4. Adapters in Machine Translation

This chapter aims to study the impact of pre-trained models when fine-tuning machine translation models with adapters. Specifically, we are interested in understanding the contribution of good representation in the pre-trained language model to the adapters during fine-tuning. Furthermore, we are also interested in understanding the capability of adapters to perform with artificially degraded pre-trained models. We propose to use transformer-based architecture such as BERT (pre-trained or trained from scratch) for both the encoder and decoder components while using adapters for fine-tuning. To be more specific, we divide the experiments into four different pre-trained model setups:

- Use BERT weights[1] as the pre-trained weights (`Pre-trained BERT`). We use this model as the baseline for our experiments in this chapter.
- Use randomly initialised BERT models and pre-train the models with MLM objective on IWSLT and WMT data (`Pre-trained Transformer`). This setup is used to understand the impact of different data volumes and domains on the quality of the pre-trained models that will be fine-tuned with adapters in the later phase.

[1]We use publicly available BERT weights from Huggingface hub `https://huggingface.co`

- Use the pre-trained BERT model where the weights within the same layer are shuffled (`Pre-trained shuffled`). We use this setup to understand whether the adapters can capture important features from the original BERT weights and restore the performance even though the weights are no longer in the original position.
- Use randomly initialised BERT models with no pre-training (`Pre-trained random`). We use this setup as a complement of `Pre-trained shuffled` experiments to understand the impact of pre-trained models that contain relatively inferior knowledge than the original BERT model on the adapters when fine-tuned in the MT task.

4.1 Experiments Setup

4.1.1 Language Model

Dataset

From [29], we understand that BERT was trained with billions of words from various sources and domains. However, we do not fully understand the proper condition to stop adding more sentences to the pre-training data so that we can reduce the hours of training the model. Additionally, we do not know the impact of combining different domains in the pre-training data on the final performance of the model. For those reasons, we are reducing the scope of the pre-training data by restricting the creation of pre-training data only from machine translation datasets in two different domains and constructing the pre-trained models with different sizes of data. We use WMT and IWSLT for the experiment as WMT and IWSLT contain different domains, and WMT has a significantly larger volume and longer sentences. Specifically, we construct three different datasets with different volumes:

1. A standalone IWSLT.

Dataset	**Initial Weights**	**Used for pre-training?**	**Used for fine-tuning?**
Standalone IWSLT	Pre-trained BERT	No	Yes
	Random BERT	Yes	Yes
IWSLT + WMT 500k	Random BERT	Yes	No
IWSLT + WMT 2m	Random BERT	Yes	No

Table 4.1: The initial weights represent the weights used on both the encoder and decoder. Pre-trained BERT refers to the pre-trained models that use BERT. Random BERT models use BERT configuration but not the weights. Therefore, contrary to the Pre-trained BERT, Random BERT models undergo a pre-training process before fine-tuning. The pre-trained and fine-tuned columns represent the boolean value to mark whether the models initialized with the targeted weights are pre-trained or fine-tuned with the respective data. For example, Huggingface BERT weights are not pre-trained with IWSLT but use IWSLT for fine-tuning.

2. A combination of IWSLT and WMT data with a total of 500k sentences. We add the WMT data on top of IWSLT to increase the volume of our pre-training data. We should also note that the domain of WMT differs from the IWSLT, and we only test the models on the IWSLT test set. The WMT data can thus lead to worse translation performance due to domain mismatch.

3. A combination of IWSLT and WMT data with 2 million sentences. The same as (2) but with larger volumes.

Datasets (2) and (3) were constructed with a simple approach by using all training data from IWSLT, randomly selecting sentences from the WMT dataset, and combining them until we met the required number of sentences mentioned in the previous paragraph.

To illustrate how the datasets are used in the experiments, we refer to Table 4.1. We can see that datasets other than standalone IWSLT are used only for pre-training. The IWSLT is always used for fine-tuning, and depending on the initial weights, we also use it as pre-training.

Model

To pre-train the model, we follow the work of [29] by using the Masked Language Model (MLM) objective. In every sentence, some words will be masked, and the model has to predict the original words. A complete illustration can be found in Figure 4.1.

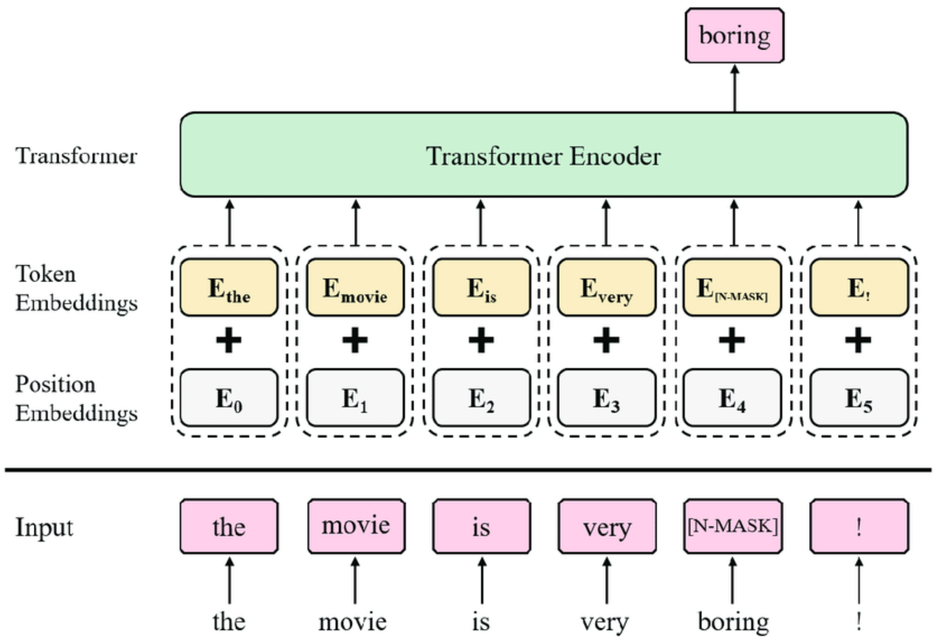

Figure 4.1: Illustration of the MLM objective during the pre-training. The illustration is reprinted from [53].

The MLM is introduced as an alternative objective to the auto-regressive language model. The auto-regressive objective is inherently different from the MLM. In auto-regressive, the models have to predict the word at step t, and they only have the ability to look at the previous words ($t-1...0$) as the context. Auto-regressive is used in a couple of works such as GPT models ([15, 63, 62]) for pre-training. One of the advantages of using the MLM compared to the auto-regressive is that we can exploit the bidirectional context rather than only predicting the words from left to right.

We use the default BERT configuration from Huggingface to construct the model. The complete list of hyperparameters can be found in Table 4.2.

Name	**En**	**De**	**Description**
vocab_size	30522	31102	Vocabulary size of the BERT model
hidden_size	768	768	Dimensionality for each of the layers
num_hidden_layers	12	12	Number of hidden layers in the transformer
num_attention_heads	12	12	Number of attention heads for each attention layer
intermediate_size	3072	3072	Dimensionality of the feed-forward layer
hidden_act	gelu	gelu	The activation function within the layer
learning_rate	0.0005	0.0005	Learning rate used for the experiments
optimizer	adam	adam	Model optimizer
batch_size	64	64	Batch size used for the experiments
hidden_dropout_prob	0.1	0.1	The dropout probability for all fully connected layers in the embeddings, encoder, and pooler
attention_probs_dropout_prob	0.1	0.1	The dropout ratio for the attention probabilities

Table 4.2: Transformer BERT hyperparameters for English (based on `bert-base-uncased`) and German (based on `bert-base-german-dbmdz-uncased`).

We train the model until convergence with a maximum of 500 epochs. We define convergence by training for at least one day, with the validation loss no longer improving. We train the models in the MetaCentrum cluster with a specific configuration where the running jobs will be automatically killed after 24 hours. The loss usually starts to converge in less than 24 hours, but in some cases, we still see some models that are not yet converged. In this case, we continue the training by running another job process and loading the last checkpoint. On the other hand, if the loss starts to diverge before 24 hours or 500 epochs, we stop the training manually and select the checkpoint with the lowest loss score. Each language may end up converging in a different number of steps.

4.1.2 Machine Translation

Dataset

In machine translation experiments, there are several scenarios in which we use both WMT and IWSLT datasets. The IWSLT dataset is mainly used to fine-tune and evaluate the final model. The WMT, on the other hand, will be combined with IWSLT for training the baseline models. We use three different datasets for the experiments similar to the language model setup: IWSLT standalone, IWSLT + WMT (500k), IWSLT + WMT (2 million). To be more specific, we use the IWSLT standalone dataset for training, evaluation, and testing. The remaining data setups are used only for training, and we limit ourselves to the IWSLT data for evaluation and testing.

Model

We use the seq2seq architecture by [78]. The seq2seq architecture contains two different components, the encoder and the decoder. This architecture's details and illustration have been described in Section 1.2.3. The encoder and decoder use the same model and hyperparameters as described in Section 4.1.1. The only modification made from the original language model is on the decoder side. On the encoder side, we only use the self-attention layers to gather some context from the neighbours within the same layer. On the other hand, the decoder requires further context to generate its outputs by including the vector representation from the encoder as additional features. For this reason, an extra component named `cross_attention` layer is introduced in the decoder, and it is trained from scratch in all experiments.

