# Hyperdecoders: Instance-specific decoders for multi-task NLP

**Hamish Ivison** and **Matthew E. Peters**
Allen Institute for AI
{hamishi, matthewp}@allenai.org

## Abstract

We investigate input-conditioned hypernetworks for multi-tasking in NLP, generating parameter-efficient adaptations for a decoder using a hypernetwork conditioned on the output of an encoder. This approach produces a unique decoder adaptation for every input instance, allowing the network a larger degree of flexibility than prior work that only produces one decoder adaptation per task. We apply our method to sequence classification tasks, extractive QA, and summarisation and find that it surpasses previous parameter efficient fine-tuning methods and often outperforms fully finetuning the underlying model. An analysis of the embeddings used by our hypernetwork shows that they are sensitive to output label and type, suggesting that our approach better maps from encoder representations to output labels. Our code is publicly available at https://github.com/allenai/hyperdecoders.

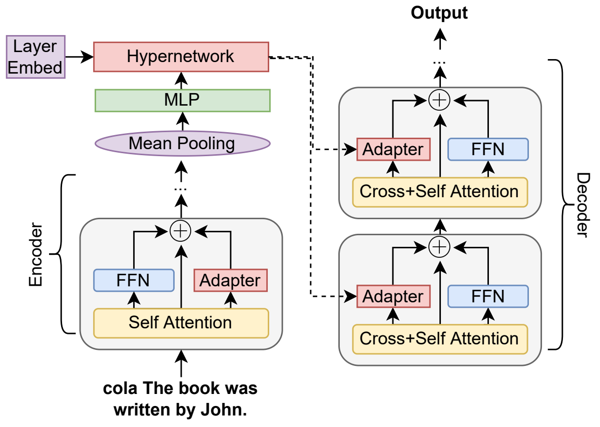

Figure 1: An overview of our proposed approach, where a hypernetwork generates the adapters for the decoder in an encoder-decoder model. Given an input instance and task name, an encoder produces an embedding which is used to generate decoder adapter parameters using a hypernetwork.

## 1 Introduction

Recent work in NLP has examined the performance of large pretrained transformer-based models in multi-task settings, where a single model is evaluated on multiple tasks simultaneously, often with tasks converted to a shared sequence-to-sequence format (Raffel et al., 2020; Brown et al., 2020). This greatly simplifies model training and deployment, requiring only one deployed model and format to handle multiple tasks. Additionally, training across multiple tasks can result in greatly improved performance for similar tasks (Phang et al., 2018), as well as tasks not seen during training (Sanh et al., 2022; Wei et al., 2022). However, not all tasks work well together, and jointly training on certain task pairs can reduce performance on both ('negative transfer') (Aribandi et al., 2022).

In this paper, we propose a new method for multi-task NLP using a hypernetwork to generate an instance-specific decoder from the output of an encoder. We effectively explore if a model can learn to adapt *itself* through learning how to generate adapter layers. Our approach produces significant gains over prior approaches for efficient multi-task fine-tuning, often matching or exceeding full fine-tuning the underlying model.

We build on parameter-efficient learning methods, where one trains a small set of parameters within a much larger model. These parameters may be newly introduced (Houlsby et al., 2019; Li and Liang, 2021) or already exist within the model (Zaken et al., 2021), and are kept as few as possible. This means these methods often do not contain the capacity to learn multiple tasks at once, losing potential transfer benefits. One way to remain parameter-efficient while still handling a variety of tasks is to instead learn to generate these parameters, making use of an auxiliary network ('hypernetwork') to generate the weights used during inference (Tay et al., 2021; Pilault et al., 2021; Ye and Ren, 2021; Karimi Mahabadi et al., 2021). This allows the model to benefit from positive transfer between tasks through the shared hypernetwork while reducing negative transfer by allowing the generated parameters to be unique per task.

However, hypernetwork-based approaches generally condition their parameters on a learnt task embedding, meaning (a) the model is the same for every example within a task, and (b) adapting to new tasks requires further training to learn new task embeddings. We increase the flexibility of this approach by instead *generating unique parameters for every input*, allowing the model to make use of similarities between samples across datasets and avoid potential interference between samples within the same dataset. This is achieved by using a shared encoder across all tasks and then generating adapter layers for the decoder only by feeding the encoded inputs into a hypernetwork. Furthermore, by conditioning on inputs rather than task embeddings, our approach allows simple transfer to out-of-domain data, as the shared hypernetwork and encoder learn to map from text to parameters. Figure 1 illustrates our approach.

We apply our approach to a diverse set of tasks, including sequence classification, extractive question answering, and summarisation, and find that our approach outperforms existing parameter-efficient approaches and matches or outperforms full-finetuning. An analysis of our approach shows that sharing parameters in the encoder, but generating them in the decoder is more effective than other possible setups, suggesting that the encoder benefits from multi-task training while the decoder does not. Our results suggest that pretrained encoders can be easily adapted and trained to produce adaptations that enhance multi-task transfer learning for decoders, while producing useful adaptations for encoders is much more difficult.

To summarise, our core contributions are:

1. We propose a new method for parameter-efficient multi-tasking, generating unique decoder layers for every input into a model.
2. We show that our approach performs strongly against other parameter-efficient baselines and fully finetuning the underlying model across a diverse set of NLP tasks.
3. We show the embeddings learnt by our hypernetwork are sensitive to both dataset and output label, suggesting the hypernetwork is effectively controlling the decoder.

## 2 Encoder-conditioned Decoders

We make use of T5 as the underlying model in our experiments, which is a popular encoder-decoder model for sequence-to-sequence multi-tasking (Raffel et al., 2020) and a common starting point for previous hypernetwork-based approaches (Karimi Mahabadi et al., 2021; Tay et al., 2021). However, our overall approach can be applied to generic encoder-decoder transformer models.

### 2.1 Adapter Layers

We augment our underlying model with adapter layers (Houlsby et al., 2019). These are small bottleneck networks with the following form:

$$\text{Adapter}(\mathbf{x}) = \mathbf{W}_u(f(\mathbf{W}_d\mathbf{x} + \mathbf{b}_d)) + \mathbf{b}_u \quad (1)$$

Where $f$ is the ReLU activation function. We insert these layers in parallel with the feedforward module of each layer of a larger pretrained model, following He et al. (2022a):

$$\mathbf{y} = \text{FF}(\text{LayerNorm}(\mathbf{x})) + \text{Adapter}(\mathbf{x}) \quad (2)$$

Where $\mathbf{y}$ is the output for the layer, FF is the feedforward module, and $\mathbf{x}$ is the output from the attention module(s). We find using LayerNorm$(\mathbf{x})$ as input to the adapter less effective than directly using $\mathbf{x}$ (see section 5.2). During training, the underlying network is frozen and only $\mathbf{W}_u$, $\mathbf{W}_d$, $\mathbf{b}_u$, and $\mathbf{b}_d$ are updated.

### 2.2 Adapter-generating Hypernetworks

A hypernetwork is a network that produces the parameters used for another network (Ha et al., 2017; Schmidhuber, 1991). We use a simple two-layer network to produce adapter parameters $\mathbf{W}_d$, $\mathbf{W}_u$, $\mathbf{b}_d$, $\mathbf{b}_u$. Given some input $x$ to the hypernetwork, we generate these parameters as follows:

$$\mathbf{h} = \text{ReLU}(\mathbf{W}_0\mathbf{x} + \mathbf{b}_0) \quad (3)$$
$$\mathbf{W}_u = \mathbf{W}_1\mathbf{h} + \mathbf{b}_1 \quad (4)$$
$$\mathbf{W}_d = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 \quad (5)$$
$$\mathbf{b}_u = \mathbf{W}_3\mathbf{h} + \mathbf{b}_3 \quad (6)$$
$$\mathbf{b}_d = \mathbf{W}_4\mathbf{h} + \mathbf{b}_4 \quad (7)$$

We re-use this hypernetwork to generate the adapters for every layer by (partially) conditioning the input on layer embeddings, greatly improving the parameter efficiency of this approach. The hypernetwork parameters are initialised using the method proposed in Chang et al. (2020).

