \boldsymbol{e}_N(1) \end{bmatrix}, \ldots, \begin{bmatrix} \boldsymbol{s}_{n_S} \\ \boldsymbol{e}_N(n_S) \end{bmatrix}, \begin{bmatrix} \boldsymbol{E}_{:,\mathtt{X}_1} \\ \boldsymbol{e}_N(n_S + 1) \end{bmatrix}, \ldots, \begin{bmatrix} \boldsymbol{E}_{:,\mathtt{X}_{n_X}} \\ \boldsymbol{e}_N(n_S + n_X) \end{bmatrix} \right) \tag{4}$$

with $\boldsymbol{s}_i$ chosen to maximize the likelihood of a target response $Y = (\mathtt{Y}_1, ..., \mathtt{Y}_{n_Y})$, i.e., $\arg\max_{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{n_S} \in \mathbb{R}^{d_e}} \sum_{j=1}^{n_Y} \log \boldsymbol{y}_{n_S + n_X + j, \mathtt{Y}_j}$, where $\boldsymbol{y}_{n_S + n_X + j}$ are autoregressively generated.

[1] For the first block, d_{in} must be $d_e + N$ but may be different for the deeper blocks.

[2] A causal model has $A_{ij} = 0$ for $j > i$. This does not affect our results so we will skip the masking step.

Prefix-tuning. Prefix-tuning applies soft prompting across the depth of the model (Li and Liang, 2021; Liu et al., 2022). The first n_S positions for all attention blocks are set as learnable parameters, replacing the input $(\boldsymbol{x}_1^l, ..., \boldsymbol{x}_{n_X}^l)$ for layer l with $(\boldsymbol{s}_1^l, ..., \boldsymbol{s}_{n_S}^l, \boldsymbol{x}_1^l, ..., \boldsymbol{x}_{n_X}^l)$. Hence, prefix-tuning can be formulated as: $\arg\max_{\{\boldsymbol{s}_i^1,...,\boldsymbol{s}_i^L\}_{i=1}^{n_S}} \sum_{j=1}^{n_Y} \log \boldsymbol{y}_{n_S+n_X+j,\mathtt{Y}_j}$. Prefix-tuning has been very successful for fine-tuning models (Vu et al., 2022; Wu and Shi, 2022; Choi and Lee, 2023; Ouyang et al., 2023; Bai et al., 2023), leading to calls for language models provided as a service (La Malfa et al., 2023) to allow providing prefixes instead of token-based prompts (Sun et al., 2022).

Any token-based prompt $(\mathtt{S}_1, ..., \mathtt{S}_{n_S})$ has a corresponding soft prompt ($\boldsymbol{s}_i$=$\boldsymbol{E}_{:,\mathtt{S}_i}$). Similarly, every soft prompt $(\boldsymbol{s}_1, ..., \boldsymbol{s}_{n_S})$ can be represented as a prefix by setting the deeper prefixes to be the values that the model would compute ($\boldsymbol{s}_i^l$=$(\mathcal{A}_1 \fatsemi ... \fatsemi \mathcal{L}_{l-1})([\boldsymbol{s}_1^\top, \boldsymbol{e}_N^\top(1)]^\top, ..., [\boldsymbol{s}_l^\top, \boldsymbol{e}_N^\top(l)]^\top)$). A hierarchy emerges: *prompting* $<$ *soft prompting* $<$ *prefix-tuning*, with prefix-tuning the most powerful of the three. Hence, we focus on prefix-tuning but our findings hold for prompting and soft prompting.

3 Soft prompting has more capacity than prompting

The success of soft prompting (and by extension, prefix-tuning) is commonly attributed to the larger capacity of the uncountably infinite embedding space compared to the finite token space. Yet, increased capacity from this larger domain is beneficial only if the model can utilize it. This section confirms that this is indeed the case by constructing a transformer that can generate exponentially more completions by varying a single virtual token than by varying a hard token.

Theorem 1 (Conditional generation capacity for a single virtual token (n_X=n_Y=1))**.** *For any V>0, there exists a transformer with vocabulary size V, context size N=2, embedding size d_e=V, one attention layer with two heads and a three-layer MLP that reproduces any map m:$[1, ..., V]$→$[1, ..., V]$ from a user input token to a model response token when conditioned on a single virtual token $\boldsymbol{s}_1$=$\left(m(1)/V, ..., m(V)/V\right)$. That is, by selecting $\boldsymbol{s}_1$ we control the model response to any user input.*

Theorem 1 shows that soft prompting is more expressive for governing the conditional behavior of a transformer model. This also holds for longer responses $n_Y > 1$ by increasing the length of the soft prompt, or longer user inputs $n_X > 1$, by increasing the depth of the model. We provide explicit constructions in Appendix A and working Python implementations.

4 Prefix-tuning can only bias the output of an attention head

We just saw that soft prompting and prefix-tuning can fully control the conditional behavior of a transformer. However, that assumed a specific design for the network weights. Hence, if we have a fixed pretrained model, rather than a carefully crafted one, is prefix-tuning still as powerful, and specifically, is it as powerful as full fine-tuning? In this section we show that a prefix S cannot change the relative attention over the content X, Y and can only bias the attention block outputs in a subspace of rank n_S, the length of the prefix, making it strictly less powerful than full fine-tuning.

Prefix-tuning cannot change the attention pattern of an attention head while full fine-tuning can. Recall the attention $\boldsymbol{A}_{ij}$ position i gives to position j for a trained transformer (Equation (1)):

$$\boldsymbol{A}_{ij} = \frac{\exp\left(T/\sqrt{k}\ \boldsymbol{x}_i^\top \boldsymbol{W}_Q^\top \boldsymbol{W}_K \boldsymbol{x}_j\right)}{\sum_{r=1}^{p} \exp\left(T/\sqrt{k}\ \boldsymbol{x}_i^\top \boldsymbol{W}_Q^\top \boldsymbol{W}_K \boldsymbol{x}_r\right)} = \frac{\exp\left(T/\sqrt{k}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{x}_j\right)}{\sum_{r=1}^{p} \exp\left(T/\sqrt{k}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{x}_r\right)}, \tag{5}$$

where $\boldsymbol{W}_Q^\top \boldsymbol{W}_K$=$\boldsymbol{H}$. Full fine-tuning can enact arbitrary changes to $\boldsymbol{W}_Q$ and $\boldsymbol{W}_K$ and hence, assuming the input does not change (e.g., at the first attention layer), we get the following attention:

$$\boldsymbol{A}_{ij}^{\text{ft}} = \frac{\exp\left(T/\sqrt{k}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{x}_j + T/\sqrt{k}\ \boldsymbol{x}_i^\top \Delta\boldsymbol{H} \boldsymbol{x}_j\right)}{\sum_{r=1}^{p} \exp\left(T/\sqrt{k}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{x}_r + T/\sqrt{k}\ \boldsymbol{x}_i^\top \Delta\boldsymbol{H} \boldsymbol{x}_r\right)},$$

where the changes to $\boldsymbol{W}_Q$ and $\boldsymbol{W}_K$ are folded into $\Delta\boldsymbol{H}$. It is clear that by varying $\Delta\boldsymbol{H}$ full fine-tuning can change the attention patterns arbitrarily. However, let us see how is attention affected by the presence of a prefix. For now, assume we have a prefix of length one ($\boldsymbol{s}_1$) at position 0.

$$\boldsymbol{A}_{i0}^{\text{pt}} = \frac{\exp\left(\frac{T}{\sqrt{k}}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{s}_1\right)}{\exp\left(\frac{T}{\sqrt{k}}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{s}_1\right) + \sum_{r=1}^{p} \exp\left(\frac{T}{\sqrt{k}}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{x}_r\right)},\ \boldsymbol{A}_{ij}^{\text{pt}} = \frac{\exp\left(\frac{T}{\sqrt{k}}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{x}_j\right)}{\exp\left(\frac{T}{\sqrt{k}}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{s}_1\right) + \sum_{r=1}^{p} \exp\left(\frac{T}{\sqrt{k}}\ \boldsymbol{x}_i^\top \boldsymbol{H} \boldsymbol{x}_r\right)} \text{ for } j \geq 1.$$

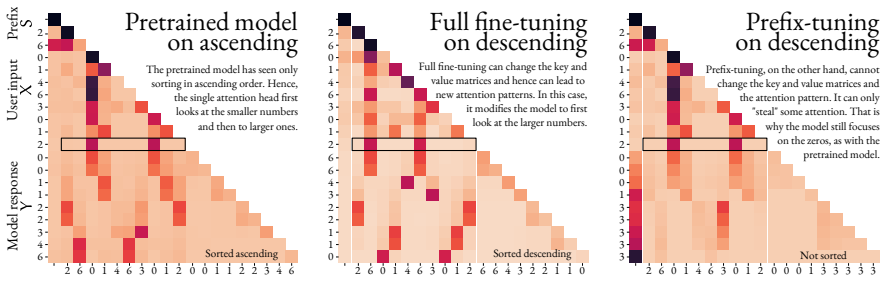

Figure 1: Attention patterns of the head of a small transformer pretrained on sorting in ascending order. The model is provided the prefix S and user input X and generates Y. Full fine-tuning successfully sorts in descending order but prefix-tuning cannot as it cannot change the learned attention.

