# Transformer Modules

## Transferable & Parameter Efficient LLM Fine Tuning

by

Jahson O'Dwyer Wha Binda

To obtain the degree of Master of Science at the Delft University of Technology,to be defended publicly on the 5th of June 2024

| | |
|---|---|
| Student Number: | 4772288 |
| Thesis Advisor: | Pradeep Murukannaiah |
| Daily co-supervisor: | Enrico Liscio |
| Project Duration: | Oct 2023 - Jun 2024 |
| Research Group: | Interactive Intelligence |
| Faculty: | Electrical Engineering, Mathematics and Computer Science |
| Thesis Committee: | Pradeep Murukannaiah |
| | Enrico Liscio |
| | Jan van Gemert |

| | |
|---|---|
| Style: | Association for Computational Linguistics (ACL) conference template |

TUDelft

# Transformer Modules: Transferable & Parameter Efficient LLM Fine Tuning

**Jahson O'Dwyer Binda**
Delft University Of Technology
`j.t.odwyerwhabinda@student.tudelft.nl`

## Abstract

With the increasing popularity of Large Language Models (LLMs), fine-tuning them has become increasingly computationally expensive. Parameter Efficient Fine-Tuning (PEFT) methods like LoRA and Adapters, introduced by Microsoft and Google, respectively, aim to reduce the number of trainable parameters, with the current state-of-the-art combining both methods as LoRA Adapters. This paper introduces Transformer Modules as a PEFT method. These modules utilize Modular Transformer Blocks (MTBs) inserted into a frozen pre-trained model, achieving competitive performance while significantly reducing computation costs. Compared to the current state-of-the-art using GPT-2, BERT, and T5, Transformer Modules further reduced compute time by 39.7% and training memory by 72.7%, with a performance cost of 4.5±2.51% on the GLUE benchmark. Additionally, the paper presents the Transformer Bridge, a continuous vector transformer designed to transfer Transformer Modules across different models. This could enable cross-model fine-tuning, allowing model-agnostic modules, such as an ethics or medical module, to be used across various LLMs without retraining or access to the original dataset. Although the current implementation of the Transformer Bridge did not fully succeed in mapping embedding spaces, analysis of the results suggests that further refinements using traditional model distillation techniques could lead to success in future iterations.

## 1 Introduction

Transformer architectures, first introduced in the paper "Attention Is All You Need" (Vaswani et al., 2017), represent a significant shift in the design of neural networks for processing sequential data, especially in the field of natural language processing (NLP).

With the recent surge in the popularity of Large Language Models (LLMs), increasingly massive Transformer models are being trained and served. Some examples include popular models like OpenAI's GPT-4 (OpenAI, 2023) and the open-source LLM Llama 2 (Touvron et al., 2023) by Meta. Just these two models have 1.7 Trillion and 70 Billion parameters respectively. As these models grow in size, tuning for a specific use case by traditionally adjusting their weights becomes increasingly expensive and potentially destructive to the model's performance.

To solve this problem an area of study has arisen to tune these models using a reduced set of trainable parameters. This is called Parameter Efficient Fine Tuning (PEFT), which has a variety of different methods of tuning a Transformer's output, some popular methods include Prompt Tuning (Lester et al., 2021), Transformer Adapters (Houlsby et al., 2019), and Low-Rank Adaptation (LoRA) (Hu et al., 2022). All of these methods either lack competitive performance, at times still have a significant computational expense and are highly dependent on the model they are originally trained on making effective cross-model transfer learning more difficult.

This paper introduces tuning **Modular Transformer Blocks (MTB)** to create **Transformer Modules** as a form of PEFT. A Transformer can generally be seen as a sequence of Transformer blocks typically these blocks primarily consist of a self-attention mechanism followed by a feed-forward neural network with Normalization layers appended after each level.

A Transformer Module is a series of MTBs, that have been inserted into a pre-trained transformer, to minimise the dependency on the original model and facilitate transfer learning they are typically placed at the head or tail ends of the model but can generally be placed anywhere. During training the original model is frozen and only the weights of the added blocks are adjusted. This allows for fine-tuning of a transformer in a way that not only reduces the number of trainable parameters but also utilises the showcased strength of the self-attention mechanism demonstrated in the original "Attention Is All You Need" paper(Vaswani et al., 2017).

The intuition behind a Transformer Module is that the internal blocks learn to guide and provide additional domain knowledge to the already trained model, by learning how to use and alter the existing embedding space created by the model's internal weights.

As mentioned before, an area where other PEFT methods fall short is that the added learnings made during training are either dependent on the embedding space created by the original model, or alter the embedding space of the original model. This lack of modularity makes the complete transfer of tuned behaviour impossible without additional fine-tuning. The process of transferring pre-trained weights followed by further fine-tuning is often referred to as *transfer learning*. The Transformer Module has been created to not only serve as a form of PEFT but also as a form of experimenting with the idea of "Modular Learning". This allows Modules to be created for a task, an example being added "domain" knowledge like a medical module, which can be trained once and used across many different models without further training.

In this paper, I also introduce the **Transformer Bridge** which is an Encoder-Decoder style continuous vector Transformer trained to map (or bridge) the embedding space of one model to another model. It does this by encoding the embedding space of one model to a shared latent representation. From this, the decoder can create an equivalent embedding from the second model's embedding space. When used together with MTBs Modules can be created that can be used in models other than the one the MTBs were originally trained on. Transformer Bridges only require to be trained once, meaning if a Bridge already exists a Transformer Module can be used on another model without any additional training. This comes with many advantages like saving on compute as well for tuned behaviours when a dataset is not publicly available.

## 2 Related Works

Given the theme of this paper related research falls into two categories. The first is research into PEFT and the second being research into transfer-learning and embedding space manipulation.

### 2.1 Parameter Efficient Fine-tuning (PEFT)

Regarding PEFT methods for transformer models, several new approaches have risen as the popularity of large transformers and LLMs increased. Notably introduced by (Hu et al., 2022), *Low-Rank Adaption* (LoRA) was a significant advancement in PEFT and is commonly used today to tune large LLMs efficiently. This method employs the idea of decomposing the weight matrices of the model into a low-rank version with a smaller dimension. The low-rank matrices can be seen to represent the changes that are required in the original weight matrices which are later applied at the end of the training session to the model weights.

Another popular method was introduced by a team of primarily Google researchers (Houlsby et al., 2019), in their paper they showcase the concept of *Transformer Adapters*. Adapters are a small set of linear layers inserted into each block of a frozen Transformer model. During training only the adapter weights are updated, this not only reduced the number of parameters required to fine-tune a model but also was non-destructive to the original model weights allowing for the selected use of the adapter at inference time. Adapters have been found to allow tuning of large models whilst achieving similar performance to the baseline, a full model tune. After the introduction of LoRA, it has been found that it can be applied to only the Adapter weights further reducing the trainable parameters, LoRA Adapters are currently the state-of-the-art PEFT method. Achieving a large parameter reduction whilst maintaining competitive performance.

There have been many variants of using LoRA Adapters for fine-tuning LLMs. The paper *LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models* presents a versatile framework that integrates various adapter types, including Series and Parallel adapters, with LoRA, achieving high performance with minimal parameter adjustments across multiple datasets (Hu et al., 2023). Similarly, *Mini-Ensemble Low-Rank Adapters for Parameter-Efficient Fine-Tuning* (MELoRA) introduces an ensemble of low-rank adapters that maintain efficiency and enhance performance by training a group of mini low-rank adaptations, significantly reducing the number of trainable parameters needed for tasks like natural language understanding and instruction following (Ren et al., 2024). Finally, *Compacter: Efficient Low-Rank Hypercomplex Adapter Layers* reduces computational complexity by combining low-rank matrices and hypercomplex algebra, inserting task-specific weight matrices computed as Kronecker products, and fine-tuning only a tiny fraction of the model's parameters while maintaining or surpassing full model fine-tuning performance on benchmarks like GLUE and SuperGLUE (Karimi Mahabadi et al., 2021).

Generally for tuning methods requiring performance similar to a full-model tune a variant of LoRA and Adapters are used in combination this sense is shared in a recent survey conducted by (Xu et al., 2023), in their paper they compare multiple PEFT methods with a large subset of them being derivatives of the LoRA Adapters solidifying it as the current state-of-the-art.

New PEFT methods like *Prompt Tuning* introduced by (Lester et al., 2021) have gained popularity. Prompt tuning introduces a set of tuneable vectors to the frozen transformer's input, making it one of the most efficient and unobtrusive PEFT methods in terms of number of parameters and weight adjustments. However, while effective for simpler tasks, its efficacy in handling more complex tasks appears to be less robust compared to other methods, suggesting a potential limitation in its applicability. This makes intuitive sense as there is only a certain amount of information that can be captured in a few added vectors to the input.

### 2.2 Embedding-Space Manipulation

Embedding-Space Manipulation is a much larger field of research, with the goal of "Modular Learning", research relating to the transfer of pre-trained learning and embedding-space mapping is the primary focus. Modular Learning in the context of this paper is pre-trained learnt behaviours that are isolated in a region of the model and can be transferred to another model. Traditionally this is similar to transfer learning with the additional requirement that the model should not require additional training, this can only be achieved if the pre-trained module successfully manipulates the embedding space of the foreign model to achieve the tuned behaviour as its final output.

As noted in a recent survey on Modular *Deep* Learning (Pfeiffer et al., 2023), the research topic has been gaining traction. *Modular Deep Learning* can be seen as computations which are implemented as autonomous parameter-efficient modules similar to Adapters. The area of cross-model modules is a space that has been relatively under-researched. For such a task a simulation of a module's input embedding space would need to be created from an embedding space created by an unknown model. Embedding space simulation by manipulating the embedding space of an unknown foreign model may prove to be a difficult task.

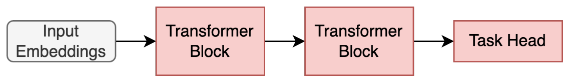
The paper "*Cross-Modal Variational Alignment of Latent Spaces*" (Theodoridis et al., 2020) demonstrates that Variational Auto-Encoders (VAEs) can map one embedding space to another by training separate VAEs for each modality and then aligning their latent spaces. By doing this, the model effectively captures the relationships and dependencies across different modalities, demonstrating the VAE's capability to map and align disparate embedding spaces. If a model can be created to do the same using sequential data, this could unlock the possibility for the creation of cross-model LLM modules.

Model Distillation introduced by (Hinton et al., 2015) shows promise in embedding-space manipulation by showing that an ensemble of models can be compressed into a single model. This suggests that the embedding space created by the compressed model successfully replicates the behaviours created by the embedding space of multiple different models. The idea of using Model Distillation with Transformer models has been used as a form of model compression as proposed by (Lu et al., 2022), in their paper the final layers of BERT had been compressed successfully, this was done by using an architecturally similar student model. There have not been many attempts for cross-model embedding mappings using model distillation techniques especially when they are potentially architecturally different, which is what we aim to do in this paper.

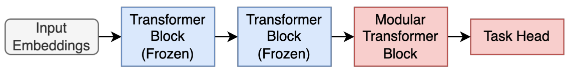
## 3 Methodology

### 3.1 Modular Transformer Blocks

Modular Transformer Blocks (MTBs) are a neural network similar to the ones introduced in the *"Attention Is All You Need*(Vaswani et al., 2017). This is a transformer block, which typically consists of a self-attention mechanism followed by a feed-forward neural network with normalization layers appended after each level. The hyper-parameters regarding the number of layers and trainable parameters for each MTB can be tuned for the specific



(a) Traditional Transformer tune.


Input Embeddings
Transformer Block (Frozen)
Transformer Block (Frozen)
Modular Transformer Block
Task Head

(b) MTB adapted Transformer tune.

Figure 1: Transformer tune comparison. (Red indicated trainable parameters)

task, as well as the number of individual MTBs required. Just like regular transformers the more transformer blocks that are used as MTBs, it is expected to result in an increase in performance for more difficult tasks.