4.2 Experiments Results

This section discusses the results obtained for machine translation. First, in Section 4.2.1, we conduct the comparative study of using adapters in different pre-trained scenarios. Second, in Section 4.2.2, we perform a study by replacing the BERT model with a different BERT version where the weights are shuffled. We continue the experiments using pre-trained models with completely random weights and no pre-training. Finally, in Section 4.2.3, we perform experiments comparing the performance of the randomly set weights pre-trained model that is fine-tuned with adapters with the baseline models as well as models that are pre-trained with the combination of WMT and IWSLT.

4.2.1 Adapters Comparison

Experiment Setup and Motivation

In this section, we conduct experiments to understand the contribution of adapters by comparing trained and fine-tuned models with different sizes of datasets. The definition of the dataset is the same as have already explained in the previous section. The dataset is used for two different purposes:

- Used to train the BERT-style language model from scratch, separated for source and target languages.

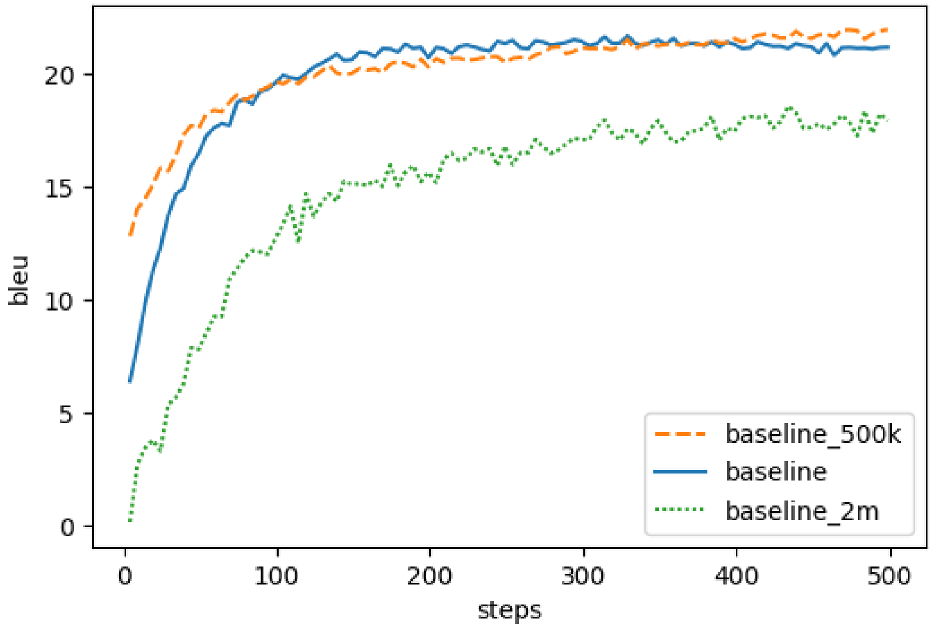

Figure 4.2: Comparison between baseline models trained with different size of datasets. `baseline` represents the model trained only using IWSLT; `baseline_500k` represents the model trained using IWSLT and WMT with total of 500k sentence pairs; `baseline_2m` represents the model trained using IWSLT and WMT with total of 2 million sentence pairs.

- Used for pre-training and later fine-tuning the seq2seq model on IWSLT parallel data.

Experiment Results

In this section, we compare the result of the baseline models with the fine-tuned models with adapters. Our first experiment compared the results from an empty BERT model (a transformer model with BERT configuration but no pre-trained BERT weights) without any pre-training and adapters and only trained the model from scratch. We refer to these models as our `baseline` for the rest of this chapter. From Figure 4.2, we can see that adding more data to the baseline models does not necessarily improve the performance. We suspect the models require more training time to get the best final performance. There is a clear gap between `baseline_2m` and the rest of the baseline models. `baseline` and `baseline_500k` perform really well from the start while `baseline_2m` lags behind. We suspect this is due to the effect of including more sentences from different domains. We found that `baseline_500k` performs the best as it balances the models to avoid overfitting in the intended domain and still benefits from out-of-domain data. `baseline_2m` shows the impact on domain difference where it has relatively lower performance than the other models. However, we must acknowledge that the model has not fully converged in the limited training time (measured by the number of steps) we have and has a chance to improve its performance further. Despite the lower performance in BLEU score, `baseline_2m` deserves a manual evaluation of the translation output. We argue that in some cases, lower BLEU may not necessarily reflect a lower quality. We perform the manual evaluation at a later stage in this chapter.

To see the impact of adapters, we compare the result on different sizes of pre-training data used for the base model. The base models are then fine-tuned with the adapters module on the IWSLT data. As we can see from Figure 4.3, pre-trained BERT with adapters achieves the best performance from the earlier steps compared to the rest of the models. In contrast to the baseline models, we see the benefit of adding more sentences to the pre-training (see `adapters_pt_2m` vs `adapters_pt_500k`). From Figure 4.2 we see that `baseline_500k` outperforms `baseline_2m` by approximately 5 BLEU despite larger training data on `baseline_2m`. Note that the training data we refer to in the baseline models is the training data for MT. On the other hand, when we add more training data on the pre-training side, we can see that the model trained using 500k data has a lower performance than the one using 2 million data. This tells us that adding larger pre-training monolingual data and then fine-tuning the models with adapters impact the performance positively.

In the model that only uses IWSLT as the pre-training data, we observe performance degradation in the middle of the fine-tuning. We found this is due to the gradient explosion in the cross-attention layer, and we attribute this instability

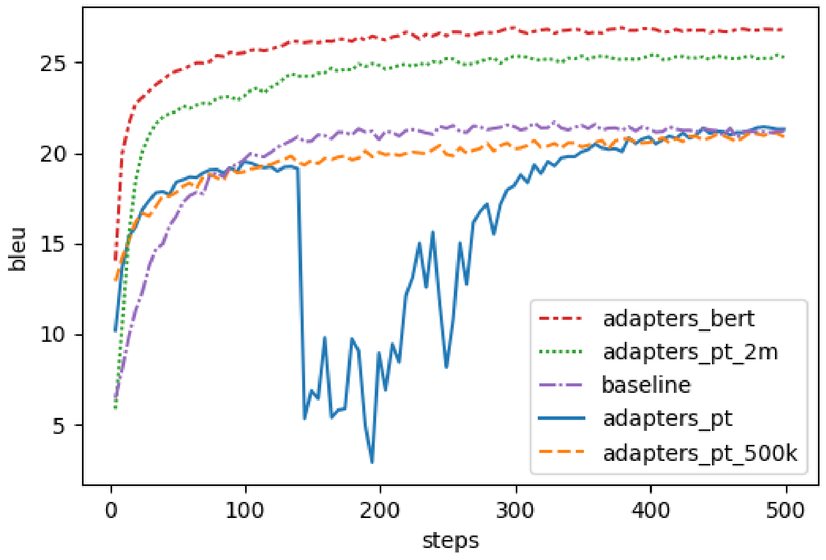

Figure 4.3: The effect of pre-training data size when the adapters are then trained on the same IWSLT data. `baseline` represents the model trained only using IWSLT; `adapters_pt` represents the model pre-trained only using IWSLT; `adapters_pt_500k` represents the model pre-trained using IWSLT and WMT with total of 500k sentence pairs; `adapters_pt_2m` represents the model pre-trained using IWSLT and WMT with total of 2 million sentence pairs; `adapters_bert` represents the model that uses BERT weights.

to the small data used in the pre-training. In the later steps, the IWSLT model eventually achieved performance similar to the 500k model.

4.2.2 Random and Shuffled Pre-trained Weights

Experiment Setup and Motivation

We perform two different categories of experiments to study how the adapters benefit from the pre-trained weights. First, we conduct experiments where we shuffle the BERT weights. We separate the weights initialization into two approaches to perform the experiments:

- We shuffled the weights from the column perspective. We shuffled the column-wise order of all the matrices while keeping the row-wise order intact.

- We shuffled the weights from both column and row perspectives.

Second, we conduct experiments where we set random weights on all base network layers and treat them as the pre-trained model. During the fine-tuning, we only update the adapter weights and keep the random weights intact.

Experiment Results

We can see from Figure 4.4 that the performance of the model that uses random pre-trained weights is more stable than the one using shuffled BERT weights. All shuffled BERT models suffer from gradient explosion similar to the IWSLT model we show in the previous section. Furthermore, shuffling the weights on both the row and column sides seems more detrimental than just shuffling on the column side. This may show that there could be some pattern in `adapters_shuffled` that the adapters can eventually help to recover during the fine-tuning, while on `adapters_shuffled_both` more patterns are missing and more challenging to recover.

