HyperTuning: Toward Adapting Large Language Models without Back-propagation

Jason Phang [1 2] **Yi Mao** [3] **Pengcheng He** [3] **Weizhu Chen** [3]

Abstract

Fine-tuning large language models for different tasks can be costly and inefficient, and even methods that reduce the number of tuned parameters still require full gradient-based optimization. We propose HyperTuning, a novel approach to model adaptation that uses a hypermodel to generate task-specific parameters for a fixed downstream model. We demonstrate a simple setup for hypertuning with HyperT5, a T5-based hypermodel that produces soft prefixes or LoRA parameters for a frozen T5 model from few-shot examples. We train HyperT5 in two stages: first, hyperpretraining with a modified conditional language modeling objective that trains a hypermodel to generate parameters; second, multi-task fine-tuning (MTF) on a large number of diverse language tasks. We evaluate HyperT5 on P3, MetaICL and Super-NaturalInstructions datasets, and show that it can effectively generate parameters for unseen tasks. Moreover, we show that using hypermodel-generated parameters as initializations for further parameter-efficient fine-tuning improves performance. HyperTuning can thus be a flexible and efficient way to leverage large language models for diverse downstream applications.

1. Introduction

While language models (LMs) have achieved remarkable capabilities with increasing model size (Brown et al., 2020), fine-tuning them on specific downstream tasks introduces significant engineering challenges and computational costs. Although large models can perform zero-shot, instruction-prompted, and few-shot learning (Sanh et al., 2022; Wei et al., 2022), they are usually outperformed by fully fine-tuned models when sufficient training data is available.

To reduce the computational and memory overhead of fine-tuning LMs, parameter-efficient fine-tuning (PEFT) methods have been proposed, such as adapters (Houlsby et al., 2019), prefix tuning (Li & Liang, 2021), and prompt tuning (Lester et al., 2021). These methods update only a small subset of (possibly new) parameters of the LM, and have achieved competitive performance with full fine-tuning (Ding et al., 2022). However, PEFT methods still require full back-propagation through the LM during training, which is computationally expensive and memory intensive. Given that (1) only a small number of parameters need to be updated to adapt an LM to a given task, and (2) very large LMs have demonstrated strong in-context learning capabilities on a forward pass, we hypothesize that it is possible to train a separate model to perform the optimization or adaptation procedure entirely using only a forward pass.

To avoid the costly computation of back-propagating through the LM to produce parameter updates, especially for thousands or millions of iterations during training, we propose a new paradigm of **hypertuning**: using a *hypermodel* to adapt a *downstream* LM to a desired application. As a concrete proof of concept, we explore a simple setup where hypermodels take as input a set of few-shot examples from a given task, and output the PEFT parameters corresponding to that task in a single forward pass.

To demonstrate the feasibility of this approach, we train *HyperT5*: a set of T5-based hypermodels that output soft prefixes (Li & Liang, 2021) or LoRA parameters (Hu et al., 2022), to be incorporated into a frozen downstream T5 LM. To train HyperT5, we introduce a two-stage procedure for training hypermodels: *hyperpretraining*, where we adapt a pretrained LM to generate PEFT parameters via a modified language modeling objective, followed by *multi-task fine-tuning* (MTF) the hypermodel. HyperT5 models can take few-shot examples from unseen tasks and generate the corresponding PEFT parameters, allowing us to adapt a downstream LM without back-propagation. We show that hypertuning is effective on P3, Super-NaturalInstructions and MetaICL datasets. Furthermore, we show that when the hypermodel-generated parameters are used as initializations for further parameter-efficient fine-tuning, they achieve

[*]Equal contribution [1]Center for Data Science, New York University, NY, USA [2]EleutherAI [3]Microsoft Azure AI, WA, USA. Correspondence to: Jason phang <jasonphang@nyu.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

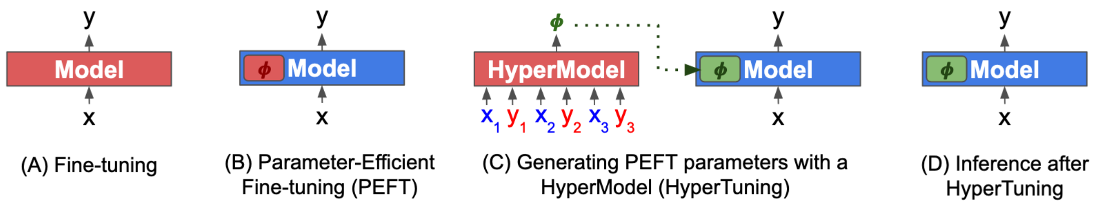
y
Model
x
(A) Fine-tuning
y
ϕ Model
x
(B) Parameter-Efficient Fine-tuning (PEFT)
ϕ
HyperModel
x_1 y_1 x_2 y_2 x_3 y_3
y
ϕ Model
x
(C) Generating PEFT parameters with a HyperModel (HyperTuning)
y
ϕ Model
x
(D) Inference after HyperTuning

Figure 1: Overview of HyperTuning. (A) Fine-tuning, where all model parameters are updated (red). (B) Parameter-efficient fine-tuning (PEFT), where all model parameters are frozen (blue) and only a small number of parameters, ϕ, are updated. (C) HyperTuning, where a hypermodel is used to generate parameters ϕ for a frozen downstream model. For instance, a hypermodel may take a set of few-shot examples to determine what ϕ to generate. Only the hypermodel's parameters are updated during training. (D) At inference time, the parameters ϕ only need to be generated once, and thereafter only need to store ϕ, with no need to retain the few-shot examples.

faster training convergence and better performance.

This work serves as a first step toward hypertuning, and we are aware of the limitations of this preliminary setup. Because our formulation of hypermodels can only take a small number of examples as input, its performance cannot compare to full parameter-efficient fine-tuning or full fine-tuning. HyperT5 also generally underperforms T5 explicitly trained for few-shot in-context learning with full attention across examples, although we note that the latter is more computationally expensive at inference time. Nevertheless, we believe that our results demonstrate a promising step toward model adaptation without the need for back-propagation.

2. Related Work

HyperNetworks Several works have explored the concept of ”hypernetworks,” where an auxiliary network is used to generate parameters for a primary network. This terminology was first introduced by Ha et al. (2017) and applied to LSTMs. Among Transformer-based language models, Karimi Mahabadi et al. (2021) and He et al. (2022) incorporated hypernetworks into T5 models for knowledge sharing during multitask fine-tuning. Peebles et al. (2022) utilized a Transformer with diffusion for generating full model parameters for image-recognition and Cartpole tasks. Similarly, Lester et al. (2022) trained models to generate soft prompts for transferring between downstream models. Deb et al. (2022) also used a hypernetwork to modify downstream model parameters and applied their approach to Super-NaturalInstuctions (S-NI). They found that incorporating instructions via a hypernetwork trained with MAML (Finn et al., 2017) improved downstream performance. In work that became available shortly before submission, Ivison et al. (2022) applied hypernetworks to generate soft prefixes for T5 models. In contrast to this work, they fully tune the downstream model and apply fusion-in-decoder attention over the hyper-encoder inputs.

Multi-task Training and Transfer A crucial ingredient to hypertuning is the transferrability of task knowledge and generalization to novel tasks. Many past works (Phang et al., 2018; Pruksachatkun et al., 2020; Vu et al., 2020) have explored the effectiveness of single- and multi-task transfer learning. More recent work has shown that large-scale multi-task training tends allows models to generalize to unseen tasks (Sanh et al., 2022; Wei et al., 2022; Wang et al., 2022; Chung et al., 2022). Min et al. (2022) and Chen et al. (2022) show that few-shot learning also benefits from multi-task training. Pfeiffer et al. (2020), Vu et al. (2021), Gu et al. (2021), and Su et al. (2022) have also explored transfer learning among PEFT methods.

3. HyperTuning

The impetus for using hypermodels for adapting downstream models derives from two recent developments in natural language processing:

1) Large language models can perform in-context learning effectively. Large language models have been shown to be able to learn from the context of a small number of task examples or instructions, without any prior training on that task (Brown et al., 2020; Min et al., 2022; Wang et al., 2022). This suggests that models can “understand” what the task is and how to tackle it based on a few samples or a task description. This capability improves as the models get larger or are trained on more relevant data (Chowdhery et al., 2022; Ouyang et al., 2022; Bai et al., 2022).

2) Large language models can be adapted to downstream tasks by tuning a small set of parameters. Along with the growth in model sizes, there have been significant advances in fine-tuning methods that only modify a small number of parameters (possibly adding some new ones) in a frozen language model to adapt it to a specific task (Houlsby et al., 2019; Li & Liang, 2021; Lester et al., 2021; Ding et al., 2022). These methods often achieve performance comparable to fine-tuning all parameters in the model. Importantly, the number of parameters that need to be changed is small enough that it is feasible to train a model to generate them (Qin et al., 2021; Lester et al., 2022).

