Explainable Soft Prompts

Raphael Reimann and **Frederic Sadrieh**
Hasso-Plattner Institut
{*firstname.lastname*}@student.hpi.de

Abstract

Prompt tuning in natural language processing allows leveraging Large Language Models (LLMs) efficiently. Yet soft prompts struggle with transferability between models and interpretability. This study introduces a method of tuning soft prompts across multiple models to uncover the "token splitting effect". Prompt tokens align with specific models, lying in their embedding spaces and playing a crucial role in their performance. We show how this is hindering transferability, but allowing prompt compression after training. Our research highlights the soft prompt functionality across multiple models and sets a foundation for optimizing soft prompts through strategic compression and a deeper understanding of token importance.[1]

1 Introduction

Traditionally, pre-trained language models have been used to perform a downstream task by fine-tuning the existing language model on the task (Radford et al., 2018). While this method leads to good performance, the approach consumes a lot of computational resources, as each new task requires the fine-tuning of a separate model instance.

Parameter-efficient fine-tuning provides an alternative to this challenge by fine-tuning a selected subset of model weights or introducing and optimizing a smaller number of new weights trained on the downstream task. Most parameters of the model remain unchanged, making the approach more resource-efficient.

Another approach is introduced with in-context learning. Here, no weights are optimized, but the model is prompted with manually crafted input-output pairs. However, it is sensitive to the nuances of prompt design and requires extensive human labor and labeled data to identify the most effective prompts (Liu et al., 2021). To address these challenges, soft prompts, a technique that optimizes prompts in the model's embedding space (Lester et al., 2021), have been introduced. By design, soft prompts require much fewer parameters than traditional fine-tuning methods but have been shown to match the performance of full model fine-tuning.

The exploration of soft prompts raises several research questions that this paper seeks to address. First, we investigate the interpretability of soft prompts when trained on multiple distinct language models (**RQ1**). Second, we investigate the transferability of soft prompts (**RQ2**), by assessing their performance and stability when applied to a distinct, untrained model across a variety of NLP tasks. By answering these research questions, this paper contributes to the broader discourse on optimizing language models in a resource-constrained setting, highlighting the potential of soft prompts as a parameter-efficient approach.

Our main contributions include:

- Identifying the token splitting effect and its implications for soft prompt behavior across multiple models, as well as the challenges it poses for transferability to out-of-distribution models.
- Examining the relationship between a token's position within the prompt and its importance to the model's performance.
- Demonstrating the feasibility of post-training prompt compression, which results in a performance penalty, but provides an approach for further efficiency gains.

2 Related Work

2.1 Prompt Engineering Approaches

Li and Liang (2021) propose prefix-tuning as a simpler alternative to full model fine-tuning. Prefix-tuning freezes the parameters of the language

[1] Our code is available at `https://github.com/FSadrieh/explainable-soft-prompts`

model and trains only a continuous task-specific prefix for any given downstream task. The prefix acts as a sequence of "virtual tokens" (not directly interpretable as real tokens) that the model can attend to. Virtual tokens are not constrained to the embeddings of real words and can take full advantage of the embedding space during optimization. Lester et al. (2021) suggest prompt-tuning as a simplification of prefix tuning. Prompt tuning uses only a single prompt representation (a soft prompt) in the embedding layer, while prefix tuning prepends the prefix at each transformer layer. Prompt tuning also does not require reparameterization of the prefix, which reduces the number of parameters during training. P-tuning (Liu et al., 2023) is another method for automatically searching prompts in the continuous space, which mixes hard and soft prompt tokens.

2.2 Transfer of Soft Prompts

Vu et al. (2022) build on Lester et al. (2021) by proposing a method for soft prompt transfer. First, training a prompt on one or more source tasks, then using the resulting prompt to initialize training for the target task. Qin and Eisner (2021) use prompt transfer to optimize a mixture of prompts on a downstream task. They specifically extract knowledge from pre-trained language models using a learned ensemble of soft prompts. Su et al. (2022) found that soft prompts are effectively transferable to similar tasks on the same pre-trained language model. They train a projector model to transfer a trained soft prompt to another pre-trained language model. In addition, trained soft prompts from similar tasks can be used as initialization, to significantly accelerate training and improve the performance of prompt tuning (Su et al., 2022). Lester et al. (2022) investigate the recycling of trained prompts for new target models without requiring re-training. While they find some improvements with a trained recycler that learns embedding differences between two models, they acknowledge that reusing prompts across models is difficult. Overall the literature does not find that direct prompt transfer is possible and many opt to train an intermediate model to help with the transfer (Su et al., 2022; Lester et al., 2022).