Although the performance of the random model (`adapters_random`) is still below the baseline model, it is interesting to see that fine-tuning only the adapters, cross-attention, and output layers is sufficient to achieve a reasonable BLEU score, considering that the pre-trained model does not contain meaningful information relative to the original BERT weights. Furthermore, it is also interesting to see that compared to `adapters_shuffled_both`, we can get better final performance even though we can categorize `adapters_shuffled_both` as a model that uses randomly set weights pre-trained model. We hypothesize that this is probably related to the difference in the distribution of the weights between the models. In `adapters_random` we are limited to the distribution of the weights in the initialization algorithm used in BERT, while in `adapters_shuffled_*` the distribution may be different as the weights in pre-trained BERT have already been optimized with the MLM objective.

Since we are relying on the pre-trained model with a random set of weights and with no further training, there is a possibility that our method may only work

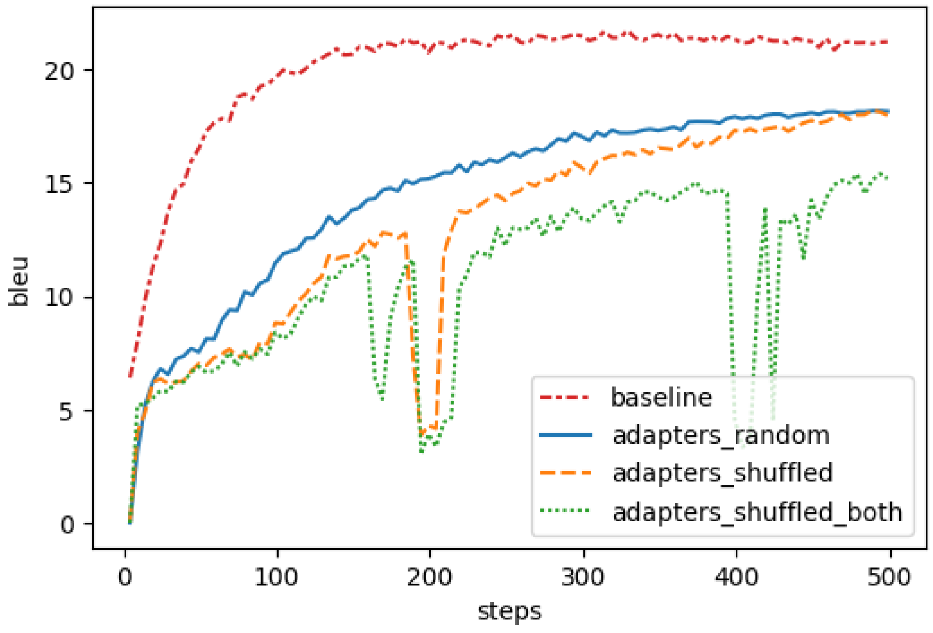

Figure 4.4: Comparison between adapters using shuffled BERT and random weights as the pre-trained models. `baseline` represents the model trained only using IWSLT; `adapters_random` represents the model pre-trained only using random weights; `adapters_shuffled` represents the model pre-trained using column-wise shuffled BERT model; `adapters_shuffled_both` represents the model pre-trained using shuffled BERT model.

in a single random seed. To ensure a robust experiment, we repeat the random experiments ten times with ten different random seeds. We can see from the result in Figure 4.5 that all the random seed performs similarly to one another.

4.2.3 Random Pretrained vs. Out-of-Domain Data

Experiment Setup and Motivation

In this experiment, we use the same setup as in Sections 4.2.1 and 4.2.2. Specifically, we are interested in investigating the model's performance that uses randomly set weights as the pre-trained model compared to the baseline model. We recall that we can gain a reasonable score when fine-tuning the model with adapters from the previous experiments using randomly set pre-trained weights on the transformer model. In this experiment, we are conducting further study to understand the

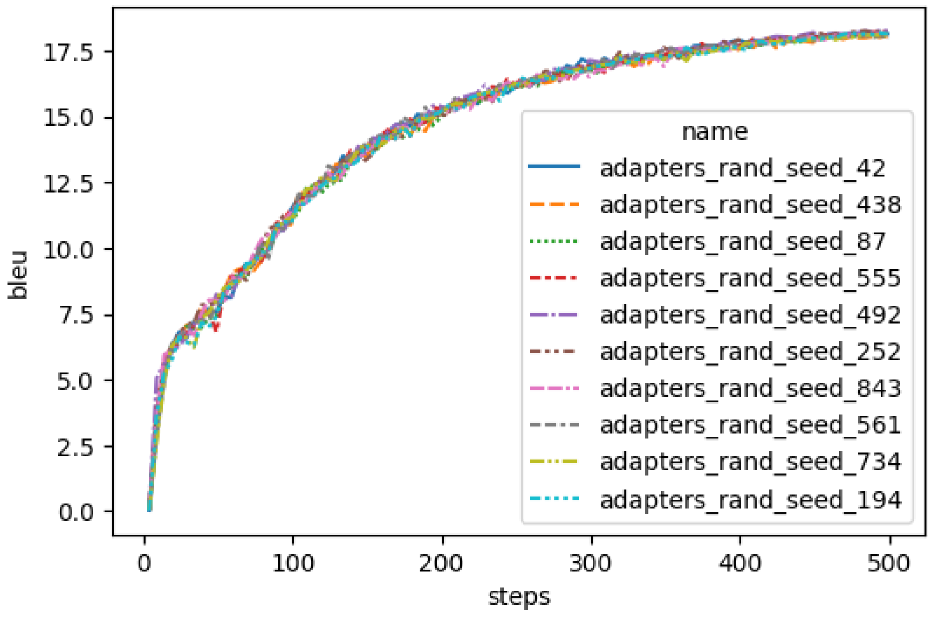

Figure 4.5: Comparison of different random seed for randomly set weights based models.

performance relative to the `baseline` model, where the model was trained using only IWSLT data and a different baseline (`baseline_2m`), where the model was trained using a mix of WMT and IWSLT. This comparative study aims to understand whether we can see the benefits of using adapters versus training the whole transformer model with bigger data sizes.

Experiment Results

We analyze the random weights by comparing the result with the best-performing baseline and our pre-trained transformer models. From Figure 4.6, we can see that the performance of the random model achieves a similar result as the baseline model that uses 2 million training sentence pairs. Recall that the baseline model is the BERT-style transformer model trained from scratch without fine-tuning and adapters. This tells us that training the whole model with bigger data does not necessarily improve the model's performance. It may need further tuning to gain the benefits of bigger data and bigger models because any departure from the

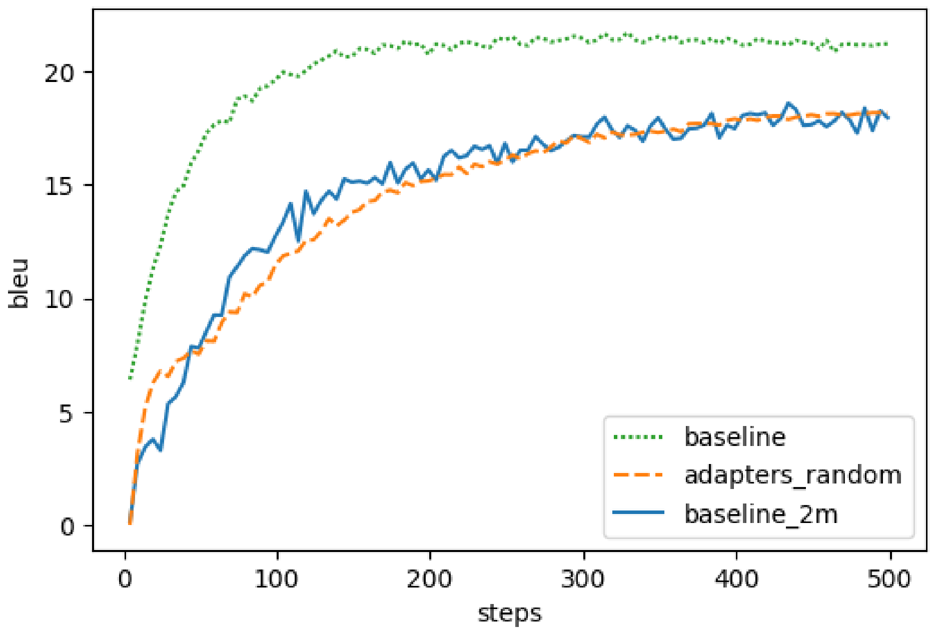

Figure 4.6: Comparison between pre-trained random weights and baseline model. `baseline` represents the model trained only using IWSLT; `baseline_2m` represents the baseline model trained with a combination of IWSLT and WMT sentence pairs; `adapters_random` represents the model pre-trained only using random weights.

domain of the test set can be more harmful than useful. We can see the result of using random weights as a potential alternative for training the model with small data such as IWSLT. While the performance is still far from the baseline (no adapters, full model training on IWSLT, i.e. in-domain data), this result shows that the base model's structure helps the adapter achieve a meaningful performance with a tiny portion of weights trained in the fine-tuning.