Taken together, these findings suggest that we may be able to use an auxiliary model that can first extract some task-relevant knowledge from some input that describes the task (e.g. instruction, few-shot examples), and then generate a small number of adaptive parameters, thereby changing the main model's behavior to suit the task. This approach, if successful, would enable us to adapt models to downstream applications without using backpropagation, or storing the encoded representations of few-shot examples in memory. In other words, we can delegate the work of model adaptation to a separate model.

We call this approach **hypertuning**, inspired by the work on hypernetworks by Ha et al. (2017). Hypertuning uses a *hypermodel* to adapt a *downstream model* to a target downstream task or application. This differs from *fine-tuning*, which uses backpropagation and a gradient descent algorithm to update model parameters. In this work, we present one possible formulation of hypertuning using few-shot examples and generating a small set of parameters with a single forward pass through the hypermodel. However, this is just one possible way of performing hypertuning, and the idea of adapting models with hypermodels can be generalized to many other cases. For example, hypermodels could also be trained to predict gradients or generate parameter updates based on input-output pairs. This way, hypermodels could work with large training sets, not just a few examples. Ultimately, with sufficiently general and well-trained hypermodels, we may be able to replace gradient-descent-based fine-tuning pipelines with hypertuning for many applications, while achieving similar or better performance.

3.1. HyperTuning with Fewshot Examples

Let M be a model with parameters θ, initialized at θ_0 from pretraining, and $\mathbb{L}$ a loss function. Given a dataset of size N with input-output pairs $\{(x, y)\}$, standard fine-tuning minimizes the following objective over θ:

$$\arg\min_{\theta} \frac{1}{N} \sum_{\{(x,y)\}} \mathbb{L}\Big(y, M(\theta; x)\Big) \tag{1}$$

In the case of parameter-efficient fine-tuning (PEFT), we fix $\theta = \theta_0$ and introduce a small set of trainable parameters ϕ (e.g. adapter parameters, soft prompts) that are injected into M. We optimize only over ϕ:

$$\arg\min_{\phi} \frac{1}{N} \sum_{\{(x,y)\}} \mathbb{L}\Big(y, M(\theta_0; x, \phi)\Big) \tag{2}$$

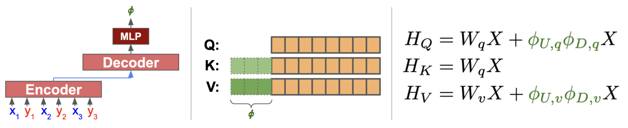
MLP
Decoder
Encoder
x_1 y_1 x_2 y_2 x_3 y_3
ϕ
Q:
K:
V:
ϕ
$H_Q = W_q X + \phi_{U,q}\phi_{D,q}X$
$H_K = W_q X$
$H_V = W_v X + \phi_{U,v}\phi_{D,v}X$
(A) HyperT5
(B) HyperT5-Prefix
(C) HyperT5-LoRA

Figure 2: Overview of HyperT5. (A) HyperT5 takes as input few-shot examples and outputs PEFT parameters ϕ. The model is initialized from an LM-adapted T5. (B) In HyperT5-Prefix, ϕ are key and value prefixes for every attention layer. (C) In HyperT5-LoRA, ϕ are additive low-rank modifications to the query and value linear maps.

For hypertuning, we further define a *hypermodel* H with parameters ξ that produces PEFT parameters $\hat{\phi}$ based on its input, which can be a set of few-shot examples or task instructions. For example, if the hypermodel input is a set of few-shot examples $\{(x_i, y_i)\}_K$, we have:

$$\hat{\phi} = H\Big(\xi; \{(x_i, y_i)\}_K\Big) \tag{3}$$

One way to train the hypermodel (H, ξ) is to perform PEFT on many tasks and use the resulting ϕ as targets. However, this is costly in computation, requiring many fine-tuning runs, and does not leverage cross-task knowledge transfer. Instead, we propose to train the hypermodel end-to-end, optimizing through the frozen model (M, θ_0). Hence, the hypermodel training objective is:

$$\arg\min_{\xi} \frac{1}{N} \sum_{\substack{\{(x,y)\} \\ \{\{(x_i,y_i)\}_K\}}} \mathbb{L}\Big(y, M(\theta_0; x, H(\xi; \{(x_i, y_i)\}_K))\Big) \tag{4}$$

At each training step, we sample a *target example* (x, y) and non-overlapping few-shot examples $\{(x_i, y_i)\}_K$. We generate $\hat{\phi}$ from the few-shot examples and compute the loss with respect to (x, y) and $\hat{\phi}$. We then back-propagate the gradients through both M and H to update ξ.

Note that since $\hat{\phi}$ does not depend on x, it can be computed once for a given set of few-shot examples and reused for downstream predictions. At inference time, we can use $\hat{\phi}$ directly without storing or recomputing the representations for $\{(x, y)\}, \{(x_i, y_i)\}_K$, saving memory and computation.[1]

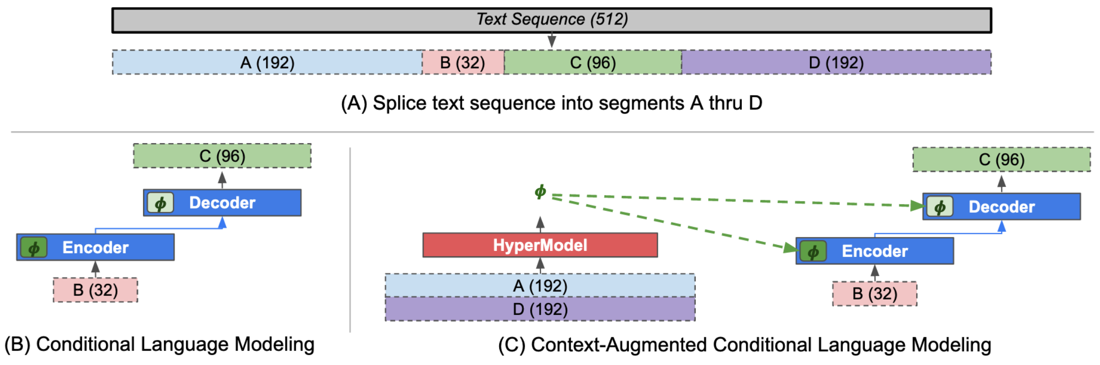
Text Sequence (512)
A (192)
B (32)
C (96)
D (192)
(A) Splice text sequence into segments A thru D
C (96)
Decoder
Encoder
B (32)
(B) Conditional Language Modeling
ϕ
HyperModel
A (192)
D (192)
C (96)
Decoder
Encoder
B (32)
(C) Context-Augmented Conditional Language Modeling

Figure 3: Overview of HyperPretraining using the Context-Augmented Conditional Language Modeling (CACLM) objective to train a hypermodel to predict PEFT parameters ϕ. (A) Sample a sequence of 512 tokens from a pretraining corpus, and splice into 4 segments A–D. (B) The frozen downstream model takes as input B and predicts continuation C. (C) The hypermodel is trained to encode additional context A and D into PEFT parameters ϕ, providing additional information to the downstream model to predict C.

4. HyperT5: A T5-Based HyperModel

4.1. Architecture and Setup

To demonstrate the feasibility of hypertuning, we propose *HyperT5*, a hypermodel based on T5, where both the hypermodel and the downstream model share a T5 backbone (Figure 2A). We use a frozen LM-adapted T5[2] as the downstream model. The hypermodel is also initialized with LM-adapted T5 parameters, but with some architectural changes. As defined in Equation 3, the hypermodel encoder takes the few-shot examples (and/or task definitions, in the case of S-NI) as input. The hypermodel decoder takes a fixed set of newly learned token embeddings as input, and its output representations are fed to a set of MLPs to generate the PEFT parameters ϕ for the downstream model. We also remove the causal masking from the decoder, since the hypermodel does not perform autoregressive generation.

We experiment with two PEFT methods: prefix tuning (Li & Liang, 2021) and LoRA (Hu et al., 2022). Prefix tuning (Figure 2B) prepends a set of learned key and value representations within each attention layer, while LoRA (Figure 2C) learns a low-rank additive modification to the query and value linear maps. Both PEFT methods have been shown to achieve good performance across a wide range of tasks (Ding et al., 2022). Chan et al. (2022) also suggest that modifying in-context representations and model weights can lead to different model behaviors, and we seek to demonstrate that hypertuning is applicable to very different PEFT methods. We name the respective hypermodels *HyperT5-Prefix* and *HyperT5-LoRA*. Additional architectural details and pseudo-code for both HyperT5-Prefix and HyperT5-LoRA models can be found in Appendix C.

[1]By construction, few-shot examples occupy at least K times the memory of the target input x.

[2]This is the model introduced by Lester et al. (2021). We use the T5 v1.1 architecture and initialize all experiments with the LM-adapted parameters, unless stated otherwise.

4.2. HyperPretraining

To train HyperT5, we first undergo an additional stage of pretraining to adapt the hypermodel to generate parameters ϕ for the downstream model, which we call *hyperpretraining*. As we show in Section 5.5, hyperpretraining is crucial for good hypermodel performance.