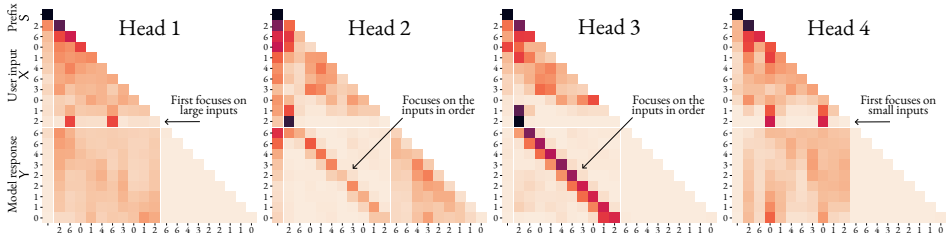

Figure 2: The heads pretrained on the four tasks specialize in the skills necessary to solve them.

The attention position i gives to the content positions $j \geq 1$ is simply scaled down by the attention it now gives to the prefix. Prefix-tuning cannot affect the relative attention patterns across the content, it will only scale them down. This becomes evident by rewriting $\boldsymbol{A}^{\text{pt}}_{ij}$:

$$\boldsymbol{A}^{\text{pt}}_{ij} = \boldsymbol{A}_{ij} \sum_{j=1}^{p} \boldsymbol{A}^{\text{pt}}_{ij} = \boldsymbol{A}_{ij}(1 - \boldsymbol{A}^{\text{pt}}_{i0}). \tag{6}$$

Prefix-tuning only adds a bias to the attention block output. Let's see how this attention scaling down affects the output of the attention block. Following Equation (2), the output at position i for the pretrained ($\boldsymbol{t}_i$), the fully fine-tuned ($\boldsymbol{t}^{\text{ft}}_i$) and the prefix-tuned ($\boldsymbol{t}^{\text{pt}}_i$) models are as follows:[3]

$$\begin{aligned} \boldsymbol{t}_i &= \textstyle\sum_{j=1}^{p} \boldsymbol{A}_{ij} \boldsymbol{W}_V \boldsymbol{x}_j, \qquad & \boldsymbol{t}^{\text{ft}}_i &= \textstyle\sum_{j=1}^{p} \boldsymbol{A}^{\text{ft}}_{ij} (\boldsymbol{W}_V + \Delta \boldsymbol{W}_V) \boldsymbol{x}_j, \\ \boldsymbol{t}^{\text{pt}}_i &= \boldsymbol{A}^{\text{pt}}_{i0} \boldsymbol{W}_V \boldsymbol{s}_1 + \textstyle\sum_{j=1}^{p} \boldsymbol{A}^{\text{pt}}_{ij} \boldsymbol{W}_V \boldsymbol{x}_j \overset{(6)}{=} \boldsymbol{A}^{\text{pt}}_{i0} \boldsymbol{W}_V \boldsymbol{s}_1 + \textstyle\sum_{j=1}^{p} \boldsymbol{A}_{ij} (1 - \boldsymbol{A}^{\text{pt}}_{i0}) \boldsymbol{W}_V \boldsymbol{x}_j = \boldsymbol{A}^{\text{pt}}_{i0} \boldsymbol{W}_V \boldsymbol{s}_1 + (1 - \boldsymbol{A}^{\text{pt}}_{i0}) \boldsymbol{t}_i. \end{aligned} \tag{7}$$

Therefore, prefix-tuning can only bias the output of the attention block towards the constant vector $\boldsymbol{W}_V \boldsymbol{s}_1$, which is independent of the content $(\boldsymbol{x}_1, ..., \boldsymbol{x}_p)$. The content only affects the scale $\boldsymbol{A}^{\text{pt}}_{i0}$ of the bias via the amount of attention on the prefix. This is in contrast with full fine-tuning where $\boldsymbol{W}_V$ can change arbitrarily, allowing for a content-dependent change of the value computation. We show that this also happens for real world transformers in Figures 5 and 6. Appendix B shows that longer prefixes allow for the bias to be coming from a larger subspace but that subspace is not fully utilized.

So, is prefix-tuning equivalent to full fine-tuning or is it less powerful than full fine-tuning? The constructions for the results in Section 3 steer the transformer precisely by acting as a bias to determine which token to generate. Therefore, the observations in this section do not contradict them. Soft prompting and prefix-tuning can be on par with full fine-tuning but in limited circumstances: when all the knowledge is represented in the virtual token as a lookup table and the model simply extracts the relevant entry. Transformers do not behave like this in practice. Moreover, if we had a lookup table of the responses to each input we would not need a learning algorithm in the first place.

5 When and why prefix-tuning can work?

Prefix-tuning cannot change the content attention and only acts as a bias to the attention block, making it strictly less expressive than full fine-tuning. To explain the prior empirical successes, we show

[3]He et al. (2021a) show a similar analysis but do not study the expressiveness of prefix-tuning.

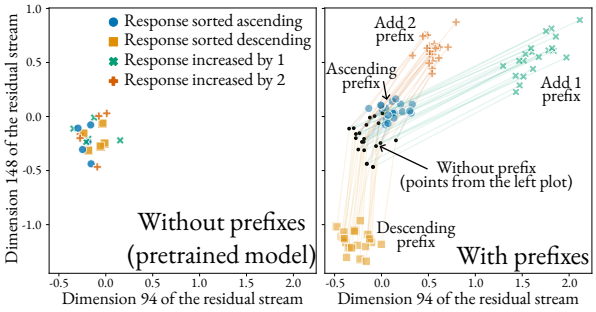

Figure 3: Attention block activations at the last input position (10) when pretrained on the four tasks. The left plot shows the pretrained activations $\boldsymbol{t}_{10}$ are not predictive of the completion. The right plot shows prefixes cluster the activations $\boldsymbol{t}^{\text{pt}}_{10}$. Connecting the pretrained and prefixed activations highlights the bias. No dimensionality reduction is used; the clustering is due to the prefixes.

that this bias can be sufficient for some fine-tuning tasks. Additionally, Appendix C discusses the non-linear effects of the prefix on the deeper layers, which also contribute to additional expressivity. Pretraining exposes a model to different classes of completions for the same token sequence, e.g., text completion, sentiment analysis, or translation. Hence, following our results from Section 4, we hypothesise that prefix-tuning cannot be used to gain *new knowledge* but can elicit *latent knowledge* present in the pretrained model.[4] We test this via small transformers (Karpathy, 2020).

Prefix-tuning cannot learn a new task requiring a different attention pattern. To check if prefix-tuning can learn a new task, we train a 1-layer, 1-head transformer to sort numbers into ascending order and then fine-tune it to sort in descending order. The model sees random sequences of 10 digits from 0 to 7 followed by them sorted in ascending order. The test accuracy of the pretrained model is 91.41%. Full fine-tuning on the descending task has 84.91% test accuracy, hence full fine-tuning successfully learns the new task. However, prefix-tuning (n_S=1) results in 0% test accuracy, hence prefix-tuning fails to learn the new task at all.

Table 1: A transformer pretrained on sorting in ascending order cannot be prefix-tuned to sort in descending order.