We perform a quick check of the output of the model in Table 4.3. From the first two lines, we can see that the model has difficulty capturing complex phrases. In the first row, the model missed **tanzen im tempel** which means **dancing in the temple**. For the second row, the model confuses **knowledge of the forest** and outputs **audience** instead. Furthermore, the model also does not translate the word **kenntnis** and makes the translation unclear since the object of the sentence is missing. Despite those mistakes, the model still can capture simple sentences, as shown in the third row. Another observation that we noticed in the generated

Random Weights + Adapters
Input: wir tanzen im tempel und werden zu gott. & quot ; **Reference**: we dance in the temple and become god . &quot; **Hypothesis**: we & apos ; re going to be able to become god. & quot ;
Input: aber gleichzeitig hatten sie eine klare kenntnis des waldes, die erstaunlich war. **Reference**: but at the same time they had a perspicacious knowledge of the forest that was astonishing. **Hypothesis**: but at the same time, they had a clear of the audience who was amazing.
Input: es ist so wunderbar. ihr musst es beschutzen. & quot ; **Reference**: it is that beautiful . it is yours to protect . &quot; **Hypothesis**: it & apos ; s wonderful. you have to protect it. & quot ;

Table 4.3: Prediction results from randomly set pre-trained model fine-tuned with adapters.

output is the tokenization of `&quot` and `&amp`. Instead of being treated as a single token, the tokenizer treats the token as three different subword tokens. We notice that this is due to the unavailability of the aforementioned token in the pre-trained BERT vocabulary.

4.3 Qualitative Comparison

We perform a manual check on the generated translations of some of our models. From Table 4.4, we can see that none of the models generated the correct result for the first example. However, BERT + adapters and 2 million pre-trained base models generate the proper context where the result is still discussing **the patient**. The wrong part is when the model generates an incorrect translation regarding the patient's disease. The second example shows that the BERT + adapters model creates a better quality of translation than the other models. The final example shows that BERT + adapters generate an interesting output where it manages to remove unimportant characters such as "`--`" and produce readable output. People may vary in their opinion on this example as the 2 million pre-trained base model

Sample Translation Output
Input: erinnerst du dich an den patienten mit dem gereizten rachen?
Reference: do you remember that patient you saw with the sore throat?
Hypothesis 1: do you remember reading to the patients? on the
Hypothesis 2: do you remember the patient with the tingling revenge?
Hypothesis 3: remember the patient with the bruised remorse?
Input: großartig, sagte ich. legte auf.
Reference: great, i said. got off the phone.
Hypothesis 1: great, i said. got up..
Hypothesis 2: great, i said. put on..
Hypothesis 3: great, i said. put it down.
Input: - - aber in unserer entdeckung der welt, haben wir alle arten unterschiedlicher methoden.
Reference: but in our discovery around the world, we have all kinds of other methods.
Hypothesis 1: but in our discovery of the world, we & apos ; ve got all sorts of different
Hypothesis 2: - - but in our discovery of the world, we have all kinds of different methods
Hypothesis 3: but in our discovery of the world, we have all sorts of different ways of doing things.

Table 4.4: Prediction results from **Hypothesis 1**: Baseline model trained with only IWSLT data; **Hypothesis 2**: Pre-trained model with adapters where we pre-train the model with IWSLT and WMT with a total of 2 million pre-training data; **Hypothesis 3**: BERT with adapters.

Input	**Output**
& quot ; moneyball & quot ; erscheint bald und dreht sich um statistiken und um diese zu nutzen ein großartiges baseball team aufzustellen.	& quot ; devilball & quot ; appears soon, and it & apos ;
moneyball erscheint bald und dreht sich um statistiken und um diese zu nutzen ein großartiges baseball team aufzustellen.	fatball soon appears and it turns out statistics and to use that to build a great baseball team

Table 4.5: An example of bad model behaviour when the input contains unknown tokens like `&quot` (top row). The translation is not abruptly cut if these symbols are removed (bottom row).

generates a more concise output.

We also notice that the BERT-based models have difficulties generating long sentences. The models always cut the translation short when an unavailable token such as `&quot` appears in the sentence. To give a more explicit example, Table 4.5 illustrates the significant difference in the outputs when the token `&quot` is observed in the input. This shows that tokenization plays an essential role in the model where tokens unavailable in the vocabulary list may affect the generated output significantly.

5. Adapters Effectiveness in Machine Translation

We continue the study from the previous chapter to understand more about the relation between adapters and pre-trained models. Similar to the previous chapter, we use BERT and its variants as the pre-trained models and fine-tune them with adapters. This study aims to evaluate the combination of adapters and BERT in machine translation and study the effectiveness of adapters by putting them only in the encoder or the decoder. We also experiment with down-scaling the pre-trained model size and try to recover the performance of the full-sized model. We separate the experiments into three different areas:

- Use BERT weights[1] as the pre-trained weights and investigate the importance of adapters in encoder or decoder.
- Use BERT weights and investigate their importance compared to random weights in the encoder or decoder while fine-tuning with adapters.
- Down-scaling BERT weights by either zeroing out half of BERT's weights (`zbert`) or completely removing them from the weight matrices, squashing the matrices (`zsbert`). We use the down-scaling technique to understand

[1]We use publicly available BERT model from Huggingface hub `https://huggingface.co`

whether we can use adapters to recover the performance of the original BERT (without adapters) while using fewer parameters.

5.1 Fixed Variable Parameters of Experients

5.1.1 Framework

As we mentioned at the beginning of the chapter, we have several scenarios we use to conduct the experiments. We start by describing the variables that we fixed throughout the experiments. As we have mentioned in Chapter 3, we use Huggingface as our main framework with added modifications for adapters. Contrary to Chapter 4, we do not investigate language models that we train ourselves, but instead, we focus only on the BERT language model.

The model and hyperparameters that we use throughout the experiment remain the same as described in Chapter 4. We use transformer model with seq2seq architecture and the BERT-based hyperparameter configuration to initialize both the encoder and the decoder.

5.1.2 Dataset

As mentioned in the previous section, our focus in this chapter is on machine translation. We use IWSLT dataset to perform the fine-tuning as well as the evaluation for the models.

5.2 Original BERT

5.2.1 Size of Adapters

Experiment Setup and Motivation

In these experiments, we freeze both the encoder and decoder and modify the reduction ratio parameter in the adapters. The adapter serves as a bottleneck layer with two dense layers and a non-linear function between them. The reduction ratio is defined as the fraction of the original representation dimension divided by the adapter vector size. For instance, if we use 16 as the reduction ratio, we reduce the original layers by 16 with the first dense layer and then scale it back to the original size with the second dense layer.

We try out various sizes of reduction ratios to compare the results. This reduction aims to see whether we can further benefit from enlarging the adapters' bottleneck size. We use 16, 8, 4, 2, and 1 as the ratio values for this experiment. We compare the results with the baseline BERT that we fine-tuned by only training the cross-attention and output layers. We will refer to this baseline as `baseline_bert` for the entirety of this chapter.

Experiment Results

In this section, we compare the results of the `baseline_bert` with BERT models that are fine-tuned with adapters in different reduction ratios. We are fine-tuning the cross-attention and output layers for the `baseline_bert` model and freeze the rest of the model. We can see in Figure 5.1 that even the smallest model (`adapt_bert_reduc_16`) can already outperform the baseline by around 2 BLEU points. This shows that the adapters can help improve the model's performance by adding only a small number of weights during the fine-tuning.

Despite better performance than the baseline model, the difference between the ratios is minimal. It suggests that there is not much benefit in expanding the size of adapters for the normal size BERT. It is possible that it is no longer trivial to simply append larger-size adapters for fine-tuning the model and getting better performance. Further changes may be required to handle the different nature of BERT's output as it is naturally different from the common auto-regressive machine translation objective. Furthermore, we recall that we also have problems where the models struggle in generating good translations when the input contains an unavailable token such as `&quot` .

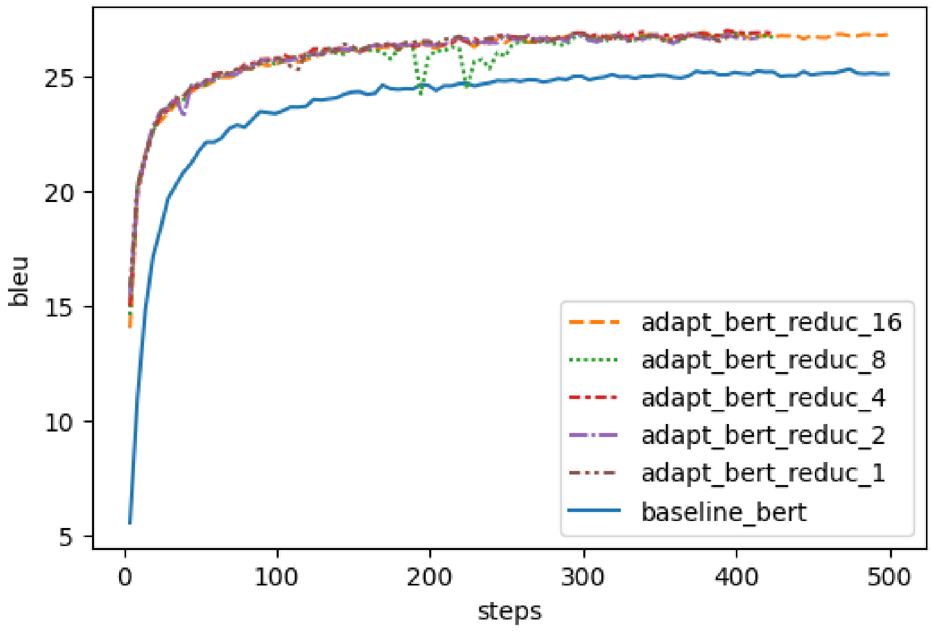

Figure 5.1: Comparison between baseline BERT model and adapters model with different ratio (16, 8, 4, 2, 1).