We propose a simple scheme for hyperpretraining using a *Context-Augmented Conditional Language Modeling* (CACLM) objective, which extends the conditional language-modeling (CLM) objective of T5 LM-adaptation. As shown in Figure 3, we sample a 512-token sequence from a pretraining corpus and split it into four consecutive segments A–D. The downstream model receives segment B as input and predicts segment C, following the CLM objective. The hypermodel receives segments A and D as input, which provide additional context from the same document, and outputs PEFT parameters for the downstream model.[3] The hypermodel thus compresses contextual information to assist the downstream model in its CLM task. We also make segment B very short (32 tokens) to encourage the downstream model to depend on the hypermodel information for accurate prediction of tokens in C. We compare to a prefix-only hyperpretraining scheme in Appendix D.

During hyperpretraining, we freeze the downstream model and only update the hypermodel parameters, training for 100K steps on C4 (Raffel et al., 2020). We perform hyperpretraining separately for HyperT5-Prefix and HyperT5-LoRA models. Hyperparameters can be found in Appendix A.

[3]Segments A and D are marked by sentinel tokens.

5. Multi-Task Fine-Tuning with HyperT5

5.1. Multitask Fine-Tuning (MTF)

After hyperpretraining, we conduct a second stage of training to train the hypermodel to generate task-specific PEFT parameters based on a small number of examples that we provide as input (Figure 1C). By performing multi-task fine-tuning on a sufficiently large number of tasks, we hope to have the hypermodel learn to generalize to generate parameters for unseen tasks. We adopt a similar training setup to MetaICL (Min et al., 2022), which uses multi-task fine-tuning (Sanh et al., 2022; Wei et al., 2022) with both a target input example (x) and a set of few-shot input-output pairs $\{(x_i, y_i)\}_K$ as inputs. The hypermodel takes the randomly sampled few-shot pairs as input, while the downstream model takes the target example as input, as shown in Equation 3. We fine-tune only the hypermodel parameters and keep the downstream model parameters fixed, unless otherwise stated. Appendix A.1 shows how we format the few-shot inputs.

We compare our approach with two baselines: multi-task fine-tuning of a T5 model without few-shot inputs, and MetaICL (multi-task fine-tuning with few-shot inputs). In MetaICL, the few-shot pairs are concatenated with the target example as input, both during training and evaluation on new tasks. We also include baselines that use PEFT methods for multi-task fine-tuning, i.e. learning a single set of prefix tuning or LoRA parameters.

We perform multi-task fine-tuning for 10,000 steps with a batch size of 256. For models that use few-shot inputs (MTF with fewshot, and hypermodels), we use up to 16 examples, and truncate tokens that exceed the maximum input length. At evaluation time, we randomly sample the few-shot inputs from the training sets of the respective datasets, for example evaluated example, though we fix the few-shot inputs seen by different models by using the same random seed. Appendix B provides more details on the datasets.

5.2. Datasets

To demonstrate the generality of our approach, we conduct experiments on three different multi-task training datasets, each with different held-out tasks and evaluation protocols.

Public Pool of Prompts (P3) (Sanh et al., 2022) consists of 62 task datasets, and was used to train the T0 models. Prompts are formatted with 0-shot inference in mind, and often contain instructions or answer options. For training our models, we use the T0-train subset. In order to fit multiple examples into the hypermodel's context, we further exclude dataset-prompt subsets with average input sequence lengths longer than 320 tokens. The list of included dataset-prompts can be found in Figure 7. Evaluation is performed on a fixed set of held-out tasks, based on multiple-choice scoring with accuracy. We exclude StoryCloze from evaluation as the task is not distributed with training data.

MetaICL (Min et al., 2022) introduced a few-shot multi-task training dataset, which is an extension of CrossFit (Ye et al., 2021) with UnifiedQA (Khashabi et al., 2020) and the addition of training data. For brevity, we will refer to this dataset as MetaICL. Unlike P3 and S-NI, the task inputs are not formatted for 0-shot inference; for instance, the task inputs may give no clue as to the goal of the task, or what the output space is. They provide several different train-task splits for tasks, of which we run our experiments on three (HR$\rightarrow$LR, Non-NLI$\rightarrow$NLI, Non-Class$\rightarrow$Class) to economize on computation costs. Evaluation is performed on held-out tasks, with ROUGE or Macro-F1 on model generations depending on the task.

Super-NaturalInstructions (S-NI) (Wang et al., 2022) consists of over 1,600 task datasets, each with a task definition and fixed sets of positive and negative examples. Following their findings, we focus our experiments on two settings: using only task definition as the hypermodel input, and using definitions and two positive examples. We only use the English tasks within the dataset. Evaluation is performed on held-out tasks using ROUGE-L on model generations.

5.3. Results

5.3.1. P3

Table 1 and Table 2 show the results of our experiments on the P3 dataset using T5-Large ($\sim$770M parameters) and T5-XL ($\sim$3B parameters), respectively.

We compare our HyperT5-Prefix and HyperT5-LoRA, which use hypermodels to generate task-specific PEFT parameters based on few-shot examples, with several baselines: prefix tuning, LoRA tuning, T5-MTF, and T5-MTF-Few-shot. T5-MTF is a model that roughly corresponds to the T0 model, and we detail the differences in Appendix B.1.

We find that HyperT5-Prefix and HyperT5-LoRA significantly outperform prefix and LoRA tuning baselines, indicating the effectiveness of using hypermodels to adapt the frozen downstream T5 LM to unseen tasks. HyperT5-Prefix achieves performance close to T5-MTF, while T5-MTF-Few-shot performs the best, in line with the findings of Min et al. (2022). These trends are consistent across T5-Large and T5-XL,[4] demonstrating the scalability of hypertuning.

We emphasize that HyperT5-Prefix/LoRA only introduces a very small number of PEFT parameters in the frozen downstream T5 model, whereas all parameters are tuned in the T5-MTF and T5-MTF-Few-shot models. Moreover,

[4]We note that T0-XL performs much worse than our trained T5-MTF, which is in agreement with other work (Wu et al., 2022) that report similar results in replicating T0.

	ANLI	HSwag	CB	COPA	RTE	WiC	WSC	WGD	AVG
Full Fine-Tuning									
T5-MTF	33.4	28.0	63.0	77.9	71.1	50.8	61.0	53.4	54.8
T5-MTF-Few-shot	35.3	27.5	68.6	70.5	75.2	51.7	62.1	52.2	55.4
Parameter-Efficient Fine-Tuning (PEFT)									
T5-MTF (Prefix)	33.1	26.1	53.9	67.8	60.5	49.8	54.7	51.4	49.7
T5-MTF (LoRA)	32.9	26.0	36.0	59.7	49.8	51.2	58.1	50.5	45.5
HyperTuning									
HyperT5-Prefix	33.4	32.3	60.1	73.9	71.5	51.1	63.0	51.1	54.6
HyperT5-LoRA	33.6	33.0	49.5	74.2	67.4	52.0	64.0	52.9	53.3
HyperTuning + Fine-Tuning									
HyperT5-Prefix+	34.5	32.2	58.1	78.4	76.5	50.4	63.8	54.3	56.0
HyperT5-LoRA+	33.9	30.7	62.1	75.8	72.3	50.8	64.6	54.5	55.6

Table 1: Results on P3 on held-out tasks (dev) with T5-Large models. T0 results taken from Sanh et al. (2022).

	ANLI	HSwag	CB	COPA	RTE	WiC	WSC	WGD	AVG
Full Fine-Tuning									
T5-MTF	39.9	29.4	64.5	88.0	80.8	51.7	60.7	57.9	59.1
T5-MTF-Few-shot	37.9	30.9	67.6	90.5	76.6	51.2	63.3	61.1	59.9
Parameter-Efficient Fine-Tuning (PEFT)									
T5-MTF (Prefix)	38.3	31.2	61.4	82.4	78.6	52.6	57.0	54.3	57.0
T5-MTF (LoRA)	33.9	26.4	47.1	67.2	53.3	50.8	51.5	50.3	47.6
HyperTuning									
HyperT5-Prefix	38.7	33.6	69.6	88.4	79.5	53.1	57.6	56.6	59.6
HyperT5-LoRA	35.3	30.8	66.4	83.3	68.5	50.3	60.0	56.1	56.4
Other results									
T0	33.4	27.3	45.4	73.1	64.5	50.7	65.0	51.0	51.3

Table 2: Results on P3 on held-out tasks (dev) with T5-XL models. T0 results taken from Sanh et al. (2022).

	HR →LR	Non-NLI →NLI	Non-Class →Class
Full Fine-Tuning			
T5-MTF	34.3	48.8	30.3
T5-MTF-Few-shot	41.0	56.7	40.6
Parameter-Efficient Fine-Tuning (PEFT)			
T5-MTF (Prefix)	29.8	42.8	29.6
T5-MTF (LoRA)	31.5	41.3	28.7
HyperTuning			
HyperT5-Prefix	38.0	58.3	38.6
HyperT5-LoRA	35.4	54.2	34.8

Table 3: Results on MetaICL (Test) with T5-Large models.