### 2.3 Encoder-conditioned Decoders

The core idea of our approach is to condition a hypernetwork on the output of the encoder to generate the adapters used for the decoder in an encoder-decoder model. We place regular (non-generated) adapter layers in the encoder to allow it to adapt to tasks in a parameter-efficient way. We then feed the encoder output to a hypernetwork to generate custom decoder adapters for every input to the encoder. This allows our approach to flexibly differentiate between samples within a task, rather than remaining static per task. This potentially allows more flexible transfer learning both between samples within datasets and with samples across datasets. We explore other potential configurations in section 5.2 and find ours works best overall. We name this approach 'hyperdecoder' in the following experiments.

Concretely, given some input, we first pass it through the T5 encoder to construct a hidden representation $\mathbf{h}$. The T5 encoder is equipped with adapter layers, which are trained during fine-tuning while the rest of the encoder is kept frozen. We mean-pool this representation and pass it through a two-layer network with a ReLU activation to construct a vector embedding of the input:

$$\mathbf{e} = \mathrm{MLP}(\mathrm{mean}(\mathbf{h})) \tag{8}$$

We then generate the parameters for an adapter in each layer $i$ of the decoder by concatenating a learnt layer embedding $\mathbf{l}_i$ to the embedding and passing it through the hypernetwork described in section 2.2.

$$\mathrm{Adapter}_i = \mathrm{Hypernetwork}([\mathrm{e}; \mathbf{l}_i]) \tag{9}$$

All parameters in the decoder are frozen and the hypernetwork is trained along with the encoder adapters during fine-tuning. This approach is summarised in Figure 1.

### 2.4 Multi-tasking

We focus on a multi-task setup, where a model is trained on multiple tasks simultaneously. The underlying model parameters $\theta$ are kept unchanged during training and only the adapter and hypernetwork parameters $\theta'$ are updated. All parameters are shared between all tasks, and the model is trained in a seq2seq setting with cross-entropy loss, following Raffel et al. (2020).

## 3 Experiments

This section details our experimental setup and baselines. Results are given in Section 4.

### 3.1 Datasets

We evaluate our approach in three settings: GLUE (Wang et al., 2018), the 2019 MRQA shared task (Fisch et al., 2019), and a set of summarisation and NLI datasets. For each setting, we sample from sub-tasks proportionally to their size, as initial experiments showed more complex sampling techniques provided little to no benefit. All tasks are in English.

#### 3.1.1 GLUE

GLUE (Wang et al., 2018) is a set of sequence classification tasks including paraphrase detection (QQP and MRPC; Dolan and Brockett, 2005), semantic similarity (STS-B; Agirre et al., 2007), natural language inference (MNLI; Williams et al., 2018), (QNLI; Rajpurkar et al., 2016), (RTE; Dagan et al., 2006; Bar Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), linguistic acceptibility (CoLA; Warstadt et al., 2018), and sentiment classification (SST-2; Socher et al., 2013). Following Karimi Mahabadi et al. (2021), for datasets with small training sets (RTE, MRPC, STS-B, CoLA), we split the validation set in half into test and validation sets, for other larger datasets we split out 1000 examples to use as the validation set and use the original validation set as a test set, and for MNLI we use the mismatched validation set as the test set. Following prior work we do not evaluate on WNLI. We preprocess the GLUE inputs to follow the format used by Raffel et al. (2020) (including task prefix).

#### 3.1.2 MRQA

The MRQA 2019 shared task dataset (Fisch et al., 2019) is a collection of 12 QA datasets, all modified to the same format of extractive QA. 6 datasets are used for training and evaluation: HotpotQA (Yang et al., 2018), Natural Questions (Kwiatkowski et al., 2019), NewsQA (Trischler et al., 2017), SQuAD (Rajpurkar et al., 2016), SearchQA (Dunn et al., 2017), and TriviaQA (Joshi et al., 2017). Another 6 are used for out-of-domain evaluation: BioASQ (Tsatsaronis et al., 2015), DROP (Dua et al., 2019), DuoRC (Saha et al., 2018), RACE (Lai et al., 2017), RelationExtraction (Levy et al., 2017), and TextbookQA (Kembhavi et al., 2017). This effectively tests a model's ability to generalise to out-of-domain data. We evaluate on the validation split of all 12 datasets after all training steps are complete. We preprocess all MRQA data to follow the SQuAD template used by Raffel et al. (2020)[1]. Notably, this prompt does not provide any indication as to which dataset a given input belongs.

[1]'`question: <question> context: <context>`'.

| Model | % Trainable Param. | CoLA | SST-2 | STS-B | MRPC | QQP | MNLI | QNLI | RTE | Avg |
|---|---|---|---|---|---|---|---|---|---|---|
| Full Finetuning | 100% | **63.6** | 94.8 | **92.0** / 91.6 | 88.7 / 91.8 | **92.2 / 89.5** | 88.6 | 93.3 | 77.5 | 86.3 |
| Hyperformer | 8.8% | 19.2 | 87.3 | 86.2 / 85.8 | 73.4 / 81.3 | 87.0 / 82.8 | 77.7 | 84.2 | 55.1 | 71.5 |
| Hyperformer++ | 4.6% | 35.5 | 88.3 | 87.9 / 87.6 | 78.8 / 85.2 | 86.4 / 82.3 | 88.3 | 84.8 | 64.5 | 76.9 |
| Task Hypernet | 2.7% | 0.0 | 82.1 | 16.4 / 16.4 | 70.4 / 81.4 | 89.8 / 86.5 | 56.6 | 64.8 | 50.7 | 54.3 |
| Modular Hypernet | 4.8% | 51.2 | 96.2 | 91.9 / **92.0** | **90.1 / 93.0** | 89.4 / 86.1 | 89.5 | 93.5 | 74.6 | 84.5 |
| Adapter | 2.9% | $58.5_{3.1}$ | $95.7_{0.3}$ | $90.1_{1.8}$ / $90.3_{1.9}$ | $89.4_{1.5}$ / $92.2_{1.0}$ | $91.4_{1.9}$ / $88.6_{0.3}$ | $89.8_{0.1}$ | $94.1_{0.2}$ | $80.7_{1.8}$ | $86.2_{0.6}$ |
| **Hyperdecoder (ours)** | 2.9% | $58.7_{2.3}$ | $\mathbf{95.9}_{0.4}$* | $91.8_{0.7}$ / $\mathbf{92.0}_{0.4}$* | $89.2_{1.5}$ / $92.0_{0.9}$ | $91.1_{0.2}$ / $88.3_{0.4}$ | $\mathbf{90.0}_{0.2}$* | $\mathbf{94.2}_{0.4}$ | $\mathbf{80.8}_{2.2}$ | $\mathbf{86.5}_{0.5}$† |

Table 1: Performance of models using **T5$_{\text{large}}$ v1.1 + LM** as the base model on GLUE test splits described in section 3.1.1. Trainable parameters is % of parameters trained in the model compared to fully finetuning T5. We report mean and standard deviation over 25 runs for Adapter and Hyperdecoder. * indicates value is statistically significant ($p < 0.05$). † Hyperdecoder is statistically significantly better when CoLA is removed from average.

We split contexts into chunks of length 512 tokens with an overlap of 128 tokens. We pair chunks with answers with the answer found in the chunk, and chunks without answers with empty strings. At evaluation time, we produce an answer for all chunks and take the most likely non-empty string as the final answer.