5.2.2 Position of Adapters (Encoder vs Decoder)

Experiment Setup and Motivation

We would like to see the importance of adapters when they are put in different places. Since we are working with seq2seq architecture in this work, we would like to see whether only incorporating adapters on either the encoder or the decoder can already be beneficial and reduce the number of parameters added to the model.

Experiment Result

We see in Figure 5.2 that adding adapters just in the encoder brings an improvement and outperforms the baseline. Adapters only in the encoder train the fastest at the beginning, and their final performance is almost the same as if we added the adapters on both sides. For the decoder, on the other hand, we can see that aside from a more promising start, there is no benefit as there is no improvement in terms of late BLEU scores compared to the baseline. With this finding, we can

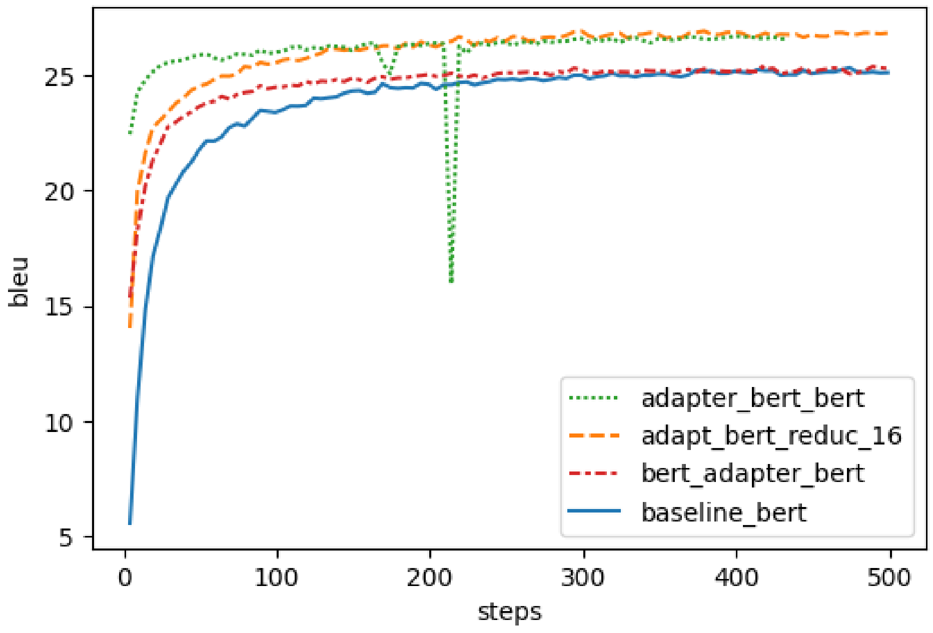

Figure 5.2: Comparison between baseline BERT model and adapters model where the adapters are placed in three different setups: 1) Adapters in both encoder and decoder (`adapt_bert_reduc_16`); 2) Adapters only in encoder (`adapter_bert_bert`); 3) Adapters only in decoder (`bert_adapter_bert`).

reduce the cost of fine-tuning further by half when we do not include the adapters on the decoder.

5.2.3 True BERT in Encoder vs Decoder

Experiment Setup and Motivation

This section investigates the importance of pre-trained BERT in the encoder or the decoder when the adapters are used for fine-tuning the models.

The previous chapter introduced an experiment where we instantiate the base transformer model with only random fixed weights. We are then fine-tuning the base transformer model by only updating the cross-attention, adapters, and output layers. In this setup, we are doing experiments in a similar concept. We use random (fixed) weights instead of the original pre-trained BERT weights in the encoder or the decoder. We thus have a seq2seq model with the random weights encoder followed by the BERT decoder and vice versa. In the fine-tuning stage, we update the adapters in the encoder and decoder, the cross-attention (or cross-attention layer only in our baseline models), and the output layers.

The purposes of the experiments are:

- We want to understand further the importance of the pre-training model when fine-tuning with adapters. By initializing the models with BERT only in one component, we can see whether it is necessary to use BERT on both components when adapters are incorporated.

- We want to understand the capability of adapters when either one of the components does not contain useful information (relative to BERT). We would like to see whether the adapters can recover or even outperform some of the performance that we have already gathered from the previous chapters and sections.

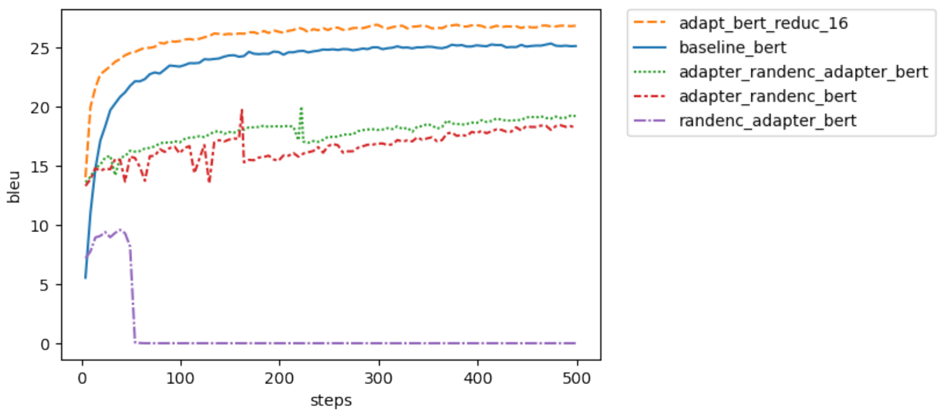

Figure 5.3: Random + BERT: Comparison between baseline BERT model and adapters model where the adapters are placed in three different setups: 1) Adapters in both encoder and decoder (`adapt_bert_reduc_16`); 2) Adapters only in encoder (`adapter_bert_bert`); 3) Adapters only in decoder (`bert_adapter_bert`) and the decoder is initialized with BERT while the encoder is initialized with random numbers.

Experiment Results

This section compares models that use adapters in either or both the encoder and decoder while only initializing one of these components with pre-trained BERT and the other one with (fixed) random weights.

Randomly Set Weights on Encoder In this part of the section, we want to answer the main question: "To what extent can the adapters restore the missing gap when the encoder does not contain useful information (relative to BERT)?"

We can see from Figure 5.3 that when adapters are used in both components to a model with random encoder weights (`adapter_randenc_adapter_bert`), we get to almost 20 BLEU points. This is relatively higher than the other two setups: adapters only in the encoder (`adapter_randenc_bert`) and only in the decoder (`randenc_adapter_bert`). However, compared to the baseline, we are missing 4 BLEU points when we set the encoder with completely random weights. This means that the base encoder model did contain relatively essential information that the adapters can not simply restore during the fine-tuning.

When the adapters are removed from the decoder (`adapter_randenc_bert`), we see a degradation in performance about 1 BLEU point compared to the model that uses adapters on both side (`adapter_randenc_adapter_bert`). However, when the adapters are removed from the encoder (`randenc_adapter_bert`), the performance is completely depleted to zero during the training. We also see the same behaviour in the next section when the weights on the decoder are set randomly. This tells us that it is not trivial to simply fine-tune the cross-attention without further modifying the encoder's parameters when the parameters on the encoder are completely random.

Randomly Set Weights on Decoder Similar to the previous section, the main question in this experiment is, "To what extent can the adapters restore the performance when the decoder does not contain useful information (relative to BERT)?"

In contrast to when the randomly set weights are on the encoder side, we can see from Figure 5.4 that fixing a random decoder leads to performance comparable to the one we have on `bert_baseline`. This tells us that the pre-trained weights in the encoder are more important than in the decoder when we have adapters on both sides. However, when removing the adapters on the encoder, we see similar performance as in the previous section, where the performance drops to zero in the middle of training. This further strengthens our argument that adapters are necessary to adjust the weights in the model so that the cross-attention layer can work properly.

On the other hand, when we remove the adapters from the decoder side, we can see that the performance is not as bad as when the adapters are removed from the encoder, but we still see a reduction in performance. We see a reduction around less than 1 BLEU point when the model reaches 400k steps in the training stage. It is possible that even with random weights on the decoder side, the adapters help the cross-attention layers produce a good vector representation with meaningful features for the decoder to generate reasonable translation outputs.