	AVG
Full Fine-Tuning	
T5-MTF (Def)	40.6
T5-MTF (Def+2Pos)	47.6
HyperTuning	
HyperT5-Prefix (Def)	37.1
HyperT5-Prefix (Def+2Pos)	43.5
HyperT5-LoRA (Def)	34.9
HyperT5-LoRA (Def+2Pos)	42.0
Other Results	
Tk-Instruct (Def+2Pos)	48.0

Table 4: Results on Super-NaturalInstuctions (S-NI; Test) with T5-Large models. Tk-Instruct results taken from Wang et al. (2022).

the P3 examples are written with prompt templates that are optimized for zero-shot inference, which is the ideal input format for T5-MTF. Furthermore, T5-MTF-Fewshot has full, bidirectional self-attention between the target input x and the few-shot examples, whereas HyperT5-Prefix and HyperT5-Lora only incorporate information from the few-shot examples via the respective PEFT parameters.

To investigate whether the hypermodel benefits are complementary to updating the downstream model parameters, we conduct an additional set of experiments where we jointly train both the hypermodel and the downstream model (HyperTuning + Fine-Tuning), with results shown at the bottom of Table 1. We observe that both HyperT5-Prefix+ and

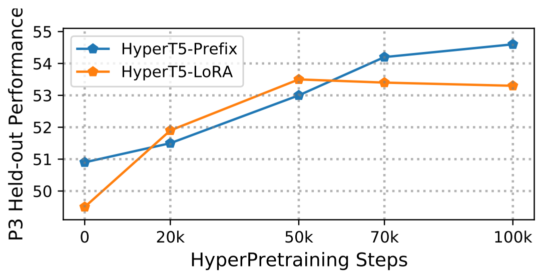

Figure 4: Performance of HyperT5 models on P3 evaluation with different amounts of hyperpretraining. HyperPretraining is crucial for good performance of the hypermodels. However, hyperpretraining for too many steps can also hurt performance (as see in the case of HyperT5-LoRA).

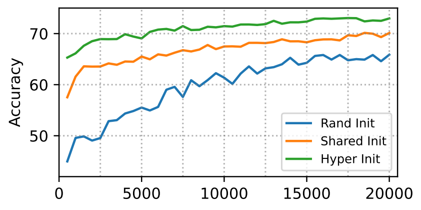

Figure 5: Prefix Tuning

Figure 6: Average performance on P3 held-out tasks with prefix tuning, using different parameter initializations. Using hypermodel-generated initializations starts with higher performance and continues to perform better on average over the course of training.

HyperT5-Lora+ slightly surpass T5-MTF-Fewshot, suggesting that the hypermodels can further enhance the performance of fine-tuned downstream models.

5.3.2. MetaICL

Table 3 shows the results on three MetaICL task splits. As in the previous experiments, both HyperT5 models outperform the PEFT models and T5-MTF. T5-MTF-Few-shot, outperforms HyperT5 models in all cases except Non-NLI→NLI, where HyperT5-Prefix achieves a higher score. T5-MTF performs poorly in MetaICL experiments as the MetaICL inputs are not designed for zero-shot inference.

5.3.3. Super-NaturalInstructions (S-NI)

We report the results on the different S-NI settings in Table 4 for T5-Large and Table 6 for T5-XL, using both Def (definition-only) and Def+2Pos (definition and two fixed positive examples) settings. The T5-MTF (Def) and T5-MTF (Def+2Pos) models are similar to the corresponding Tk-Instruct variants (Wang et al., 2022), with a slight difference in input formatting (see Appendix A.1). For the hypermodels, we prepend the task definitions to the few-shot examples and treat them as part of the hypermodel input. On average, the HyperT5 with Def+2Pos outperforms T5-MTF (Def) by a large margin, but still underperforms T5-MTF (Def+2Pos), in line with the above results.

5.4. Discussion

Above, we evaluated hypermodels on three multi-task datasets, where they generate task-specific soft prefixes or LoRA parameters from a few examples or instructions. In general, HyperT5 matched or exceeded T5-MTF models, but lagged behind T5-MTF-Fewshot models (or Def+2Pos models, in the case of S-NI). This gap is expected, as T5-MTF-Fewshot uses full self-attention between the examples and the target input x, while HyperT5 encodes the examples into PEFT parameters that are independent of x. We attribute some of the gap to this limitation.

However, this limitation also confers efficiency advantages to HyperT5 at inference time compared to T5-MTF-Fewshot. In encoder-decoders such as T5, the full self-attention between the examples and x means that a new forward pass is needed for each new x. In contrast, for hypermodels only need to encode the examples to generate parameters once, and the generated parameters can be reused for all inputs.

We also observe a consistent trend where HyperT5-Prefix outperforms HyperT5-LoRA. We speculate that it is easier for hypermodels to learn to generate soft prefixes as compared to LoRA weights, since soft prefix are effectively model-internal hidden states, and the generated PEFT parameters are themselves transformations of the hypermodel hidden states. Incidentally, another possible interpretation of the HyperT5-Prefix model is that the combination of the hypermodel and the downstream model can be seen as a dual-encoder, single-decoder model with separate encoders for the few-shot examples and the target example.

Lastly, most the experiments were conducted with minimal hyperparameter-tuning, and the current results primarily serve as a proof-of-concept of hypertuning being a viable approach to adapt downstream models. We expect that further exploration of hyperpretraining and MTF hyperparameters as well as hypermodel architectures may lead to better results and overcome some of the limitations we identified.

5.5. Is HyperPretraining Necessary?

We demonstrate the benefits of hyperpretraining for the hypermodels in this section. As mentioned in Section 3, we hyperpretrained the hypermodels for 100k steps before multi-task fine-tuning. To examine the impact of hyperpretraining, we also multi-task fine-tune HyperT5-Prefix and HyperT5-Lora models on the intermediate checkpoints of hyperpretraining. Figure 4 shows the average scores on the

	ANLI	HSwg	CB	COPA	RTE	WiC	WSC	WGD	AVG
Prefix (Rand Init)	54.6	50.5	98.8	79.0	78.8	71.6	63.5	52.2	68.6
Prefix (Shared Init)	60.8	51.6	99.4	85.7	84.8	72.4	72.6	65.1	74.0
Prefix (Hyper Init)	61.4	51.5	97.6	84.3	87.1	71.2	76.5	71.6	75.2
LoRA (Rand Init)	59.5	51.3	93.5	78.0	82.6	73.5	77.9	65.1	72.7
LoRA (Shared Init)	57.9	51.6	99.4	83.0	83.8	73.1	73.3	67.9	73.7
LoRA (Hyper Init)	57.7	48.4	99.4	87.3	84.1	73.0	83.9	66.2	75.0

Table 5: Prefix and LoRA tuning on T5-Large with different initializations on P3 held-out tasks. Using HyperT5-generated parameters as an initialization achieves better average performance than using MTF or randomly initialized PEFT parameters.

held-out tasks for these models. Both HyperT5 models perform very poorly without any hyperpretraining, achieving scores similar to PEFT-only (see Table 1). With hyperpretraining, the performance of both hypermodels significantly improves. While HyperT5-Prefix appears to consistently improve over the course of 100k steps, HyperT5-LoRA performance slightly declines after 50k steps. Hypermodels targeting different PEFT methods may benefit from different amounts of hyperpretraining, and our choice of hyperpretraining steps is by no means considered to be optimal.[5]

6. HyperModels for Improved Parameter Initialization

Thus far, we have discussed hypermodels in the context of generating PEFT parameters in a single forward pass through the hypermodel. We can also consider an alternative use of hypermodels: Instead of randomly initializing new parameters, we can use hypermodels to produce task-specific PEFT parameters based on a few examples from the task. This can be seen as using task knowledge acquired by the hypermodel during training to provide a first approximation of PEFT parameters, and thereafter refining the parmaeters via regular PEFT training.

In conventional PEFT, wherever new parameters are introduced into the model, they are either initialized randomly, or with fixed initial values (e.g. the up-projection weights in LoRA are initialized to 0)–for brevity, we will refer to this simply as random initialization. Beyond random initialization, Vu et al. (2021, SPoT) and Gu et al. (2021, PPT) have explored transfer-learning within PEFT, first doing PEFT on one or more upstream tasks, and then using the learned PEFT parameters as an initialization for downstream PEFT.

This approach has two advantages over conventional PEFT initializations. First, the hypermodel-generated parameters already perform well on the task, as shown in Section 5.3, so PEFT training can reach good performance faster. Second, the hypermodel can transfer relevant knowledge from previous tasks to the new task, similar to SPoT and PPT, except we let the hypermodel determine what previously learned task knowledge is most applicable to the new task.

[5]We chose 100k steps based on the T5 LM-adaptation procedure (Lester et al., 2021).

To investigate the effectiveness of using hypermodels to generate PEFT initializations, we use the P3-trained models from Section 5.3.1, and perform prefix and LoRA tuning on held-out tasks individually.[6] For each method-task pair, we sweep across learning rates $\{1e^{-3}, 1e^{-4}, 1e^{-5}\}$ and take the best average result over 3 random seeds.

