ATTEMPT – Parameter-Efficient Multi-task Tuning via Attentional Mixtures of Soft Prompts: A Replication Study

Róbert Belanec[12], **Simon Ostermann**[3], **Ivan Srba**[1] and **Mária Bieliková**[1]
[1] Kempelen Institute of Intelligent Technologies, Bratislava, Slovakia
[2] Faculty of Information Technology, Brno University of Technology, Brno, Czechia
[3] German Research Institute for Artificial Intelligence (DFKI),
Saarland Informatics Campus, Germany
`{robert.belanec, ivan.srba, maria.bielikova}@kinit.sk`
`simon.ostermann@dfki.de`

Abstract

Generative language models gained an increase in popularity shortly after the introduction of the transformer architecture (Vaswani et al., 2017), which resulted in a fast increase in the number of model parameters. Currently, large language models contain billions of trainable parameters, which makes them power and cost inefficient. Large language models also require significant amounts of training data, which especially benefits well-resourced languages. To address these problems, parameter-efficient fine-tuning methods have emerged. Parameter-efficient fine-tuning methods aim to fine-tune generally pre-trained language models while training only a fraction of parameters. In the scientific and academic community, authors often compare with the current state-of-the-art. However, for the results to be relevant and trustworthy, both of the works (state-of-the-art and compared) must be reproducible. In our work, we present the methodology and the results of our replication study of a parameter-efficient fine-tuning method introduced in the paper ATTEMPT: Parameter-Efficient Multi-task Tuning via Attentional Mixtures of Soft Prompts. To replicate the results provided by the authors, we have conducted a series of experiments and we show that better-performing source prompts may contribute more to the overall results. We also point out a stability issue and provide examples of results that have a better score but are harder to replicate due to the randomness factors. Finally, we compare our results to the results provided by the authors and derive a conclusion based on a discussion.

1 Introduction

In recent years, generative language models have experienced a steady increase in popularity. After the introduction of the transformer architecture (Vaswani et al., 2017) for natural language processing, there has been a fast increase in the number of model parameters. The first widely-used transformer models contained millions of trainable parameters (e.g. BERT-Large having 340 million parameters (Devlin et al., 2019) and GPT having 117 million parameters (Radford et al., 2018)). Recent architectures contain billions of trainable parameters (e.g. GPT-2 having 1.5 billion parameters (Radford et al., 2019) and GPT-3 having 175 billion parameters (Brown et al., 2020)). With the rising trend of increasing the number of parameters to achieve better results, models often require a vast amount of computational resources for training. Besides their parameter hunger, large language models also require significant amounts of training data, which especially benefits well-resourced languages. The newest language models often perform sub-par for low-resourced languages, decreasing the exhibited trust in such models. Significant trust decrease is also caused by the loss of interpretability that correlates with the size of the newest language models.

Consequently, there is a strong motivation in the natural language processing research community to decrease the number of trained parameters and the need for large amounts of training data, while maintaining the results on downstream tasks. To address these problems, parameter-efficient and data-efficient fine-tuning methods have emerged. Parameter-efficient fine-tuning methods aim to fine-tune generally pre-trained language models while training only a fraction of the model parameters. Data-efficient finetuning methods aim to leverage the power of large pre-trained models and try to adapt them to specific tasks or domains with only minimal amounts of training data. Many state-of-the-art parameter-efficient models have been shown also to require less training data, which is why both problems can often be alleviated with a single method (Yu et al., 2022; Gu et al., 2022).

In the scientific and academic community, authors often compare with the current state-of-the-art. However, for the results to be relevant and trustworthy, both of the works (state-of-the-art and compared) must be reproducible. This means, that by following the authors' publication, we should be able to derive the same results as provided by the authors. In our work, we present the methodology and the results of our replication study of a parameter-efficient fine-tuning method presented in the paper ATTEMPT: Parameter-Efficient Multi-task Tuning via Attentional Mixtures of Soft Prompts (Asai et al., 2022).

2 Related Work

Language models. Recently, generative language models have experienced a breakthrough that started by introducing the transformer architecture (Vaswani et al., 2017), which was preceded by the introduction of novel methods in machine learning translation like Sequence to Sequence models and attention (Sutskever et al., 2014; Bahdanau et al., 2014). For natural language generation, the GPT (Radford et al., 2018) model was introduced. Shortly after, BERT (Kenton and Toutanova, 2019) architecture was introduced, replacing the bi-directional LSTM (Peters et al., 2018) with a bidirectional transformer architecture pre-trained to de-mask masked parts of a text sequence.

Recently, with the introduction of large language models (Radford et al., 2018, 2019; Touvron et al., 2023a,b; Jiang et al., 2023), the number of model parameters has risen from millions of trainable parameters to billions. These models require large amounts of computational resources and large amounts of data to train.

Large language models have also been trained to solve multiple natural language processing tasks. For example, authors of the T5 model (Raffel et al., 2020) pre-trained their model on a set of a multi-task mixture of unsupervised and supervised tasks. The T5 model is designed to solve text-to-text denoising problems by training on text-to-text format datasets with span corruption. Building upon T5 versatility, its improved version Flan-T5 (Chung et al., 2022) was introduced shortly after. Nevertheless, fine-tuning large language models (e.g. to perform in a multi-task setting) is also a parameter-heavy task, therefore, new methods of training have been introduced.