### 3.1.3 Summarisation and NLI

To investigate transfer learning with difficult tasks, we evaluate on summarisation and NLI tasks that are known to cause negative interference (Aribandi et al., 2022) and are difficult for current parameter-efficient techniques (He et al., 2022a). For summarisation, we use Xsum (Narayan et al., 2018), CNN/Daily Mail (Hermann et al., 2015), and the English WikiLingua split from Gehrmann et al. (2021). For NLI, we use MNLI (Williams et al., 2018), abductive NLI (Bhagavatula et al., 2020), and adversarial NLI (Nie et al., 2020). We jointly train and evaluate on all tasks. We preprocess all datasets following the templates used in Raffel et al. (2020)[2]. We evaluate on the provided test splits for all datasets except abductive NLI and MNLI. For abductive NLI, we split 1000 samples from the validation set and use the existing validation set as the test set. We treat MNLI as detailed in section 3.1.1.

## 3.2 Experimental Details

We build on the Hyperformer codebase (Karimi Mahabadi et al., 2021), making use of the transformers implementation of T5 (Wolf et al., 2020). We finetune all models using AdamW (Loshchilov and Hutter, 2019), with a learning rate of 3e-4 with linear decay and 500 warmup steps. For GLUE tasks, we train for 65k steps with an effective batch size of 128, evaluate every 1000 steps on the development set, and test on the overall best performing checkpoint. For MRQA, we train for 4 epochs and evaluate on the final model (initial experiments showed that taking checkpoints always resulted in evaluating on the final model regardless). For summarisation and NLI tasks, we train for 100k steps with a batch size of 64, evaluate every 5000 steps on the development set, and test using the single overall best-performing checkpoint. All non-hypernetwork and non-adapter parameters are frozen throughout training. All experiments start from the T5 v1.1+LM checkpoints (Lester et al., 2021) unless otherwise stated. Further details can be found in Appendix B.

[2]This means that the NLI tasks have their dataset name prepended onto the prompt, while summarisation tasks do not.

## 3.3 Baselines

We primarily compare against three strong baselines: fully-finetuning the underlying model, training only adapter layers placed in parallel with all feedforward modules ('adapter') and using task-conditioned hypernetworks to generate adapter layers in the encoder and decoder ('task hypernet'), similar to the Hyperformer (Karimi Mahabadi et al., 2021). Apart from full finetuning, we keep the number of trainable parameters roughly equal across methods. More details are provided in Appendix A.

We additionally compare against relevant prior work. For GLUE, we compare against the Hyperformer and Hyperformer++ models proposed in Karimi Mahabadi et al. (2021) with an increased number of trainable parameters[3], and the modular task hypernetwork (Ponti et al., 2022). For MRQA, we compare against CA-MTL (Pilault et al., 2021), which modifies a BERT model with several task-conditional modules and uses a novel

[3]We increased the number of trainable parameters for Hyperformer as a best-faith effort to provide a strong baseline as the default settings tended to result in worse performance in initial experiments.

| Model | % Trainable Param. | CoLA | SST-2 | STS-B | MRPC | QQP | MNLI | QNLI | RTE | Avg |
|---|---|---|---|---|---|---|---|---|---|---|
| Hyperformer* | ∼4% | **58.9** | 95.7 | 91.6 / 91.5 | 92.7 / 90.0 | 87.7 / 90.7 | 89.8 | 94.5 | 87.0 | 87.3 |
| HyperPrompt* | ∼4% | 57.5 | **96.7** | **91.9 / 92.0** | 93.6 / 91.2 | 87.0 / 90.1 | 90.3 | **95.0** | 87.7 | 87.5 |
| **Hyperdecoder (ours)** | 4.2% | 58.2 | 96.4 | 91.5 / 91.6 | **93.8 / 91.4** | **89.3 / 91.9** | **90.7** | 94.8 | **88.4** | **87.9** |

Table 2: Performance of models using **$T5_{large}$ v1.1 + LM** as the base model on GLUE dev set splits, picking the best task performance across all checkpoints. Trainable parameters is % of the trainable parameters used compared to fully finetuning T5. * Results from He et al. (2022b).

| Model | % Trainable Param. | CoLA | SST-2 | STS-B | MRPC | QQP | MNLI | QNLI | RTE | Avg |
|---|---|---|---|---|---|---|---|---|---|---|
| Full Finetuning* | 100% | 54.9 | 92.5 | 88.8 / 88.5 | 90.2 / 93.0 | **91.1 / 88.1** | 85.7 | 92.0 | 75.4 | 83.8 |
| Hyperformer* | 54% | 61.3 | 93.8 | 89.6 / 89.0 | **90.6 / 93.3** | 87.2 / 90.1 | **86.3** | 92.8 | **78.3** | **85.3** |
| Hyperformer++* | 2% | **63.7** | **94.0** | 90.0 / 89.7 | 89.7 / 92.6 | 87.2 / 90.3 | 85.7 | 93.0 | 75.4 | 85.2 |
| **Hyperdecoder (ours)** | 7.4% | 54.8 | 93.8 | **90.3 / 90.2** | 86.2 / 90.5 | 90.5 / 87.3 | 85.8 | **93.4** | 71.0 | 83.3 |

Table 3: Performance of models using **$T5_{base}$ vanilla** as the base model on GLUE test splits described in section 3.1.1. * Results reported by Karimi Mahabadi et al. (2021).

sampling technique. We also compare to UnifiedQA (Khashabi et al., 2020) results reported by Friedman et al. (2021) to show out-of-domain split performance from alternate T5-based QA model.

## 4 Results

This section details our main experimental results. Ablations and analysis follow in Section 5.

### 4.1 GLUE

We report our results on the GLUE benchmark in Tables 1 and 3. Our approach improves greatly over the Hyperformer and full finetuning when using T5 v1.1 + LM as the underlying model. Notable improvements are made in SST-2 and RTE tasks, the latter of which is known to benefit from transfer learning (Phang et al., 2018). This suggests our approach enhances positive transfer benefits over adapter-only and full finetuning approaches. The worst performing approach is the 'task hypernet' method, which follows the Hyperformer approach but with our adapter placement. This suggests that the adapter placement difference between our method and the Hyperformer is not the reason for our improved performance, but rather the use of an encoder-conditioned decoder. Additionally, our approach remains parameter efficient, training 0.03× fewer parameters than full finetuning. Our approach also outperforms HyperPrompt (He et al., 2022b) when using a matching evaluation setup[4], as seen in 2.

[4]He et al. (2022b) do not release their code, so we are limited to comparing against the numbers they report.

However, we note that in table 3 our approach underperforms when using the original T5 model as the underlying model. As T5 was originally pretrained with a mix of self-supervised span-infilling and supervised tasks (including GLUE), this suggests the Hyperformer is able to effectively adapt the model to tasks seen or similar to those seen during pretraining. However, when we remove these tasks from the pretraining mixture (as was done for T5 v1.1+LM), the Hyperformer struggles to adapt the model, as seen in Table 1. Overall, this suggests being exposed to the underlying task in pretraining can make a large difference in the evaluation of parameter-efficient methods. As we wish to evaluate how well our approach can adapt models to completely unseen tasks, we restrict our underlying model to T5 v1.1+LM.