MTBs are inserted into a frozen model where only the weights of the MTBs and possibly the task head can be changed during training. A simplistic comparison of an MTB-based tune with a traditional transformer fine-tune can be seen in Figure 1

The intuition of an MTB is that for many tasks introducing a small number of new trainable parameters to an already trained large Transformer model is more efficient at train time than mutating the original model weights, in terms of the number of trainable parameters. This follows a similar logic to Adapters introduced by the team at Google Deepmind (Houlsby et al., 2019).

This allows the original model to avoid mutation and maintain its performance on its original training tasks as the pre-trained weights can be selectively added during inference. Fine-tuning a transformer using MTBs is as simple as injecting new transformer blocks (MTBs) into the frozen model transformer block sequence.

During training, only the weights of the MTBs are adjusted. After training all learning is contained in the MTBs. A practical example is that given a large medical dataset, a medical module can be trained by inserting MTBs into a frozen LLM. Instead of training the full model, we update the weights contained only in the MTBs. This leads to the learnt behaviour for this task being contained completely in the weights of the MTBs.

#### 3.1.1 Differences when compared with Adapters

Especially when compared with Adapters MTBs essentially can be used in the same way. The key difference of an MTB is that an MTB contains attention layers. Self-attention is one of the main driving forces behind a LLM's performance. And for complex tasks such as ethics modelling, language inference etc. It may prove helpful as it allows the model to learn new ways to attend to the input sequence to make its predictions. This (potentially) comes with the disadvantage of an increased number of trainable parameters, this can be offset because Adapters are typically inserted in each block while the number of MTBs are inserted as required for the given task. One MTB could be equivalent to several Adapters. Other PEFT methods like LoRA can further reduce the number of trainable parameters by applying it only to the MTBs.

Another key difference is their placement in the model, as mentioned before Adapters are typically inserted at each Transformer block, making Adapters dependent on the (mutated) embedding space created by the (mutated) model beneath it. This has 2 disadvantages, the first being that it does not facilitate the opportunity for transferable "Modular Learning" in the same way MTBs do. MTBs are meant to be used together sequentially to create a Module, the module has a single dependence, the embedding space used to sample its inputs during training. Due to its less "entangled" nature with the original model, an MTB sequence in theory should only require a mapping of a single embedding space to achieve inter-model operability. The second disadvantage is that the Adapter's entangled nature requires that gradients and intermediate outputs for large portions of the model must be calculated and stored during training, even for the frozen parameters that are not being updated, when compared to the contiguous nature of the Transformer Modules the use of MTBs is expected to be significantly more efficient in both memory and training time.

### 3.2 Transformer Bridge

When a Transformer model has been trained it creates an *embedding space* at the output of each of its Transformer blocks. Typically the embedding space created by the final block of a model is used for tasks such as contextual embeddings or to train additional linear layers to create task-specific heads.

As seen in Figure 1b, MTBs sample inputs from the embedding space created by the downstream model to use for a prediction. This becomes a problem when you transfer an MTB from one model to another as the models can have completely different embedding spaces.

In more simplistic terms, we can not directly take an MTB trained using GPT-4 embeddings and insert it into another model like Llama 2 and expect good results, this is because the embedding space created at the output of the two models is completely different. To address this problem, given a sequence of embeddings from Llama 2 we need to generate a sequence of equivalent GPT-4 embeddings. From this generated sequence, a GPT-4 based MTB could theoretically use LLama 2 embeddings to make its prediction given that the mapping is accurate.

Inspired by Variational Auto-Encoders, which have been shown to map vectors from one embedding space to another, by creating an Encoder-Decoder architecture with a shared latent space. We can create a model to do the same, with the key difference being that it should be able to map a *sequence* of inter-dependant vector embedding from one space to another. Such a model will be called a Transformer Bridge as it has the goal of "bridging" the embedding space of two different transformer models.

Using *translated* embeddings from a Transformer Bridge, any MTB should be able to be used in a model outside of the one it was trained out without further training, given that a Bridge exists that can accurately create equivalent embeddings.

#### 3.2.1 Transformer Bridge Architecture

The Transformer Bridge is built using the transformer architecture, specifically the Encoder-Decoder style transformer to act similarly to a Variational Auto-Encoder but with the ability to work for sequential data. In this paper's implementation of the Bridge, the popular T5 Model by Google Deepmind (Roberts et al., 2019) will serve as the base. T5 was chosen due to its proven history of being a model that generalises well for multiple tasks, an example being that it is one of the top 10 models on the GLUE benchmark, as well as being an Encoder-Decoder style transformer. The modified architecture can be seen in Figure 2.

T5 like all other transformer models use tokens for their predictions. Transformer models have a finite number of tokens, in the case of T5 its vocabulary is around 32,000 tokens. In the final layers of a traditional transformer, probabilities are generated for every token in the vocabulary where the highest probability token is predicted as the next token in the sequence.

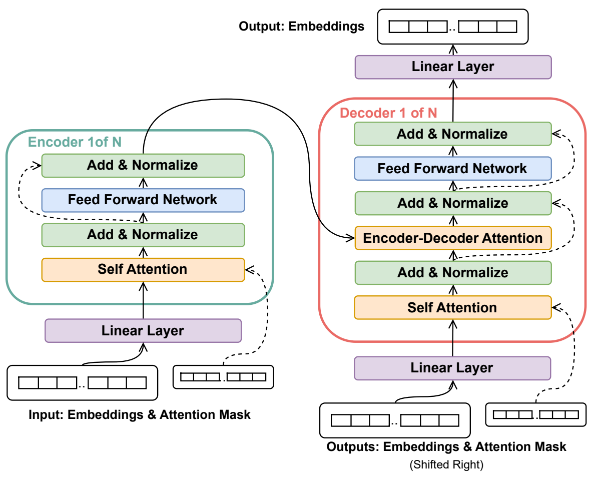


Figure 2: The Transformer Bridge architecture based on T5. Token and positional embedding layers have been removed. Linear *adaptive* layers convert input embedding dimensions to match the Transformer bridge for both Encoder and Decoder. A final adaptive layer converts embeddings back to the target dimensions at the Decoder's output.

This does not align well with the task of embedding translations, as the final output is an N-dimension vector, the set of possible outputs is essentially infinite. The lack of a finite set of possible outputs means that a traditional transformer can not create probabilities for what embedding should be used at a given time step in the sequence.

Because of this the T5 model used to build the Bridge has been modified, starting with removing all layers relating to token and positional embeddings. This leaves only the individual Encoder & Decoder Blocks of the model. The base T5 model's Blocks use a sequence of 768 dimension vectors as its input and output, to allow for the bridge to map embedding spaces of dimensions other than 768, linear layers have been added as inputs to both the Encoder and Decoder of the Bridge, as well as the output of the Decoder. These linear layers can be seen as an *adaptive layer*, where the main purpose is to convert the N-dimension input embeddings to 768 dimensions for the Bridge to use and output to an M-dimension embedding space.

The main challenge for the Bridge is due to the lack of tokens in the architecture two major aspects of a transformer are lost. The first is the lack of an end-of-the-sequence (EOS) token, auto-regressive models use the EOS token to indicate that it has completed the prediction of the whole sequence. This token is part of the model's vocabulary set, which allows it to be predicted. The second is the lack of a padding (PAD) token, the PAD token is used to maintain the shape of batched inputs. This allows transformers to ingest large amounts of data and process them in parallel efficiently on the GPU.

To address the lack of a PAD token, this paper's implementation of the Transformer Bridge uses the tokenizer of the target model to guide the embedding generation. When translating embeddings from Model *A* to Model *B*. The input string that was originally passed to Tokenizer *A* at generation time will also pass through Tokenizer *B*. Tokenizer *B* is used to create the output attention mask for the Bridge, the Bridge uses the mask as an indicator of the embedding sequence length for sentences in the current batch.

In other words, given an attention mask of shape: $[B, N]$, where $B$ is batch size and $N$ is sequence length for a model with an embedding dimension $D$, the Bridge now knows that it should generate an output tensor of shape: $[B, N, D]$ to be used with the provided attention mask.

The next challenge is how the model handles the lack of an EOS token. Traditionally transformers have a *forward* function, which is used to generate the next token for a sequence, and a *generate* function which repeatedly calls the forward function until an EOS token is predicted indicating the end of the sequence. The Bridge's *generate* function instead looks for an EOS condition, this condition is that the sequence should match the length of the target attention mask provided by the foreign tokenizer, i.e the tokenizer used by the model which is using the Bridges output.

#### 3.2.2 Training The Bridge

The Transformer Bridge is expected to perform best when creating mappings from 2 models that have ingested data from similar domains. In the case of LLMs and NLP this would be a contextual understanding of the meaning behind an input sentence which is generally acquired by ingesting large text corpora of scraped internet data.

To train a Bridge from Model A to B a corpus of inputs for both models must be created. For every item in the corpus, each model should be used to get their corresponding embedding and attention mask. The embeddings along with their masks are used to train the bridge.

#### 3.2.3 Alternative Bridge Designs

Multiple configurations and training setups were experimented with to create a Transformer Bridge.

**A dual-output Transformer Bridge** was experimented with to predict both the embedding and the attention at the current step of the sequence. One version used a stacked task-head design and another a forked task-head design, the design of these versions of the Bridge can be seen in Appendix C Figure 12.

A dual-output bridge would mean we would no longer need to use the output of a foreign tokenizer as an EOS condition. Instead, an EOS condition can be inferred from the attention output of the bridge, such as an attention of 0 being predicted after a 1. The Bridge's *generate* function in this scenario looks for an EOS condition. During inference, the generate function considers the sequence complete when the model has predicted a PAD vector following a relevant embedding. In other words, at every time-step *t*, both an embedding and an attention probability are generated by the bridge, if an embedding with an attention probability of 0 is predicted following a 1 at step *t-1* this can be assumed to be a PAD vector indicating that the completed sequence has been predicted. If required the Bridge will continuously predict PAD vectors to maintain the shape of the current batch to allow for efficient processing for the upstream-stream model.

During experimentation, the attention prediction was shown to be much less accurate than the embedding prediction. Because of this, the attention prediction aspect was removed in favour of more focused experimentation on embedding generation.

### 3.3 Transformer Module

A **Transformer Module** is a set of pre-trained MTBs, the set can be added to a model to achieve the fine-tuned behaviour during inference.

There are two types of modules.

1. **Native Module:** Modules that are used within the original model they have been trained on.

2. **Foreign Module:** Modules that are used within a model that the MTBs were not originally trained on.

As seen in Figure 3, the main differentiating aspect between a Native and a Foreign Module is that a Foreign Module contains a pre-trained Transformer Bridge to translate the foreign input embeddings into the equivalent embeddings from the embedding space the MTBs were originally trained on.

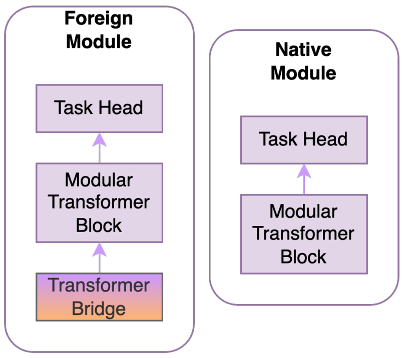


Figure 3: Transformer Module Types

The Transformer Bridge is trained independently of the MTBs, so if a Bridge already exists then the learnt behaviour of a Native Module can be transferred as a Foreign Module without any additional data or training. This represents, to the best of our knowledge, the first example of transferable fine-tuning which requires no additional training which I will refer to as *"Modular Learning"* in this paper.

Returning to the practical example made in Section 3.1. A medical module could be trained using a model like GPT-4, a large dataset and MTBs. This would create a GPT-4 (OpenAI, 2023) Native Module for the medical data. The learnt behaviour from the medical data can be used in other models say LLama 2 (Touvron et al., 2023) by creating a Foreign Module using a GPT-4 to LLama 2 Transformer Bridge. This would allow us to create a fine-tuned medical LLama 2 model with no additional training and without access to the original dataset. Such a scenario saves both in infrastructure costs but also allows the creation of models in scenarios where a dataset is not available.