	Ascending	Descending
Pretrain on asc.	91.41%	0.00%
Full fine-tune on desc.	0.00%	84.91%
Prefix-tune on desc.	0.09%	0.00%

The attention patterns in Figure 1 show why this is the case: the pretrained model learns to attend first to the smallest numbers and then to the larger ones. When fully fine-tuned, the attention patterns are reversed: they now first attend to the largest values. However, following Section 4, prefix-tuning cannot change the attention pattern over the input sequence and will still attend to the smallest values. Therefore, prefix-tuning indeed cannot learn a new task if it requires new attention patterns.

Prefix-tuning can elicit a skill from the pretrained model. The second part of our hypothesis was that prefix-tuning can elicit latent skills in the pretrained model. We test it by pretraining a 1-layer, 4-head model with solutions sorted in ascending or descending order, or adding one or two to each element of the input sequence. Each solution is shown with 25% probability. We provide no indication of what the task is, leading to test accuracy between 15% and 32% for all tasks. Full fine-tuning for each task naturally results in high accuracy. However, prefix-tuning can also reach very high accuracy for all tasks: above 83%. Compared to the previous case, prefix-tuning is more successful here because the pretrained model contains the attention mechanisms for solving the four tasks (Figure 2).

Table 2: A transformer pretrained on several tasks can be prefix-tuned for one of them.

	Asc.	Desc.	Add 1	Add 2
Pretrain on random	31.4%	19.2%	30.0%	15.6%
Prefix-tune on asc.	94.1%	0.0%	0.0%	0.0%
Prefix-tune on desc.	0.0%	83.7%	0.2%	0.7%
Prefix-tune on add 1	0.0%	0.0%	99.6%	0.0%
Prefix-tune on add 2	0.0%	0.0%	0.2%	99.8%

If all a prefix does is bias the attention layer activations in some direction, how can it steer the model towards a single task? This is likely due to the attention block placing the solutions for all tasks in different subspaces of the residual stream (Elhage et al., 2021). As the MLP needs to select one of these solutions to generate, a further indicator on the selected task (or lack of selection thereof) should also be represented. The bias induced by the prefix then acts on this "selection subspace" to nudge the MLP to select the solution of the desired task. This can be seen from the activations of the attention layer at the last input position ($\texttt{X}_{n_X}$) where the task selection happens. The activations of the pretrained model show no correlation with the output it generates, demonstrating that the choice of completion is not determined by the activation block (Figure 3). However, the prefix-tuned

[4] A similar hypothesis has also been proposed by Reynolds and McDonell (2021) for fine-tuning in general.

activations for the same inputs are clustered as a result of the prefix-induced bias. Hence, the bias acts as a "task selector" of the subspace of the residual stream specializing in the desired task.

Prefix-tuning can combine knowledge from pretraining tasks to solve new tasks. Prefix-tuning eliciting one type of completion learned in pretraining starts to explain its practical utility. Still, it seems to be successful also at tasks that the pretrained model has not seen. This can happen if the "skill" required to solve the new task is a combination of "skills" of the pretrained model.

We pretrain a 4-layer 4-head model with the same tasks and prefix-tune (n_S=10) for incrementing the ascending sorted sequence by 1. The pretrained model has never seen such a task but prefix-tuning results in 34.5% accuracy. Hence, it successfully combines two pretraining skills to solve a novel task. Still, prefix-tuning for a new task is more challenging to optimize than for a pre-training task: 25 times more iterations for a third of the accuracy. Hence, the ease of prefix-tuning can be used as a proxy for a task's similarity to pretraining tasks.

Table 3: Prefix tuning can combine pretraining skills to solve a new task requiring the same skills.

	Iterations	Pretraining tasks	Asc.+1
Pretrain on random tasks	40 000	8.2%–39.6%	0.0%
Prefix-tune on a pretraining task	20 000	99.8%– 100%	0.0%
Prefix-tune on ascending + 1	500 000	0.0%	34.5%

6 Discussion and related works

Understanding fine-tuning and prefix-tuning. Prior works show that prefixes have low intrinsic dimension allowing transfer to similar tasks (Qin et al., 2021; Su et al., 2022; Zhong et al., 2022; Wang et al., 2022b; Zheng et al., 2023). This work offered theoretical insights to their results: this subspace is the span of the prefix-induced bias. Other works shows that skills can be localized in the parameter space of pretrained models (Wang et al., 2022a; Panigrahi et al., 2023). We showed that it is also possible to identify subspaces of the residual stream corresponding to individual tasks.

Prompting and in-context learning. Prompting and in-context learning are a special case of prefix-tuning so prompting is unlikely to enable the model to solve a completely new task. Our theory thus explains why Kossen et al. (2023) observed that in-context examples cannot overcome pre-training skills. While context-based fine-tuning approaches cannot learn *arbitrary* new tasks, as shown in Section 5, they can leverage pre-trained skills. For instance, transformers can learn linear models in-context by mimicking gradient descent (Von Oswald et al., 2023) or by approximating matrix inversion (Akyürek et al., 2022). This is consistent with our theory: the prediction updates are enacted as biases in the activations of the attention block. Still, there are non-algorithmic applications where our theory predicts that in-context learning will fail, for example, translating to a language that the model has never seen before, even if large number of translation pair are provided in-context.

Implications for catastrophic forgetting and model alignment. The lack of expressiveness of context-based fine-tuning can be a feature: desirable properties will be maintained. Full fine-tuning can result in catastrophic forgetting (He et al., 2021b; Luo et al., 2023; Mukhoti et al., 2023). Our theory shows that context-based methods will not destroy pretrained skills. Model alignment poses the reverse problem: ensuring that the model cannot pick up undesirable skills during fine-tuning. Our results show that prompting and prefix-tuning cannot steer the model towards new adversarial behaviors. Hence, the recent successes in adversarial prompting (Zou et al., 2023) indicate that current model alignment methods just mask the undesirable skills rather than removing them.

Implications for model interpretability. An open question for language model interpretability is whether attention is sufficient for explainability (Jain and Wallace, 2019; Wiegreffe and Pinter, 2019). Section 5 points toward the negative: by interfering in the *output* of the attention layer, we can drastically change the behavior of the model, without changing its attention patterns.

Acknowledgments and Disclosure of Funding

This work is supported by a UKRI grant Turing AI Fellowship (EP/W002981/1) and the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (EP/S024050/1). AB has received funding from the Amazon Research Awards. We also thank the Royal Academy of Engineering and FiveAI.

References

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. 2022. What learning algorithm is in-context learning? Investigations with linear models. In *International Conference on Learning Representations*.

Jiaqi Bai, Zhao Yan, Jian Yang, Xinnian Liang, Hongcheng Guo, and Zhoujun Li. 2023. Knowprefix-tuning: A two-stage prefix-tuning framework for knowledge-grounded dialogue generation. *arXiv preprint arXiv:2306.15430*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*.

YunSeok Choi and Jee-Hyong Lee. 2023. CodePrompt: Task-agnostic prefix tuning for program and language generation. In *Findings of the Association for Computational Linguistics: ACL 2023*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *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)*.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2021. A mathematical framework for transformer circuits. *Transformer Circuits Thread*.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: Word-level Adversarial ReProgramming. In *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)*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021a. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

Tianxing He, Jun Liu, Kyunghyun Cho, Myle Ott, Bing Liu, James Glass, and Fuchun Peng. 2021b. Analyzing the forgetting problem in pretrain-finetuning of open-domain dialogue response models. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. 2023. LLM-Adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not explanation. In *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)*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Andrej Karpathy. 2020. minGPT GitHub Repository.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*.

Jannik Kossen, Tom Rainforth, and Yarin Gal. 2023. In-context learning in large language models learns label relationships but is not conventional learning. *arXiv preprint arXiv:2307.12375*.

Emanuele La Malfa, Aleksandar Petrov, Christoph Weinhuber, Simon Frieder, Ryan Burnell, Anthony G. Cohn, Nigel Shadbolt, and Michael Wooldridge. 2023. The ARRT of Language-Models-as-a-Service: Overview of a new paradigm and its challenges. *arXiv preprint arXiv:2309.16573*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing continuous prompts for generation. In *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)*.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-Tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2021. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*.