### 4.2 MRQA

We report our results on the MRQA dataset in Tables 4 and 5. We note that during experimentation, we found it beneficial to increase the encoder adapter size and reduce the decoder adapter size, keeping the overall parameter budget roughly the same (full results detailed in Appendix G). Overall, our approach outperforms other parameter-efficient approaches and full finetuning (69.4 vs 69.9 overall average F1 between our method and full-finetuning). Gains are especially large in the out-of-domain test sets, likely due to the fact that freezing the underlying model preserves knowledge useful to these new domains. We also note our method performs especially well on long-context out-of-domain datasets (DuoRC, TextbookQA),

| Model | SQuAD | HotpotQA | TriviaQA | NewsQA | SearchQA | Natural Qs | Avg |
|---|---|---|---|---|---|---|---|
| BERT-base | 86.7 | 76.6 | 71.6 | 66.8 | 76.7 | 77.4 | 62.6 |
| BERT-large | 88.4 | 79.0 | 74.7 | 66.3 | 79.0 | **79.8** | 77.9 |
| CA-MTL (BERT-large)* | - | - | - | - | - | - | 79.5 |
| Full Finetuning | 91.0 | **80.6** | **76.2** | **69.0** | **83.4** | 79.6 | **80.0** |
| Adapter | 90.8 | 79.5 | 75.0 | 68.6 | 82.9 | 78.6 | 79.2 |
| Task Hypernet | 88.9 | 77.4 | 70.6 | 66.0 | 81.0 | 75.0 | 76.5 |
| **Hyperdecoder (ours)** | **91.3** | 79.9 | 75.0 | 68.6 | 82.9 | 79.1 | 79.5 |

Table 4: F1 score of models on in-domain MRQA validation split. * results from Pilault et al. (2021), who do not report performance on individual datasets and train on additional data. All non-BERT models use T5$_{\text{base}}$ v1.1 + LM.

| Model | BioASQ | DROP | DuoRC | RACE | Relation Ext. | TextbookQA | Avg |
|---|---|---|---|---|---|---|---|
| BERT-base | 62.7 | 34.5 | 54.6 | 41.4 | 83.8 | 53.9 | 55.2 |
| BERT-large | 66.8 | 43.8 | 58.0 | 42.5 | 85.2 | 55.7 | 58.7 |
| UnifiedQA* | 59.7 | 45.7 | 30.4 | 51.4† | 82.0 | 35.9 | 50.9 |
| Full Finetuning | 63.9 | 47.9 | 54.5 | 47.3 | 84.0 | 55.1 | 58.8 |
| Single Adapter | **66.0** | **48.2** | 55.6 | **47.7** | 84.0 | 57.5 | 59.8 |
| Task Hypernet** | 63.9 | 36.6 | 53.0 | 44.8 | 82.0 | 53.0 | 55.6 |
| **Hyperdecoder (ours)** | 65.8 | 47.0 | **58.1** | 46.3 | **84.3** | **59.5** | **60.2** |

Table 5: F1 score of models on out-of-domain MRQA validation split. All non-BERT models use T5$_{\text{base}}$ v1.1 + LM. * results from Friedman et al. (2021). **Task Hypernet uses an average of the learnt in-domain task embeddings to condition its adapters. † RACE was part of UnifiedQA's training data.

| Model | Sum. Avg. | NLI Avg. | Overall Avg. |
|---|---|---|---|
| Full-finetuning | **18.4** | 64.8 | 41.6 |
| Adapter | 17.0 | **66.3** | **41.7** |
| Task Hypernet | 14.0 | 62.0 | 38.0 |
| **Hyperdecoder (ours)** | 16.8 | 65.6 | 41.2 |

Table 6: Average performance across Summarisation and NLI datasets using T5$_{\text{base}}$ v1.1 + LM. Summarisation performance is average of R2 scores, while NLI is average of accuracy scores. Overall average is the arithmetic mean of the two.

suggesting it is especially effective at identifying when the question cannot be answered given a subsection of context. This shows our approach is still able to work well even when the underlying datasets are not revealed to the model in the prompt and that it can generalise well to out-of-domain data. Additionally, our approach matches CA-MTL, which uses an underlying model with approximately 90 million extra parameters (BERT-large vs T5$_{\text{base}}$) and makes use of additional training data. Overall, this suggests our approach is able to generalise well to out-of-domain data, even when underlying datasets are not distinguished.

### 4.3 Summarisation & NLI

Finally, to explore a setting where strong negative interference is present, we experiment on a combination of summarisation and NLI tasks following Aribandi et al. (2022). As shown in Table 6, we find that while no approach is able to match full-finetuning in summarisation, both regular adapters and our approach are able to perform well for NLI. This suggests that while our approach is able to avoid some of the negative interference that results in lower NLI scores for the fully-finetuned model, it still struggles to overcome it. We note that similar to MRQA, we found here that placing a larger parameter budget into the encoder outperformed evenly splitting the budget between encoder and decoder (see Appendix H for details).

## 5 Analysis

This section presents an analysis of the Hyperdecoder's embeddings and ablations, establishing the efficacy of our architecture design choices.

### 5.1 Hypernetwork Embeddings

We visualise the embeddings learnt by the encoder before being passed to the hypernetwork ('**e**' in Equation 8) by utilising dimensionality reduction with t-SNE (van der Maaten and Hinton, 2008) and PCA (Pearson, 1901). Although Figure 2 suggests the hypernetwork does customise to different datasets to a degree, Figure 3 shows the most

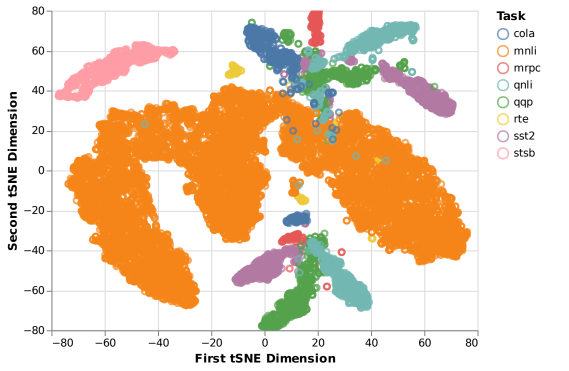

Figure 2: t-SNE visualisation of GLUE validation set hypernetwork embeddings produced by our approach.

| Model | Encoder-decoder | Encoder-only |
|---|---|---|
| **Hyperdecoder (ours)** | **75.23** | 74.94 |
| Adapter | 74.69 | 72.65 |

Table 7: GLUE validation set performance for encoder-only and encoder-decoder models excluding STS-B. We detail the encoder-only setup in Appendix C.

salient difference between samples is the predicted label. Note that following Karimi Mahabadi et al. (2021) we train our models to output numeric rather than text labels, meaning that while the labels may have semantic differences between datasets, the actual output from the decoder is identical between datasets. This suggests the hypernetwork has learnt to map from embedding space to the text labels, and the encoder is doing much of the classification work. We further investigate this by training simple linear classifiers for all datasets apart from STS-B on top of the T5 encoder (with learnt adapters) in Table 7. We note that we can recover much of the performance of our model in this case, suggesting that the decoder is largely working to map from represented space to text label space. Furthermore, the efficacy of our model at long-context out-of-domain datasets for MRQA further suggests that the hypernetwork can effectively control the decoder to output specific labels when needed, *but can flexibly swap to generate arbitrary text output when needed*, unlike a simple linear classifier. This is especially important for multi-tasking and long-context documents where the model must swap between generating short set labels and arbitrary longer text. A visualisation of the hypernetwork embeddings generated for NewsQA in Figure 4 further shows that empty string answers are generally clustered together.

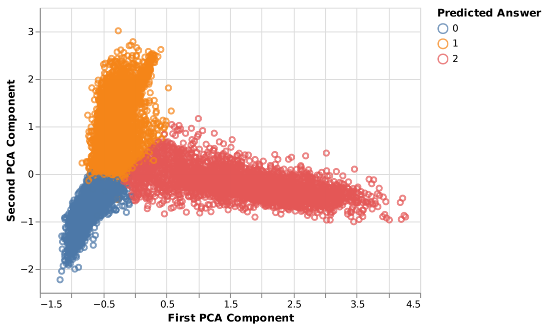

Figure 3: PCA visualisation of GLUE validation set hypernetwork embeddings, coloured by predicted label. STS-B examples removed for simplicity, as it is cast to a 21-class classification task for T5.