## 4 Experiment Design

In this section, we introduce the main experiments of the paper. We begin by introducing the experimental bench-marking tasks and models (Sections 4.1 and 4.2), then we describe the evaluation of the MTB concept (Section 4.3), the creation, training and evaluation of the Transformer Bridge (Section 4.4) and finally the evaluation of the Transformer Module concept putting it all together (Section 4.5

### 4.1 Experimental Tasks

Following the example set from the paper introducing Transformer Adapters (Houlsby et al., 2019), the General Language Understanding Evaluation or GLUE Benchmark introduced by (Wang et al., 2019) can be used for our experimentation.

The GLUE benchmark is a collection of diverse natural language understanding tasks designed to evaluate and compare the performance of machine learning models on tasks like sentiment analysis, question answering, and textual entailment. It provides a standardized and comprehensive test suite to assess a model's ability to understand context and reason. For each Glue task, 3 models have been trained using 5 different methods, these methods are visualized in Figure 4 and listed below.

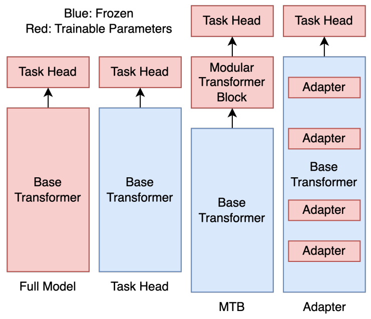


Figure 4: Fine-Tuning Setup

1. A Full-Model and task head fine-tune.
2. A Task-Head only fine-tune, where all parameters except for the task head are frozen.
3. A MTB fine-tune, where the last Transformer Block of the model is copied and added to a frozen model paired with a task head.
4. A LoRA MTB fine-tune, similar to the MTB setup except the trainable parameters are further reduced using LoRA.
5. An Adapter fine-tune, where LoRA adapters are inserted to the frozen model along with a task head.

This set of fine-tuning methods allows for not only a comparison with a baseline, being the full model tune, but also with the current state-of-the-art PEFT method, LoRA Adapters.

Evaluation of the models in all experiments will be task-dependent. For the GLUE Benchmarks tasks the recommended standard metrics will be used. These depend on the task and are either; Test Set Accuracy, Matthews Correlation, F1 Score or Pearson-Spearman Correlation.

For details on the hardware and software used for all experiments in this paper, see Appendix D.

### 4.2 Experimental Models

Due to the theme of this paper being about efficient and transferable fine-tuning, three models are used during the experimentation. The models chosen were the Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019), Generative Pre-trained Transformer 2 (GPT-2) (Radford et al., 2019) and the Text-to-Text Transfer Transformer (T5) (Roberts et al., 2019).

These three models were chosen because they are large, open source and well-researched. This set additionally has the added complexity that each model is an architecturally different style of the transformer model. Where BERT is an Encoder-only model, GPT-2 is Decoder-only and T5 is a standard Encoder-Decoder model.

While all three models are transformers the nature of how they are trained and generate answers are very different each with their strengths and weaknesses, making bridging the embedding spaces between them an interesting experiment. One such strength is that while GPT-2 is good at language generation tasks, BERT is known to perform better on GLUE and other classification tasks.

For each model 3 sets of experiments have been conducted to verify different aspects of the Transformer Module concept, validating MTBs as a PEFT method, validating the Transformer Bridge's ability to translate embeddings for every model pair and validating Foreign MTBs as transferable and modular fine-tuning.

### 4.3 Modular Transformer Blocks

The primary goal of MTB experiments is to validate that MTBs are a viable PEFT method. To this end, we are evaluating if MTB-based fine-tuning achieved competitive performance whilst reducing trainable parameters significantly, In this experiment, all three models are fine-tuned and evaluated on each task using different training methods as seen in Figure 4.

GLUE tasks are either classification or regression tasks, because of this, the frozen pre-trained model weights will be downloaded using the Huggingface library (Wolf et al., 2020). Due to memory constraints, the smallest version of the 3 models will be used, for all models, the small version consists of a total of 12 transformer blocks.

Following the example made by Google's Adapter paper (Houlsby et al., 2019) a hyper-parameter search has been conducted for each task and each tuning method. A grid-search was performed for the following parameters; learning rate $\{3 \cdot 10^{-5}, 3 \cdot 10^{-4}, 3 \cdot 10^{-3}\}$ and number of epochs $\{3, 20\}$. The highest performing hyper-parameters for each tuning method on the validation set were used to conduct the experiment.

All training sessions used a batch size of 32 and an AdamW (Loshchilov and Hutter, 2019) optimizer paired with a linear scheduler with 0 warm-up steps.

### 4.4 Transformer Bridge

The experiments for the Transformer Bridge have the main goal of validating the successful translation of vectors from one embedding space to another. To this end, we are evaluating if a trained Bridge can successfully be used with another model's task head. This experiment is a two-step process, first the creation and training of the Transformer Bridge and secondly, the use of the Bridge to achieve embedding mappings.

#### 4.4.1 Creating the Bridge

To train the Transformer Bridge a sentence corpus has been created. This corpus is made from 50% of the input sentences for all GLUE Tasks. This is so that during training the Bridge has some representation of the domain it is expected to build a mapping to. This corpus has a size of 1.4 Million strings. Each row of the corpus has been passed through all 3 models to build a dataset of their respective model embeddings and attention masks, this is used as the Input and Target during training. The reason for using GLUE sentences instead of a general sentence corpus is to allow the Bridge to focus on mapping embeddings for a subset of sentences structured and relating to the downstream task, this is so the bridge does not need to learn to translate sentences structured in an unrelated way. If successful a more generalized Bridge can be experimented on with a larger and more varied dataset.

Multiple Bridges were created using the architecture defined in the Methodology, since the theme of the paper is Parameter efficiency the goal is to make the Bridge as small as possible so we limited the number of Transformer Blocks to a maximum of 4. This allows the creation of 4 different Bridges with the shapes; *1x1, 1x2, 2x1, 2x2*, where *ExD* represent the number of Encoder blocks used (*E*) and Decoder blocks used (*D*). A total of 24 Bridges were trained to create bridges for all input and output combinations of our 3 models.

To leverage an opportunity for transfer learning, the Transformer Bridge used the last N-blocks of the pre-trained T5 base model. This is called a warm-start and is popularly used by Google to allow models to converge faster during training. As showcased in the paper *Leveraging Pre-trained Checkpoints for Sequence Generation Tasks* (Rothe et al., 2020)

### 4.4.2 Bridge Training and Parameters

As the implementation of the Bridge is based on T5 (Roberts et al., 2019), in this paper the same training loop and hyper-parameters are used to fine-tune the T5 model for the task of embedding generation.

During training a batch size of 128 was used with a constant learning rate of 0.001. The T5 paper denoted the duration of training using "training steps", a training step accounts for a single back-propagation. The paper had a fine-tuning phase consisting of $2^{18}$ steps. After every 5000 steps, a checkpoint of the model was saved, like the T5 paper, the checkpoint with the highest score on the validation set was selected for the experiment.

We also experimented with using variable learning rate using a *"inverse square root"* scheduler from the original T5 paper, denoted as $1/\sqrt{\max(n, k)}$, where $n$ is the current iteration and $k$ is the number of warm-up steps set as 10,000. The inverse square root scheduler had training curves that were worse than the constant learning rate after about 15,000 steps resulting in an early termination of the training session.

To train the Bridge teacher-forcing (Lamb et al., 2016) was used. At every training step the full input sequence is passed to the encoder and the full target sequence is passed to the decoder. While traditionally Cross-Entropy loss is used to train transformer models due to the lack of a vocabulary this paper's implementation of the Bridge optimizes to minimize the Mean Squared Difference (MSE) between the predicted and target vector. MSE is measured only for the embeddings in the sequence that are being attended to according to the attention mask and averaged.

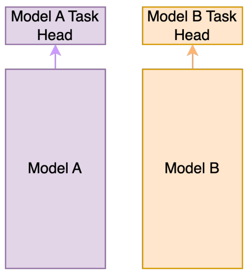
The MSE values between pairs of vectors in distinct embedding spaces, such as those of GPT2 and T5, can reach the tens of thousands. To prevent the exploding gradient issue during back-propagation, a logarithmically scaled MSE is employed. This log-scaled MSE is expressed as:

$$\log_{\text{mse}} = \log(1 + \text{MSE})$$

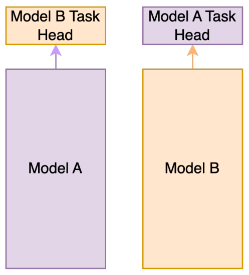
Cosine Loss training was also experimented with, initially showing good results as the Bridge successfully mapped embedding from two different models with an error as low as *0.0086*. The results and training graphs of which can be seen in Appendix C.In practice, this version of the Bridge did not perform well on the downstream task. This is partially because Cosine-Loss optimizes for the direction of vectors in space and does not account for the magnitude of those vectors. Embedding magnitude is expected to be an important feature in LLM spaces.

### 4.4.3 Bridge Experimentation

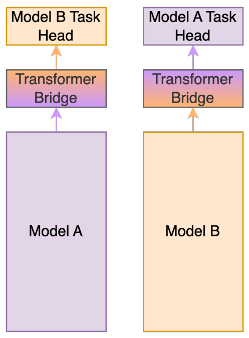
After completing the MTB experiments detailed in Section 4.3, we have tuned models for all GLUE tasks. Particularly, for every task we have both a task head model where only the task head was updated during training and a *Native Module* model where a Transformer Module was created for the tasks and updated. These models are labelled as "Task Head" and "Module" in Figure 4.



(a) Pre-trained models using original task-heads



(b) Pre-trained models using foreign task-head



(c) Pre-trained models using foreign task-head and Bridge

Figure 5: Transformer Bridge Experimental Setup

This allowed us to conduct the Transformer Bridge experiment with no further training. To do this a Bridge can be used to swap the task head for every pair of models as seen in Figure 5.

For every GLUE task we built a set of models, these models include:

1. An unmodified model to serve as the baseline.
2. A model using a foreign task head and no bridge.
3. A model using a foreign task head after a Bridge.

The models without any further training were evaluated on the same test set as the unmodified models.

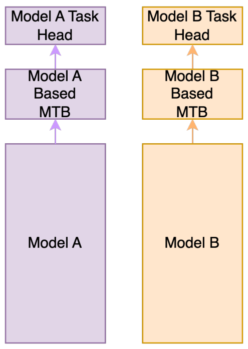
The bridge can be deemed successful if the model using a Bridge and a foreign task head can achieve similar performance to the same model using its native task head. It should also be significantly better than the models without the bridge using a foreign task head. If true, this has shown that the bridge has successfully bridged the base model embeddings to the foreign embedding space as a pre-trained task head expects samples from its native embedding space to create predictions.

### 4.5 Transformer Modules

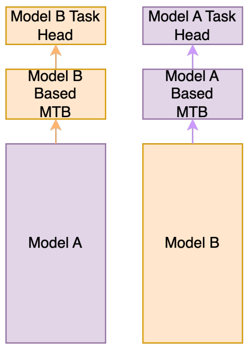
The Transformer Modules experiment is similar to the Transformer Bridge experiment in goals and execution. In this experiment, we aim to evaluate that pre-trained *Foreign Modules* can be created and used to add a tuned behaviour from one model to another. The success of *Native Modules* is already being explored by the PEFT experiments in Section 4.3, so in this set the focus is primarily on *Foreign Modules*.

Similar to the Transformer Bridge experiments we can leverage the previous pre-trained modules from our PEFT experiments. This allows us to validate the foreign module concept without further training. The difference is instead of using just a pre-trained task head we will use a pre-trained MTB and task head to create *Modules*. The Foreign Modules will include a bridge to translate the input embedding space as seen in Figure 6.

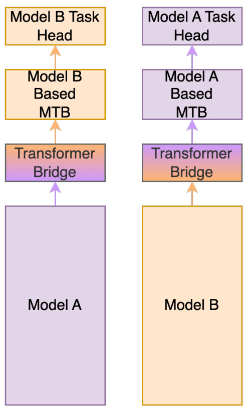
This experiment follows the same success criteria as the task-head-only experiment. The Transformer Module as a form of modular and transferable learning can be deemed successful if, for every task, the model using a Foreign Transformer



(a) Pre-trained models with native Modules



(b) Pre-trained models using foreign Modules



(c) Pre-trained models using foreign Modules and a Bridge

Figure 6: Transferable Transformer Modules Experimental Setup

Module achieves similar performance to the performance of the same Module in its native model.