2.3 Interpretability of Soft Prompts

Back translation In their paper Lester et al. (2021) explore the interpretability of soft prompts by mapping the soft tokens to their nearest-neighbor natural language token. While they find that the sequence of learned prompts shows little interpretability, some learned tokens lie within proximity of lexically similar clusters from the training task, suggesting that the prompt may prime the model to interpret input in a specific domain. According to Bailey et al. (2023), soft prompts occupy regions in the embedding space that are distinct from those containing natural language. Direct comparisons between these regions can be misleading. The drawbacks of trying to interpret soft prompts through back translation are also shown in Khashabi et al. (2022). The paper introduces a new loss function that optimizes prompt performance while confining the soft prompt to the neighborhood of a certain hard prompt. This allows for the training of a soft prompt that is always back-translated to the same hard prompt regardless of the task. Khashabi et al. demonstrate this constraints only inflicts a small loss penalty, allowing one to freely control the interpretability of the soft prompt.

Prompt similarity To measure the transferability of soft prompts between tasks, Vu et al. (2022) measure the similarity between the soft prompts trained on those tasks. They find that the similarity of soft prompts correlates with the transferability of soft prompts between the trained on tasks (Vu et al., 2022).

2.4 Multi-Model training

Zou et al. (2023) show the possibility of attacking LLMs with certain hard prompt suffixes. The attack leads to adverse behavior of the assistant (i.e., providing a recipe for bomb building (Zou et al., 2023)). To be able to attack black-box models like GPT4 they use open-source LLMs like Vicuna (Zheng et al., 2024) to train their suffix. They introduce "Greedy Coordinate Gradient" Zou et al. (2023), which greedily replaces hard tokens and tries to minimize the loss. This supervised training of the suffix leads to a well-suited attack on the trained model. To improve performance on black-box models, they use "Universal Prompt Optimization" Zou et al. (2023), which extends token selection across different models. Training across multiple open-source LLMs results in a suffix that performs well on the trained-on and out-of-distribution models. These findings serve as a foundation for our interest in soft prompt multi-model training. The difference between soft prompts and hard prompts is the worse transferability between models for soft prompts (as seen in subsection 2.2).

3 Methods

Soft Prompt Design Soft prompts are a parameter-efficient fine-tuning method that adapts a pre-trained language model with specific tasks without extensive parameter retraining. In this approach, we prepend k d-dimensional vectors to the input, where d is the dimensionality of the embedding space of the pre-trained model. This insertion effectively reduces the available input space, highlighting the importance of selecting a value k that ensures it is large enough to encapsulate the task complexity, yet small enough to maximize the input space. One limitation of soft prompts is their slower convergence time, which makes it appealing to explore transferability as a means to enhance efficiency (Su et al., 2022).

Multi-Model Training Our methodology extends to multi-model training, following principles outlined in previous work (Zou et al., 2023). Specifically, we prepend the same soft prompt to the input for multiple models. We then calculate the average loss across all models after the forward pass and use this aggregated feedback to update the soft prompt. This process assumes uniformity in the embedding dimensions across the selected models, as differences here would make training non-trivial.

Transfer of Soft Prompts In contrast to Vu et al. (2022) and Lester et al. (2022), we transfer the soft prompts without an intermediate model. This is achieved by saving the optimized soft prompt and prepending it to the input of a third model not included in the initial training. Again, this requires compatibility in embedding dimensions across the models involved.

3.1 Interpretability

Back translation Building on existing research (Lester et al., 2021), we calculate the back translation via the nearest discrete neighbor of each soft prompt token. First, the vocabulary of the tokenizer is embedded. Then the distances between the soft prompt tokens and the embedded vocabulary tokens are calculated. Finally, the nearest neighbor is selected. This discrete approximation inherently involves a loss of information (Khashabi et al., 2022), highlighting the challenges of translating continuous optimization spaces back into interpretable language tokens.