Jishnu Mukhoti, Yarin Gal, Philip H. S. Torr, and Puneet K. Dokania. 2023. Fine-tuning can cripple your foundation model; preserving features may be the solution. *arXiv preprint arXiv:2308.13320*.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Mutethia Mutuma, Yasin Tarabar, Ankit Gupta, Tao Yu, Yi Chern Tan, Xi Victoria Lin, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. 2021. DART: Open-domain structured data record to text generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Yawen Ouyang, Yongchang Cao, Yuan Gao, Zhen Wu, Jianbing Zhang, and Xinyu Dai. 2023. On prefix-tuning for lightweight out-of-distribution detection. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Abhishek Panigrahi, Nikunj Saunshi, Haoyu Zhao, and Sanjeev Arora. 2023. Task-specific skill localization in fine-tuned language models. In *International Conference on Machine Learning*.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying LMs with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Yujia Qin, Xiaozhi Wang, Yusheng Su, Yankai Lin, Ning Ding, Jing Yi, Weize Chen, Zhiyuan Liu, Juanzi Li, Lei Hou, et al. 2021. Exploring universal intrinsic task subspace via prompt tuning. *arXiv preprint arXiv:2110.07867*.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, et al. 2021. Scaling language models: Methods, analysis & insights from training Gopher.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*.

Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen. 2018. CARER: Contextualized affect representations for emotion recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yusheng Su, Xiaozhi Wang, Yujia Qin, Chi-Min Chan, Yankai Lin, Huadong Wang, Kaiyue Wen, Zhiyuan Liu, Peng Li, Juanzi Li, Lei Hou, Maosong Sun, and Jie Zhou. 2022. On transferability of prompt tuning for natural language processing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, Joao Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. 2023. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou', and Daniel Cer. 2022. SPoT: Better frozen model adaptation through soft prompt transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Xiaozhi Wang, Kaiyue Wen, Zhengyan Zhang, Lei Hou, Zhiyuan Liu, and Juanzi Li. 2022a. Finding skill neurons in pre-trained transformer-based language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*.

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. 2022b. Multitask prompt tuning enables parameter-efficient transfer learning. In *International Conference on Learning Representations*.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Hui Wu and Xiaodong Shi. 2022. Adversarial soft prompt tuning for cross-domain sentiment analysis. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Yuanhang Zheng, Zhixing Tan, Peng Li, and Yang Liu. 2023. Black-box prompt tuning with subspace learning. *arXiv preprint arXiv:2305.03518*.

Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, and Dacheng Tao. 2022. PANDA: Prompt transfer meets knowledge distillation for efficient model adaptation. *arXiv preprint arXiv:2208.10160*.

Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

A Constructing transformers that utilize the capacity of the embedding space

A.1 Unconditional generation for a single virtual token

This section provides an explicit construction of a transformer with the properties described in **??**. The goal is to construct a transformer that, by varying the choice of the virtual token, can generate any sequence of N tokens.

First, we need to specify how we encode the target sequence $(\mathtt{Y}_1, \ldots, \mathtt{Y}_N)$ into the virtual token $\boldsymbol{s}_1$. We chose the size of the embedding (and hence of $\boldsymbol{s}_1$) to be N. This way, each element of $\boldsymbol{s}_1$ can represent one position of the target sequence. We then represent the token value by discretizing each element of $\boldsymbol{s}_1$ into V levels:

$$\boldsymbol{s}_1 = \left({}^{(\mathtt{Y}_1 - 1)}/_{V}, \ldots, {}^{(\mathtt{Y}_N - 1)}/_{V} \right).$$

Note that this means that each element of $\boldsymbol{s}_1$ is in $[0, 1)$.

When predicting the token for the $i+1$ position, the transformer needs to pick the i-th element of $\boldsymbol{s}_1$, and then decode the corresponding value as a one-hot encoding representing the $\mathtt{Y}_i$-th token.

We extract the i-th element of $\boldsymbol{s}_1$ using one attention block of two heads. The `fst` head always looks at the first position which is our virtual token $\boldsymbol{s}_1$. For that purpose we create an attention head that always has $\boldsymbol{A}^{\text{fst}}_{ij} = 1$ if $j = 1$ and $\boldsymbol{A}^{\text{fst}}_{ij} = 0$ otherwise together with a value matrix $\boldsymbol{W}^{\text{fst}}_V$ that extracts the embedding. This is achieved with

$$\boldsymbol{W}^{\text{fst}}_Q = [\boldsymbol{0}_N, \boldsymbol{1}_N], \qquad \boldsymbol{W}^{\text{fst}}_K = [\boldsymbol{0}_N, 1, \boldsymbol{1}_{N-1}], \qquad \boldsymbol{W}^{\text{fst}}_V = [\boldsymbol{I}_N, \boldsymbol{0}_{N \times N}], \tag{8}$$

and a sufficiently high inverse temperature parameter T.

The `pos` head instead extracts the one-hot encoding of the current position. This can be done with an attention head that always attends only to the current position and a value matrix $\boldsymbol{W}^{\text{pos}}_V$ that extracts the position embedding as a one-hot vector:

$$\boldsymbol{W}^{\text{pos}}_Q = [\boldsymbol{0}_{N \times N}, \boldsymbol{I}_N], \qquad \boldsymbol{W}^{\text{pos}}_K = [\boldsymbol{0}_{N \times N}, \boldsymbol{I}_N], \qquad \boldsymbol{W}^{\text{pos}}_V = [\boldsymbol{0}_{N \times N}, \boldsymbol{I}_N]. \tag{9}$$

When the outputs of these two attention heads are summed, then only the element of $\boldsymbol{s}_1$ that corresponds to the current position will be larger than 1. From Equation (2) the output at the i-th position of the attention block is:

$$\boldsymbol{t}_i = \sum_{j=1}^{p} \boldsymbol{A}^{\text{fst}}_{ij} \boldsymbol{x}_j + \sum_{j=1}^{p} \boldsymbol{A}^{\text{pos}}_{ij} \boldsymbol{e}_N(j) = \boldsymbol{s}_1 + \boldsymbol{e}_N(i),$$

where $\boldsymbol{x}_1 = \boldsymbol{s}_1$ and $\boldsymbol{x}_j = E_{:, \mathtt{Y}_{j-1}}$ for $j > 1$.

We can extract the value of $\boldsymbol{s}_1$ corresponding to the current position by substracting 1 from the hidden state and apply ReLU: $\hat{\mathcal{L}}_{\text{ex}} = \hat{\mathcal{L}}[\boldsymbol{I}_N, -\boldsymbol{1}_N]$. Now, we are left with only one non-zero entry and that's the one corresponding to the next token. We can retain only the non-zero entry if we just sum all the entries of the hidden state with $\hat{\mathcal{L}}_{\texttt{sum}} = \hat{\mathcal{L}}[\boldsymbol{1}^\top_N, 0]$.

The final step is to map this scalar to a V-dimensional vector which has its maximum value at index $\mathtt{Y}_i$. This task is equivalent to designing V linear functions, each attaining its maximum at one of $0, {}^1/_V, \ldots, {}^{(V-1)}/_V$. To construct this, we use the property of convex functions that their tangent is always under the plot of the function. Therefore, given a convex function $\gamma(x)$, we construct the i-th linear function to be simply the tangent of γ at ${}^{i-1}/_V$. If we take $\gamma(x) = (x - {}^1/_2)^2$, this results in the following linear layer:

$$\mathcal{L}_{\text{proj}} = \mathcal{L} \begin{bmatrix} \left[\frac{2(1-1)}{V} - 1, \ldots, \frac{2(V-1)}{V} - 1 \right]^T, \left[\frac{1}{4} - \frac{(1-1)^2}{V^2}, \ldots, \frac{1}{4} - \frac{(V-1)^2}{V^2} \right]^\top \end{bmatrix}. \tag{10}$$

Figure 4 shows the predictors for each individual token id.