## 5 Results

### 5.1 Modular Transformer Blocks

Due to the number of experiments conducted, the results in the paper content will primarily focus on the T5 model, trends mentioned in this section are shared across all models unless stated otherwise. For an individual breakdown of each model's scores and performance summaries of the other models, see Appendix A.

The individual GLUE scores for T5 can be seen in Table 1. Following the precedent set in Google's Adapter paper (Houlsby et al., 2019), we have excluded WNLI from our analysis due to the prevailing challenge where no current algorithm surpasses the baseline performance achieved by simply predicting the majority class. Summaries of the compute resources required for each PEFT method for

| Tune Method | Total Params | Trainable Params | COLA | SST2 | MRPC | STS-B | QQP | MNLI(m) | MNLI(mm) | QNLI | RTE | Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Full Model** | 60.8 M | 60.8 M | 30.1 | 90.7 | 88.0 | 84.8 | 71.4 | 82.8 | 82.8 | 89.6 | 56.9 | **74.3** |
| **Task Head** | 60.8 M | 263 K | 14.5 | 84.3 | 81.6 | 54.0 | 52.5 | 52.4 | 52.6 | 66.1 | 52.4 | **57.2** |
| **LoRA Adapter** | 61,2 M | 663 K | 19.6 | 90.5 | 85.8 | 84.5 | 70.7 | 81.2 | **N/A** | 89.3 | 54.4 | **72.0** |
| **MTB** | 64,9 M | 4.45 M | 8.0 | 90.3 | 80.9 | 77.9 | 67.5 | 77.1 | 77.0 | 85.5 | 53.9 | **67.6** |
| **LoRA MTB** | 65.3 M | 370 K | 28.3 | 90.0 | 84.7 | 81.2 | 67.6 | 76.5 | 76.5 | 86.1 | 53.2 | **71.0** |

Table 1: T5 Model: Results on GLUE test sets scored using the GLUE evaluation server. MRPC and QQP are evaluated using F1 score. STS-B is evaluated using Spearman's correlation coefficient. CoLA is evaluated using Matthew's Correlation. The other tasks are evaluated using accuracy. (MNLI mm excluded from Total calculation due to missing value, currently scheduled in DAIC)

| Tune Method | Model Size | Trainable Params | Train Time | Memory | GLUE Score |
|---|---|---|---|---|---|
| **Full Model** | 100.00% | 100.00% | 100.00% | 100.00% | **74.3** |
| **Task Head** | 100.00% | 0.43% | 37.72% | 6.55% | **57.2** |
| **LoRA Adapter** | 100.66% | 1.09% | 81.99% | 68.67% | **72.0** |
| **MTB** | 106.90% | 7.34% | 46.96% | 12.51% | **67.6** |
| **LoRA MTB** | 107.51% | 0.61% | 51.20% | 18.68% | **71.0** |

Table 2: T5 GLUE resource summary. (Percentages relative to "Full Model")

T5 are noted in Table 2.

The results from the fine-tuning experiments show that the Task-head tune was consistently the worst-performing tuning method, followed by MTBs and LoRA MTBs. Adapters generally were the highest-scoring PEFT method across all models. The BERT model is an exception to this trend as Adapter tuning achieved a GLUE score higher than the baseline full-model tune. This can be seen in Appendix A Table 9.

## 5.2 Transformer Bridge

### 5.2.1 Transformer Bridge Training

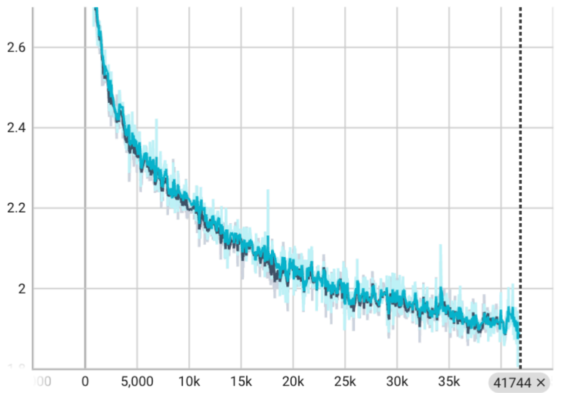
During training the performance of the bridge was measured using the log-scaled MSE of the predictions it made. Different configurations of the number of Encoder/Decoder blocks seemed to have minimal differences in their respective loss curves, because of this and due to time restraints bridges that were not the *1x1* shape were terminated early. The trend of the Bridge shape having minimal impact on the training curves was also noticed with the versions of the Bridge trained using cosine loss.

Due to the theme of this paper being parameter efficiency the smallest Bridge models were allowed to continue training. The best-performing 1x1 Bridge was selected to conduct our experiments, this correlated to the last checkpoint, Checkpoint-7 which correlates to 35,000 training steps. The performance metrics of Checkpoint-7 can be seen in Table 3.

The primary observation made during training

| Bridge | $\log_{\text{MSE}}$ |
|---|---|
| BERT to GPT2 | 6.62 |
| BERT to T5 | 1.79 |
| GPT2 to BERT | 3.83 |
| GPT2 to T5 | 1.80 |
| T5 to BERT | 3.84 |
| T5 to GPT2 | 6.54 |

Table 3: Log scaled MSE on the test set for the Transformer Bridge trained for 35k steps (Checkpoint 7).



| | Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|---|
| ● | 1x1-bert-to-t5 | 1.9002 | 1.9299 | 40,467 | 6.978 day |
| ● | 1x1-gpt2-to-t5 | 1.8858 | 1.8761 | 41,744 | 6.974 day |

Figure 7: Training $\log_{\text{MSE}}$ loss for T5 output Transformer Bridge

is the effect of the target embedding space being bridged. Across the 3 models tested; BERT, GPT2 and T5, the loss for generating embeddings for the T5 model were significantly lower than the other models. It is also observed that the input embeddings did not have much of an effect on the performance of the Bridge. This can be seen in Figure 7, where the loss curves for the T5 output Bridges are similar regardless of the input embedding space.

Another observation made by investigating the training curves of the Bridges is that there is still an opportunity for improvement. Figure 7 shows a gradient that has not yet completely stagnated, this sentiment is the same for the other bridges too though to a lesser degree. The loss curves for the BERT and GPT2 output Bridges can be found in Appendix B Figure 10. This suggests that the continuation of training may lead to better results.

The degree of improvement of the Bridge during training was also highly dependent on the target embedding space. As seen in Table 3 the GPT2 embedding seemed the most difficult for the Bridge to learn, though when considering the starting MSE at the beginning of the session was around 41 thousand which it reduced to around 690 the degree of improvement is much larger than the best performing T5 output Bridges. T5 output bridges started at a MSE of around 38 which was then reduced down to around 5.6.

#### 5.2.2 Transformer Bridge Evaluation

The combined results for all experiments using a foreign task-head with or without the bridge can be seen in Table 4. Due to the difference in the dimensionality of the T5 model (512 dimensions) with BERT and GPT-2 (768 dimensions), a no-bridge experiment for T5-related pairings was omitted.

The Bridge experiments were conducted using the GLUE validation set, this is due to time constraints and rate-limiting on the GLUE evaluation servers. We found that validation set results accurately reflected test set results when creating Table 1 and during the training of the Bridge the validation set was not used at all so it is unseen data.

During training the MSE loss suggested that some Bridges may outperform others in a real-world task. For the results in Tables 4, this does not seem to be the case overall all versions of the Bridge performed poorly. The inclusion of the Bridge achieved a slightly higher overall GLUE score than without the Bridge when using the GPT-2 space as the output, though this trend is inconsistent across all individual tasks.

## 6 Transformer Modules

The combined results for all *Foreign Module* Bridge experiments can be seen in Table 5. Generally, trends in the score share a similar resemblance to the findings in Section 5.2.2 except that the scores are higher across the board. Likewise, with Table 4 the no bridge experiments for the T5 model have been omitted due to dimensionality differences.

A few positive scenarios suggested that the use of the Bridge significantly improved prediction results. This was the case for the *SST2* and *MRPC* tasks, both of which are the easiest GLUE tasks where even the "Task-Head" tune in Table 1 was able to create a well-performing model. For SST2 the models using a Bridge outputting to the BERT embedding space achieved a relatively high score of 83%+, though this never happened again for any other task. As for the MRPC models which have a few instances of scoring higher than 80 this could very likely be due to the small size and imbalance of the dataset in scenarios like this F1 score is known to be inconsistent and possibly unreliable.

## 7 Discussion

### 7.1 Transformer Modules as a PEFT method

The experimental results for using Transformer Modules as a PEFT method look promising, whilst the overall GLUE score for the MTB base fine-tuning was lower than LoRA Adapters, most significantly for the BERT model due to it outperforming the baseline. LoRA MTBs generally seemed to be the most competitive PEFT method after LoRA Adapters.

The difference in performance between the standard and LoRA MTBs suggests that for the GLUE benchmark tasks, the additional trainable parameters in standard MTB tuning may have led to some over-fitting.

#### 7.1.1 Task Performance

When it comes to task performance the strength of Adapters is undeniable. LoRA Adapters even at times outperform the full-model baseline. This leads to the question as to why, for scenarios where a full-model tune under-performed this is most likely due to the model over-fitting on the training data. This is a well-researched phenomenon for large models being trained on simple tasks.

When investigating why Adapters scored higher on average than the MTB-based tuning the reason leads back to the placement of the trainable parameters in the model. Adapters place trainable parameters at each Transformer block in the model,

| Model | COLA | SST2 | MRPC | STS-B | QQP | MNLI(m) | MNLI(mm) | QNLI | RTE | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| **T5 Native** | **10.5** | **81.9** | **81.9** | **51.4** | **60.9** | **52.0** | 54.4 | **65.0** | **55.2** | **57.0** |
| **T5 - Bridge - GPT2** | -2.10 | 52.2 | 49.8 | 8.00 | 0.100 | 33.2 | 34.5 | 50.5 | 52.3 | 30.9 |
| **T5 - Bridge - BERT** | 4.60 | 73.5 | 0.000 | 13.7 | 0.000 | 32.8 | 33.1 | 50.5 | 47.3 | 28.4 |
| **GPT Native** | **4.40** | **83.6** | **83.4** | **66.4** | **67.3** | **53.6** | 56.3 | **66.2** | **55.6** | **59.6** |
| **GPT2 - Bridge - T5** | 0.000 | 53.7 | 62.6 | -2.50 | 42.5 | 35.9 | 35.4 | 51.2 | 47.7 | 36.3 |
| **GPT2 - Bridge - BERT** | 4.60 | 74.8 | 0.000 | 8.50 | 0.000 | 32.9 | 33.2 | 50.5 | 47.3 | 28.0 |
| **GPT2 - BERT (no bridge)** | 0.000 | 50.9 | 0.700 | 5.60 | 53.7 | 32.9 | 33.2 | 50.5 | 52.7 | 31.2 |
| **BERT Native** | **29.3** | **81.0** | **81.1** | **68.5** | **36.2** | **42.3** | 43.2 | **68.7** | **57.8** | **56.4** |
| **BERT - Bridge - T5** | 4.60 | 58.3 | 70.5 | 13.2 | 38.6 | 38.3 | 39.2 | 50.4 | 49.1 | 40.2 |
| **BERT - Bridge - GPT2** | 0.000 | 53.0 | 51.6 | 6.70 | 0.100 | 33.9 | 34.3 | 51.3 | 54.9 | 31.8 |
| **BERT - GPT2 (no bridge)** | -2.80 | 47.5 | 0.000 | -3.50 | 53.8 | 33.1 | 33.0 | 50.1 | 52.7 | 29.3 |