Parameter-efficient fine-tuning. The core idea of parameter-efficient fine-tuning is to train a neural network model while backpropagating only over a small fraction of parameters. One of the first works towards more efficient fine-tuning of the language models was a work introducing sequential *adapters* (Houlsby et al., 2019), which are small trainable feedforward neural network modules, that are inserted into transformer architecture layers, while keeping the rest of the model frozen. Adapters and their variations (Pfeiffer et al., 2021; He et al., 2022; Chronopoulou et al., 2023) are still heavily used since they provide flexibility for multi-task problems (e.g. by training multiple adapters for each task separately and swapping between them on demand).

Some parameter-efficient fine-tuning methods focus on the reparameterization of the original weights by introducing a smaller matrix that is then transformed into a bigger matrix that represents the δW that will be added to the base model weights. For example, the Intrinsic SAID (Aghajanyan et al., 2020) method uses a Fastfood transform to transform from the low-rank decomposing, but it is not that effective due to the high memory complexity of the Fastfood transform (Le et al., 2013). Building on top of the Intrinsic SAID method LoRA (Hu et al., 2021) introduced two separate matrices that form the resulting δW matrix. After the introduction of LoRA, other methods based on LoRA appeared. For example, QLoRA (Dettmers et al., 2023) uses quantization of model parameters to 4-bit NormalFloat and uses a paged optimizer to deal with the memory spikes.

Another parameter-efficient fine-tuning method that adds modules to the base models is *prompt-tuning* (Lester et al., 2021). Prompt-tuning trains embeddings (in a separate embedding module) that are prepended to the input embeddings before inserting them into the base model. Prompt tuning requires only less than 0.01% of the original parameters to train the model to a specific task. In parallel with prompt-tuning, *prefix-tuning* (Li and Liang, 2021) was developed. Instead of prepending a single matrix of weights to the first layer, in prefix-tuning, a matrix is prepended to each separate layer of a transformer architecture. Therefore, it requires around 1% of the original parameters, still a relatively small number. These methods can be classified as *soft prompt fine-tuning* methods (Liu et al., 2023; Vu et al., 2022; Asai et al., 2022; Hambardzumyan et al., 2021; Wang et al., 2023), as they are fine-tuning parameter-efficient soft prompts (i.e., which are not made by humans, when compared to hard prompts). These methods provide more significant parameter reduction than some methods incorporating adapters but also sacrifice a portion of the model input context.

Some recent works also focus on transferring soft prompt information like SPoT (Vu et al., 2022) and ATTEMPT (Asai et al., 2022). The SPoT method investigates the transferability of soft prompts on 160 task combinations. ATTEMPT focuses on fine-tuning the model using prompt tuning on multiple tasks.

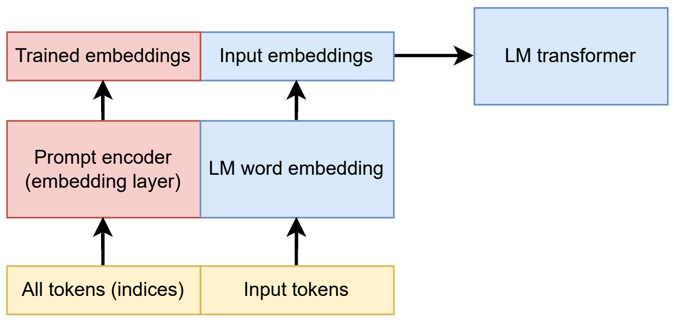

Figure 1: Diagram representing training process of the prompt tuning method. The blue color represents components with frozen parameters and the red color represents components with trainable parameters. The yellow color represents components without any weights (i.e. model inputs and outputs or utility functions).

After training the source soft prompts for each source task, ATTEMPT then trains a target soft prompt to solve the target tasks using a trainable attention layer to incorporate the source prompts accordingly. This method comes from the idea that learning to solve different tasks may contribute to solving other tasks. ATTEMPT is still very parameter-efficient as it trains only 0.4% of the original model parameters.

3 Replicated Methods

We have divided the ATTEMTP parameter-efficient fine-tuning method replication into two main parts: 1) the prompt tuning (Lester et al., 2021) replication, which we will use to train the source soft prompts and target soft prompts for prompt transfer, and 2) the ATTEMPT method replication. ATTEMPT is built on top of the prompt tuning and heavily relates to it. At the time of execution of this replication study, the prompt tuning parameter-efficient fine-tuning method is implemented in the publicly available parameter-efficient fine-tuning module (Mangrulkar et al., 2022). Regardless, we have decided to replicate the prompt tuning method, since we can build on it when implementing ATTEMPT. In this chapter, we will further describe both replicated methods.

3.1 Parameter-Efficient Prompt Tuning

The first parameter-efficient fine-tuning we will describe is prompt tuning (Lester et al., 2021). Prompt tuning is a parameter-efficient fine-tuning method that prepends a trainable embedding (prompt embedding) to the input embeddings to be forwarded as input to the base model. When training, the prompt embedding guides the language model to produce better results. Prompt tuning can therefore be seen as automatic prompt generation (which is also similar to adversarial reprogramming that can be seen in computer vision tasks (Elsayed et al., 2019)). This automatically trained prompt embedding is called a *soft prompt*.