With just two attention heads and three linear layers, the transformer $\mathcal{A}[(\boldsymbol{W}^{\text{fst}}_Q, \boldsymbol{W}^{\text{pos}}_Q), (\boldsymbol{W}^{\text{fst}}_K, \boldsymbol{W}^{\text{pos}}_K), (\boldsymbol{W}^{\text{fst}}_V, \boldsymbol{W}^{\text{pos}}_V)] \fatsemi \hat{\mathcal{L}}_{\text{ex}} \fatsemi \hat{\mathcal{L}}_{\text{sum}} \fatsemi \mathcal{L}_{\text{proj}} \fatsemi \text{softmax}$ achieves the upper bound of V^N unique outputs by controlling a single virtual token at its input. Note that for this construction, the choice of embedding matrix $E \in \mathbb{R}^{N \times V}$ does not matter. The same transformer

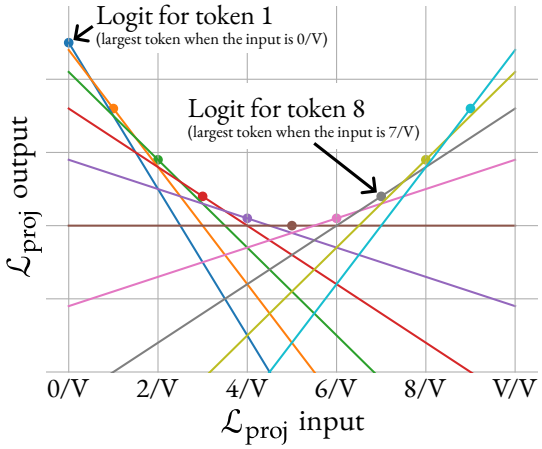

Figure 4: Illustration of the predictors for each token in the $\mathcal{L}_{\text{proj}}$ linear layer for $V = 10$. The layer is constructed in such a way that the i-th token has the highest confidence when the input is $^{i-1}/_{V}$.

architecture can generate only V unique outputs if we only control the first token instead. Therefore, it is indeed the case that the embedding space has exponentially more capacity for control than the token space. You can see this transformer implemented and running in practice in Section 2 of the `constructions` Jupyter notebook.

A.2 Conditional generation for a single virtual token ($n_X = n_Y = 1$)

This section provides an explicit construction of a transformer with the properties described in Theorem 1. The goal is to construct a transformer that, by varying the choice of the virtual token, can cause the model to act as any map $m : [1, \dots, V] \to [1, \dots, V]$. In other words, by selecting the virtual token, we can fully control how the model will respond to any token the user may provide.

First, we need to specify how the map m will be encoded in the virtual token $\boldsymbol{s}_1$. We choose the embedding size d_e to be V. Now, we can use the same encoding scheme as before, but now each element in $\boldsymbol{s}_1$ corresponds to a different user token, rather than to a position in the generated sequence:

$$\boldsymbol{s}_1 = \left({}^{m(1)}/_{V}, \dots, {}^{m(V)}/_{V}\right).$$

Therefore, the first element of $\boldsymbol{s}_1$ designates the response if the user provides token 1, the second element is the response to the token 2, and so on.

Extracting the $\texttt{Y}_i$-th value from $\boldsymbol{s}_1$ and decoding it can be done in a very similar way as for the unconditional case. The only difference is that instead of looking at the user input position, we look at its value. Take $\boldsymbol{E} = \boldsymbol{I}_V$ and $N = 2$.

Hence we have the following `val` head (only differing in the $\boldsymbol{W}_V$ matrix from Equation (9)):

$$\boldsymbol{W}_Q^{\text{val}} = [\boldsymbol{0}_{2\times V}, \boldsymbol{I}_2], \qquad \boldsymbol{W}_K^{\text{val}} = [\boldsymbol{0}_{2\times V}, \boldsymbol{I}_2], \qquad \boldsymbol{W}_V^{\text{val}} = [\boldsymbol{I}_V, \boldsymbol{0}_{V\times 2}].$$

We also need embedding of the first token, so we have a modified version of Equation (8):

$$\boldsymbol{W}_Q^{\text{fst}} = [\boldsymbol{0}_V, 1, 1], \qquad \boldsymbol{W}_K^{\text{fst}} = [\boldsymbol{0}_V, 1, 0], \qquad \boldsymbol{W}_V^{\text{fst}} = [\boldsymbol{I}_V, \boldsymbol{0}_{V\times 2}].$$

And hence the output of this attention block at the second position would be:

$$\boldsymbol{t}_2 = \sum_{j=1}^{2} \boldsymbol{A}_{ij}^{\text{fst}} \boldsymbol{x}_j + \sum_{j=1}^{2} \boldsymbol{A}_{ij}^{\text{val}} \boldsymbol{W}_V^{\text{fst}} x_j = \boldsymbol{s}_1 + \boldsymbol{e}_V(\texttt{Y}_1).$$

Similarly to the unconditional case, only the entry of $\boldsymbol{t}_2$ corresponding to the user token will have a value above 1 and that value would be $1 + m(x_1)/V$.

We can now extract the one-hot representation of the target token using the same approach as before, just adjusting for the different hidden state size: $\hat{\mathcal{L}}_{\text{ex}} = \hat{\mathcal{L}}[\boldsymbol{I}_V, -\boldsymbol{1}_V]$, $\hat{\mathcal{L}}_{\text{sum}} =$

$\hat{\mathcal{L}}[\mathbf{1}_V^\top, 0]$, and the same projection had as before (Equation (10)). The final transformer is then: $\mathcal{A}[(\boldsymbol{W}_Q^{\text{fst}}, \boldsymbol{W}_Q^{\text{val}}), (\boldsymbol{W}_K^{\text{fst}}, \boldsymbol{W}_K^{\text{val}}), (\boldsymbol{W}_V^{\text{fst}}, \boldsymbol{W}_V^{\text{val}})] \fatsemi \hat{\mathcal{L}}_{\text{ex}} \fatsemi \hat{\mathcal{L}}_{\text{sum}} \fatsemi \mathcal{L}_{\text{proj}} \fatsemi \mathrm{softmax}$. You can see this transformer implemented and running in practice in Section 3 of the `constructions` Jupyter notebook.

A.3 Conditional generation for longer responses ($n_X = 1, n_Y > 1$)

We can obtain longer responses via a simple extension. If the response length is N_0, then we can encode the map $m : [1, \ldots, V] \to [1, \ldots, V]^{N_0}$ in N_0 virtual tokens, each corresponding to one of the target positions:

$$\boldsymbol{s}_i = \left({}^{m(1)_i}/_V, \ldots, {}^{m(V)_i}/_V\right) \text{ for } i = 1, \ldots, N_0.$$

For this model we would then have $N = 2N_0$ and $d_e = V$.

First, we need a head that always looks at the token provided by the user, which will be at position $N_o + 1$:

$$\boldsymbol{W}_Q^{\text{user}} = [\mathbf{0}_V, \mathbf{1}_N], \qquad \boldsymbol{W}_K^{\text{user}} = [\mathbf{0}_{(V+N_o)}, 1, \mathbf{0}_{(N_o-1)}], \qquad \boldsymbol{W}_V^{\text{user}} = [\boldsymbol{I}_V, \mathbf{0}_{V\times N}].$$

In order to consume the map at the right location, we need to also look at the embedding of the token N_o positions before the one we are trying to generate:

$$\boldsymbol{W}_Q^{\text{back}} = \begin{bmatrix} \mathbf{0}_{N_0\times(N_0+V)} & \boldsymbol{I}_{N_0} \\ \mathbf{0}_{N_0\times(N_0+V)} & \mathbf{0}_{N_0\times N_0} \end{bmatrix}, \qquad \boldsymbol{W}_K^{\text{back}} = [\mathbf{0}_{N\times V}, \boldsymbol{I}_N], \qquad \boldsymbol{W}_V^{\text{back}} = [\boldsymbol{I}_V, \mathbf{0}_{V\times N}].$$

From here on, the decoding is exactly the same as in the $n_X = n_Y = 1$ case. The final transformer is then: $\mathcal{A}[(\boldsymbol{W}_Q^{\text{user}}, \boldsymbol{W}_Q^{\text{back}}), (\boldsymbol{W}_K^{\text{user}}, \boldsymbol{W}_K^{\text{back}}), (\boldsymbol{W}_V^{\text{user}}, \boldsymbol{W}_V^{\text{back}})] \fatsemi \hat{\mathcal{L}}_{\text{ex}} \fatsemi \hat{\mathcal{L}}_{\text{sum}} \fatsemi \mathcal{L}_{\text{proj}} \fatsemi \mathrm{softmax}$. You can see this transformer implemented and running in practice in Section 4 of the `constructions` Jupyter notebook.