Table 4: GLUE Benchmark results of models using a **foreign task-head** and a $\log_{\text{MSE}}$ loss optimized bridge. MRPC and QQP are evaluated using F1 score. STS-B is evaluated using Spearman's correlation coefficient. CoLA is evaluated using Matthew's Correlation. The other tasks are evaluated using accuracy. (Baselines in bold)

| Model | COLA | SST2 | MRPC | STS-B | QQP | MNLI(m) | MNLI(mm) | QNLI | RTE | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| **T5 Native** | **6.60** | **88.6** | **82.5** | **83.3** | **83.7** | **76.9** | 77.9 | **85.4** | **51.3** | **70.7** |
| **T5 - Bridge - GPT2** | -1.60 | 49.9 | 81.1 | 15.0 | 0.000 | 33.7 | 33.4 | 50.6 | 53.4 | 35.1 |
| **T5 - Bridge - BERT** | 1.40 | 83.9 | 2.10 | 14.0 | 0.000 | 32.9 | 36.3 | 50.7 | 46.9 | 29.8 |
| **GPT Native** | **5.90** | **88.5** | **83.3** | **71.0** | **80.1** | **69.5** | 71.6 | **75.3** | **58.5** | **67.0** |
| **GPT2 - Bridge - T5** | -4.90 | 50.9 | 81.2 | 4.60 | 0.000 | 33.3 | 33.7 | 49.5 | 52.7 | 33.4 |
| **GPT2 - Bridge - BERT** | 7.20 | 84.3 | 2.10 | 12.2 | 0.000 | 33.0 | 36.5 | 50.6 | 45.8 | 30.2 |
| **GPT2 - BERT (no bridge)** | 0.000 | 49.1 | 81.2 | -0.100 | 46.3 | 31.8 | 31.8 | 50.6 | 52.7 | 38.1 |
| **BERT Native** | **42.7** | **90.8** | **83.3** | **85.5** | **84.4** | **46.7** | 78.4 | **55.3** | **56.3** | **69.3** |
| **BERT - Bridge - T5** | -6.90 | 50.9 | 81.2 | -0.800 | 0.000 | 34.3 | 34.5 | 49.5 | 52.7 | 32.8 |
| **BERT - Bridge - GPT2** | 0.400 | 49.1 | 79.7 | 20.4 | 0.000 | 34.6 | 34.0 | 50.6 | 54.5 | 35.9 |
| **BERT - GPT2 (no bridge)** | 16.9 | 51.3 | 0.000 | -3.00 | 33.0 | 31.8 | 31.8 | 47.7 | 51.6 | 29.0 |

Table 5: GLUE Benchmark results of models using a **foreign Transformer Module** and a $\log_{\text{MSE}}$ loss optimized bridge. MRPC and QQP are evaluated using F1 score. STS-B is evaluated using Spearman's correlation coefficient. CoLA is evaluated using Matthew's Correlation. The other tasks are evaluated using accuracy. (Baselines in bold)

whilst MTBs rely solely on the embedding space created by the final Transformer block. This difference may handicap the MTB's opportunity to learn for the upstream task. Adapters can learn how to use information from various levels of abstraction in the original model, allowing for task-relevant features from multiple stages of the model to be used during the fine-tuning process. These features can be lost as you progress down the parameters of the model negatively impacting the learning opportunity of an MTB at the tail end of a model.

| Model | Train Time | Memory | GLUE Score |
|---|---|---|---|
| **GPT-2** | -42.57% | -66.74% | -4.50% |
| **BERT** | -38.97% | -78.43% | -7.61% |
| **T5** | -37.55% | -72.80% | -1.46% |

Table 6: Average percentage difference of LoRA MTB relative to LoRA Adapters

Having said that the MTB results on GLUE still achieved decently good results, when compared with the state-of-the-art LoRA adapters across all models LoRA MTBs averaged a score within 4.5±2.51%, for the T5-based experiments MTBs were within 1.46% of LoRA adapters performance as seen in Table 6

### 7.1.2 Compute Efficiency

When considering the goals of PEFT as a form of efficient training of LLMs. MTB-based fine-tuning has shown substantial improvements over Adapter tuning in terms of compute resources. As seen in Table 2, MTB-based tuning trained significantly faster than both a Full Model tune and an Adapter tune. MTBs not only reduced the training time but they did so while using significantly less GPU memory as well. The computational efficiency of MTBs was only beaten by a task-head-only tune, which did not achieve competitive results in the benchmark.

When again compared directly with LoRA Adapters in Table 6, on average across all models LoRA MTBs reduced the required training time by 39.7±2.11% and reduced the required training memory by 72.7±4.77%.

The substantial reduction in the required compute resources to tune a model using MTBs again lies in the placement of the trainable parameters in the model. As mentioned in Section 3.1.1, the contiguous nature of the trainable parameters in MTB-based fine-tuning became one of its greatest strengths. Since Adapters are distributed across multiple layers of the model, the computing process needs to traverse through these additional layers at each stage, calculating the gradients for a significant portion of the model thus increasing the computational overhead. These gradients need to be stored during calculation and this is saved in the GPU memory, resulting in Adapters requiring significantly more processing time as well as more memory.

## 7.2 Transformer Bridge and the creation of Foreign Modules

Experiments involving the Transformer Bridge showed poor results in most cases, this could be due to various factors. Initially, the most prevailing one was that the MSE loss of generated embeddings was not low enough to accurately translate the embedding space between two models. To decide whether or not this is the case a further investigation into the training dynamics and the embedding spaces created by the bridge was necessary.

### 7.2.1 Training differences between models

The first observation of the bridge experiments was the performance differences during training. As seen in Figure 7 and Figure 10 it is seen that the biggest indicator of the training performance of the Bridge was the output embedding space. Where T5 embeddings achieved the lowest MSE followed by BERT, the GPT-2 embedding space seemed most difficult to model with the MSE being in the magnitude of the 10s of thousands and by the 7th checkpoint still being in the thousands range.

The low loss for the T5 space is most likely due to the use of a pre-trained T5 model for the bridge architecture, using the pre-trained weights for T5 introduces an opportunity for transfer learning as the embedding space created by the bridge before training is likely aligned with the original T5 embedding space. This is the reason why the T5 output bridges started at a relatively low MSE and got lower from there.

To explain the training differences between the BERT and GPT-2 embedding spaces a visualization was created. To do this a sample of 1000 sentences were created, each one being passed through all 3 models. For each output embedding sequence a *"key embedding"* was selected, this correlates to the embedding which has a contextual representation of the complete sentence. For BERT this correlates to the first embedding in the sequence correlating to the **<CLS>** token and for GPT-2 and T5 due to their auto-regressive nature this correlated to the last non-padded embedding. Due to a miss-match in the embedding dimensions of the 3 models, all 3 were reduced to a dimension of 50 using Principle Component Analysis (PCA). The reduced embeddings were then visualized using t-distributed Stochastic Neighbor Embedding (t-SNE), separating them into clusters the resulting visualization can be seen in Figure 8.

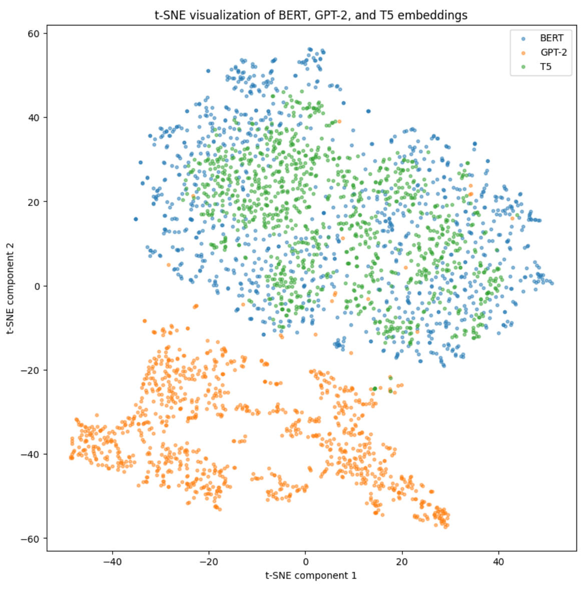


Figure 8: t-SNE visualization of GPT-2, BERT and T5 Embeddings (Perplexity=30)

Figure 8, we can see 2 distinct clusters where both the BERT and T5 embedding spaces have been grouped in the same cluster. This indicates that the BERT and T5 embedding spaces are more closely aligned to each other than GPT-2, this alignment is the most likely reason for BERT-based bridges outperforming the GPT-2-based bridges in terms of loss, as BERTs embedding space is more aligned with the initial embedding space created by the Bridge.

As for why the input embedding space and size of the model seemed to have a minimal impact on the training performance. This can be due to all 3 embedding spaces being sufficiently rich and informative for the task, or because the task itself may not be completely aligned with what is being optimized during training. The latter of which is discussed further in Section 7.2.2.

### 7.2.2 Exploring the Transformer Bridge performance

When analysing the performance of the Bridge its failure in the task of adequately simulating the embedding space of the target model is undeniable, this is seen in both Table 5 and Table 4.

Initially when looking at the training curves of the models in Figure 7 and Figure 10, it could be argued that the model has not been trained sufficiently to be evaluated on a real-world task like the GLUE benchmark. If this is the case we can take a deeper look into the bridge and its learning progress for its main task of embedding-space simulation. As the loss of the bridge decreases then the expectation is that the embedding space created by the bridge more closely resembles the target embedding space, to verify this we can visualize the embedding space of the bridge and its target at various stages of training, given that the 7th checkpoint was used for the evaluation we have 6 more to explore.

To do this similarly to how the embedding spaces of each model were explored in Section 7.2.1, we can visualize the generated embedding space of the bridge and compare it with the target using PCA and t-SNE. If the model is in fact learning, as the training graphs imply, then the embedding spaces of the bridge and its target model should appear to be converging to a single cluster. If this is the case it can be argued that the bridge has not been trained enough and must reach a loss "threshold" for which it can be used.

Visualizations of the embedding space created by the BERT to T5 bridge using t-SNE can be seen in Figure 9. Checkpoints 1,3,5 and 7 were selected to show the progress of the convergence of the Bridge's output embedding space with the target T5 embedding space. As seen in Figure 9, the t-SNE algorithm was successfully able to separate the two embedding spaces into clear clusters. This separation of embedding space suggests that there is an intrinsic difference between the generated T5 embedding space and the target embedding space which does not seem to improve as training progresses. This sentiment is shared in a similar analysis using PCA visualizations in Appendix B.1 Figure 11, while t-SNE actively attempts to create clusters directly visualizing with PCA we can create a visualization without this feature. In both figures, we can see that the two embedding spaces maintain two obvious clusters that do not seem to improve as training progresses. This lack of improvement is further showcased by the highlighted sentence pairs for each embedding space where the expectation is that over time the highlighted sentences should move closer to their respective pairing in the other embedding space.

The figures in this section and Appendix B.1 have shown that there is a disconnect between the optimization metric of the bridge during training and the actual training goal. There does not seem to be a relationship between a distance metric like MSE and the ability to model the embedding space of the model. This lack of alignment can be an indicator as to why the performance of the bridge when used on a real-world task like GLUE was relatively low.

### 7.2.3 Aligning the Bridge training for the downstream task

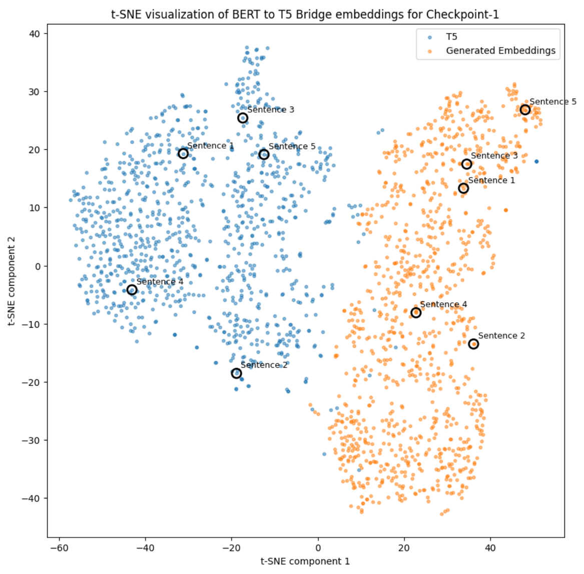
The training of the bridge is based on two assumptions:

1. For an embedding in Model *A*'s space there exists a contextual equivalent embedding in Model *B*'s space.

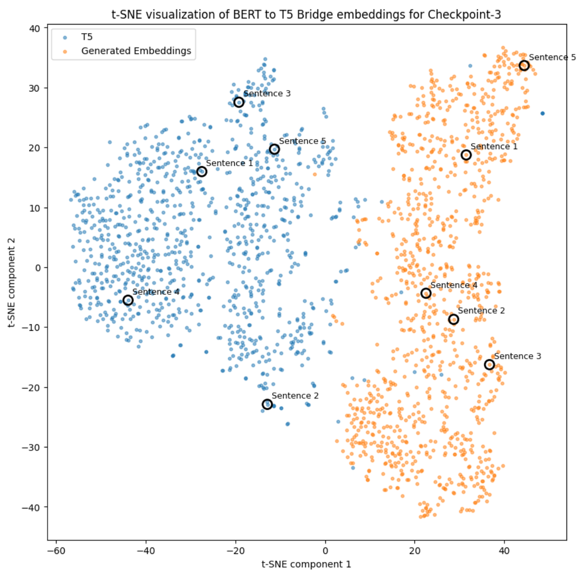
2. For a given embedding within a model's space, contextually similar embeddings are situated in close proximity to it.