Soft prompts are often compared with hard prompts. Hard prompts are prompts, that are made by a human (prompt engineer) to improve the results of already trained language models without any weight updates. This comparison of soft prompts and hard prompts can sometimes mislead the reader as it suggests that soft prompts are interpretable and readable in human language. Interpretation of a soft prompt in human language is not straight forward as the prompt embedding is trained separately. Therefore it has its own set of tokens (indices of the embedding) which is not a subset of the base model tokens and, therefore cannot be detokenized to the base model's vocabulary.

In a text-to-text approach using T5 (Raffel et al., 2020) as a base model we can interpret the language model as a conditional probability $Pr_\theta(Y|X)$ where Y is a sequence of tokens conditioned by a sequence of input tokens X parametrized by models weights θ. Prompting is a method that incorporates creating a hard prompt P which is a set of tokens prepended to input tokens $[P;X]$. Prompt tuning builds upon this idea and introduces parametrization of P with its weights θ_P. The conditional probability of generating Y is now $Pr_{\theta;\theta_P}(Y|[P;X])$. T5 embeds the set of input tokens into a matrix $X_e \in \mathbb{R}^{n\times e}$ where n is the length of the input token sequence e is the dimension of T5 embeddings. Prompt tuning represents soft prompts as a matrix of parameters $P \in \mathbb{R}^{p\times e}$ where p is the length of the soft prompt.

To gain a better overview of the prompt tuning parameter-efficient fine-tuning method, we provide a method diagram that can be seen in Figure 1. After the training, soft prompts include information about the tasks that they were trained on. This can also mean that combining multiple soft prompts benefits in solving multi-task problems. The ATTEMPT method further builds upon this idea.

3.2 ATTEMPT – Attentional Mixtures of Soft Prompts

The ATTEMPT method takes advantage of the prompt tuning and builds on top of the method by introducing an attention module to create a mixture of pre-trained soft prompts based on how much they contribute to the result. The main hypothesis of the ATTEMPT method is when transferring the information from one soft prompt trained on a specific task, it can also contribute to solving other tasks, which is parallel to the language model transfer learning (e.g. model trained to summarize texts in the English language has already learned to understand the English language grammar and therefore can be trained easily to solve other English language tasks).

ATTEMPT can be trained in multiple ways – in a single-task setting (training each dataset separately) and in multi-task training on multiple concatenated datasets with an option to share the attention module across multiple tasks. In both of these settings, ATTEMPT trains a set of **target prompts** for each task (i.e. in a single task setting and a multi-task setting without a shared attention module the number of target prompts is 1) and uses a set of soft prompts to calculate the addition to the target prompt. We will further describe each of these training settings in the following paragraphs. The overview of the ATTEMPT method can be seen in Figure 2.

Single-task training setting. To train ATTEMPT in a single-task setting, we first need a set of pre-trained soft prompts (that authors call **source prompts**) and choose a soft prompt to initialize the target prompt (the target prompt can be also initialized with random weights, but authors used one of the pre-trained source prompts to initialize the target prompt). The target prompt is trained similarly to the prompt-tuning prompt (a matrix of parameters that is prepended to the matrix of input embeddings). What ATTEMPT does on top of that is to add a weighted sum of source prompts to the target prompt to produce an **instance prompt**. The weighted sum is calculated using attention scores from the attention module.

Multi-task training setting. To train ATTEMPT in a multi-task setting, we can train the target prompt similarly to the single-task setting, but concatenate multiple datasets into a single training dataset. We can then train the target prompt on a single train set and evaluate it on multiple evaluation sets separately. Multi-task ATTEMPT can be also trained with a shared attention module for multiple tasks. This means that for each dataset, we have a separate target prompt identified by a task ID. We then assign a task ID to each dataset before training. During training, we then retrieve the right target prompt based on the task ID of the input data. The process of retrieving the right target prompt is depicted in Figure 3. After we retrieved the right prompts for every input in the batch, we can continue with the instance prompt calculation as mentioned in the single-task training setting. This will increase the overall trained parameters, but the usage of only a single shared attention module for multiple target prompts compensates for the increase.

Attention module. The role of the attention module is to determine a score for the contribution of each source prompt based on the model input X, source prompts P, and the target prompts P_{target}. Since $X \in \mathbb{R}^{n \times e}$ and $P_j \in \mathbb{R}^{p \times e}$ have different sequence lengths, the attention module first does max pooling over the model input and source prompts to get $\hat{X} \in \mathbb{R}^e$ and $\hat{P}_j \in \mathbb{R}^e$. After the max pooling of a sub-network $\mathcal{G}$ projects the input $\hat{X}$ into the space of source prompts. The sub-network $\mathcal{G}$ consists of one downsampling fully connected input layer $H_{down} = W_{down}^T(\hat{X})$ and one upsampling fully connected layer with a SiLU (Elfwing et al., 2018) non-linear activation function $H_{up} = W_{up}^T(SiLU(H_{down}))$. As an output layer, there is a layer norm layer $H_{out} = LayerNorm(H_{up})$ after the upsampling layer. Finally, the attention module computes the attention score a_j by multiplying the $\hat{P}_j$ and H_{out} and applies a softmax over the scores as follows.