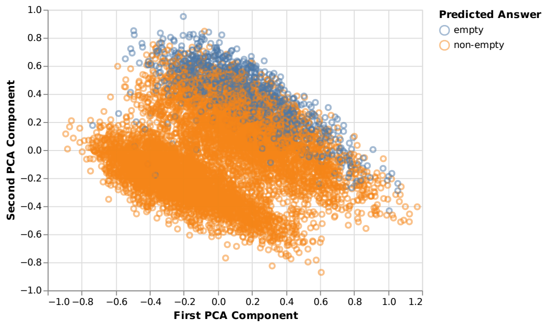

Figure 4: PCA visualisation of hypernetwork embeddings for NewsQA instances. Blue indicates predicted answer is the empty string, orange non-empty string.

## 5.2 Ablations

**Placement of adapter and parameter generators** We investigate alternate adapter generation possibilities by varying regular, task, and encoder-conditioned adapters independently in the encoder and decoder while keeping the total number of trainable parameters roughly constant. We use the same setup as our previous GLUE experiments. As seen in Table 8, input-conditioned or task-conditioned adapters do not perform as well as regular adapters in the encoder. The task hypernetwork struggles to learn useful adapters at all, while the encoder-conditioned adapters perform better but still do not match directly learnt adapters, likely due to the regular adapters being more effectively able to share knowledge or simply being easier to optimise. However, it seems much easier to learn to generate adapters for the decoder, with both task and encoder-conditioned adapters performing well. Our approach, using regular adapters in the encoder and encoder-conditioned adapters in the decoder, performs best overall.

| Encoder↓ Decoder→ | Full-Finetuning | Adapters | Task-Conditioned Adapters | Encoder-Conditioned Adapters |
|---|---|---|---|---|
| Full-Finetuning | 86.3 | - | - | - |
| Adapters | - | 86.4 | 86.3 | **86.9** |
| Task-Conditioned Adapters | - | 51.5 | 54.3 | 57.9 |
| Encoder-Conditioned Adapters | - | 83.4 | 82.3 | 82.8 |

Table 8: Average GLUE benchmark performance across varied encoder/decoder adapter configurations using $\text{T5}_{\text{large}}$ v1.1 + LM as the base model. Number of trainable parameters is kept similar across approaches apart from full-finetuning. More details given in Appendix F.

| Model | GLUE Avg |
|---|---|
| Hyperdecoder | **86.9** |
| - MLP | 86.7 |
| + post-layernorm input | 83.2 |

Table 9: GLUE performance over different ablations.

**Other Elements** We also investigate removing the MLP used in Equation 8 and using layer normalisation outputs as input to the adapters ('post-layernorm input'). The results in Table 9 suggest the MLP provides some utility and using inputs pre-layer normalisation works better.

## 6 Related Work

### 6.1 Multi-task Models

Neural networks have long been known to uncover and make use of task relatedness when training across multiple tasks (Caruana, 1997). Applications of this approach in NLP initially have usually involved generating shared representations and passing these to task-specific layers (Collobert and Weston, 2008; Liu et al., 2019). Newer approaches have opted to use a single set of parameters for all tasks, achieved by casting them to unified formats (Raffel et al., 2020). This allows massive multi-tasking approaches where extremely large models are trained across a wide variety of tasks (Aghajanyan et al., 2021; Aribandi et al., 2022), often with benefits to few or zero-shot performance (Sanh et al., 2022; Wei et al., 2022). This requires finetuning large models for long amounts of time over a large number of tasks, which may be out of reach with a limited compute budget.

### 6.2 Parameter-efficient Tuning

Numerous parameter-efficient approaches to finetuning large models have been proposed, including adapters (Houlsby et al., 2019), prefix-tuning (Li and Liang, 2021), prompt-tuning (Lester et al., 2021), and p-tuning (Liu et al., 2021a,b), all of which involve learning a small set of parameters in carefully chosen locations to achieve performance close to fully-finetuning the model. Recent studies have shown the effectiveness of these methods can be increased with differing placements (Pfeiffer et al., 2021; He et al., 2022a) and that parameters learnt for one task or language can be combined to allow better performance across a wide variety of tasks or languages (Pfeiffer et al., 2021, 2020).

### 6.3 Hypernetwork-based Adaption Methods

Generating network weights with another network was originally proposed by Schmidhuber (1991) in a general setting. More recently, Ha et al. (2017) showed modulating weights in CNNs and LSTMs could improve performance on various tasks, including language modelling. With the rise of large pretrained transformers in NLP, much recent work has explored generating adaptations for these models. Tay et al. (2021) and Karimi Mahabadi et al. (2021) investigate generating adapter (or adapter-like) layers using task embeddings and hypernetworks for multitasking, evaluating primarily on sequence classification tasks using T5 v1.0. Ye and Ren (2021) investigate generating adapters from task descriptions for zero-shot tasks and find this provides improvements over just fully-finetuning the underlying model. Concurrently, Volk et al. (2022) use a T5 model to convert input instances into domain 'signatures', which they condition a hypernetwork on to generate classifier weights. In contrast, we directly generate weights for adapters from the encoded input representation. Recent work has further applied hypernetwork-based adaptation methods to vision and language tasks (Sung et al., 2021), prompt-tuning (He et al., 2022b), and few-shot-based multitasking (Ponti et al., 2022).

Outside of sequence-to-sequence-based approaches, Pilault et al. (2021) find modifying multiple parts of BERT to be conditional on a task embedding to be effective for sequence classification tasks. Üstün et al. (2020) and Ansell et al. (2021) also explore generating multilingual adapters for mBERT using linguistic typological features, the former using parameter generators in every layer.

## 7 Conclusion

We propose a novel method for generating adapters conditioned on a model's input and show that this improves performance in multi-task settings across a variety of tasks. We explore the effectiveness of our approach for sequence classification, QA, and summarisation tasks, and find that it often outperforms strong parameter-efficient baselines. Future work could examine applying our approach to other architectures (e.g. decoder-only models) or explore the tradeoffs between shared and generated parameters across different layers. An analysis of our approach suggests the primary benefits come from improved control of the encoder over the decoder, enhancing the effects of positive transfer from the shared encoder. This allows our approach to efficiently adapt a pretrained language model to multiple tasks unseen during pretraining while still benefiting strongly from positive transfer.

## Limitations

Our work explores a novel idea within parameter-efficient finetuning by conditioning a model on itself, enabling greater flexibility for multi-tasking while adding relatively few parameters. However, this flexibility has limits: some tasks are more difficult to adapt with our method than others, as seen in the summarisation results in section 4.3. Examining other parameter efficient training methods such as LoRA (Hu et al., 2022) or prefix-tuning (Li and Liang, 2021) on top of adapters may yield further improvements (He et al., 2022a), but may also uncover a dependence on the particular adapter setup used by our approach. Additionally, parts of our design involve compressing information (in particular, the mean-pooling used to condition the hypernetwork), and further experiments on tasks with long inputs or outputs (such as summarisation) may reveal potential limitations of this approach and suggest further improvements. Additionally, our work only examines English-based tasks, so the ability of the model to handle alternate or potentially multiple languages at once is unknown. We note that existing work has shown adapters and hypernetworks to be useful for multilingual adaptation (Pfeiffer et al., 2020; Üstün et al., 2020; Ansell et al., 2021), suggesting that our approach may be effective for multilingual multitasking when combined with these existing multilingual adaptation methods. Finally, the nature of the hypernetwork approach means that it may be difficult to scale to massive multi-tasking on the scale of exT5 (Aribandi et al., 2022) or T0 (Sanh et al., 2022), as the hypernetwork itself has limited capacity to store tasks. We do not investigate how this approach scales with respect to tasks, instead focussing on how the hypernetwork improves positive transfer and mitigates negative transfer for tasks whose positive and negative transfer effects are well-known.