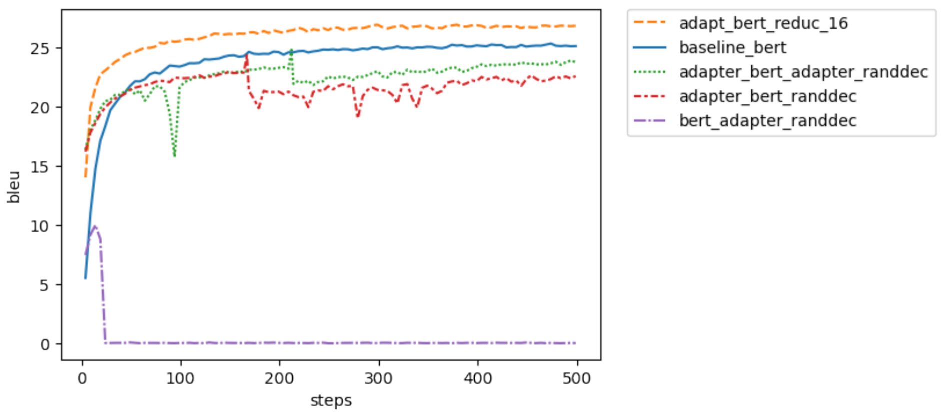

Figure 5.4: BERT + Random: Comparison between baseline BERT model and adapters model where the adapters are placed in three different setups: 1) Adapters in both encoder and decoder (`adapt_bert_reduc_16`); 2) Adapters only in encoder (`adapter_bert_bert`); 3) Adapters only in decoder (`bert_adapter_bert`) and the encoder is initialized with BERT while the decoder is initialized with random numbers.

5.3 BERT Size Reduction

5.3.1 Zeroing Columns

Experiment Setup and Motivation

In this experiment, we will focus on the soft reduction of BERT weights by zeroing the weight matrices on every even column and row indices within the transformer body as well as in the embedding. We load the pre-trained BERT weights, manually edit them and then continue the experiments by fine-tuning the cross-attention, adapters, and output layers. We refer to this setup as `zbert` for the rest of this chapter.

Besides removing the columns, we also perform experiments where we put the adapters either on the encoder or the decoder. This particular experiment aims to understand the model's behaviour when the pre-trained BERT is replaced with this particular setup.

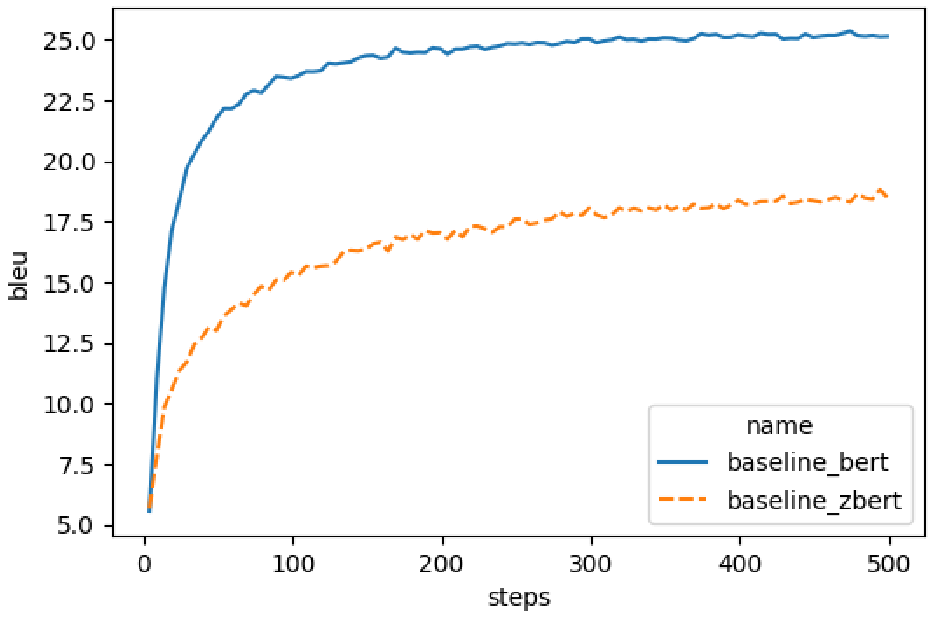

Figure 5.5: Comparison between baseline BERT model and baseline `zbert` models.

Comparison with BERT Baseline (Full BERT Fine-tuning)

We first compare the `zbert` model without adapters and only fine-tune the cross-attention and output layers. We use `zbert` weights on both the encoder and decoder so that it is comparable to the model that uses full-weight BERT. We use the full-weight BERT as the baseline in this experiment.

We can see in Figure 5.5 that we are losing performance of about 4 BLEU points. This is significant as we lose essential features from the original BERT model. To see whether we can recover some of the performance with adapters, we continue our experiment by fine-tuning the `zbert` model that is instantiated on both encoder and decoder sides with adapters. We can see from Figure 5.6 that we only managed to recover 1 BLEU point with a reduction ratio of 16.

From Figure 5.7, when we increase the size of the reduction ratio, initially, we can see some improvement in the BLEU score compared to the higher ratio. However, they eventually converge to a similar performance by the end of training with no significant difference between different ratios. From this result, we can understand that depending on the base pre-trained model, adapters still have a

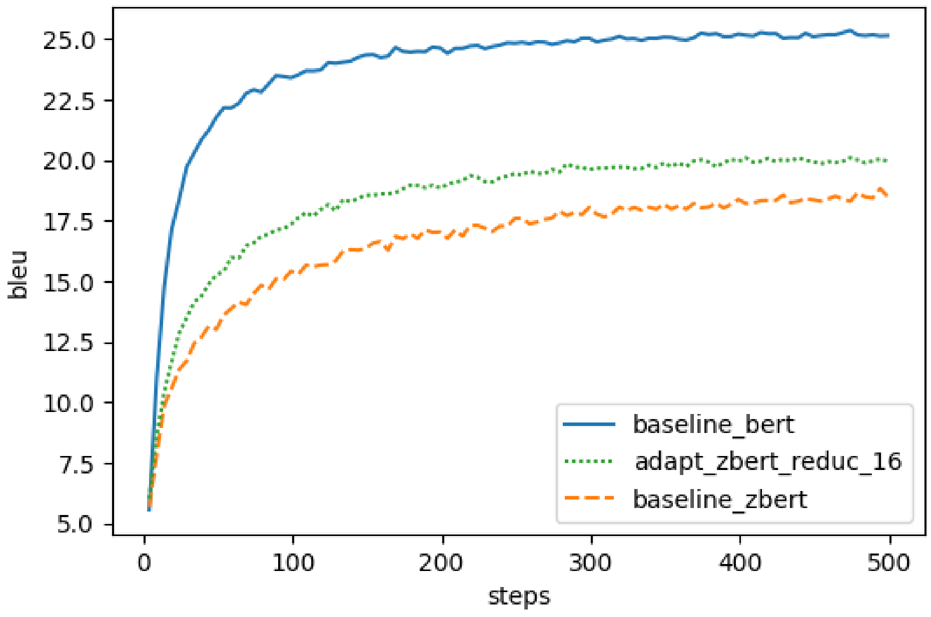

Figure 5.6: Comparison between baseline BERT model, baseline `zbert` and adapters `zbert` models.

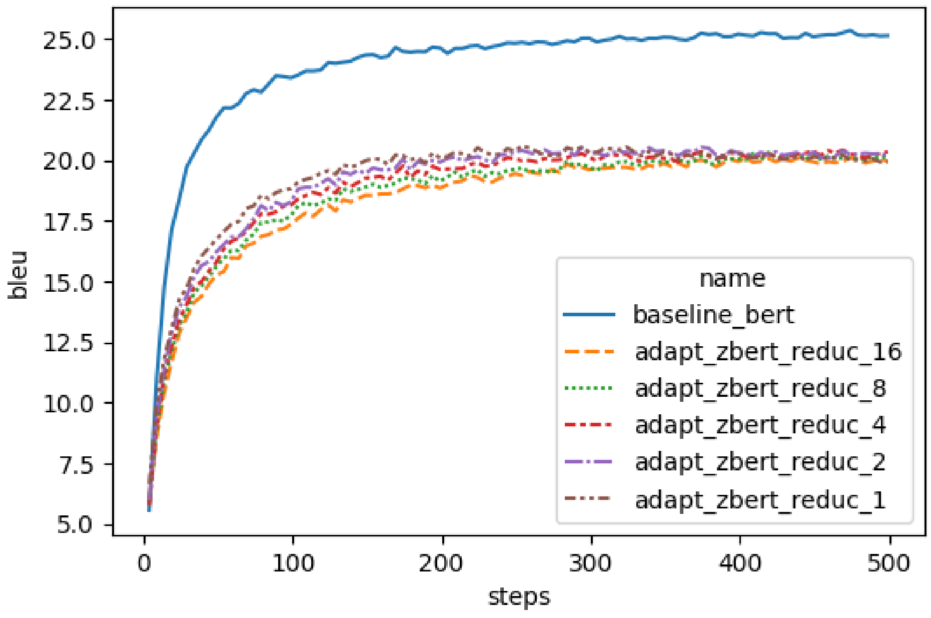

Figure 5.7: Comparison between baseline BERT model and different reduction ratio of `zbert` models.

limitation in achieving certain performance.