ATTEMPT also scales the attention scores with temperature T (Radford et al., 2021) to avoid making the attention over-confident. To calculate an instance prompt ATTEMPT adds a weighted sum to the target prompt as follows:

$$P_{instance} = P_{target} + \sum_{j=1}^{t+1} a_j P_j \tag{1}$$

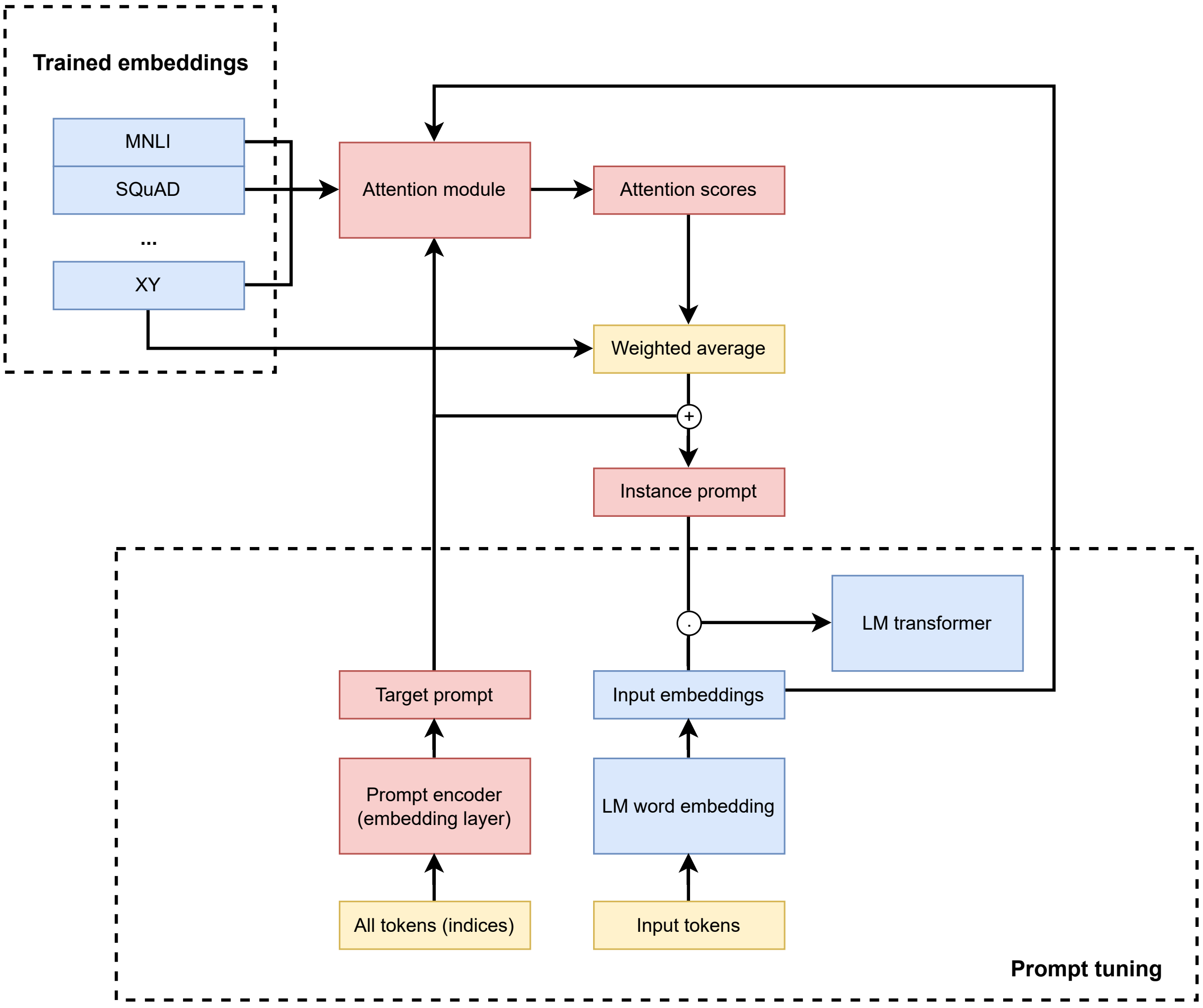

Figure 2: Diagram representing training process of the ATTEMPT method. The blue color represents components with frozen parameters and the red color represents components with trainable parameters. The yellow color represents components without any weights (i.e. model inputs and outputs or utility functions). The dot sign operation represents prepending the instance prompt to the input embeddings. The plus sign operation represents the addition of weighted average interpolation and target prompt from eq. 1.