The choice of using a distance metric for the loss of generated embeddings is based on these assumptions. The first assumption for most complex LLMs is most likely true as the models both model the same domain, the contextual meaning of natural language. As for the second assumption, it is assumed that if the bridge generates an embedding that is close in proximity to the target then the generated embedding shares a similar contextual meaning to the target as well. This allows the model to generate contextual embeddings while also having a non-zero loss, as long as it is generally close to the target it should have contextually the same meaning allowing it to be used successfully for the upstream task.

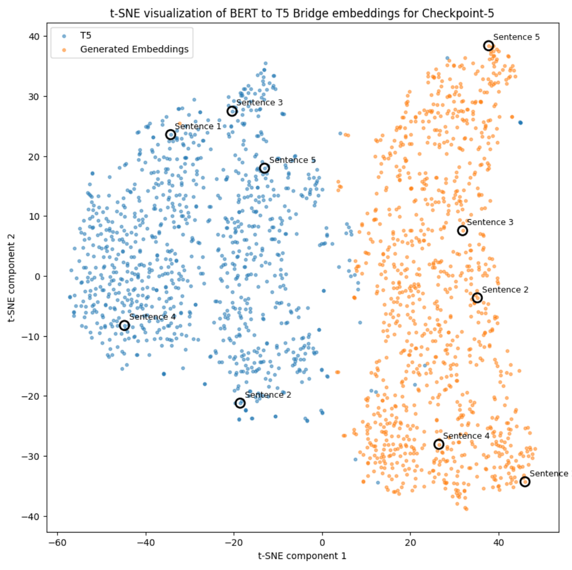
To investigate the validity of the second assumption a small experiment was conducted. For each model, the "key" embedding for different sets of sentences was acquired. These sets are a set of contextually similar sentences, a set of contextually



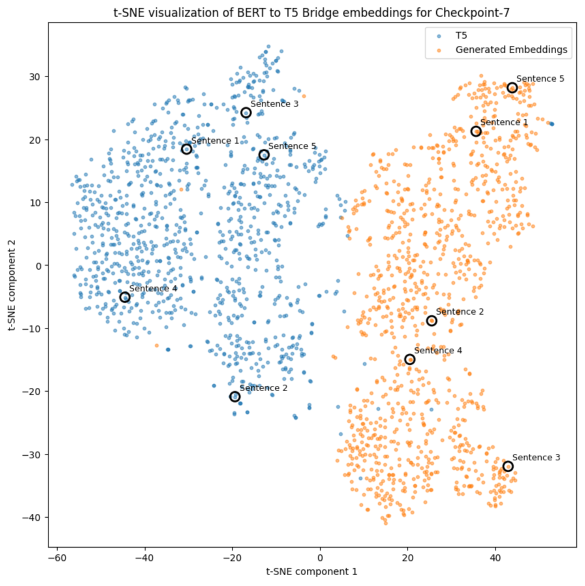
(a) Checkpoint 1 (5000 Training Steps)



(b) Checkpoint 3 (15000 Training Steps)



(c) Checkpoint 5 (25000 Training Steps)



(d) Checkpoint 7 (35000 Training Steps)

Figure 9: t-SNE visualization of BERT to T5 Transformer Bridge and the Target embedding space, the same sentence pairs are highlighted for each space and checkpoint. (Sample size 1000, Perplexity = 30)

different sentences and a set of the same sentence repeated. The repeated sentence set contains repetitions of the sentence "The quick brown fox jumps over the lazy dog.", while the remaining sets contain the following:

**Contextually similar sentence set:**

1. "The quick brown fox jumps over the lazy dog."
2. "The fast coffee coloured fox leaps above the lethargic dog."
3. "The swift brown vulpine hops over a slothful canine."

**Contextually different sentence set:**

1. "Tomorrow's weather forecast shows rain."
2. "Mathematics is the study of numbers, shapes, and patterns."
3. "Elephants are the largest land animals on Earth."

The MSE for each pair of sentences in the respective sets were calculated, for the set containing identical sentences the models consistently returned the same embedding meaning that the models are deterministic with the provided input and scored an

MSE of 0. Table 7 show the MSE of each sentence pair in both the similar and different sets, for the same comparison using Cosine-Loss see Appendix C.4. In it, we can see that MSE does not seem to be a good indicator of sentence similarity as both sentences that contextually mean the same and completely different things when compared have similar MSE values.

| Model | Pair | MSE Similar | MSE Different |
|---|---|---|---|
| BERT | 1 - 2 | 0.252 | 0.378 |
| BERT | 1 - 3 | 0.286 | 0.268 |
| BERT | 2 - 3 | 0.317 | 0.343 |
| GPT-2 | 1 - 2 | 5.83 | 13.3 |
| GPT-2 | 1 - 3 | 5.84 | 5.71 |
| GPT-2 | 2 - 3 | 2.66 | 10.2 |
| T5 | 1 - 2 | 0.136 | 0.129 |
| T5 | 1 - 3 | 0.209 | 0.152 |
| T5 | 2 - 3 | 0.165 | 0.149 |

Table 7: Comparison of MSE values for similar and different sentences across models. Sentence indices refer to the pre-listed sentence groups for similar and different contexts.

This shows that a distance metric-based loss like MSE and Cosine Similarity, may not be optimal for embedding mapping. In practice, a near 0 MSE would be required to translate embeddings from one space to another accurately. This leads to the question of how can a model be trained for such a task. For this, a different strategy should be employed. Originally in this paper for an embedding in space $A$ we are optimizing the model to find the exact match for that embedding in space $B$ for the given input, this task may be too difficult for a model to learn given the infinite size of the respective spaces.

We can formulate the problem in a new way, let $A$ be an embedding space and $B$ another embedding space. We can define the *equivalence set* $S \subseteq B$ such that for any embedding $a \in A$, there exists at least one embedding $b \in S$ which contextually represents similar information to $a$. Thus, $S$ contains the embeddings from $B$ that, while not necessarily close in spatial terms, share critical features necessary for achieving equivalent predictions in an upstream task.

To train a model for this task we can look at how the compression of transformer models is achieved using a more traditional *model distillation* technique. In a recent paper researchers from the Alibaba Group (Brown et al., 2023) suggest the best practices for performing model distillation on transformer models. Adapting these recommendations for the use case of embedding translation, the model of the target embedding space can be used as a teacher. To build the student model, a Transformer Bridge's input model can be frozen, after which a bridge can be appended to the model along with a frozen task head from the teacher model, this setup would be similar to the one shown in Figure 5c with the exception that only the bridge has trainable parameters. The student model undergoes similar training to the target model, in the case of LLMs this can involve undergoing multiple tasks, loss functions and datasets for some even requiring different task heads. The teacher model provides *soft-labels* which is used in combination with the *hard-label* which is the true label for the current task. In such a scenario the model containing the bridge can only succeed if it is successfully predicting embeddings that contextually represent the same information as the teacher while not optimizing on the much harder task of generating an exact match.

## 8 Conclusion

Given the ever-increasing size of LLMs the study of PEFT methods becomes increasingly important, and given the availability, and cost of training LLMs with large datasets the need for a transferable pre-trained fine-tuning becomes an interesting problem to tackle. In this paper, we introduce Transformer Modules a form of Parameter Efficient Fine-Tuning (PEFT) that facilitates inter-model transfer-ability. The transfer-ability of pre-trained Transformer Modules is facilitated by a Transformer Bridge, a neural network trained to translate one transformer model embedding space into another. A pre-trained Module paired with a Bridge should allow Modules to be used in models outside of the original one it is trained with.

This work would allow low-resource entities to train specialized transformer models without the requirement of a large infrastructure. The ability of cross-model transfer-ability of pre-trained modules would allow for tuned models to be created with no training, circumventing the need for large infrastructure, time and access to large datasets making Transformer Modules the first PEFT method to support such a use-case.

Transformer Modules as a PEFT method showed promising results, significantly reducing the requirement in terms of compute and GPU memory to train LLMs on the GLUE benchmark. This reduction in compute came at a slight cost to the peak performance of the model as generally a full-model tune or the use of Transformer Adapters achieved higher GLUE scores, both used significantly more compute resources. The choice of a PEFT method depends on the training goals and available resources, there are scenarios where MTB-based tuning is favourable and others where Adapter-based tuning is.

With the current training setup, the Transformer Bridge failed to accurately translate embeddings created by two models. The experimentation led to key findings on how best to tackle the problem in the future suggesting that by using traditional Model Distillation techniques an accurate bridge can be trained.

## 9 Limitations

The primary limitation of the study on Transformer Modules revolves around the generality and transferability of the proposed approach. Although the results are promising, several aspects would benefit from further exploration and validation:

### 9.1 Limited Model Scope

The experiments conducted in this study were limited to three specific models: BERT, GPT-2, and T5. While these models are popular and well-researched, the applicability of Transformer Modules to other models, including newer or more diverse architectures, remains untested. This potentially limits the generalizability of the findings and calls for broader experimentation with a wider range of transformer models to establish the robustness of the approach. If tested on newer models like GPT-4 or domain-specific models like BioBERT, we might see variations in performance efficiency and accuracy.

### 9.2 Benchmark Diversity

The evaluation of Transformer Modules was conducted solely on the GLUE benchmark. While GLUE is a comprehensive and widely used benchmark for natural language understanding tasks, it does not cover all possible types of tasks or domains to which transformer models might be applied. Testing on benchmarks like ImageNet for image classification or LibriSpeech for speech recognition could reveal different computational efficiencies and fine-tuning effectiveness.

### 9.3 Real-World Applicability

The practical application of Transformer Modules in real-world scenarios has not been fully explored. Factors such as the complexity of real-world data, the variability of input distributions, and the specific requirements of different use cases might affect the performance and utility of Transformer Modules. Extensive testing in varied and practical settings is required to validate the approach's effectiveness outside of controlled experimental conditions.

### 9.4 Scalability and Efficiency

While the study demonstrates that Transformer Modules can reduce computational requirements during fine-tuning, the scalability of this approach to extremely large models and datasets, which are common in real-world applications, needs further investigation. The balance between achieving efficiency and maintaining high performance is crucial for practical deployment. When applied to large-scale datasets or models with billions of parameters, the reduction in computational requirements may not scale linearly.

## References

Nathan Brown, Ashton Williamson, Tahj Anderson, and Logan Lawrence. 2023. Efficient transformer knowledge distillation: A performance review. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 54–65, Singapore. Association for Computational Linguistics.

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.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

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

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. 2023. LLM-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5254–5276, Singapore. Association for Computational Linguistics.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *Advances in Neural Information Processing Systems*, volume 34, pages 1022–1035. Curran Associates, Inc.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. 2016. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

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.

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

Chengqiang Lu, Jianwei Zhang, Yunfei Chu, Zhengyu Chen, Jingren Zhou, Fei Wu, Haiqing Chen, and Hongxia Yang. 2022. Knowledge distillation of transformer-based language models revisited.

OpenAI. 2023. Gpt-4 technical report.

Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Ponti. 2023. Modular deep learning. *arXiv*.

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.

Pengjie Ren, Chengshun Shi, Shiguang Wu, Mengqi Zhang, Zhaochun Ren, Maarten de Rijke, Zhumin Chen, and Jiahuan Pei. 2024. Mini-ensemble low-rank adapters for parameter-efficient fine-tuning.

Adam Roberts, Colin Raffel, Katherine Lee, Michael Matena, Noam Shazeer, Peter J. Liu, Sharan Narang, Wei Li, and Yanqi Zhou. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. Technical report, Google.

Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. 2020. Leveraging pre-trained checkpoints for sequence generation tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280.

Thomas Theodoridis, Theocharis Chatzis, Vassilios Solachidis, Kosmas Dimitropoulos, and Petros Daras. 2020. Cross-modal variational alignment of latent spaces. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4127–4136.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In the Proceedings of ICLR.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment.

## A PEFT GLUE Results

### A.1 Individual Glue Scores

| Tune Method | Total Params | Trainable Params | COLA | SST2 | MRPC | STS-B | QQP | MNLI(m) | MNLI(mm) | QNLI | RTE | Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Full Model** | 124 M | 124 M | 35.5 | 92.6 | 83.2 | 81.1 | 69.7 | 81.8 | **N/A** | 89.1 | 55.1 | **72.3** |
| **Task Head** | 124 M | 1.54 K | 6.0 | 84.4 | 80.9 | 65.1 | 55.1 | 54.2 | 55.1 | 64.5 | 51.8 | **58.3** |
| **LoRA Adapter** | 125 M | 896 K | 0.0 | 92.9 | 84.9 | 79.5 | 69.3 | **N/A** | 80.8 | 87.1 | 55.3 | **67.0** |
| **MTB** | 132 M | 7.09 M | 7.0 | 86.7 | 80.7 | 63.9 | 64.3 | 69.6 | 70.1 | 74.6 | 53.0 | **61.5** |
| **LoRA MTB** | 132 M | 69.1 K | 8.6 | 88.4 | 80.9 | 73.7 | 63.7 | 70.7 | 72.9 | 79.9 | 52.7 | **64.0** |

Table 8: GPT-2 Model: Results on GLUE test sets scored using the GLUE evaluation server. MRPC and QQP are evaluated using F1 score. STS-B is evaluated using Spearman's correlation coefficient. CoLA is evaluated using Matthew's Correlation. The other tasks are evaluated using accuracy. (MNLI excluded from Total calculation due to missing values, job is currently scheduled in DAIC)

| Tune Method | Total Params | Trainable Params | COLA | SST2 | MRPC | STS-B | QQP | MNLI(m) | MNLI(mm) | QNLI | RTE | Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Full Model** | 109 M | 109 M | 51.7 | 93.6 | 79.9 | 83.0 | 71.6 | 84.9 | **N/A** | 91.0 | 50.3 | **75.8** |
| **Task Head** | 109 M | 1.54 K | 27.7 | 82.0 | 79.8 | 50.6 | 24.5 | 42.5 | 43.4 | 69.8 | 56.3 | **54.2** |
| **LoRA Adapter** | 110 M | 896 K | 47.9 | 91.2 | 83.8 | 83.9 | 70.8 | 83.9 | 83.8 | 91.1 | 62.3 | **76.9** |
| **MTB** | 117 M | 7.09 M | 39.2 | 90.4 | 81.5 | 78.2 | 67.5 | 78.5 | 78.2 | 86.1 | 59.2 | **67.6** |
| **LoRA MTB** | 117 M | 112 K | 28.3 | 90.0 | 84.7 | 81.7 | 67.6 | 76.5 | 77.2 | 86.1 | 53.2 | **71.0** |

Table 9: BERT Model: Results on GLUE test sets scored using the GLUE evaluation server. MRPC and QQP are evaluated using F1 score. STS-B is evaluated using Spearman's correlation coefficient. CoLA is evaluated using Matthew's Correlation. The other tasks are evaluated using accuracy. (MNLI_mm excluded from Total calculation due to missing values)

### A.2 PEFT Compute Resource Summaries

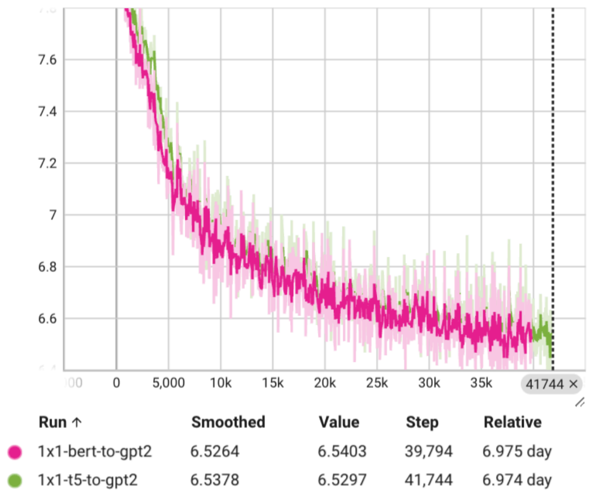
| Tune Method | Model Size | Trainable Params | Train Time | Memory | GLUE Score |
|---|---|---|---|---|---|
| **Full Model** | 100.00% | 100.00% | 100.00% | 100.00% | **72.3** |
| **Task Head** | 100.000% | 0.001% | 36.13% | 15.20% | **58.3** |
| **LoRA Adapter** | 100.719% | 0.72% | 82.24% | 76.01% | **67.0** |
| **MTB** | 105.696% | 5.70% | 44.18% | 20.41% | **61.5** |
| **LoRA MTB** | 105.751% | 0.06% | 47.23% | 25.28% | **64.0** |

Table 10: GPT-2 GLUE resource summary. (Percentages relative to "Full Model")

| Tune Method | Model Size | Trainable Params | Train Time | Memory | GLUE Score |
|---|---|---|---|---|---|
| **Full Model** | 100.00% | 100.00% | 100.00% | 100.00% | **72.3** |
| **Task Head** | 100.00% | 0.001% | 37.84% | 5.68% | **58.3** |
| **LoRA Adapter** | 100.82% | 0.82% | 79.73% | 69.56% | **67.0** |
| **MTB** | 106.47% | 6.48% | 45.48% | 9.73% | **61.5** |
| **LoRA MTB** | 106.58% | 0.10% | 48.66% | 15.00% | **64.0** |

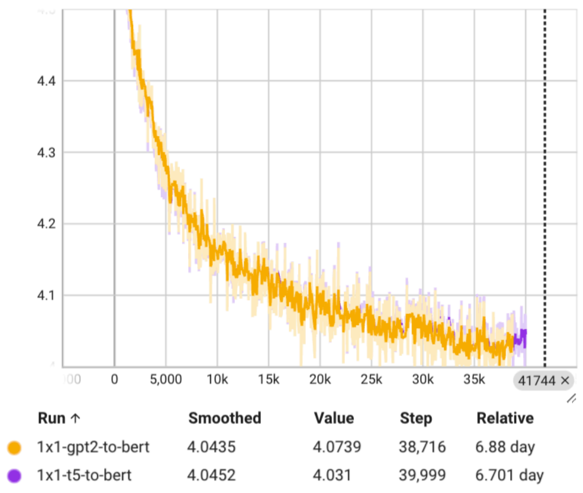
Table 11: BERT GLUE resource summary. (Percentages relative to "Full Model")

## B Transformer Bridge Graphs



| | Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|---|
| ● | 1x1-bert-to-gpt2 | 6.5264 | 6.5403 | 39,794 | 6.975 day |
| ● | 1x1-t5-to-gpt2 | 6.5378 | 6.5297 | 41,744 | 6.974 day |

(a) GPT-2 Output Bridges



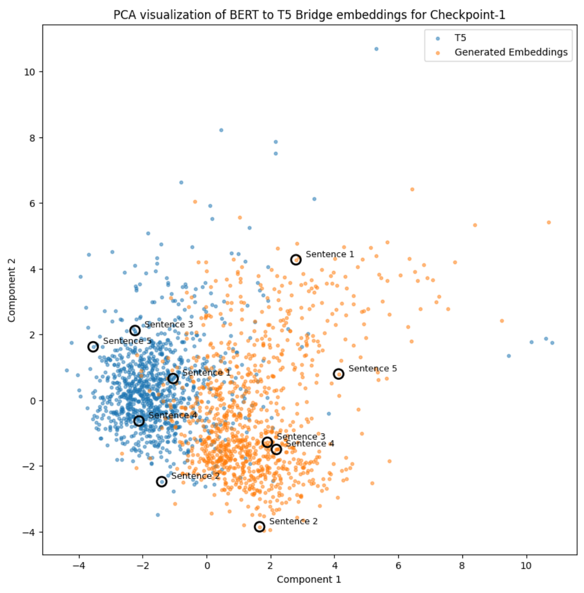
| | Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|---|
| ● | 1x1-gpt2-to-bert | 4.0435 | 4.0739 | 38,716 | 6.88 day |
| ● | 1x1-t5-to-bert | 4.0452 | 4.031 | 39,999 | 6.701 day |

(b) BERT Output Bridges

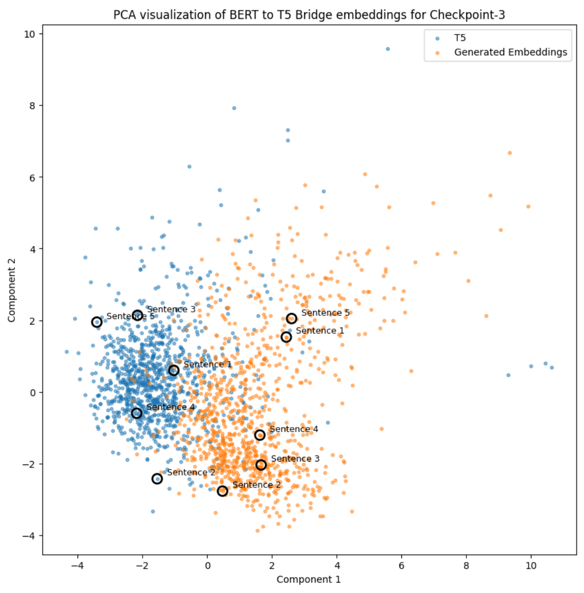
Figure 10: Training $\log_{\text{MSE}}$ loss for Transformer Bridges

## B.1 Embedding Space Exploration

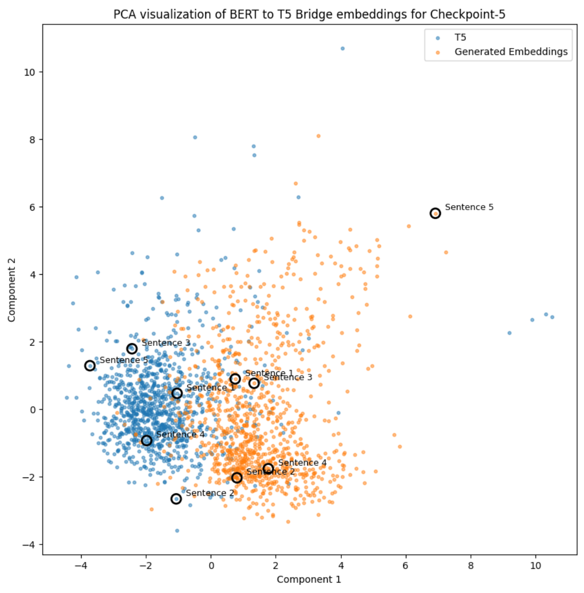
### B.1.1 BERT to T5 Bridge PCA Visualization