We consider two baselines for initializations: random initialization (Rand Init) and using the multi-task fine-tuned PEFT parameters from Section 5.3.1 as initializations (Shared Init). The hypermodel-generated initialization (Hyper Init) is generated using a randomly sampled set of 16 examples from the respective training sets.

We show the results of prefix tuning[7] and LoRA tuning with different initialization schemes in Table 5. We observe that for both prefix tuning and LoRA tuning, shared initialization significantly performs random initialization, while using a hypermodel-generated initialization outperforms both on average. We also show the average performance across tasks over the course of tuning in Figure 6 and Figure 11. We observe that hypermodel-generated initializations start with much better performance compared to the other two initialization schemes, and continue to outperform them over the course of fine-tuning. Hence, hypermodels can be complementary to a standard PEFT pipeline, providing both performance gains and computational cost savings.

7. Societal Impact

While this work presents only an initial step toward hypertuning, it is worth discussing the societal impact of this line of research. The primary benefits of this line of work is to reduce the cost of model adaptation for downstream application, either to specific tasks or personalization of models. This has benefits in reducing both the economic and environmental costs of creating specialized models.

[6]We use one prompt for each task, listed in Appendix B.1.

[7]Prefix tuning is performed via a reparameterization, in line with standard practice. Refer to Appendix E for details.

However, it is also worth considering the negative impact of this research. One limitation is that the hypermodel's capacity to adapt the downstream model is fairly limited: both in terms of the extent to which the downstream model can be modified (only via parameter-efficient tuning methods, to reduce the output space of the hypermodel), but also the training data for the hypermodel. Compared to gradient descent-based optimization, we expect that hypertuning will be more constrained to model adaptation that is within the training data already seen by the hypermodel during training, and would not be able to adapt a model to an entirely new domain (e.g. fine-tuning a model to tackle a different language). This may leave low-resource domains or speakers or consumers of low-resource languages underserved by hypertuning-based model adaptation.

8. Conclusion

We introduce the concept of *hypertuning*, which leverages a hypermodel to adapt a downstream model to a specific downstream application. We present a basic framework for hypertuning, where a hypermodel is trained to produce parameters for a downstream model from few-shot examples in one forward pass, and we apply this framework to train HyperT5-Prefix and HyperT5-LoRA models that can adapt a fixed downstream T5 model. We find that a two-stage training procedure of hyperpretraining and multi-task fine-tuning is effective for training hypermodels, and we evaluate the HyperT5 models on P3, MetaICL and S-NI datasets, showing that they can generate PEFT parameters that enable the downstream T5 models to perform well on unseen tasks. Furthermore, the parameters generated by hypertuning can also serve as improved parameter initializations for parameter-efficient fine-tuning. We regard these findings as an initial but encouraging indication of the potential of adapting large language models without back-propagation.

Acknowledgements

We would like to thank Sam Bowman for his valuable feedback on this manuscript.

References

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., Joseph, N., Kadavath, S., Kernion, J., Conerly, T., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Hume, T., Johnston, S., Kravec, S., Lovitt, L., Nanda, N., Olsson, C., Amodei, D., Brown, T., Clark, J., McCandlish, S., Olah, C., Mann, B., and Kaplan, J. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback, 2022. URL https://arxiv.org/abs/2204.05862.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-shot Learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Chan, S. C. Y., Dasgupta, I., Kim, J., Kumaran, D., Lampinen, A. K., and Hill, F. Transformers generalize differently from information stored in context vs in weights, 2022. URL https://arxiv.org/abs/2210.05675.

Chen, Y., Zhong, R., Zha, S., Karypis, G., and He, H. Meta-learning via Language Model In-context Tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 719–730, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.53. URL https://aclanthology.org/2022.acl-long.53.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. PaLM: Scaling Language Modeling with Pathways, 2022. URL https://arxiv.org/abs/2204.02311.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, E., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei,

J. Scaling Instruction-finetuned Language Models, 2022. URL https://arxiv.org/abs/2210.11416.

Deb, B., Zheng, G., and Awadallah, A. H. Boosting Natural Language Generation from Instructions with Meta-learning, 2022. URL https://arxiv.org/abs/2210.11617.

Dettmers, T., Lewis, M., Shleifer, S., and Zettlemoyer, L. 8-bit Optimizers via Block-wise Quantization. *9th International Conference on Learning Representations, ICLR*, 2022.

Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chan, C.-M., Chen, W., Yi, J., Zhao, W., Wang, X., Liu, Z., Zheng, H.-T., Chen, J., Liu, Y., Tang, J., Li, J., and Sun, M. Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models, 2022. URL https://arxiv.org/abs/2203.06904.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic Meta-learning for Fast Adaptation of Deep Networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/finn17a.html.

Gu, Y., Han, X., Liu, Z., and Huang, M. PPT: Pre-trained Prompt Tuning for Few-shot Learning, 2021. URL https://arxiv.org/abs/2109.04332.

Ha, D., Dai, A. M., and Le, Q. V. HyperNetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

He, Y., Zheng, S., Tay, Y., Gupta, J., Du, Y., Aribandi, V., Zhao, Z., Li, Y., Chen, Z., Metzler, D., Cheng, H.-T., and Chi, E. H. HyperPrompt: Prompt-based task-conditioning of transformers. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 8678–8690. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/he22f.html.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-Efficient Transfer Learning for NLP. *CoRR*, abs/1902.00751, 2019. URL http://arxiv.org/abs/1902.00751.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Ivison, H., Bhagia, A., Wang, Y., Hajishirzi, H., and Peters, M. Hint: Hypernetwork instruction tuning for efficient zero-shot generalisation, 2022. URL https://arxiv.org/abs/2212.10315.

Karimi Mahabadi, R., Ruder, S., Dehghani, M., and Henderson, J. Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks. 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)*, pp. 565–576, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.47. URL https://aclanthology.org/2021.acl-long.47.

Khashabi, D., Min, S., Khot, T., Sabharwal, A., Tafjord, O., Clark, P., and Hajishirzi, H. UNIFIEDQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1896–1907, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.171. URL https://www.aclweb.org/anthology/2020.findings-emnlp.171.

Lester, B., Al-Rfou, R., and Constant, N. The Power of Scale for Parameter-Efficient Prompt Tuning, 2021. URL https://arxiv.org/abs/2104.08691.

Lester, B., Yurtsever, J., Shakeri, S., and Constant, N. Reducing Retraining by Recycling Parameter-Efficient Prompts, 2022. URL https://arxiv.org/abs/2208.05577.

Li, X. L. and Liang, P. 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)*, pp. 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.

Min, S., Lewis, M., Zettlemoyer, L., and Hajishirzi, H. MetaICL: Learning to Learn In Context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics:*

Human Language Technologies, pp. 2791–2809, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.201. URL https://aclanthology.org/2022.naacl-main.201.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Peebles, W., Radosavovic, I., Brooks, T., Efros, A. A., and Malik, J. Learning to learn with generative models of neural network checkpoints, 2022. URL https://arxiv.org/abs/2209.12892.

Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. AdapterFusion: Non-Destructive Task Composition for Transfer Learning. *CoRR*, abs/2005.00247, 2020. URL https://arxiv.org/abs/2005.00247.

Phang, J., Févry, T., and Bowman, S. R. Sentence Encoders on STILTs: Supplementary Training on Intermediate Labeled-data Tasks, 2018. URL https://arxiv.org/abs/1811.01088.

Pruksachatkun, Y., Phang, J., Liu, H., Htut, P. M., Zhang, X., Pang, R. Y., Vania, C., Kann, K., and Bowman, S. R. Intermediate-task transfer learning with pretrained language models: When and why does it work? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5231–5247, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.467. URL https://www.aclweb.org/anthology/2020.acl-main.467.

Qin, Y., Wang, X., Su, Y., Lin, Y., Ding, N., Yi, J., Chen, W., Liu, Z., Li, J., Hou, L., Li, P., Sun, M., and Zhou, J. Exploring Universal Intrinsic Task Subspace via Prompt Tuning, 2021. URL https://arxiv.org/abs/2110.07867.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. ZeRO: Memory Optimizations toward Training Trillion Parameter Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

Sanh, V., Webson, A., Raffel, C., Bach, S., Sutawika, L., Alyafeai, Z., Chaffin, A., Stiegler, A., Raja, A., Dey, M., Bari, M. S., Xu, C., Thakker, U., Sharma, S. S., Szczechla, E., Kim, T., Chhablani, G., Nayak, N., Datta, D., Chang, J., Jiang, M. T.-J., Wang, H., Manica, M., Shen, S., Yong, Z. X., Pandey, H., Bawden, R., Wang, T., Neeraj, T., Rozen, J., Sharma, A., Santilli, A., Fevry, T., Fries, J. A., Teehan, R., Scao, T. L., Biderman, S., Gao, L., Wolf, T., and Rush, A. M. Multitask Prompted Training Enables Zero-Shot Task Generalization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9Vrb9D0WI4.

Su, Y., Wang, X., Qin, Y., Chan, C.-M., Lin, Y., Wang, H., Wen, K., Liu, Z., Li, P., Li, J., Hou, L., Sun, M., and Zhou, J. 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*, pp. 3949–3969, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.290. URL https://aclanthology.org/2022.naacl-main.290.