Similarity There are several ways to define similarity between soft prompts. Vu et al. (2022) define two similarity metrics:

$$\text{avg\_sim}(p^1, p^2) = \cos\left(\frac{1}{L}\sum_{i=1}^{L} t_i^1, \frac{1}{L}\sum_{i=1}^{L} t_i^2\right)$$

$$\text{per\_token\_sim}(p^1, p^2) = \frac{1}{L^2}\sum_{i=1}^{L}\sum_{j=1}^{L}\cos(t_i^1, t_j^2)$$

Where p^1, p^2 are soft prompts of length L assuming p^1 and p^2 are of equal length. t_i^1, t_i^2 are the tokens at index i of that soft prompt. The avg_sim calculates the similarity between the averaged representation of the soft prompt. The per_token_sim is the per-token similarity between the soft prompts. The second formula introduces a new attribute of soft prompts that we call "inner complexity". This means $\text{per\_token\_sim}(p^1, p^1) \neq 1$ because the soft prompts lie distributed within the embedding space. Through soft prompt pre-averaging, this complexity is lost.

We present a third option to calculate the similarity of soft prompts:

$$\text{pairwise\_sim}(p^1, p^2) = \frac{1}{L}\sum_{i=1}^{L}\cos(t_i^1, t_i^2)$$

It calculates the pairwise similarity between prompt tokens. Here, we try to answer whether the token position is relevant in a soft prompt. Given two soft prompts p^1, p^2, with the same token values but in a shuffled order, the per token similarity will give both soft prompts the same inner complexity and $\text{per\_token\_sim}(p^1, p^2) = \text{per\_token\_sim}(p^1, p^1)$. Pairwise similarity ignores the inner complexity of soft prompts giving us $\text{pairwise\_sim}(p_1, p_1) = 1$, but a different similarity for the soft prompts p^1, p^2.

All of these metrics can be calculated using cosine similarity (Vu et al., 2022) or euclidean distances. This enables 6 different similarity metrics.

Dimensionalty Reduction Soft prompts can be viewed as a list of points in a high-dimensional space. Together with embedding spaces, they can not be interpreted directly by humans. One possible solution to this problem is dimensionality reduction.

With dimensionality reduction, it is possible to reduce the high-dimensional space of soft prompts to a 2-dimensional space. Reducing the embedding spaces of the trained models and the soft prompts can help to visualize where the prompt tokens lie within these embedding spaces.

We use the popular dimensionality reduction methods t-SNE (Van der Maaten and Hinton, 2008) and UMAP (McInnes et al., 2018). These methods suffer computationally when dimensionality is high. Therefore, simpler methods such as PCA (Pearson, 1901) are used to reduce the dimensions to 50. From this reduced state we use either UMAP or t-SNE to reduce it to 2 dimensions.

Prompt token model mapping Another way to explain the relationship between prompt tokens and embedding spaces is a k-nearest neighbor algorithm (Fix and Hodges, 1989). While dimensionality reduction can be evaluated qualitatively and can give humans insight into soft prompts, the quantitative k-nearest neighbor algorithm approach has the scalability to test soft prompts automatically. The k-nearest neighbor algorithm calculates the k nearest neighbors of each soft prompt token in all embedding spaces combined. A majority vote then decides to which embedding space the token is assigned. This is only possible under the assumption that the embedding spaces are somewhat separated and that prompt tokens lie within these spaces.

For our studies, we use both cosine similarity and euclidean distance to calculate the neighbors. We choose $k = 7$.

Token masking An interesting question in interpretability is the importance of each element (e.g., Lundberg and Lee (2017)). We want to answer how important a prompt token is to a model. To evaluate the importance of a token to the performance of a model, we use token masking, by replacing a token, t_i with a zero vector, z, of equivalent dimensionality, and observing the model's output. We define a new token importance metric for a soft prompt token t_i and a model m:

$$\text{importance}(t_i, m) = \text{Loss}_m\left([t_{<i}, z, t_{>i}]; \text{input}\right)$$

With z being a zero vector of the soft prompt dimension. A significant loss indicates the token plays an important role in the model's inference process since the model performs poorly without the token.

To evaluate the importance of each token on each model, must repeat this forward pass for each model and each token. This algorithm has a complexity of $O(M \cdot T)$, where M is the number of models and T is the prompt length.