A.4 Conditional generation for longer user inputs ($n_X > 1$, $n_Y = 1$)

Finally, we consider the case when the user input `X` is longer. This is a bit more complicated because we need to search through a domain of size V^V. We will only consider the case with $n_X = 2$ where we would need two attention layers. A similar approach can be used to construct deeper models for $n_X > 2$. Finally, combining the strategy in the previous section for longer responses with the strategy in this section for longer user inputs allows us to construct transformers that map from arbitrary length user strings to arbitrary length responses.

In order to encode a map $m : [1, \ldots, V]^2 \to [1, \ldots, V]$ into a single virtual token we would need a more involved construction than before. Similarly to how we discretized each element of the virtual token $\boldsymbol{s}_1$ in V levels before, we are going to now discretize it into V^V levels. Each one of these levels would be one of the V^V possible maps from the *second* user token to the response. The first user token would be used to select the corresponding element of $\boldsymbol{s}_1$. Then this scalar will be "unpacked" into a new vector of V elements using the first attention block. Then, the second user token will select an element from this unpacked vector, which will correspond to the target token.

We construct the virtual token as follows:

$$\boldsymbol{s}_1 = \left[\sum_{i=1}^{V} m_1(i) \times \frac{V^{i-1}}{V^V}, \ldots, \sum_{i=1}^{V} m_V(i) \times \frac{V^{i-1}}{V^V}\right],$$

where $m_f(x) = m(f, x)$ is a map from the second user token to the response when the first token is fixed to be f.

An additional change from the previous constructions is that we are going to divide the residual stream into two sections. This is in line with the theory that different parts of the residual stream specialize for different communications needs by different attention heads (Elhage et al., 2021). We will use the first half of the residual stream to extract and "unpack" the correct mapping from second token to target token, while the second half of the residual stream will be used to copy the second token value so that the second attention layer can use it to extract the target. As usual, the embedding matrix will be the identity matrix: $\boldsymbol{E} = \boldsymbol{I}_V$. Finally, for convenience, we will also use a dummy zero

virtual token that we will attend to when we want to not attend to anything. This results in context size $N = 4$ with the input being

$$\left(\begin{bmatrix}0_V\\ \boldsymbol{e}_N(1)\end{bmatrix}, \begin{bmatrix}\boldsymbol{s}_1\\ \boldsymbol{e}_N(2)\end{bmatrix}, \begin{bmatrix}\boldsymbol{E}_{:,\mathtt{X}_1}\\ \boldsymbol{e}_N(3)\end{bmatrix}, \begin{bmatrix}\boldsymbol{E}_{:,\mathtt{X}_2}\\ \boldsymbol{e}_N(4)\end{bmatrix}\right) = \left(\begin{bmatrix}\mathbf{0}_V\\ \boldsymbol{e}_N(1)\end{bmatrix}, \begin{bmatrix}\boldsymbol{s}_1\\ \boldsymbol{e}_N(2)\end{bmatrix}, \begin{bmatrix}\boldsymbol{e}_V(\mathtt{X}_1)\\ \boldsymbol{e}_N(3)\end{bmatrix}, \begin{bmatrix}\boldsymbol{e}_V(\mathtt{X}_2)\\ \boldsymbol{e}_N(4)\end{bmatrix}\right).$$

We want the output at the last position to be the target $m(\mathtt{X}_1, \mathtt{X}_2)$, that is:

$$\arg\max_{u\in 1,\ldots,V} \boldsymbol{y}_{4,u} = m(\mathtt{X}_1, \mathtt{X}_2) \text{ for any } m, \mathtt{X}_1, \mathtt{X}_2.$$

The first attention block will have three attention heads.

As before, we want to extract the value of $\boldsymbol{s}_1$ that corresponds to the first token the user provided ($\mathtt{X}_1$) and place it in the first half of the residual stream. We want only the third position to do that, while the rest of the positions keep the first half of their residual stream with zeros. Hence we have the following `fst` head:

$$\boldsymbol{W}_Q^{\text{fst}} = \left[\begin{array}{c|cccc}\mathbf{0}_{2\times V} & 1 & 1 & 0 & 1\\ & 0 & 0 & 1 & 0\end{array}\right], \quad \boldsymbol{W}_K^{\text{fst}} = \left[\begin{array}{c|cccc}\mathbf{0}_{2\times V} & 1 & 0 & 0 & 0\\ & 0 & 1 & 0 & 0\end{array}\right], \quad \boldsymbol{W}_V^{\text{fst}} = \begin{bmatrix}\boldsymbol{I}_V & \mathbf{0}_{V\times N}\\ \mathbf{0}_{V\times V} & \mathbf{0}_{V\times N}\end{bmatrix}.$$

The `user1` head extracts the value of the first user-provided token ($\mathtt{X}_1$) and also places it in the first half of the residual stream:

$$\boldsymbol{W}_Q^{\text{user1}} = \left[\begin{array}{c|cccc}\mathbf{0}_{2\times V} & 1 & 1 & 0 & 1\\ & 0 & 0 & 1 & 0\end{array}\right], \boldsymbol{W}_K^{\text{user1}} = \left[\begin{array}{c|cccc}\mathbf{0}_{2\times V} & 1 & 0 & 0 & 0\\ & 0 & 0 & 1 & 0\end{array}\right], \boldsymbol{W}_V^{\text{user1}} = \begin{bmatrix}\boldsymbol{I}_V & \mathbf{0}_{V\times N}\\ \mathbf{0}_{V\times V} & \mathbf{0}_{V\times N}\end{bmatrix}.$$

And the `user2` head does the same for the value of the second user-provided token ($\mathtt{X}_2$), placing it in the second half of the residual stream:

$$\boldsymbol{W}_Q^{\text{user2}} = \left[\begin{array}{c|cccc}\mathbf{0}_{2\times V} & 1 & 1 & 1 & 0\\ & 0 & 0 & 0 & 1\end{array}\right], \boldsymbol{W}_K^{\text{user2}} = \left[\begin{array}{c|cccc}\mathbf{0}_{2\times V} & 1 & 0 & 0 & 0\\ & 0 & 0 & 0 & 1\end{array}\right], \boldsymbol{W}_V^{\text{user2}} = \begin{bmatrix}\mathbf{0}_{V\times V} & \mathbf{0}_{V\times N}\\ 2\boldsymbol{I}_V & \mathbf{0}_{V\times N}\end{bmatrix},$$

where the factor 2 is there because, as usual, the first linear layer will subtract 1 from everything in order to extract the value selected by the first token.

This linear layer looks as usual: $\hat{\mathcal{L}}_{\text{ex2}} = \hat{\mathcal{L}}[\boldsymbol{I}_{2V}, -\mathbf{1}_{2V}]$. The result is that the first V elements will be 0 except one which designates which map from second user token to output we should use, and the second V elements have a one hot-encoding of the second user token. Constructing an MLP that unpacks the mapping can become quite involved so we do not provide an explicit form for it. But from the universal approximation theorems and the finiteness of the domain and range, we know that such an MLP should exist. We thus designate by `unpack` the MLP that decodes the first half of the residual stream to:

$$\left(\frac{m_{\mathtt{X}_1}(1)}{V}, \ldots, \frac{m_{\mathtt{X}_1}(V)}{V}\right)$$

and keeps the second half unchanged.