Vu, T., Wang, T., Munkhdalai, T., Sordoni, A., Trischler, A., Mattarella-Micke, A., Maji, S., and Iyyer, M. Exploring and Predicting Transferability across NLP Tasks, 2020. URL https://arxiv.org/abs/2005.00770.

Vu, T., Lester, B., Constant, N., Al-Rfou, R., and Cer, D. SPoT: Better Frozen Model Adaptation through Soft Prompt Transfer. *CoRR*, abs/2110.07904, 2021. URL https://arxiv.org/abs/2110.07904.

Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Arunkumar, A., Ashok, A., Dhanasekaran, A. S., Naik, A., Stap, D., Pathak, E., Karamanolakis, G., Lai, H. G., Purohit, I., Mondal, I., Anderson, J., Kuznia, K., Doshi, K., Patel, M., Pal, K. K., Moradshahi, M., Parmar, M., Purohit, M., Varshney, N., Kaza, P. R., Verma, P., Puri, R. S., Karia, R., Sampat, S. K., Doshi, S., Mishra, S., Reddy, S., Patro, S., Dixit, T., Shen, X., Baral, C., Choi, Y., Smith, N. A., Hajishirzi, H., and Khashabi, D. Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks, 2022. URL https://arxiv.org/abs/2204.07705.

Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned Language Models are Zero-Shot Learners. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gEZrGCozdqR.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Wu, Z., IV, R. L. L., Walsh, P., Bhagia, A., Groeneveld, D., Singh, S., and Beltagy, I. Continued Pretraining for Better Zero- and Few-Shot Promptability. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, December 2022.

Ye, Q., Lin, B. Y., and Ren, X. CrossFit: A Few-shot Learning Challenge for Cross-task Generalization in NLP. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7163–7189, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.572. URL https://aclanthology.org/2021.emnlp-main.572.

A. Training Details

All experiments are trained with 1-bit Adam (Dettmers et al., 2022) and batch size of 256, a learning rate of 5e-5, and a linear decay schedule. Training was performed with ZeRO (Rajbhandari et al., 2020) and Transformers (Wolf et al., 2020). For hypermodels, the hypermodel's max input sequence length is 1024 tokens and the downstream model's max input sequence length is 384 tokens. Correspondingly, the max input sequence length for all non-few-shot models (e.g. T5-MTF, T5-MTF(Prefix)) is 384. The max input sequence length of few-shot models (e.g. T5-MTF-Few-shot) is thus conservatively set at 1024+384=1408 tokens. The max target sequence length is set to 128 for all experiments.

A.1. Input Formatting

Few-shot examples for hypermodels are formatted in the following manner:

```
<x> Input 1 <y> Target 1 <x> Input 2 <y> Target 2 <x> Input 3 <y> Target 3
```

where `<x>` and `<y>` and special tokens.

For S-NI, the task definitions are treated as just another example:

```
<x> Instruction <x> Input 1 <y>Target 1 <x> Input 2 <y>Target 2
```

B. Dataset-specific Details

B.1. P3 / T0

We highlight some differences our T0 baselines and the T0 setup described in the original paper (Sanh et al., 2022). Besides the different optimizers and batch sizes listed above, we do not use packing to process our training data. Moreover, because our focus is on few-shot learning, we remove a number of tasks formulations with longer inputs from the T0-train dataset, listed in Section 7. For T0, we use an input sequence length of 384 and output length of 128, which matches the input and output lengths of the downstream model in our hypermodel setup. For T5-MTF-Few-shot, we use an input sequence length of 1024+384=1408, which is the combined input lengths of the hypermodel and downstream model. We believe that these changes can meaningfully modify the performance of the T0 models, but provide a fairer baseline to the hypermodel setup.

adversarial_qa_dbert_answer_the_following_q, adversarial_qa_dbert_based_on, adversarial_qa_dbert_generate_question, adversarial_qa_dbert_question_context_answer, adversarial_qa_dbert_tell_what_it_is,
adversarial_qa_dbidaf_answer_the_following_q, adversarial_qa_dbidaf_based_on, adversarial_qa_dbidaf_generate_question, adversarial_qa_dbidaf_question_context_answer,
adversarial_qa_dbidaf_tell_what_it_is, adversarial_qa_droberta_answer_the_following_q, adversarial_qa_droberta_based_on, adversarial_qa_droberta_generate_question,
adversarial_qa_droberta_question_context_answer, adversarial_qa_droberta_tell_what_it_is, ag_news_classify, ag_news_classify_question_first, ag_news_classify_with_choices,
ag_news_classify_with_choices_question_first, ag_news_recommend, ag_news_which_section, ag_news_which_section_choices, amazon_polarity_Is_this_product_review_positive,
amazon_polarity_Is_this_review, amazon_polarity_Is_this_review_negative, amazon_polarity_User_recommend_this_product, amazon_polarity_convey_negative_or_positive_sentiment,
amazon_polarity_flattering_or_not, amazon_polarity_negative_or_positive_tone, amazon_polarity_user_satisfied, amazon_polarity_would_you_buy, app_reviews_categorize_rating_using_review,
app_reviews_convert_to_rating, app_reviews_convert_to_star_rating, app_reviews_generate_review, cnn_dailymail_3.0.0_generate_story, cnn_dailymail_3.0.0_spice_up_story, common_gen_Example_prompt,
common_gen_Given_concepts_type_1, common_gen_Given_concepts_type_2, common_gen_Put_together, common_gen_choice_in_concept_centric_sentence_generation,
common_gen_random_task_template_prompt, common_gen_sentence_to_concepts, common_gen_topic_to_sentence, common_gen_topics_from_the_sentence, cos_e_v1.11_aligned_with_common_sense,
cos_e_v1.11_description_question_option_id, cos_e_v1.11_description_question_option_text, cos_e_v1.11_explain_why_human, cos_e_v1.11_generate_explanation_given_text, cos_e_v1.11_i_think,
cos_e_v1.11_question_description_option_id, cos_e_v1.11_question_description_option_text, cos_e_v1.11_question_option_description_id, cos_e_v1.11_question_option_description_text,
cos_e_v1.11_rationale, cosmos_qa_context_answer_to_question, cosmos_qa_context_description_question_answer_id, cosmos_qa_context_description_question_answer_text,
cosmos_qa_context_description_question_text, cosmos_qa_context_question_description_answer_id, cosmos_qa_context_question_description_answer_text, cosmos_qa_context_question_description_text,
cosmos_qa_description_context_question_answer_id, cosmos_qa_description_context_question_answer_text, cosmos_qa_description_context_question_text, cosmos_qa_no_prompt_id,
cosmos_qa_no_prompt_text, cosmos_qa_only_question_answer, dbpedia_14_given_a_choice_of_categories_, dbpedia_14_given_a_list_of_category_what_does_the_title_belong_to,
dbpedia_14_given_list_what_category_does_the_paragraph_belong_to, dbpedia_14_pick_one_category_for_the_following_text, dream_answer_to_dialogue, dream_baseline, dream_generate_first_utterance,
dream_generate_last_utterance, dream_read_the_following_conversation_and_answer_the_question, duorc_ParaphraseRC_build_story_around_qa, duorc_SelfRC_build_story_around_qa, gigaword_TLDR,
gigaword_first_sentence_title, gigaword_generate_summary_for_this, gigaword_in_a_nutshell, gigaword_make_a_title, gigaword_reverse_writing, gigaword_write_a_title_for_this_sentence,
gigaword_write_an_article, gigaword_write_its_sentence, glue_mrpc_equivalent, glue_mrpc_generate_paraphrase, glue_mrpc_generate_sentence, glue_mrpc_paraphrase, glue_mrpc_replace,
glue_mrpc_same_thing, glue_mrpc_want_to_know, glue_qqp_answer, glue_qqp_duplicate, glue_qqp_duplicate_or_not, glue_qqp_meaning, glue_qqp_quora, glue_qqp_same_thing,
imdb_Movie_Expressed_Sentiment, imdb_Movie_Expressed_Sentiment_2, imdb_Negation_template_for_positive_and_negative, imdb_Reviewer_Enjoyment, imdb_Reviewer_Enjoyment_Yes_No,
imdb_Reviewer_Expressed_Sentiment, imdb_Reviewer_Opinion_bad_good_choices, imdb_Reviewer_Sentiment_Feeling, imdb_Sentiment_with_choices_, imdb_Text_Expressed_Sentiment,
imdb_Writer_Expressed_Sentiment, kilt tasks_hotpotqa_combining_facts, kilt tasks_hotpotqa_complex_question, kilt tasks_hotpotqa_final_exam, kilt tasks_hotpotqa_formulate,
kilt tasks_hotpotqa_straighforward_qa, paws_labeled_final_Concatenation, paws_labeled_final_Concatenation_no_label, paws_labeled_final_Meaning, paws_labeled_final_Meaning_no_label,
paws_labeled_final_PAWS_ANLI_GPT3, paws_labeled_final_PAWS_ANLI_GPT3_no_label, paws_labeled_final_Rewrite, paws_labeled_final_Rewrite_no_label, paws_labeled_final_context_question,
paws_labeled_final_context_question_no_label, paws_labeled_final_paraphrase_task, paws_labeled_final_task_description_no_label, qasc_is_correct_1, qasc_is_correct_2, qasc_qa_with_combined_facts_1,
qasc_qa_with_separated_facts_1, qasc_qa_with_separated_facts_2, qasc_qa_with_separated_facts_3, qasc_qa_with_separated_facts_4, qasc_qa_with_separated_facts_5, quarel_choose_between, quarel_do_not_use,
quarel_heres_a_story, quarel_logic_test, quarel_testing_students, quartz_answer_question_based_on, quartz_answer_question_below, quartz_given_the_fact_answer_the_q, quartz_having_read_above_passage,
quartz_paragraph_question_plain_concat, quartz_read_passage_below_choose, quartz_use_info_from_paragraph_question, quartz_use_info_from_question_paragraph, ropes_background_new_situation_answer,
ropes_background_situation_middle, ropes_given_background_situation, ropes_new_situation_background_answer, ropes_plain_background_situation, ropes_plain_bottom_hint, ropes_plain_no_background,
ropes_prompt_beginning, ropes_prompt_bottom_hint_beginning, ropes_prompt_bottom_no_hint, ropes_prompt_mix, ropes_read_background_situation, rotten_tomatoes_Movie_Expressed_Sentiment,
rotten_tomatoes_Movie_Expressed_Sentiment_2, rotten_tomatoes_Reviewer_Enjoyment, rotten_tomatoes_Reviewer_Enjoyment_Yes_No, rotten_tomatoes_Reviewer_Expressed_Sentiment,
rotten_tomatoes_Reviewer_Opinion_bad_good_choices, rotten_tomatoes_Reviewer_Sentiment_Feeling, rotten_tomatoes_Sentiment_with_choices_, rotten_tomatoes_Text_Expressed_Sentiment,
rotten_tomatoes_Writer_Expressed_Sentiment, samsum_Generate_a_summary_for_this_dialogue, samsum_Given_the_above_dialogue_write_a_summary, samsum_Sum_up_the_following_dialogue,
samsum_Summarize_, samsum_Summarize_this_dialogue_, samsum_To_sum_up_this_dialog, samsum_Write_a_dialogue_that_match_this_summary, sciq_Direct_Question, sciq_Direct_Question_Closed_Book_,
sciq_Multiple_Choice, sciq_Multiple_Choice_Closed_Book_, sciq_Multiple_Choice_Question_First, social_i_qa_Check_if_a_random_answer_is_valid_or_not, social_i_qa_Generate_answer,
social_i_qa_Generate_the_question_from_the_answer, social_i_qa_I_was_wondering, social_i_qa_Show_choices_and_generate_answer, social_i_qa_Show_choices_and_generate_index,
trec_fine_grained_ABBR, trec_fine_grained_ABBR_context_first, trec_fine_grained_DESC, trec_fine_grained_DESC_context_first, trec_fine_grained_ENTY, trec_fine_grained_HUM,
trec_fine_grained_HUM_context_first, trec_fine_grained_LOC, trec_fine_grained_LOC_context_first, trec_fine_grained_NUM, trec_fine_grained_NUM_context_first, trec_fine_grained_open,
trec_fine_grained_open_context_first, trec_pick_the_best_descriptor, trec_trec1, trec_trec2, trec_what_category_best_describe, trec_which_category_best_describes, wiki_bio_comprehension,
wiki_bio_guess_person, wiki_bio_key_content, wiki_bio_what_content, wiki_bio_who, wiki_qa_Decide_good_answer, wiki_qa_Direct_Answer_to_Question, wiki_qa_Generate_Question_from_Topic,
wiki_qa_Is_This_True_, wiki_qa_Jeopardy_style, wiki_qa_Topic_Prediction_Answer_Only, wiki_qa_Topic_Prediction_Question_Only, wiki_qa_Topic_Prediction_Question_and_Answer_Pair,
wiki_qa_automatic_system, wiki_qa_exercise, wiki_qa_found_on_google, wiqa_does_the_supposed_perturbation_have_an_effect, wiqa_effect_with_label_answer, wiqa_effect_with_string_answer,
wiqa_what_is_the_final_step_of_the_following_process, wiqa_what_is_the_missing_first_step, wiqa_what_might_be_the_first_step_of_the_process, wiqa_what_might_be_the_last_step_of_the_process,
wiqa_which_of_the_following_is_the_supposed_perturbation, yelp_review_full_based_on_that, yelp_review_full_format_rating, yelp_review_full_format_score, yelp_review_full_format_star,
yelp_review_full_on_a_scale, yelp_review_full_so_i_would, yelp_review_full_this_place