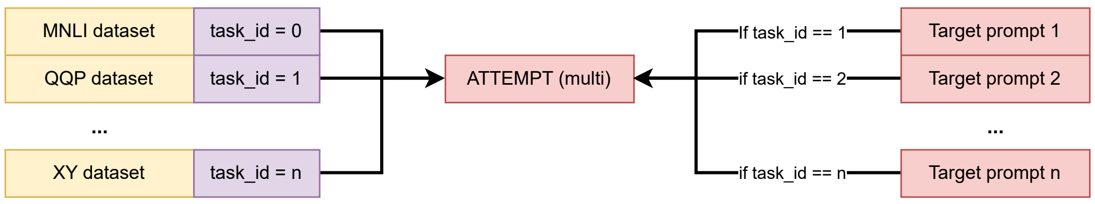

Figure 3: Diagram representing the process of target prompt selection when using shared attention across multiple target prompts. The red color represents components with trainable parameters and the yellow color represents components without any weights (i.e. model inputs and outputs or utility functions). The purple color is to represent added information to datasets.

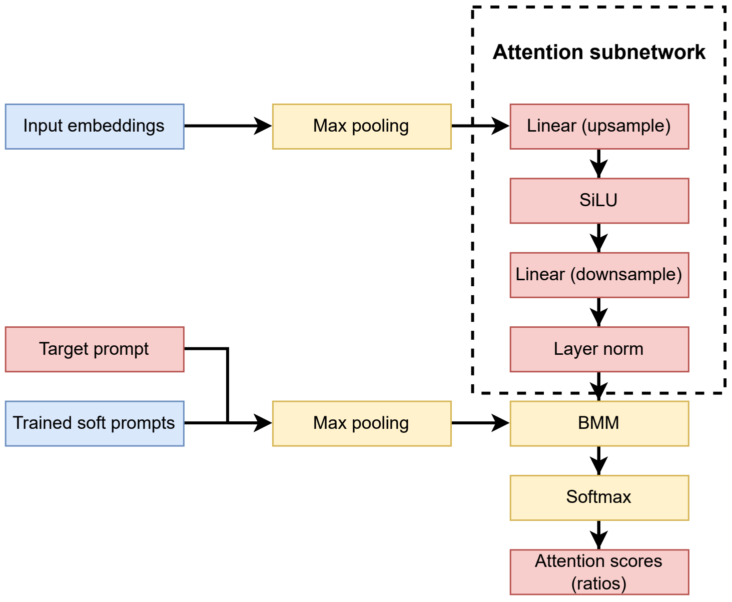

Figure 4: Diagram representing the architecture of attention module. The blue color represents components with frozen parameters and the red color represents components with trainable parameters.

Where P_{target} represents the task-specific part and the weighted sum represents a composition from different tasks that differs from different instances of the same task. As shown from eq. 1, the selection of $1 + a_{t+1}$ weights for the target task enables the ATTEMPT to use the knowledge from the target prompt when the knowledge from soft prompts is insufficient. These are some of the theoretical details from the ATTEMPT article, that we have built upon in the implementation phase of our replication study.

4 Implementation

Implementing the replicated method is an important part of our replication study. We implement the prompt tuning parameter-efficient fine-tuning method as well as the ATTEMPT parameter-efficient fine-tuning method. We use Python with deep learning modules (i.e. PyTorch, Transformers). All of our source code can be found in our GitHub repository[1]. Required Python packages are in the *requirements.txt* file. The original ATTEMPT implementation can be found in the authors' repository[2].

The *run.py* script creates a *PeftTraning* object that contains all the information about a single training run and handles the data pre-processing and training in the *run* method. During the dataset pre-processing, each dataset has a specified preprocessor function (in *tasks/tasks.py* file) to transform data into text-to-text setting and a formater function to put the transformed data into seq-2-seq format. The ATTEMPT authors use the preprocessing available in the T5 implementation, but instead of using words for classification (i.e. entailment, neutral, contradiction), the authors used numbers for classification (i.e. 0, 1, 2). We suspect that this change may result in pre-trained T5 model sub-optimal performance[3]. However, to achieve similar results as ATTEMPT, we used the same preprocessing as the authors did. The datasets are then split into training, validation, and test sets. Large datasets (over 10k samples) have a validation set split into 1000 validation samples and the rest for the test set; the small datasets (less or equal to 10k samples) have a validation set split into two halves, which are validation and test sets. We have used seed 42 to match the authors' seed for the dataset shuffle.

The *PeftTraning* also creates a *PeftConfig* and initializes a pre-trained version of the T5 model (Raffel et al., 2020). The *PeftConfig* is then inserted into the *get_peft_model* method, which creates and initializes the *PeftModel* based on the config. The config contains the information about the type of task (in our case seq-2-seq language model) and the type of parameter-efficient fine-tuning method (in our case prompt tuning or ATTEMPT). We also implement *save_pretrained*, *from_pretrained*, *forward* and *generate* methods. Since we have decided to implement prompt tuning from scratch, we built a parameter-efficient

[1] `https://github.com/DisAI-Replication-Challenge/ATTEMPT`
[2] `https://github.com/AkariAsai/ATTEMPT`
[3] We have also approached the authors to discuss this (and other) possible issues, but unfortunately, we did not receive an answer at the time of writing this report.

fine-tuning framework called **CPEFT** for **C**ustom **PEFT**, which is a custom remake of Huggingface parameter-efficient fine-tuning module (Mangrulkar et al., 2022) that we took inspiration from.