And now, by using two attention heads, the second attention block extracts the value of the above vector at the position designated by the second token, in a fashion not dissimilar to all the previous cases:

$$\boldsymbol{W}_Q^{\text{emb}} = [\mathbf{0}_V^\top, \mathbf{1}_V^\top], \qquad \boldsymbol{W}_K^{\text{emb}} = [\mathbf{1}_V^\top, \mathbf{0}_V^\top], \qquad \boldsymbol{W}_V^{\text{emb}} = [\boldsymbol{I}_V \quad \mathbf{0}_{V\times V}],$$

$$\boldsymbol{W}_Q^{\text{user2'}} = [\mathbf{0}_V^\top, \mathbf{1}_V^\top], \qquad \boldsymbol{W}_K^{\text{user2'}} = [\mathbf{0}_V^\top, \mathbf{1}_V^\top], \qquad \boldsymbol{W}_V^{\text{user2'}} = [\mathbf{0}_{V\times V} \quad \boldsymbol{I}_V],$$

And finally, with $\hat{\mathcal{L}}_{\text{ex}} = \hat{\mathcal{L}}[\boldsymbol{I}_V, -\mathbf{1}_V]$, $\hat{\mathcal{L}}_{\text{sum}} = \hat{\mathcal{L}}[\mathbf{1}_V^\top, 0]$, and the same projection had as before (Equation (10)), we get the target token. The final transformer is then: $\mathcal{A}[(\boldsymbol{W}_Q^{\text{fst}}, \boldsymbol{W}_Q^{\text{user1}}, \boldsymbol{W}_Q^{\text{user2}}), (\boldsymbol{W}_K^{\text{fst}}, \boldsymbol{W}_K^{\text{user1}}, \boldsymbol{W}_K^{\text{user2}}), (\boldsymbol{W}_V^{\text{fst}}, \boldsymbol{W}_V^{\text{user1}}, \boldsymbol{W}_V^{\text{user2}})] \fatsemi \hat{\mathcal{L}}_{\text{ex2}} \fatsemi \texttt{unpack} \fatsemi \mathcal{A}[(\boldsymbol{W}_Q^{\text{emb}}, \boldsymbol{W}_Q^{\text{user2'}}), (\boldsymbol{W}_K^{\text{emb}}, \boldsymbol{W}_K^{\text{user2'}}), (\boldsymbol{W}_V^{\text{emb}}, \boldsymbol{W}_V^{\text{user2'}})] \fatsemi \hat{\mathcal{L}}_{\text{ex}} \fatsemi \hat{\mathcal{L}}_{\text{sum}} \fatsemi \mathcal{L}_{\text{proj}} \fatsemi \text{softmax}$. You can see this transformer implemented and running in practice in Section 5 of the `constructions` Jupyter notebook.

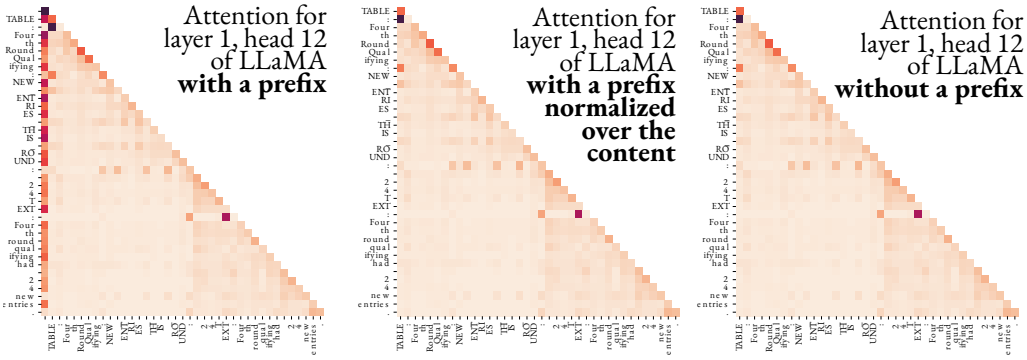

Figure 5: The attention of the twelfth head of the first layer of LLaMA (Touvron et al., 2023). The left plot shows the attention with a prefix of length one. The second plot shows the same attention but normalized such that the attenion over the non-prefix positions sums to 1. The right plot shows the attention of the pre-trained model (without prefix). The center and the right plots are the same, illustrating that the presence of the prefix indeed only scales down the attention over the content (non-prefix positions) but does not change its relative distribution, providing empirical validation of Equation (6). The test sequence is `TABLE: Fourth Round Qualifying : NEW_ENTRIES_THIS_ROUND : 24 TEXT: Fourth round qualifying had 24 new entries.` from the DART table-to-test dataset (Nan et al., 2021).

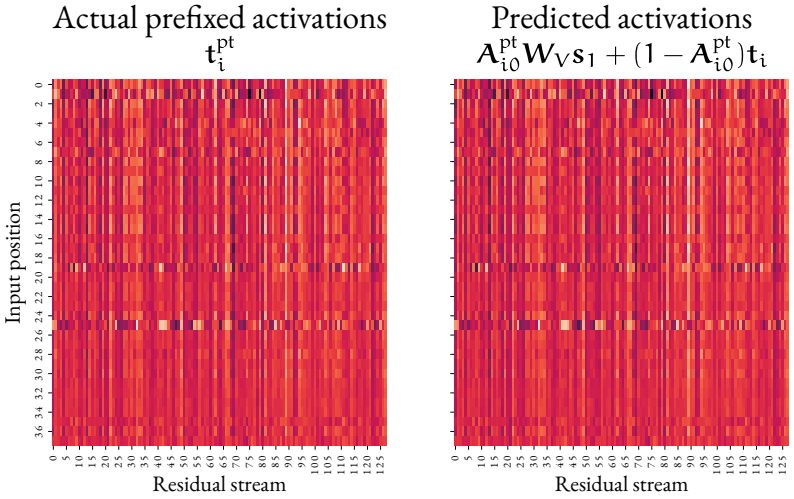

Figure 6: The activations of the twelfth head of the first layer of LLaMA (Touvron et al., 2023). The left plot shows the activations in the presence of the prefix. The right plot shows the activations $\boldsymbol{t}_i$ of the pretrained model, scaled by one minus the attention that the prefix would take and then biased in the direction $\boldsymbol{W}_V\boldsymbol{s}_1$. The two plots are the same, illustrating that our theory, Equation (7) in particular, also holds for real-world large transformer models. The test sequence is the same as in Figure 5.

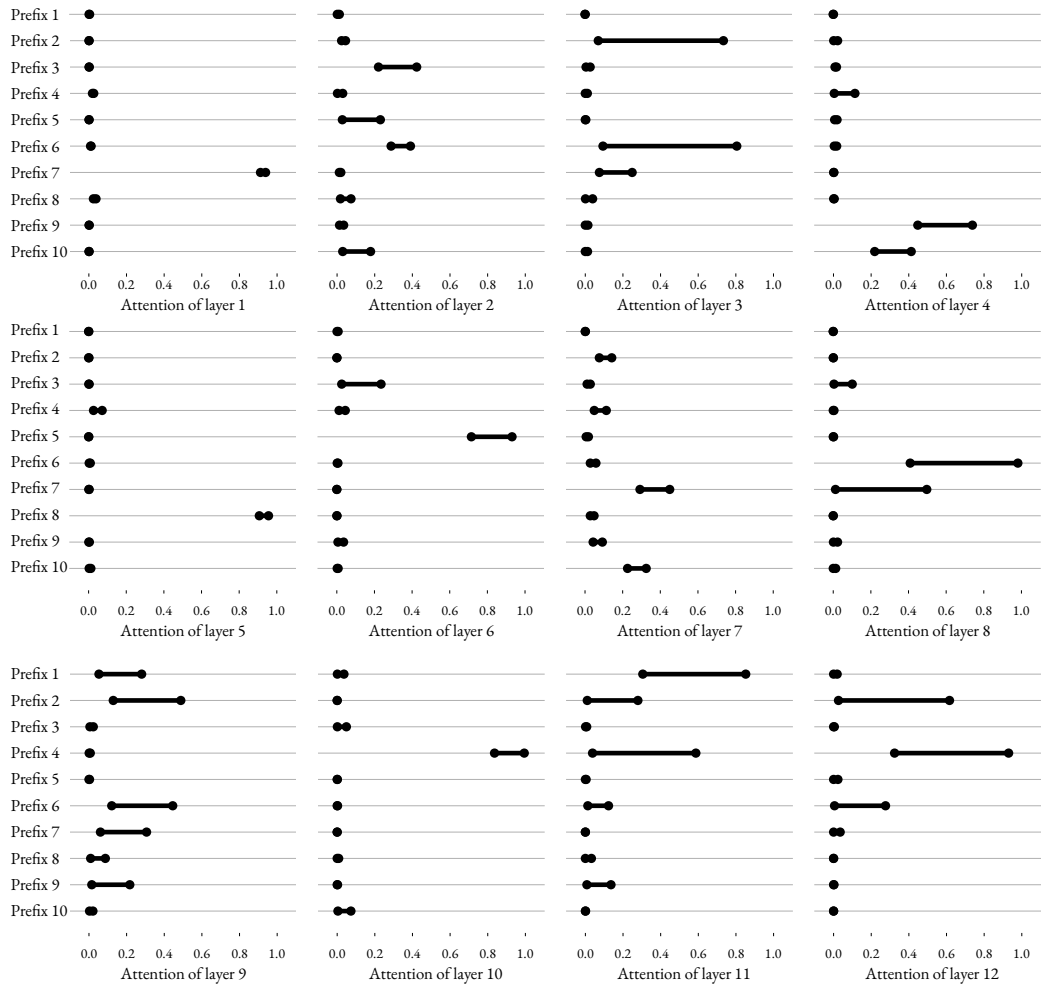

Figure 7: The range of attention (1st to 99th percentile) for a single GPT-2 (Radford et al., 2019) prefix trained on the Emotion dataset (Saravia et al., 2018). The prefix is of size 10 ($n_S = 10$). This is the attention of the last user input token (n_X) because this is the position at which the class prediction is done. For illustration purposes, we have normalized the attention so that the attention over the 10 prefix positions sums to 1. The range of attention over the 10 positions for each layer are shown.