## References

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5799–5811, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Eneko Agirre, Lluís Màrquez, and Richard Wicentowski, editors. 2007. *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*. Association for Computational Linguistics, Prague, Czech Republic.

Alan Ansell, Edoardo Maria Ponti, Jonas Pfeiffer, Sebastian Ruder, Goran Glavaš, Ivan Vulić, and Anna Korhonen. 2021. MAD-G: Multilingual adapter generation for efficient cross-lingual transfer. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4762–4781, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Vamsi Aribandi, Yi Tay, Tal Schuster, Jinfeng Rao, Huaixiu Steven Zheng, Sanket Vaibhav Mehta, Honglei Zhuang, Vinh Q. Tran, Dara Bahri, Jianmo Ni, Jai Gupta, Kai Hui, Sebastian Ruder, and Donald Metzler. 2022. Ext5: Towards extreme multi-task scaling for transfer learning. In *International Conference on Learning Representations*.

Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth PASCAL recognizing textual entailment challenge.

Chandra Bhagavatula, Ronan Le Bras, Chaitanya Malaviya, Keisuke Sakaguchi, Ari Holtzman, Hannah Rashkin, Doug Downey, Wen tau Yih, and Yejin Choi. 2020. Abductive commonsense reasoning. In *International Conference on Learning Representations*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Rich Caruana. 1997. Multitask learning. *Machine Learning*, 28(1):41–75.

Oscar Chang, Lampros Flokas, and Hod Lipson. 2020. Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA. Association for Computing Machinery.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pages 177–190. Springer.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. 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)*, pages 2368–2378, Minneapolis, Minnesota. Association for Computational Linguistics.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Güney, Volkan Cirik, and Kyunghyun Cho. 2017. Searchqa: A new q&a dataset augmented with context from a search engine. *CoRR*, abs/1704.05179.

Adam Fisch, Alon Talmor, Robin Jia, Minjoon Seo, Eunsol Choi, and Danqi Chen. 2019. MRQA 2019 shared task: Evaluating generalization in reading comprehension. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 1–13, Hong Kong, China. Association for Computational Linguistics.

Dan Friedman, Ben Dodge, and Danqi Chen. 2021. Single-dataset experts for multi-dataset question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6128–6137, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Sebastian Gehrmann, Tosin Adewumi, Karmanya Aggarwal, Pawan Sasanka Ammanamanchi, Anuoluwapo Aremu, Antoine Bosselut, Khyathi Raghavi Chandu, Miruna-Adriana Clinciu, Dipanjan Das, Kaustubh Dhole, Wanyu Du, Esin Durmus, Ondřej Dušek, Chris Chinenye Emezue, Varun Gangal, Cristina Garbacea, Tatsunori Hashimoto, Yufang Hou, Yacine Jernite, Harsh Jhamtani, Yangfeng Ji, Shailza Jolly, Mihir Kale, Dhruv Kumar, Faisal Ladhak, Aman Madaan, Mounica Maddela, Khyati Mahajan, Saad Mahamood, Bodhisattwa Prasad Majumder, Pedro Henrique Martins, Angelina McMillan-Major, Simon Mille, Emiel van Miltenburg, Moin Nadeem, Shashi Narayan, Vitaly Nikolaev, Andre Niyongabo Rubungo, Salomey Osei, Ankur Parikh, Laura Perez-Beltrachini, Niranjan Ramesh Rao, Vikas Raunak, Juan Diego Rodriguez, Sashank Santhanam, João Sedoc, Thibault Sellam, Samira Shaikh, Anastasia Shimorina, Marco Antonio Sobrevilla Cabezudo, Hendrik Strobelt, Nishant Subramani, Wei Xu, Diyi Yang, Akhila Yerukola, and Jiawei Zhou. 2021. The GEM benchmark: Natural language generation, its evaluation and metrics. In *Proceedings of the 1st Workshop on Natural Language Generation, Evaluation, and Metrics (GEM 2021)*, pages 96–120, Online. Association for Computational Linguistics.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics.

David Ha, Andrew M. Dai, and Quoc V. Le. 2017. Hypernetworks. In *International Conference on Learning Representations*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022a. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

Yun He, Huaixiu Zheng, Yi Tay, Jai Gupta, Yu Du, V. Aribandi, Zhe Zhao, YaGuang Li, Zhao Chen, Donald Metzler, Heng-Tze Cheng, and Ed H. Chi. 2022b. Hyperprompt: Prompt-based task-conditioning of transformers. *ArXiv*, abs/2203.00759.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information*

*Processing Systems*, volume 28. Curran Associates, Inc.

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 *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. 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)*, pages 565–576, Online. Association for Computational Linguistics.

Aniruddha Kembhavi, Minjoon Seo, Dustin Schwenk, Jonghyun Choi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Are you smarter than a sixth grader? textbook question answering for multimodal machine comprehension. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. UNIFIEDQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1896–1907, Online. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark. Association for Computational Linguistics.

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*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2011. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, page 47.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 333–342, Vancouver, Canada. Association for Computational Linguistics.

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)*, pages 4582–4597, Online. Association for Computational Linguistics.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021a. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *CoRR*, abs/2110.07602.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021b. Gpt understands, too. *arXiv:2103.10385*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial

NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4885–4901, Online. Association for Computational Linguistics.

Karl Pearson. 1901. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Jason Phang, Thibault Févry, and Samuel R. Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *CoRR*, abs/1811.01088.

Jonathan Pilault, Amine El hattami, and Christopher Pal. 2021. Conditionally adaptive multi-task learning: Improving transfer learning in {nlp} using fewer parameters & less data. In *International Conference on Learning Representations*.

Edoardo M Ponti, Alessandro Sordoni, and Siva Reddy. 2022. Combining modular skills in multitask learning. *arXiv preprint arXiv:2202.13914*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, pages 2383–2392. Association for Computational Linguistics.

Amrita Saha, Rahul Aralikatte, Mitesh M. Khapra, and Karthik Sankaranarayanan. 2018. DuoRC: Towards complex language understanding with paraphrased reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1683–1693, Melbourne, Australia. Association for Computational Linguistics.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

Jurgen Schmidhuber. 1991. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. Technical Report FKI147-91, Institut fur Informatik, Technische Universitat Munchen.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pages 1631–1642.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2021. Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks. *ArXiv*, abs/2112.06825.

Yi Tay, Zhe Zhao, Dara Bahri, Donald Metzler, and Da-Cheng Juan. 2021. Hypergrid transformers: Towards a single model for multiple tasks. In *International Conference on Learning Representations*.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A machine comprehension dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 191–200, Vancouver, Canada. Association for Computational Linguistics.

George Tsatsaronis, Georgios Balikas, Prodromos Malakasiotis, Ioannis Partalas, Matthias Zschunke, Michael R. Alvers, Dirk Weissenborn, Anastasia Krithara, Sergios Petridis, Dimitris Polychronopoulos, Yannis Almirantis, John Pavlopoulos, Nicolas Baskiotis, Patrick Gallinari, Thierry Artiéres, Axel-Cyrille Ngonga Ngomo, Norman Heino, Eric Gaussier, Liliana Barrio-Alvers, Michael Schroeder, Ion Androutsopoulos, and Georgios Paliouras. 2015. An overview of the bioasq large-scale biomedical semantic indexing and question answering competition. *BMC Bioinformatics*, 16(1):138.

Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. 2020. UDapter: Language adaptation for truly Universal Dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2302–2315, Online. Association for Computational Linguistics.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Tomer Volk, Eyal Ben-David, Ohad Amosy, Gal Chechik, and Roi Reichart. 2022. Example-based hypernetworks for out-of-distribution generalization. *arXiv preprint arXiv:2203.14276*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2018. Neural network acceptability judgments. *arXiv preprint 1805.12471*.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Qinyuan Ye and Xiang Ren. 2021. Learning to generate task-specific adapters from task description. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 646–653, Online. Association for Computational Linguistics.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *CoRR*, abs/2106.10199.

## A Baseline Details

### A.1 Single Adapter

Our single adapter baseline involves placing parallel adapters (He et al., 2022a) in each layer of the encoder and decoder and directly learning the parameters. For most tasks, we make use of adapters with size 410 in order to keep the trainable parameter budget equivalent to our approach. For MRQA, we achieve better results with adapters of size 800 in the encoder and size 36 in the decoder.

### A.2 Task Hypernet

This model uses two hypernetworks to produce adapters for the encoder and decoder respectively, based on a learnt task embedding. The hypernetwork layout and adapter placement are identical to the one proposed in our method, with the exception that the initial input is a task embedding and not the mean-pooled encoder output. The inclusion of this baseline shows that the benefits of our method come from the use of input-specific decoders instead of improved adapter placement. For this approach, we use adapters of size 50 and hypernetworks of size 100, keeping the overall number of trainable parameters close to our approach.

### A.3 Modular Task Hypernetwork

This is the method proposed by Ponti et al. (2022), which learns a mapping from task id to a sparse set of skills, each of which corresponds to a set of LoRA (Hu et al., 2022) parameters. These parameters are then combined to create task-specific adaptations. We set the number of skills $|\mathcal{S}| = 4$ and use a two-speed learning rate as suggested by the authors. We set the main learning rate as $3e-4$, and the secondary learning rate for $Z$ as $1e-2$. We use the same learning rate schedule as detailed in section 3.2. The rank of the LoRA adaptations is set to 16, following the defaults used in Ponti et al. (2022).

## B Training Details

We run experiments on a single NVIDIA A100 80GB GPU, with all reported results from one training run using the provided hyperparameters. A single training run (including preprocessing and all evaluation) of our approach on the GLUE benchmark (with T5$_{\text{large}}$ v1.1 + LM as the underlying model) takes 18 hours. A single training run of our approach on the MRQA datasets (with T5$_{\text{base}}$ v1.1 + LM as the underlying model), including final evaluation, takes 13 hours. A training single run of our model on the Summarisation and NLI datasets (with T5$_{\text{base}}$ v1.1 + LM as the underlying model), including evaluation, takes 22.5 hours. We note that T5$_{\text{base}}$ v1.0 (or 'vanilla') has roughly 220 million parameters total, T5$_{\text{base}}$ v1.1 + LM has roughly 250 million parameters total, and T5$_{\text{large}}$ v1.1 + LM has roughly 800 million parameters total.

## C Encoder-Only Model Details

The encoder-only models used in Table 7 consist of a T5 encoder with adapters inserted as done for the adapter-only and our model. We train a unique linear layer per task that maps from the hypernetwork embedding to logits, which are passed through a softmax layer to make a final prediction. We initialise the T5 encoder with the trained adapter layers in encoders, using the checkpoints reported in Table 1. This encoder is kept frozen during training. To match the MLP used to generate the hypernetwork embeddings in equation 8, we place an identical MLP on top of the adapter-augmented T5 encoder and pass its output to the linear classifier. This MLP is trained along with the classifier. To train, we use the AdamW optimiser with a learning rate of 2e-5 for 3 epochs with linear learning rate warmup and decay with 500 warmup steps.

## D Parameters

We calculate the number of parameters for each method as follows. In each case, $l$ is the number of layers, $d$ the hidden size of the model, $a$ the adapter dimension, $t$ the number of tasks, and $b$ the hypernetwork bottleneck size. For simplicity, we will assume that the encoder and decoder adapter sizes are the same below, but it is straightforward to calculate the parameters used in the encoder and decoder separately and sum these for cases where the encoder and decoder sizes differ. We leave out bias parameters for simplicity.

### D.1 Adapters

We only consider placing adapters in the feedforward module of the transformer layer. Every layer has one adapter consisting of two linear layers and a non-linearity, giving $l(2ad + a + d)$ overall new parameters.

| Dataset | Train Split Size | Validation Split Size | Test Split Size |
|---|---|---|---|
| CoLA | 8551 | 521 | 522 |
| SST-2 | 66349 | 1000 | 872 |
| STS-B | 5749 | 750 | 750 |
| MRPC | 3668 | 204 | 204 |
| QQP | 362846 | 1000 | 40430 |
| MNLI | 392702 | 9832 | 9815 |
| QNLI | 103743 | 1000 | 5463 |
| RTE | 2490 | 138 | 139 |

Table 10: Summary statistics of splits used when evaluating GLUE.

### D.2 Task Hypernetwork

For the task-embedding-based hypernetwork approach, we generate all adapters with two hypernetworks, one for the encoder and one for the decoder. the main parameter costs come in the form of the final layer of the hypernetwork, which consists of four linear layers producing the weights and biases of the two linear layers making up the adapter, thus costing $b(2ad+a+d)$. Given task embedding size $e_t$ and layer embedding size $e_l$, the total cost of the hypernetwork is $te_t+le_l+(e_t+e_l)b+b(2ad+a+d)$. This is then multiplied by two as we have hypernetworks for the encoder and decoder.

### D.3 Encoder-conditioned Decoders

The cost of the decoder is similar to the task hypernetwork, except we add the cost of the MLP and use the hidden size of the model instead of the task embedding size: $2d^2 + (d+e_l)b + b(2ad+a+d)$. Note $b(2ad + a + d) \gg d^2$ for the adapter and model size choices used in our work. The encoder costs the same as the adapters case: $l(2ad+a+d)$. Our overall cost is simply the sum of the two.

## E Dataset Statistics

In Tables 10, 11 and 12 provide some summary statistics of each dataset used and split sizes.

## F Full GLUE Ablation Results

In Table 13 we provide the full results from the ablations performed in Tables 8 and 9. Model names are in format <encoder type>-<decoder type>. 'Generated' refers to encoder-conditioned adapters, 'manual' regular adapters, and 'task' task-conditioned adapters.

## G MRQA Varied Encoder Results

In Tables 14 and 15 we provide results from experiments varying encoder and decoder sizes, as

| Dataset | Train Split Size | Validation Split Size |
|---|---|---|
| SQuAD | 86588 | 10507 |
| HotpotQA | 72928 | 5901 |
| TriviaQA | 61688 | 7785 |
| NewsQA | 74160 | 4212 |
| SearchQA | 117384 | 16980 |
| Natural Qs | 104071 | 12836 |
| BioASQ | - | 1504 |
| DROP | - | 1503 |
| DuoRC | - | 1501 |
| RACE | - | 674 |
| Relation Ext. | - | 2948 |
| TextbookQA | - | 1503 |

Table 11: Summary statistics of splits used when evaluating MRQA, before applying preprocessing.

| Dataset | Train Split Size | Validation Split Size | Test Split Size |
|---|---|---|---|
| XSum | 204045 | 11332 | 11334 |
| CNN/Daily Mail | 287113 | 13368 | 11490 |
| Wiki Lingua | 99020 | 13823 | 28614 |
| MNLI | 392702 | 9832 | 9815 |
| Abductive NLI | 168654 | 1000 | 1532 |
| Adversarial NLI | 162865 | 3200 | 3200 |

Table 12: Summary statistics of splits used when evaluating summarisation and NLI datasets. Note we combine all Adversarial NLI rounds (r1, r2, r3).

mentioned in section 4.2. For the final results reported in Tables 4 and 5, we use an encoder size of 512 for adapters and the hyperdecoder.