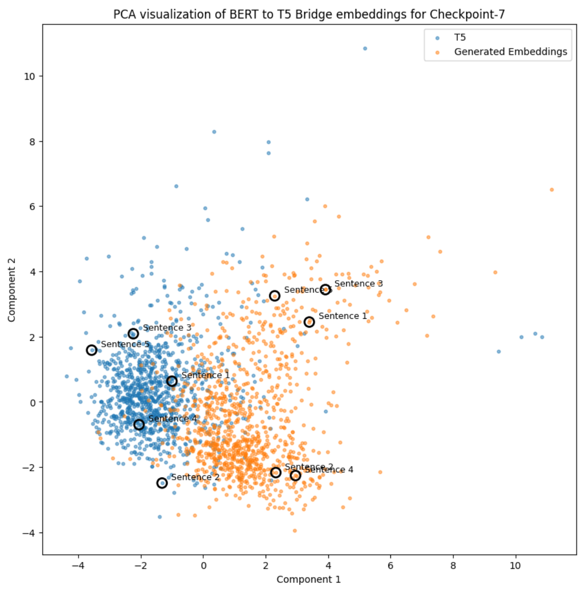
(a) Checkpoint 1 (5000 Training Steps)



(b) Checkpoint 3 (15000 Training Steps)



(c) Checkpoint 5 (25000 Training Steps)

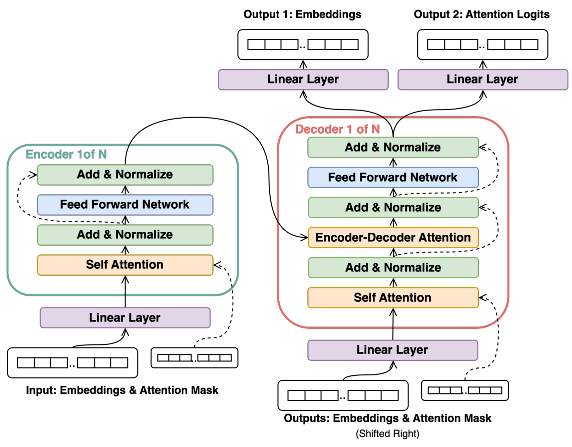


(d) Checkpoint 7 (35000 Training Steps)

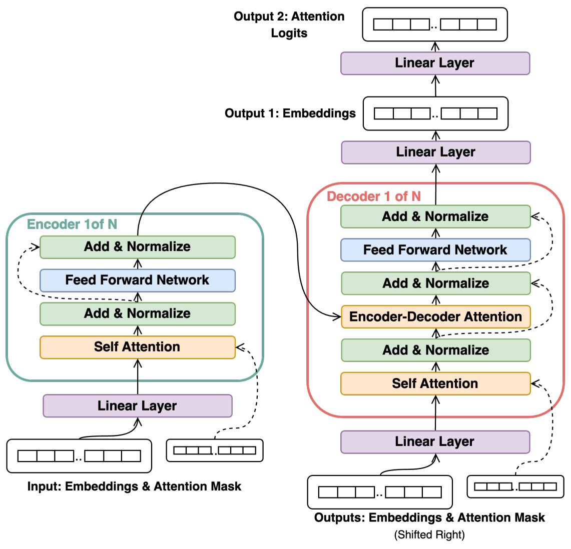
Figure 11: PCA visualization of BERT to T5 Transformer Bridge and the Target embedding space, the same sentence pairs are highlighted for each space and checkpoint. (Sample size 1000, dimension reduced to 2 components)

## C Alternative Transformer Bridge

### C.1 Alternative Architectural Designs



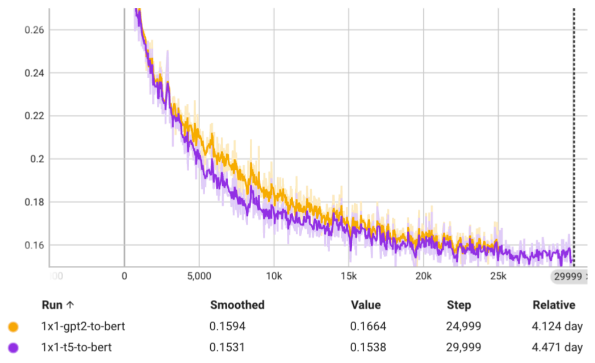
(a) Forked Dual-Task Head



(b) Stacked Dual-Task Head

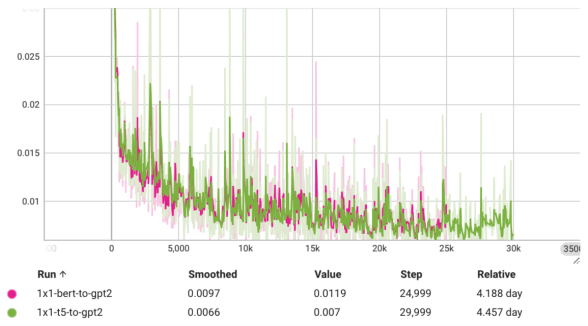
Figure 12: Alternative Transformer Bridge Designs experimented on while writing this paper.

### C.2 Cosine Loss Training Graphs



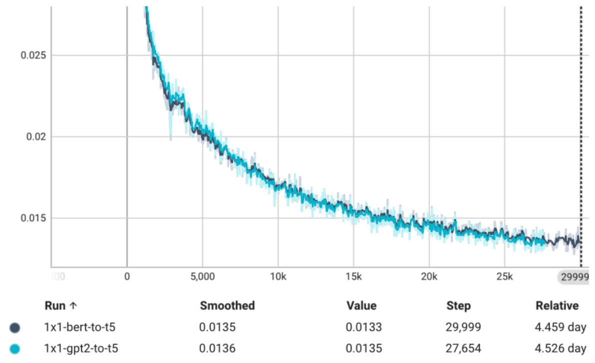
| | Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|---|
| ● | 1x1-gpt2-to-bert | 0.1594 | 0.1664 | 24,999 | 4.124 day |
| ● | 1x1-t5-to-bert | 0.1531 | 0.1538 | 29,999 | 4.471 day |

(a) BERT Output Bridges



| | Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|---|
| ● | 1x1-bert-to-gpt2 | 0.0097 | 0.0119 | 24,999 | 4.188 day |
| ● | 1x1-t5-to-gpt2 | 0.0066 | 0.007 | 29,999 | 4.457 day |

(b) GPT-2 Output Bridges



| | Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|---|
| ● | 1x1-bert-to-t5 | 0.0135 | 0.0133 | 29,999 | 4.459 day |
| ● | 1x1-gpt2-to-t5 | 0.0136 | 0.0135 | 27,654 | 4.526 day |

(c) T5 Output Bridges

Figure 13: Training Loss Graphs for Cosine loss based Transformer Bridge

### C.3 Cosine Loss GLUE Evaluation

| **Bridge** | **Cosine Loss** |
|---|---|
| BERT to GPT2 | 0.0087 |
| T5 to GPT2 | 0.0086 |
| BERT to T5 | 0.0150 |
| GPT2 to T5 | 0.0146 |
| GPT2 to BERT | 0.1558 |
| T5 to BERT | 0.1482 |

Table 12: Cosine Loss for Transformer Bridge at the Checkpoint 2

| | **COLA** | **SST2** | **MRPC** | **STS-B** | **QQP** | **QNLI** | **RTE** | **Total** |
|---|---|---|---|---|---|---|---|---|
| **T5 Native** | **10.5** | **81.9** | **81.9** | **51.4** | **60.9** | | **55.2** | **58.1** |
| **T5 - Bridge - GPT2** | 2.90 | 54.1 | 33.0 | 2.20 | 6.40 | 52.4 | 49.1 | 28.6 |
| **T5 - Bridge - BERT** | 0.000 | 60.1 | 0.000 | 14.2 | 0.000 | 50.5 | 47.3 | 24.6 |
| **GPT Native** | **4.40** | **83.6** | **83.4** | **66.4** | **67.3** | **66.2** | **55.6** | **61.0** |
| **GPT2 - Bridge - T5** | 0.000 | 51.1 | 81.1 | 1.40 | 55.0 | 53.4 | 48.0 | 41.4 |
| **GPT2 - Bridge - BERT** | 0.000 | 70.4 | 0.000 | 7.10 | 0.000 | 50.5 | 47.3 | 25.1 |
| **GPT2 - BERT (no bridge)** | 42.2 | 50.9 | 0.700 | 5.60 | 53.8 | 50.5 | 52.7 | 36.6 |
| **BERT Native** | **29.3** | **81.0** | **81.1** | **68.5** | **36.2** | **68.7** | **57.8** | **60.4** |
| **BERT - Bridge - T5** | 0.000 | 55.3 | 81.4 | 8.70 | 25.4 | 49.2 | 48.7 | 38.4 |
| **BERT - Bridge - GPT2** | 1.80 | 51.1 | 35.4 | 9.30 | 1.40 | 52.0 | 50.9 | 28.9 |
| **BERT - GPT2 (no bridge)** | -2.80 | 47.5 | 0.000 | -3.50 | 53.7 | 50.1 | 52.7 | 28.2 |

Table 13: GLUE Benchmark results of models using a **foreign task-head** and a Cosine loss optimized bridge. MNLI GLUE Tasks excluded due to not being completed by the time of experimentation. (Baselines in bold, B = includes bridge)

| | **COLA** | **SST2** | **MRPC** | **STS-B** | **QQP** | **QNLI** | **RTE** | **Total** |
|---|---|---|---|---|---|---|---|---|
| **T5 Native** | **6.60** | **88.6** | **82.5** | **83.3** | **83.7** | **85.4** | **51.3** | **68.8** |
| **T5 - Bridge - GPT2** | 0.000 | 49.1 | 81.2 | 11.2 | 0.000 | 51.1 | 51.3 | 34.8 |
| **T5 - Bridge - BERT** | 0.000 | 75.0 | 0.000 | 12.8 | 0.000 | 50.7 | 46.9 | 26.5 |
| **GPT Native** | **5.90** | **88.5** | **83.3** | **71.0** | **80.1** | **75.3** | **58.5** | **66.1** |
| **GPT2 - Bridge - T5** | -1.70 | 50.9 | 81.2 | 7.10 | 0.000 | 49.6 | 52.7 | 34.3 |
| **GPT2 - Bridge - BERT** | -2.10 | 75.1 | 0.000 | 10.9 | 0.000 | 50.4 | 47.3 | 25.9 |
| **GPT2 - BERT (no bridge)** | 0.000 | 49.1 | 81.2 | -0.100 | 46.3 | 50.6 | 52.7 | 40.0 |
| **BERT Native** | **42.7** | **90.8** | **83.3** | **85.5** | **84.4** | **55.3** | **56.3** | **71.2** |
| **BERT - Bridge - T5** | 2.30 | 50.9 | 81.2 | 10.4 | 0.000 | 49.5 | 52.7 | 35.3 |
| **BERT - Bridge - GPT2** | 0.000 | 49.1 | 80.6 | 13.5 | 0.000 | 50.4 | 54.5 | 35.5 |
| **BERT - GPT2 (no bridge)** | 16.9 | 51.3 | 0.000 | -3.00 | 33.0 | 47.7 | 51.6 | 28.2 |

Table 14: GLUE Benchmark results of models using a **foreign Transformer Module** and a Cosine loss optimized bridge. MNLI GLUE Tasks excluded due to not being completed by the time of experimentation. (Baselines in bold, B = includes bridge)

### C.4 Cosine Loss Sentence Comparison

| Model | Pair | Cosine Loss Similar | Cosine Loss Different |
|---|---|---|---|
| BERT | 1 - 2 | 0.150 | 0.551 |
| BERT | 1 - 3 | 0.234 | 0.454 |
| BERT | 2 - 3 | 0.229 | 0.425 |
| GPT-2 | 1 - 2 | 0.00379 | 0.0197 |
| GPT-2 | 1 - 3 | 0.00664 | 0.00609 |
| GPT-2 | 2 - 3 | 0.00364 | 0.00582 |
| T5 | 1 - 2 | 0.0112 | 0.0251 |
| T5 | 1 - 3 | 0.0224 | 0.0234 |
| T5 | 2 - 3 | 0.00498 | 0.0444 |

Table 15: Comparison of Cosine Loss values for similar and different sentences across models. Sentence indices refer to the pre-listed sentence groups for similar and different contexts.

## D Experimental Hardware And Software

| Component | Specification |
|---|---|
| Graphics Processing Unit (GPU) | NVIDIA A40 |
| Processor | AMD EPYC 7543 32-Core Processor |
| Memory | 16GB RAM |
| Software | PyTorch for model implementation and training |

Table 16: Experimental Setup Details