B Longer prefixes define larger subspaces for the bias but are not fully utilized in practice.

In the case of a longer prefix $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{n_S})$, the bias vector is in a subspace of dimensionality n_S: $\boldsymbol{t}_i^{\text{pt}} = \sum_{j=1}^{n_S} \boldsymbol{A}_{i,S_j}^{\text{pt}} \boldsymbol{W}_V \boldsymbol{s}_j + (1 - \sum_{j=1}^{n_S} \boldsymbol{A}_{i,S_j}^{\text{pt}}) \boldsymbol{t}_i$, where i goes over the content and j over the prefix positions. Larger prefixes thus have a larger subspace to modify the attention block output. The specific direction is determined by the relative distribution of attention across the prefix positions.

One would expect that different prefix positions specialize for different sub-tasks. However, when we look at the distribution of attention across the prefix positions for various inputs as in Appendix B this does not seem to be the case: the attention for each prefix position is in a limited range. That indicates that in practice, prefix-tuning does not make full use of the whole space that the vectors $\boldsymbol{W}_V \boldsymbol{s}_j$ span. We hypothesise that this is due to the two competing optimization goals for the vectors $\boldsymbol{s}_j$: at the same time they need to "grab attention" when interacting with $\boldsymbol{W}_K$ and determine the bias direction when multiplied with $\boldsymbol{W}_V$.

Longer prefixes define a subspace from which the bias for the attention block is selected. For a prefix of size n_S, that means that this subspace is n_S-dimensional. Each of prefix position j defines a basis

vector $\boldsymbol{W}_V \boldsymbol{s}_j$ for this subspace, while the attention $\boldsymbol{A}^{\text{pt}}_{i,S_j}$ on this position determines how much of this basis component contributes to the bias.

In ordered to span the whole subspace and make full use of the capacity of the prefix, $\boldsymbol{A}^{\text{pt}}_{i,S_j}$ should vary between 0 and 1 for different inputs. However, we observe that this does not happen in practice. Figure 7 shows the ranges of attention the different prefix positions take for the GPT-2 model (Radford et al., 2019). For layer 1, for example, the attention each prefix positions gets is almost constant hence, the effective subspace is collapsed and there is a single bias vector that's applied to the attention layer output, regardless of the user input X.

Some other layers show slightly higher variation. For example, layer 3 has three prefix positions with large variations. Therefore, the effective bias subspace is 3-dimensional and the user input X governs which bias vector from this subspace will be selected.

C Prefix-tuning can change the attention, albeit the one of the next layer

The empirical success of prefix-tuning is also due the non-linear behavior of the prefix-induced bias when passed through the following MLP and later attention layers. Let's see how the prefix of one attention layer affects the following attention layer. For simplicity, assume no MLP between them, residual connections or layer norms: the output $\boldsymbol{t}^{(1)}_i$ of one attention layer is the input $\boldsymbol{x}^{(2)}_i$ of the next.

The pretrained model then has outputs $\boldsymbol{t}^{(1)}_i = \sum_{j=1}^{p} \boldsymbol{A}^{(1)}_{ij} \boldsymbol{W}^{(1)}_V \boldsymbol{x}^{(1)}_j$, resulting in the following attention for the second layer: $\tilde{\boldsymbol{A}}^{(2)}_{ij} = {}^{T}\!/\!\sqrt{k}\, \boldsymbol{t}^{(1)\top}_i \boldsymbol{H}^{(2)} \boldsymbol{t}^{(1)}_j$. Here $\tilde{\boldsymbol{A}}_{ij}$ is the attention before exponentiation and normalization, i.e., $\boldsymbol{A}_{ij} = \exp \tilde{\boldsymbol{A}}_{ij} / \sum_{r=1}^{p} \exp \tilde{\boldsymbol{A}}_{ir}$. For prefix-tuning we have:

$$
\begin{aligned}
\boldsymbol{t}^{\text{pt}(1)}_i &= \boldsymbol{A}^{\text{pt}(1)}_{i0} \boldsymbol{W}_V \boldsymbol{s}^{(1)}_1 + \sum_{j=1}^{p} \boldsymbol{A}^{\text{pt}(1)}_{ij} \boldsymbol{W}^{(1)}_V \boldsymbol{x}^{(1)}_j \overset{(7)}{=} \underbrace{\boldsymbol{A}^{\text{pt}(1)}_{i0}}_{\alpha_i} \underbrace{\boldsymbol{W}_V \boldsymbol{s}^{(1)}_1}_{\boldsymbol{\mu}} + (1 - \boldsymbol{A}^{\text{pt}(1)}_{i0}) \boldsymbol{t}^{(1)}_i \\
\tilde{\boldsymbol{A}}^{\text{pt}(2)}_{ij} &= {}^{T}\!/\!\sqrt{k}\, \boldsymbol{t}^{\text{pt}(1)\top}_i \boldsymbol{H}^{(2)} \boldsymbol{t}^{\text{pt}(1)}_j \\
&= {}^{T}\!/\!\sqrt{k} (\alpha_i \alpha_j \underbrace{\boldsymbol{\mu}^\top \boldsymbol{H}^{(2)} \boldsymbol{\mu}}_{\text{constant}} + \alpha_j (1 - \alpha_i) \underbrace{\boldsymbol{t}^{(1)\top}_i \boldsymbol{H}^{(2)} \boldsymbol{\mu}}_{\text{depends only on } \boldsymbol{t}^{(1)}_i} + \alpha_i (1 - \alpha_j) \underbrace{\boldsymbol{\mu}^\top \boldsymbol{H}^{(2)} \boldsymbol{t}^{(1)}_j}_{\text{depends only on } \boldsymbol{t}^{(1)}_j} + (1 - \alpha_i)(1 - \alpha_j) \underbrace{\boldsymbol{t}^{(1)\top}_i \boldsymbol{H}^{(2)} \boldsymbol{t}^{(1)}_j}_{\tilde{\boldsymbol{A}}^{(2)}_{ij}})
\end{aligned}
$$

The presence of $\boldsymbol{\mu}$ shows that the prefix of layer 1 can result in a changed attention pattern at the following layer. In contrast to the effect of the prefix on the attention of layer 1, this change is content-specific: the second and the third terms depend on the inputs. This demonstrates that a simple bias can have complex behavior when passed through non-linear MLPs and attention blocks.

Still, even considering this cross-layer effect, prefix-tuning is more limited in its expressiveness than full fine-tuning. While the second and the third terms are input-dependent, each depends on one input position only. The prefix does not change the bilinear dependency on both the query and key. This is something that the full fine-tuning can achieve: $\tilde{A}^{\text{ft}(2)}_{ij} = {}^{T}\!/\!\sqrt{k}\, \boldsymbol{t}^{\text{ft}(1)\top}_i (H^{(2)} + \Delta H^{(2)}) \boldsymbol{t}^{\text{ft}(1)}_j$. Hence, deeper prefix-tuned models also cannot learn new tasks that fully fine-tuned models can.

Prefix-tuning is can only bias the activations of attention blocks but we showed that this is enough for many practical tasks. If the pretrained model has seen the task, or if it can be solved with "skills" the pretrained model has, then prefix-tuning can successfully fine-tune the model for it. Otherwise, if the task is completely new (e.g., fine-tuning a model trained only on English to translate Bulgarian to Arabic), then prefix-tuning is doomed to fail, no matter how much data or compute one has.