## H Full Summarisation and NLI Results

In Table 16 we provide a breakdown of the results reported in Table 6.

| Model | % Trainable Param. | CoLA | SST-2 | STS-B | MRPC | QQP | MNLI | QNLI | RTE | Avg |
|---|---|---|---|---|---|---|---|---|---|---|
| Full Finetuning | 100% | **63.6** | 94.8 | 92.0 / 91.6 | 88.7 / 91.8 | **92.2 / 89.5** | 88.6 | 93.3 | 77.5 | 86.3 |
| Manual-Manual | 2.9% | 58.5 | 95.3 | 91.7 / 91.4 | **89.7** / 92.5 | 91.2 / 88.3 | 89.8 | 94.0 | 81.2 | 86.4 |
| Manual-Task | 1.7% | 54.6 | 96.0 | 91.9 / 91.9 | **89.7 / 92.6** | 91.0 / 88.1 | **90.0** | 94.1 | **83.3** | 86.3 |
| Manual-Generated | 2.9% | 58.5 | **96.6** | 91.8 / 91.7 | **89.7** / 92.5 | 91.3 / **89.5** | 89.8 | **94.5** | 82.6 | **86.9** |
| Task-Manual | 1.7% | 8.2 | 80.7 | 38.4 / 38.4 | 69.0 / 80.1 | 78.5 / 74.2 | 61.9 | 69.8 | 51.5 | 57.7 |
| Task-Generated | 3.0% | 0.0 | 83.1 | 38.4 / 42.5 | 69.5 / 81.0 | 90.6 / 87.6 | 63.0 | 60.1 | 52.2 | 57.9 |
| Task-Task | 2.7% | 0.0 | 82.1 | 16.4 / 16.4 | 70.4 / 81.4 | 89.8 / 86.5 | 56.6 | 64.8 | 50.7 | 54.3 |
| Generated-Manual | 3.3% | 50.3 | 95.7 | 90.6 / 90.8 | 82.8 / 87.6 | 89.8 / 86.5 | 89.1 | 92.2 | 76.1 | 83.4 |
| Generated-Task | 3.3% | 41.6 | 95.3 | 82.4 / 92.7 | 83.1 / 88.3 | 89.5 / 86.2 | 88.6 | 92.1 | 79.7 | 82.3 |
| Generated-Generated | 4.4% | 45.4 | 95.8 | 88.6 / 88.4 | 84.2 / 88.5 | 89.7 / 86.3 | 89.5 | 91.8 | 76.8 | 82.8 |
| Manual-Generated, no MLP | 2.6% | 62.0 | 96.0 | **92.2 / 92.3** | 86.7 / 90.3 | 91.1 / 88.2 | **90.0** | 93.4 | 81.9 | 86.7 |
| Manual-Generated, post-layernorm adapter input | 2.9% | 45.0 | 94.8 | 90.7 / 91.1 | 82.3 / 87.5 | 89.8 / 86.7 | 89.3 | 92.8 | 79.7 | 83.2 |

Table 13: Performance of models using $T5_{large}$ v1.1 + LM as the base model on GLUE test set splits described in section 3.1.1. Trainable parameters is % of the trainable parameters used compared to fully finetuning T5.

| Model | Encoder Adapter Size | Decoder Adapter Size | Hypernet Bottleneck Size | % Trainable Param. | SQuAD | HotpotQA | TriviaQA | NewsQA | SearchQA | Natural Qs | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Hyperdecoder (ours) | 64 | 64 | 128 | 6.2% | 91.1 | 78.9 | 73.5 | 67.7 | 82.6 | 77.6 | 78.6 |
| Hyperdecoder (ours) | 512 | 36 | 72 | 5.9% | **91.3** | **79.9** | 75.0 | **68.6** | **82.9** | **79.1** | **79.5** |
| Hyperdecoder (ours) | 800 | 2 | 16 | 6.2% | 90.8 | 79.5 | 75.0 | **68.6** | **82.9** | 78.6 | 79.2 |
| Adapter | 370 | 370 | - | 5.5% | 88.7 | 75.9 | 67.4 | 64.9 | 79.5 | 74.3 | 75.1 |
| Adapter | 512 | 225 | - | 5.5% | 90.8 | 79.5 | 75.0 | **68.6** | **82.9** | 78.6 | 79.2 |
| Adapter | 800 | 2 | - | 6.0% | 91.0 | 79.6 | **75.5** | **68.6** | **82.9** | 78.9 | 79.4 |

Table 14: F1 score of models on in-domain MRQA validation split. All models use the $T5_{base}$ v1.1 + LM checkpoint as a starting point.

| Model | Encoder Adapter Size | Decoder Adapter Size | Hypernet Bottleneck Size | % Trainable Param. | BioASQ | DROP | DuoRC | RACE | Relation Ext. | TextbookQA | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Hyperdecoder (ours) | 64 | 64 | 128 | 6.2% | 66.2 | 43.3 | 57.8 | 46.8 | 85.4 | 58.4 | 59.7 |
| Hyperdecoder (ours) | 512 | 36 | 72 | 5.9% | 65.8 | 47.0 | **58.1** | 46.3 | 84.3 | **59.5** | **60.2** |
| Hyperdecoder (ours) | 800 | 2 | 16 | 6.2% | 65.7 | 44.1 | 55.2 | 45.4 | **84.6** | 54.9 | 58.3 |
| Adapter | 370 | 370 | - | 5.5% | 63.4 | 33.3 | 56.4 | 44.8 | 83.1 | 56.9 | 56.3 |
| Adapter | 512 | 225 | - | 5.5% | 66.0 | **48.2** | 55.6 | **47.7** | 84.0 | 57.5 | 59.8 |
| Adapter | 800 | 2 | - | 6.0% | **66.4** | 43.7 | 55.9 | 47.2 | 84.5 | 55.6 | 58.9 |

Table 15: F1 score of models on out-of-domain MRQA validation split. All models use the $T5_{base}$ v1.1 + LM checkpoint as a starting point.

| Model | % Trainable Param. | XSUM R2 | CNN/Daily Mail R2 | Wiki Lingua R2 | MNLI | Ab. NLI | Ad. NLI | Sum. Avg. | NLI Avg. | Overall Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| Full-finetuning | 100% | **17.4** | **19.3** | **18.4** | 85.6 | 64.3 | 44.3 | **18.4** | 64.8 | 41.6 |
| Adapter | 5.5% | 15.4 | 18.7 | 16.7 | 86.2 | 65.2 | 45.0 | 16.9 | 65.5 | 41.2 |
| Adapter (enc-heavy) | 5.5% | 15.5 | 18.7 | 16.8 | **86.5** | **67.2** | **45.2** | 17.0 | **66.3** | **41.7** |
| Hyperdecoder (ours) | 6.2% | 14.6 | 18.2 | 16.0 | 85.5 | 63.8 | 44.1 | 16.3 | 64.5 | 40.4 |
| Hyperdecoder (ours) (enc-heavy) | 5.9% | 15.3 | 18.6 | 16.5 | 86.3 | 66.4 | 44.1 | 16.8 | 65.6 | 41.2 |
| Task Hypernet | 10.4% | 11.6 | 17.0 | 13.4 | 83.9 | 61.5 | 40.5 | 14.0 | 62.0 | 38.0 |

Table 16: Performance across Summarisation and NLI datasets using $T5_{base}$ v1.1 + LM. Summarisation scores are Rouge2 scores, while NLI scores are accuracy using test splits described in section 3.1.3. 'Enc-heavy' variants use encoder adapter sizes of 512 and the matching decoder/hypernetwork sizes in Table 14.