Adapters Position In this section, we aim to understand whether the position of both adapters and the pre-trained models affect the model's performance, similar to what we have seen in Section 5.2.2. We use a similar setup as in the previous section, with the exception that we use `zbert` as the pre-trained model instead of the original BERT model.

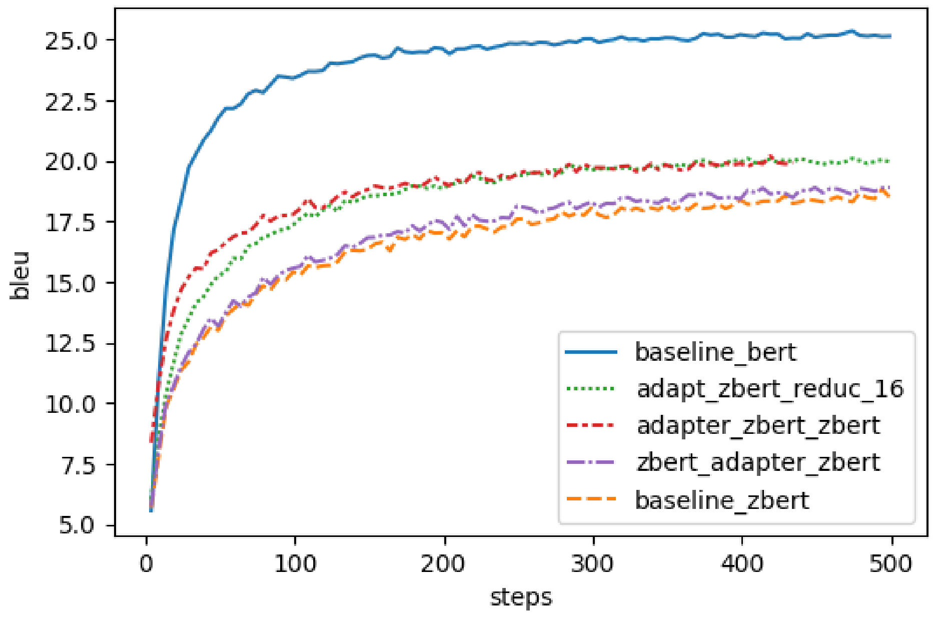

Figure 5.8: Comparison between baseline BERT model, baseline `zbert` model, adapters in both encoder and decoder of `zbert` model (`adapt_zbert_reduc_16`), adapters only in encoder of `zbert` model (`adapter_zbert_zbert`), and adapters only in decoder of `zbert` model (`zbert_adapter_zbert`).

We can see from Figure 5.8 that when we include adapters on both the encoder and decoder, we can outperform the baseline `zbert` in around 2 BLEU points. This shows that, similar to the models that use BERT as the pre-trained model, the adapters can help to improve the performance further, even though some of the information is already missing in the base model.

Furthermore, we can also see that, similar to the BERT model experiments, fine-tuning with adapters only on the encoder side (`adapter_zbert_zbert`) performs much better than on the decoder side (`zbert_adapter_zbert`). Other than that, we can also see that incorporating adapters only on the encoder side helps the model achieve better performance faster than using adapters on both sides. This further supports our hypothesis that updating the representation on the encoder side is more beneficial. Additionally, we can also see the same behaviour as the original pre-trained BERT that `zbert_adapter_zbert` performance is close to the baseline model (`baseline_zbert`), where we are only fine-tuning the cross-attention and output layers. This could mean that fine-tuning the decoder may not be enough to achieve better performance when the representation from the source side is unchanged.

5.3.2 Model Down-Scaling

Experiment Setup and Motivation

This experiment is the follow-up from `zbert`, where we zeroed out half of the elements in the matrices. More specifically, we are completely removing those elements from the matrix instead of just zeroing out the elements. The way we do this is similar to the one we do on `zsbert`. We remove the matrix elements on every even column and row in the transformer body and the embedding. We again do the weights processing offline before using it as the pre-trained model. For the rest of this writing, we refer to this setup as `zsbert`.

Furthermore, we also follow a similar setup as in Section 5.2.2 where we experiment with the position of the adapters. The goal of this experiment is to understand the behaviour of this model compared to the baseline as well as `zbert`.

Comparison with BERT Baseline and `zbert`

We begin by comparing `zsbert` with the BERT baseline. We see in Figure 5.9 that the performance degrades by more than 10 BLEU points. This is also significantly worse than `zbert`, where we only lose 7.5 BLEU points. Our initial hypothesis was

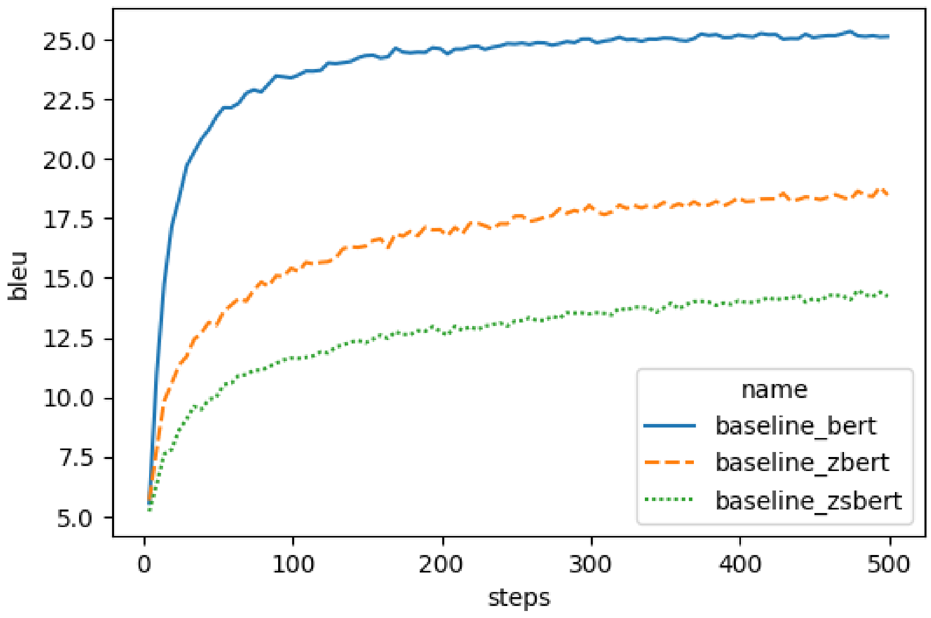

Figure 5.9: Comparison between baseline BERT model and baseline `zsbert` model.

that `zsbert`'s performance should be comparable to `zbert` as we are fundamentally performing the same reduction technique. After some investigation, we realized that removing the weights from the network and zeroing out the matrices are not directly comparable because we also need to consider the computation of layer normalization, which highly depends on the matrix dimension. We found this discrepancy by performing a manual evaluation where we used an arbitrary vector as the input to the network and monitored the output in each of the network layers. We found a slight difference in the layer's output between the zeroed and completely removed weights. Even though the difference is minimal, the output discrepancy gets propagated to the top layers, causing the final network output to differ significantly.

Next, we study the interplay of model down-scaling and adapters. We can see in Figure 5.10 that `zsbert` with a 16 ratio adapters manages to improve the performance up to 6 BLEU points compared to `zbert` without adapters. This shows that adapters can still improve the model's performance even when some weights are missing. Furthermore, despite still showing difficulties in reaching the baseline

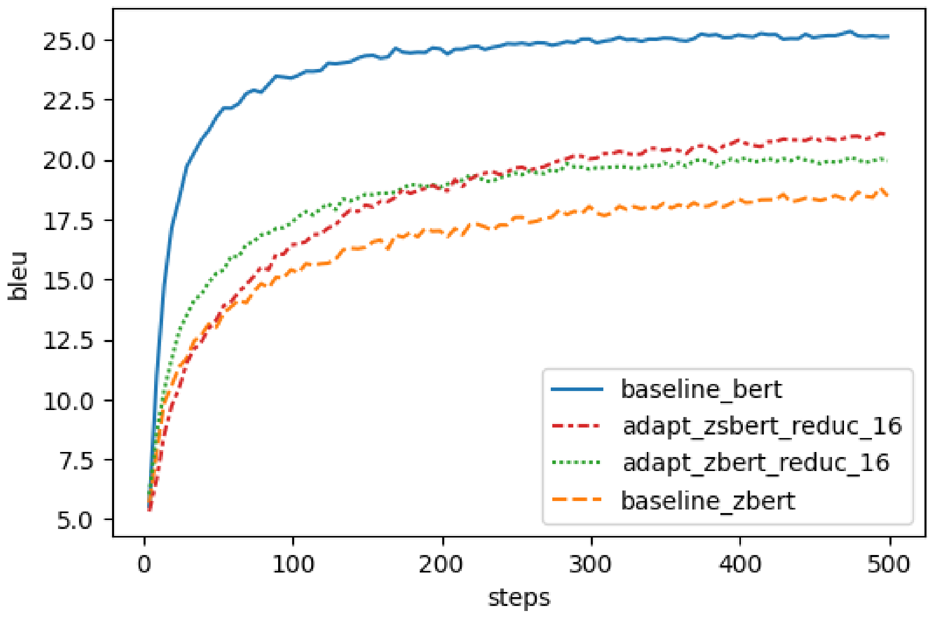

Figure 5.10: Comparison between baseline BERT model, baseline `zbert`, baseline `zsbert` and adapters `zsbert` models.

performance, Figure 5.11 shows that we can still improve the model's performance by reducing the adapters reduction ratio. We can see that the model with the lowest reduction ratio (1) manages to close the performance gap significantly with the baseline model. This is one of our prominent results because we can see from Table 5.1 that the total number of weights (including adapters) required to fine-tune the model is significantly lower than the original BERT model.