Figure 7: List of P3 dataset-prompts used for training. We chose a subset of T0-train with average input lengths shorter than 320 tokens.

For the hypermodel initialization/PEFT experiments, we do single-task parameter-efficient fine-tuning on each of the following dataset-prompts:

1. anli_GPT_3_style_r1
2. hellaswag_complete_first_then
3. super_glue_cb_GPT_3_style
4. super_glue_copa_C1_or_C2_premise_so_because_
5. super_glue_rte_GPT_3_style
6. super_glue_wic_GPT_3_prompt
7. super_glue_wsc.fixed_GPT_3_Style
8. winogrande_winogrande_debiased_Replace

B.2. S-NI / T-KI

To standardize the preprocessing across our experiments, we do not use the input formatting provided in the original work (Wang et al., 2022). Instead, we use the format described in Appendix A.1 for all experiments. Given that the same format is used in multi-task fine-tuning and evaluation, this should not unfairly advantage any model. However, because the format deviates from that of the original work, we do not directly evaluate the T-KI models.

Additionally, the Super-NaturalInstructions dataset (previously known as NaturalInstructions-v2) has undergone some changes over time. In our experiments, we use the v2.5 version of the dataset.

	AVG
Full Fine-Tuning	
T5-MTF (Def)	46.6
T5-MTF (Def+2Pos)	54.3
HyperTuning	
HyperT5-Prefix (Def)	38.9
HyperT5-Prefix (Def+2Pos)	48.6
HyperT5-LoRA (Def)	38.9
HyperT5-LoRA (Def+2Pos)	45.0
Other Results	
Tk-Instruct (Def+2Pos)	54.0

Table 6: Results on Super-NaturalInstuctions (S-NI; Test) with T5-XL models. Tk-Instruct results taken from Wang et al. (2022).

B.3. MetaICL

C. Model Details

The number of decoder input tokens and the size of the MLPs depend on the chosen PEFT method and its hyperparameters. For example, for HyperT5-Prefix that generates soft prefixes, ϕ will be of the shape $[L, 2, 2, P, H]$, where L is the number of layers, 2 is for the encoder and decoder, 2 is for the key and value prefixes, P is the number of prefix tokens, and H is the hidden size. We set the number of decoder input tokens to be $2P$. We provide pseudo-code for HyperT5-Prefix and HyperT5-LoRA models in the Figure 8 and Figure 9 in the Appendix.

We show the parameter counts for the respective hypermodels in Table 7.

```
# B = batch_size
# T = input_length
# P = number of prompt tokens
# H = hidden_dim
# L = num layers in encoder/decoder

# Shape: [B, T]
fewshot_input_ids = ...

# Shape: [B, T, H]
hyper_enc_out = hypermodel.encoder(fewshot_input_ids)

# Shape: [B, 2P, H]
# Decoder implicitly uses a fixed set of input embeddings of size 2P
hyper_dec_out = hypermodel.decoder(hyper_enc_out)

# Shape: [B, P, LH]
downstream_enc_k_prefix = hypermodel.enc_k_head(hyper_dec_out[:, :P, :])
downstream_enc_v_prefix = hypermodel.enc_v_head(hyper_dec_out[:, :P, :])
downstream_dec_k_prefix = hypermodel.dec_k_head(hyper_dec_out[:, P:, :])
downstream_dec_v_prefix = hypermodel.dec_v_head(hyper_dec_out[:, P:, :])

# Shape: [B, P, L H]
downstream_enc_k_prefix = downstream_enc_k_prefix.reshape(B, P, L, H)
downstream_enc_v_prefix = downstream_enc_v_prefix.reshape(B, P, L, H)
downstream_dec_k_prefix = downstream_dec_k_prefix.reshape(B, P, L, H)
downstream_dec_v_prefix = downstream_dec_v_prefix.reshape(B, P, L, H)
# These correspond to the per-layer learned prefixes for K and V


# where each of the heads is defined (e.g.):
hypermode.enc_k_head = nn.Sequential([
    nn.LayerNorm(),
    nn.Linear(H),
    nn.TanH(),
    nn.Linear(L*H),
])
```