4.1 Prompt Tuning Implementation

Prompt tuning introduces a new variable to set the length of the prompt to be prepended to the input and a new variable to set the initialization of the prompt. The prompt can be initialized with random numbers, with random embeddings from the model vocabulary, and with single or multiple pre-trained prompt embeddings. During initialization, the prompt encoder Pytorch module is created and appended to the base model. The prompt encoder for the seq-2-seq language model is initialized with double the size of the prompt since the seq-2-seq architecture consists of two separate networks. This is the behavior presented in the Huggingface parameter-efficient fine-tuning module (Mangrulkar et al., 2022), but it does not match the implementation from the original prompt tuning paper (Lester et al., 2021). We have decided to use the Huggingface parameter-efficient fine-tuning behavior and just halve the size of the prompt encoder embeddings in configurations.

During the forward or generate of the model the method *get_prompt* is called. This method calls the forward function of the prompt encoder which returns the whole embedding matrix of the prompt encoder. This matrix is returned for each data in the batch and prepended to the input embeddings. After that, the result is inserted into the forward function of the base model. This implementation does not require to override of the original backward method or backpropagation calculation. During the saving and loading of pre-trained *PeftModel*, only the prompt encoder embeddings are saved, and loaded.

We train the source soft prompts individually for the SQuAD (Rajpurkar et al., 2016), SST-2 (Socher et al., 2013), QQP, QNLI (Wang et al., 2018), MNLI (Williams et al., 2018), and ReCoRD (Zhang et al., 2018) datasets. The training is set for 5 epochs and a single run with evaluation after each epoch. Weight decay for the AdamW optimizer is set to 1×10^{-5} with a linear scheduler with 500 warmup steps and a learning rate of 3×10^{-1}. The size of all soft prompts is 100. We use a maximum target length of 128 and a maximum input length of 512 for SQuAD and 256 for others.

4.2 ATTEMPT Implementation

The ATTEMPT method implementation includes an initialization of the prompt encoder with single or multiple pre-trained prompt embeddings, based on whether to train ATTEMPT in a single-task setting or multi-task setting. When initializing the ATTEMPT method, the prompt encoder and the attention module are created. The attention module consists of the sub-network module and the process of creating attention scores similar to the diagram in figure 4. The instance prompt is then created in the forward method of the *PeftModel* module.

The only difference in multi-task ATTEMPT is in the prompt encoder initialization and prompt fetching. While the single-task prompt embedding was just a single embedding, in the multi-task setting there is a *ModuleList* of embeddings for each task. Each embedding is then chosen based on the task IDs of the data. Similar to the prompt tuning, the instance prompt is then forwarded to the base model. During saving and loading of the model together with the prompt encoder embeddings also the attention module is saved and loaded.

We train ATTEMPT on 8 datasets from the GLUE (Wang et al., 2018) benchmark and 5 datasets from the SuperGLUE (Wang et al., 2019) benchmark. We train datasets over 10k samples for 10 epochs and the rest of the datasets for 20 epochs. We conduct 3 runs for each training configuration, initialize the target prompt embeddings with source prompts trained on the MNLI dataset, and use all of our trained soft prompts as source prompts. Weight decay for the AdamW optimizer is set to 1×10^{-2} with a linear scheduler with 500 warmup steps and a learning rate of 3×10^{-1}. The size of all soft prompts is 100. We use a maximum target length of 128 and a maximum input length of 348 for MultiRC and 256 for others. Another different setting from prompt tuning is that we pad the input to the maximum length of the T5 input token sequence.

The same settings are used in multi-task training except that we are using shared attention in every case. We are also not using a different learning rate for the attention sub-network and we are not using pre-trained weights for attention sub-network initialization.

dataset	SQuAD	SST-2	QNLI	MNLI	QQP	ReCoRD	avg.
Authors' soft prompts	31.7	63.7	92	62.9	92.3	82.9	70.9
Our soft prompts	68.8	95.4	95.5	84.6	94.2	82.1	86.8

Table 1: Evaluation of soft prompts provided by authors and our trained soft prompts. We have used accuracy for all of the datasets.