Name	# Trainable Variables	# Untrainable Variables	# Total Variables	Percentage Trainable
Ratio 16	7.736.826	95.143.296	102.880.122	7.5%
Ratio 8	8.179.770	95.143.296	103.323.066	7.9%
Ratio 4	9.065.658	95.143.296	104.208.954	8.7%
Ratio 2	10.837.434	95.143.296	105.980.730	10.2%
Ratio 1	14.380.986	95.143.296	109.524.282	13.1%

Table 5.1: Total trainable variables in `zsbert` with adapters on different ratio vs normal BERT model.

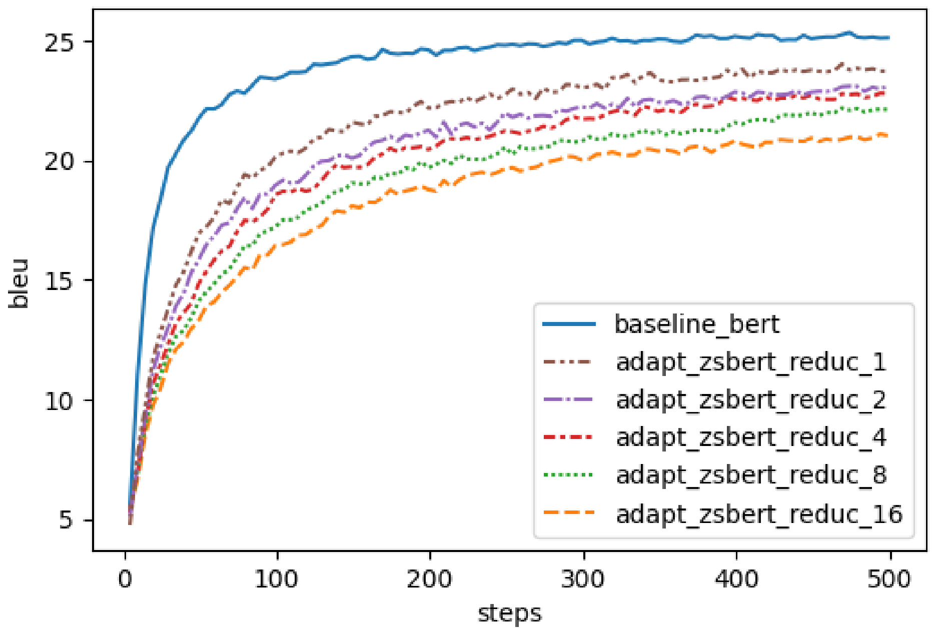

Figure 5.11: Comparison between baseline BERT model and different reduction ratio of `zsbert` models.

Adapters Position

From Figure 5.12, similar to the `zbert` experiments, we can see similar behaviour where models fine-tuned with adapters outperform the baseline `zsbert` and `zbert` models. However, compared to `zbert` experiments, we notice a bigger improvement in `zsbert`'s final performance. In `zbert`, the difference between baseline and adapters is within 5 BLEU points. On the other hand, in `zsbert`, we see the improvement is within 8 BLEU points. This result is particularly interesting for us as we expect the difference to be similar to `zbert`. We recall from 5.3.2 that this is due to the numerical error from the layer normalization, i.e. a difference in constant used to perform vector normalization in the layer normalization. In other words, we observed a worse performance in `zbert` than we got for `zsbert` with adapters.

We deep-dive further in Figure 5.13 to show the comparison between adapters in `zbert` and `zsbert`. We use a reduction ratio of 16 to compare the adapter's performance between these two setups. We notice a leap in the final performance

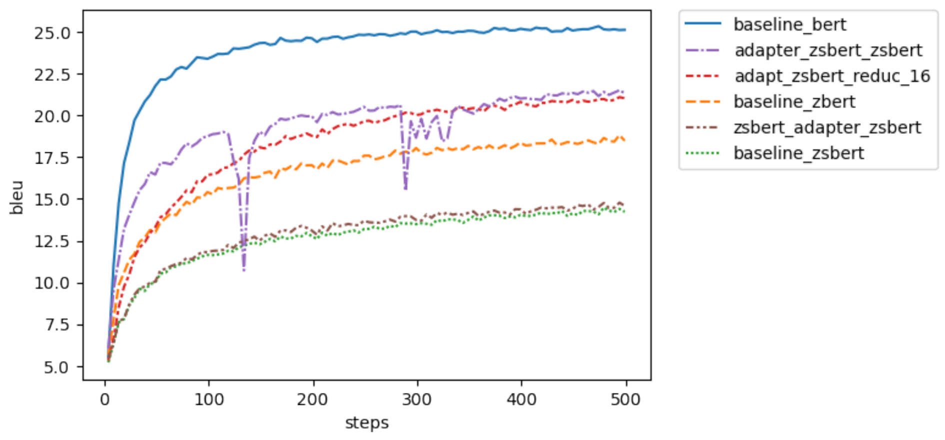

Figure 5.12: Comparison between baseline BERT model, baseline `zsbert` model, adapters in both encoder and decoder of `zsbert` model (`adapt_zsbert_reduc_16`), adapters only in encoder of `zsbert` model (`adapter_zsbert_zsbert`), and adapters only in decoder of `zsbert` model (`zsbert_adapter_zsbert`).

when we compare the adapter model with an equal reduction ratio (16) between `zbert` and `zsbert`. We can see that initially `zsbert` performs worse than `zbert`. After some steps, we can see the performance in `zbert` starting to stall but not in `zsbert`. We hypothesize that this relates to a similar reason we stated in the original BERT model, where we could not see any improvement when increasing the reduction ratio. It is possible that when we reduce the original pre-trained model's size, the adapters adjust the flow of information within the network and better replace the missing information with new knowledge that is more important for solving the task. Another possibility is simply because `zbert` is a "heavier model" than `zsbert` as it contains more parameters and thus has a harder time for the adapters to recover.

We see a similar behaviour as in `zbert` experiments concerning the position of the adapters. In Figure 5.12, the benefit of incorporating adapters on the encoder side is apparent and outperforms the decoder counterpart. We can also see a similar behaviour where the model's performance with adapters on the encoder eventually outperforms the model with adapters on both sides. Furthermore, we also see a similar behaviour as in `zbert` for models with adapters in the decoder only where

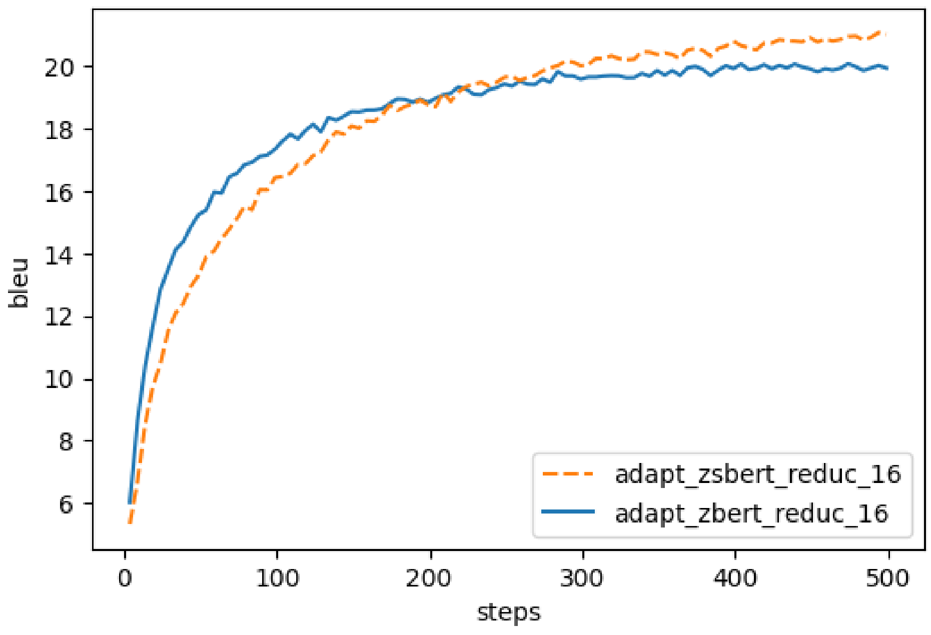

Figure 5.13: Comparison adapters performance in `zsbert` and `zbert`. Both are using reduction ratio 16 and the adapters are placed on encoder and decoder.

the performance is very close to the baseline and not improving as much as on the encoder side. We hypothesize that the same reason as we have stated in `zbert` could apply in `zsbert` as well. Essentially, we need to modify the representation on the encoder side in order to achieve better performance.