Figure 8: Pseudo-code for HyperT5-Prefix

```
# B = batch_size
# T = input_length
# R = LoRA rank
# H = hidden_dim
# L = num layers in encoder/decoder

# Shape: [B, T]
fewshot_input_ids = ...

# Shape: [B, T, H]
hyper_enc_out = hypermodel.encoder(fewshot_input_ids)

# Shape: [B, 3L, H]
# Decoder implicitly uses a fixed set of input embeddings of size 3L
hyper_dec_out = hypermodel.decoder(hyper_enc_out)

# Shape: [B, L, H]
enc_repr = hyper_dec_out[:, :L, :]
dec_repr = hyper_dec_out[:, L:2*L, :]
cross_repr = hyper_dec_out[:, 2*L:, :]

# Repeat for dec_repr, cross_repr for decoder self- and cross-attention
# Shape: [B, L, 2RH]
enc_q_repr = hypermodel.enc_q_head(enc_repr)
enc_v_repr = hypermodel.enc_v_head(enc_repr)

# Shape: [B, L, 2RH]
enc_q_repr = enc_q_repr.reshape(B, L, 2, R, H)
enc_v_repr = enc_v_repr.reshape(B, L, 2, R, H)

# raw_enc_q_gate and raw_enc_v_gate are learned parameters of size [L]
# Shape: [1, L, 1, 1, 1]
enc_q_gate = torch.tanh(raw_enc_q_gate)[None, :, None, None, None]
enc_v_gate = torch.tanh(raw_enc_v_gate)[None, :, None, None, None]

# Shape: List of [B, R, H]
enc_lora_q_up_list = [enc_q_repr[:, l, 0, :, :] for l in range(L)]
enc_lora_q_down_list = [enc_q_repr[:, l, 1, :, :] for l in range(L)]
enc_lora_v_up_list = [enc_v_repr[:, l, 0, :, :] for l in range(L)]
enc_lora_v_down_list = [enc_v_repr[:, l, 1, :, :] for l in range(L)]
# These correspond to up- and down-map deltas in LoRA in Q and V
# attention linear maps


# where each of the heads is defined (e.g.):
hypermode.enc_q_head = nn.Sequential([
    nn.LayerNorm(),
    nn.Linear(H),
    nn.TanH(),
    nn.Linear(2*R*H),
])
```

Figure 9: Pseudo-code for HyperT5-LoRA

	Body	Param Heads	Total
HyperT5-Prefix-Large	750M	105M	855M
HyperT5-LoRA-Large	750M	107M	857M
HyperT5-Prefix-XL	2.78B	420M	3.20B
HyperT5-LoRA-XL	2.78B	428M	3.21B

Table 7: Parameter counts for the respective hypermodels. Param Heads refer to the new-initialized layers used to output the Prefix or LoRA parameters for the downstream model.

	AVG
P3	
HyperT5-Prefix (Prefix-only)	54.5
HyperT5-Prefix (Prefix-and-suffix)	52.6
Super-Natural Instructions	
HyperT5-Prefix (Def+2Pos, Prefix-only)	43.6
HyperT5-Prefix (Def+2Pos, Prefix-and-suffix)	39.1

Table 8: Prefix-only and Prefix-and-suffix HyperPretraining on T5-Large models

D. Ablations for HyperPretraining

Ivison et al. (2022) found that prefix-only pretraining worked better than prefix-and-suffix pretraining for HINT, another T5-based hypermodel. There are several major differences between HyperT5 and HINT, particularly that the downstream model is fully fine-tuned in HINT, whereas in HyperT5 the downstream model is frozen. Nevertheless, we conduct a short ablation comparing prefix-only and prefix-and-suffix hyperpretraining (Table 8). We find that prefix-only hyperpretraining performs worse that prefix-and-suffix hyperpretraining, justifying our use of the prefix-and-suffix setup for HyperT5.

E. Elaboration on Prefix Tuning Comparisons

While prefix tuning is generally presented as learning a set of prepended key and value representations for each Transformer layer, in practice, the learned prefixes are not optimized directly. In the work that introduced prefix tuning (Li & Liang, 2021), Section 4.3 explains that directly optimizing the learned prefixes leads to unstable training and poorer performance, and instead recommend optimizing a set of learned embeddings and a parameterized MLP to generate the learned prefixes. (At inference time, the prefixes can be generated from the learned components–this only impacts the training process.) We confirmed in our experiments that directly optimizing prefixes leads to poor perfomance, and other works involving prefix tuning have similarly used this prefix reparamterization

Hence, we have two flavors of prefix tuning to consider: directly optimizing over prefixes ("Prefix-Flat"), and optimizing with reparamterization ("Prefix-MLP"). The T5-MTF (Prefix) model uses Prefix-MLP, which is the appropriate approach to tuning prefixes in that setting. However, because HyperT5-Prefix only generates the final prefixes, only Prefix-Flat tuning is possible. Hence, when we perform the prefix tuning with different initializations in Section 6, we cannot fairly compare the two methods directly–one which uses a reparameterization during training, and the other which uses direct optimization which we know performs worse in practice.

Instead, we compare prefix tuning in the two different settings, Prefix-Flat and Prefix-MLP, completely separately. We describe each individual initialization scheme:

Prefix-Flat

1. Prefix-Flat (Rand): Randomly initialize soft prefixes
2. Prefix-Flat (Shared): Run a forward pass through the prefix reparameterization to obtain the flat prefixes, and use them as the initialization
3. Prefix-Flat (Hyper): Generate prefixes with HyperT5-Prefix

Prefix-MLP

1. Prefix-MLP (Rand): Randomly initialize the prefix reparameterization embeddings and MLPs (i.e. conventional prefix tuning)
2. Prefix-MLP (Shared): Directly reuse the prefix reparameterization from T5-MTF (Prefix)
3. Prefix-MLP (Hyper): We train an entirely new HyperT5-Prefix-MLP model, where the parameter generation heads directly correspond to the prefix tuning reparameterization MLPs. The encoder-decoder in the hypermodel will output the "embeddings", and we directly reuse the parameter generation heads during tuning.

The results for Prefix-MLP are presented in the body of the paper in Section 6. We believe that this approach provides the fairest comparison of initializations. Importantly, both Prefix-MLP (Shared) and Preflix-MLP (Hyper) have been trained on the same number of labeled examples (not including the few-shot examples, which are inputs), but where the Prefix-MLP uses a single set of learned embeddings, HyperT5-Prefix-MLP generates the embeddings based on few-shot examples.

We present the full set of prefix tuning results in Table 9, the performance of Prefix-Flat Figure 12.

	ANLI	HSwag	CB	COPA	RTE	WiC	WSC	WGD	AVG
Prefix-Flat (Rand Init)	43.6	36.3	82.7	74.0	72.9	64.4	64.2	53.0	61.4
Prefix-Flat (Shared Init)	54.3	40.4	98.8	82.7	83.9	71.0	67.4	57.1	69.4
Prefix-Flat (Hyper Init)	56.6	43.5	91.7	84.3	85.3	69.3	73.0	67.6	71.4
Prefix-MLP (Rand Init)	54.6	50.5	98.8	79.0	78.8	71.6	63.5	52.2	68.6
Prefix-MLP (Shared Init)	60.8	51.6	99.4	85.7	84.8	72.4	72.6	65.1	74.0
Prefix-MLP (Hyper Init)	61.4	51.5	97.6	84.3	87.1	71.2	76.5	71.6	75.2
LoRA (Rand Init)	59.5	51.3	93.5	78.0	82.6	73.5	77.9	65.1	72.7
LoRA (Shared Init)	57.9	51.6	99.4	83.0	83.8	73.1	73.3	67.9	73.7
LoRA (Hyper Init)	57.7	48.4	99.4	87.3	84.1	73.0	83.9	66.2	75.0

Table 9: Prefix tuning (Flat and MLP) and LoRA fine-tuning on T5-Large with different initializations on P3 held-out tasks.

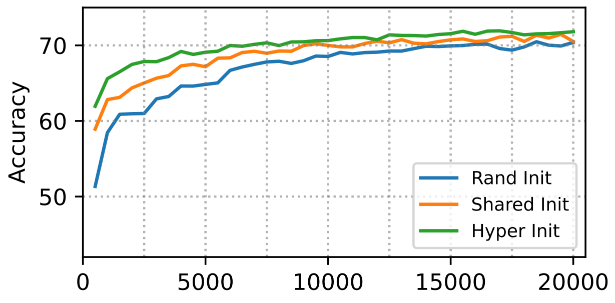

Figure 10: LoRA

Figure 11: Average performance on P3 held-out tasks with LoRA, using different parameter initializations.

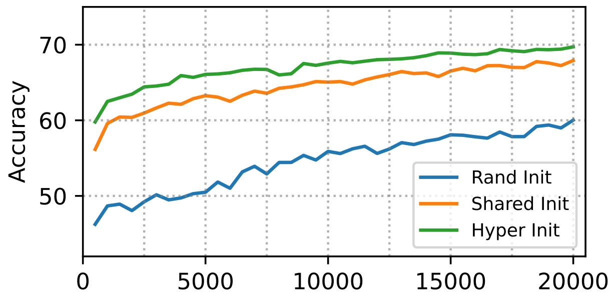

Figure 12: Average performance on P3 held-out tasks with prefix tuning (flat), using different parameter initializations