	GLUE									SuperGLUE					
dataset	MNLI	QQP	QNLI	SST-2	STS-B	MRPC	RTE	CoLA	avg.	Multi	BoolQ	WiC	WSC	CB	avg.
ATTEMPT single	**84.3**	**90.3**	93	93.2	89.7	85.7	73.4	57.4	83.4	**74.4**	**78.8**	**66.8**	53.8	78.6	70.5
ATTEMPT multi	83.7	90.1	**93.2**	**94.3**	**90.8**	**87.3**	**82.7**	64.3	**85.8**	**74.4**	78.5	66.5	**69.2**	82.1	**74.1**
Authors' soft prompts single	$72.6_{1.7}$	$\mathbf{90.3_{0}}$	$92.4_{0.3}$	$92.9_{0.2}$	$90_{0.5}$	$84.5_{2.3}$	$63.1_{1.1}$	$\mathbf{76.3_{5.9}}$	$82.8_{1.5}$	$48_{41.5}$	$70.2_{7.3}$	$60_{6.6}$	$64.7_{4.4}$	$67.9_{9.4}$	$62.2_{13.8}$
Our soft prompts single	$83.8_{0.2}$	$\mathbf{90.3_{0}}$	$92.7_{0.3}$	$89.4_{1.3}$	$90_{0.4}$	86.4_{2}	$74.8_{0.7}$	$72.8_{2.3}$	$85_{0.9}$	$71_{0.9}$	$75.1_{0.5}$	$57.8_{7.7}$	$66_{1.1}$	$77.4_{2.1}$	$69.5_{2.5}$
Authors' soft prompts multi	62.1_{7}	$87.7_{0.8}$	$90.4_{0.3}$	$91.1_{1.5}$	$89.6_{1.5}$	72.1_{1}	$47_{4.2}$	$69.4_{0.2}$	$76.2_{2.1}$	$71.7_{0.9}$	$68.9_{6.2}$	$59.6_{4.1}$	$34_{7.7}$	$65.5_{19.7}$	$59.9_{7.7}$
Our soft prompts multi	$83.5_{0.2}$	90_{0}	$92.4_{0.2}$	$90.3_{0.9}$	$90.1_{0.3}$	$81.7_{1.2}$	$73.6_{2.2}$	69.5_{0}	$83.9_{0.6}$	$68.2_{0.6}$	$75_{0.7}$	$51.3_{6.4}$	$56.4_{9.1}$	$\mathbf{84.5_{5.5}}$	$67.1_{4.5}$

Table 2: Test results of our ATTEMPT implementation compared to the results provided by authors. The results are calculated as a mean across 3 runs. We have used Pearson Correlation for STS-B, F1 macro for MultiRC (Multi), and accuracy for other datasets. The first two rows represent results provided by the authors in the ATTEMPT paper.

5 Experiments and Results

Since our replication study focuses mainly on replicating ATTEMPT results, we did not replicate the results provided by the prompt tuning authors; we only compared our results to source prompts provided by the ATTEMPT authors[4]. All of our experiment results and saved weights were documented in Weights & Biases projects, which are available online[5]. We are executing the experiments individually per configuration on a single Nvidia A10, A40, or A100. There is also a config file available for each of the set of experiments, we have created config files for prompt tuning, ATTEMPT single with authors' source prompts, ATTEMPT single with our source prompts, ATTEMPT multi with authors' source prompts, ATTEMPT multi with our source prompts. The ATTEMPT experiments set is multiplied by the number of dataset sets used.

5.1 Prompt Tuning

Better source prompt performance. Based on the results of source prompt training in Table 1, we can say that our source prompts are on average performing better than source prompts provided by authors. These results were not expected, as we followed the authors' hyperparameter settings and only trained the source prompts for 5 epochs. Since we trained the source prompts only for 1 run, we cannot determine stability across multiple runs.

The difference from the authors' results may be caused by the source prompt initialization from T5 vocabulary, which tends to increase instability as reported by ATTEMPT authors Asai et al. (2022). There may be also other randomness factors that we did not take into account, which may have caused the results to differ. Authors are also using their custom implementation of prompt which includes adapting and changing the original T5 code from the transformers library which may behave differently from our adapted CPEFT solution.

5.2 ATTEMPT

Better-performing source prompts over multi-task training. The results from ATTEMPT experiments in Table 2 show that the single-task method with our trained source soft prompt almost matched the authors' multi-task ATTEMPT results in average GLUE datasets score. This leads us to conclude that better-performing source prompts benefit the ATTEMPT performance. However, multi-task training splits the number of trained parameters over all trained tasks, which makes it more efficient compared to single-task training and more suitable for multi-task problems. Another observation is that with the increase of source prompts performance, the overall ATTEMPT performance also increases. This can mean that if the target prompt reaches a point in training in which it outperforms the source prompt attention interpolation the source prompts may start to hold back the target prompt. We can also see that better-performing source prompts tend to increase the stability of multi-task training.

[4]`https://homes.cs.washington.edu/~akari/models/attempt/source_prompts.zip`
[5]`https://github.com/DisAI-Replication-Challenge/ATTEMPT#experiment-results`

	GLUE									SuperGLUE					
dataset	MNLI	QQP	QNLI	SST-2	STS-B	MRPC	RTE	CoLA	avg.	Multi	BoolQ	WiC	WSC	CB	avg.
ATTEMPT single	84.3	90.3	93	93.2	89.7	85.7	73.4	57.4	83.4	**74.4**	**78.8**	66.8	53.8	78.6	70.5
ATTEMPT multi	83.7	90.1	93.2	94.3	**90.8**	**87.3**	**82.7**	64.3	85.8	**74.4**	78.5	66.5	**69.2**	82.1	**74.1**
Authors' soft prompts single	75.4	93.9	94.6	**95.5**	88.9	86.3	75.3	**77.2**	85.9	68.6	77.3	**67.4**	59.6	78.6	70.3
Our soft prompts single	84.6	**94.3**	**95.4**	93.4	89.5	87.2	81.9	75.6	**87.7**	74.2	76.8	65.8	67.3	82.1	73.2
Authors' soft prompts multi	75.3	93.8	93.5	95.4	88.9	80.9	60.1	68.7	82.1	68.9	75	60.2	59.6	78.6	68.5
Our soft prompts multi	**84.8**	94.2	92.5	87.1	88.6	78.9	77.5	68.7	84	69.4	76	64.3	63.5	**85.7**	71.8

Table 3: Cherry picked results of our ATTEMPT implementation - best validation results over all runs. We have used Pearson Correlation for STS-B, F1 macro for MultiRC (Multi), and accuracy for other datasets. The first two rows represent results provided by the authors in the ATTEMPT paper.

Stability problems across multiple runs. We have noticed training instability across multiple runs of ATTEMPT, especially in smaller-size datasets (less than 10k samples). The instability may be caused by the random weight initialization and since we did not use the seed for the weight (only for dataset shuffle) of the attention module, randomness factors may be another reason why our results differ from the authors' results. Since the ATTEMPT authors did not provide any information about stability, we have chosen to select also the best validation results across all of the runs to see how the results shift. These results can be seen in Table 3 and are called cherry-picked results. We can see that cherry-picked results can increase the overall GLUE and SuperGLUE score of our ATTEMPT implementation, but these results do not say anything about the true ATTEMPT performance.

The need for pre-trained attention of multi-task ATTEMPT. We were not able to match the results of multi-task ATTEMPT, but we suspect that one of the reasons why our implementation underperformed the authors' multi-task ATTEMPT implementation is the lack of pre-training of the attention module. We were not able to retrieve more information about the pre-training of the attention module from the ATTEMPT paper, therefore we have decided to not pre-train the attention module. We also did not set a separate learning rate for the attention module sub-network, which may be another cause of why we ended up with different results.

Overall ATTEMPT Results. Our experiments with single-task ATTEMPT achieved a better average GLUE benchmark score than the results reported in the ATTEMPT paper by ATTEMPT authors and almost matched the SuperGLUE benchmark scores. The multi-task ATTEMPT experiments did not achieve better results on both benchmarks and our multi-task ATTEMPT results are lower than the single-task ATTEMPT results. We suspect that the requirement of the attention module pre-training may be crucial for yielding better results for the multi-task training, since it may be harder for the attention module to adapt for multiple tasks from scratch. Another reason for not achieving the exact results as provided by the ATTEMPT authors may be the randomness factors. Our prompt initialization, data splits, and even the training environment (i.e. GPU, Python modules versions) were not necessarily the same, which may have caused differences in training.

6 Conclusion

In our replication study, we have successfully replicated the parameter-efficient fine-tuning method presented in the paper ATTEMPT: Parameter-Efficient Multi-task Tuning via Attentional Mixtures of Soft Prompts (Asai et al., 2022). Based on the results from conducted experiments, we have identified that better-performing source prompts in single-task ATTEMPT training achieve on average better results even when compared to multi-task training. We also discuss the stability problems that we have faced during ATTEMPT training and the possible need for pre-training of the attention module for multi-task ATTEMPT training.

Furthermore, we would like to conduct extended experiments with ATTEMPT and investigate how dataset size and number of trained source prompts affect the performance of ATTEMPT. At the same time, we would like to investigate the transferability of source prompts trained on tasks in multiple languages for multi-lingual tasks. Lastly, we would like to look at the architecture of ATTEMPT and its attention module to investigate, whether there are other ways how to look at attentional task transferability, like replacing the max pooling with another transformation that retains more information.

Acknowledgment

This research was partially supported by DisAI, a project funded by Horizon Europe under GA No.101079164. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Akari Asai, Mohammadreza Salehi, Matthew Peters, and Hannaneh Hajishirzi. 2022. ATTEMPT: Parameter-efficient multi-task tuning via attentional mixtures of soft prompts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6655–6672, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Alexandra Chronopoulou, Matthew Peters, Alexander Fraser, and Jesse Dodge. 2023. AdapterSoup: Weight averaging to improve generalization of pretrained language models. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 2054–2063, Dubrovnik, Croatia. Association for Computational Linguistics.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11.

Gamaleldin F. Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. 2019. Adversarial reprogramming of neural networks. In *International Conference on Learning Representations*.

Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2022. PPT: Pre-trained prompt tuning for few-shot learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8410–8423, Dublin, Ireland. Association for Computational Linguistics.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: Word-level Adversarial ReProgramming. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4921–4933, Online. Association for Computational Linguistics.

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

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2.

Quoc Le, Tamás Sarlós, and Alexander Smola. 2013. Fastfood-computing hilbert space expansions in loglinear time. In *International Conference on Machine Learning*, pages 244–252. PMLR.

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.

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, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023. Gpt understands, too. *AI Open*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapter-Fusion: 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.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. *OpenAI*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou', and Daniel Cer. 2022. SPoT: Better frozen model adaptation through soft prompt transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5039–5059, Dublin, Ireland. Association for Computational Linguistics.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

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.

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. 2023. Multitask prompt tuning enables parameter-efficient transfer learning. *arXiv preprint arXiv:2303.02861*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Ping Yu, Wei Wang, Chunyuan Li, Ruiyi Zhang, Zhanpeng Jin, and Changyou Chen. 2022. Stt: Soft template tuning for few-shot adaptation. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 941–946. IEEE.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*.