Prompt masking and compression With the information from token masking, we can use the importance measure to define important and unimportant tokens per model. Masking unimportant tokens reduces the information in the prompt, which is done by setting them to zero vectors as in token masking.

Omitting the unimportant tokens results in prompt compression. Compression allows for more input into the model, by condensing the important soft prompt tokens.

4 Experimental Results

4.1 Research parameters

Model To enable comparable multi-model training we use the MultiBERT collection (Sellam et al., 2022)[2]. These are 25 BERT-base uncased models pre-trained with different seeds on English data, using Masked Language Modeling and Next Sentence Prediction objectives. This enables us to generalize findings beyond the specific model instances, because they differ in embedding spaces and weights, but have the same architecture (Sellam et al., 2022).

Dataset To evaluate our findings we use 3 different classification datasets which serve as our downstream task to adapt pre-trained language models to.

The IMDb movie review dataset (Maas et al., 2011). The dataset contains movie reviews categorized into positive or negative sentiment (binary classification). We have generated a split of 15,000; 5,000; 5,000 (train, validation, test) from the original 25,000 training instances available through huggingface datasets repository (Lhoest et al., 2021)[3].

The emotion dataset (Saravia et al., 2018). The dataset contains English tweets and assigns one of six emotions to each ("sadness, disgust, anger, joy, surprise, and fear") (Saravia et al., 2018). The multi-class nature of the dataset allows us to analyze our methods in a different setting compared to IMDb. We have generated a split of 6,000; 5,000; 5,000 (train, validation, test) from the 16,000 train

[2]We download them from the huggingface transformer libary: For example `https://huggingface.co/google/multiberts-seed_0`

[3]`https://huggingface.co/datasets/imdb`

split of the dataset downloaded from huggingface datasets (Lhoest et al., 2021)[4].

The mnli dataset, which is part of the glue benchmark (Williams et al., 2018). The dataset contains a premise and a hypothesis sentence. We feed them into our Multibert model by concatenating them with a semi-colon (i.e., `premise;hypothesis`). The label shows whether the premise entails the hypothesis, is neutral, or contradicts it (Williams et al., 2018). We have generated a split of 10,000; 5,000; 5,000 (train, validation, test) from a sample of the 393,000 train split of the dataset downloaded from huggingface datasets (Lhoest et al., 2021)[5].

Task and Training setup We used masked language modeling for classification, appending a `[MASK]` token to each example and transforming the label to be only important at that position. As the label, we expect the tokenized version of the class name. We use huggingface transformers (Wolf et al., 2020) to download the MultiBERT models and a PyTorch lightning trainer for model training. Parts of our training and evaluation pipeline are built on top of a template (Dobler et al., 2023), incorporating best practices from previous work into our experimental design.

Hyper-parameter tuning Since performance is not a focus of this research we limited hyper-parameter tuning. After some manual testing, we tuned hyper-parameters (batch size, learning rate, warmup period, weight decay, beta1, beta2) on the IMDb dataset in 10 runs with a bayesian sweep. For the other datasets, we ran 10 runs with a bayesian sweep to only change the learning rate and the batch size. All final hyper-parameter configurations can be found in Table 7.[6] Untuned hyperparameters include the use of bfloat 16 mixed precision, a cosine learning rate scheduler, gradient clipping (Pascanu et al., 2013) and the AdamW optimizer (Loshchilov and Hutter, 2019).

Soft Prompt Following Lester et al. (2021) we prepend 16 tunable tokens to the embedded input and feed it into the model bypassing the embedding layer. The 16 768-dimensional vectors (MultiBERTs embedding dimension) are seeded and randomly initialized. In addition, we seed the training.

[4]https://huggingface.co/datasets/dair-ai/emotion

[5]https://huggingface.co/datasets/glue/viewer/mnli

[6]The configuration for each used soft prompt can be found in https://github.com/FSadrieh/explainable-soft-prompts/tree/main/cfgs

#Model	IMDb	emotion	mnli
1	**0.360** ± 0.093	**0.494** ± 0.121	1.098 ± 0.014
2	**0.332** ± 0.035	**0.506** ± 0.015	**1.086** ± 0.005
5	**0.325** ± 0.022	**0.735** ± 0.003	1.108 ± 0.009
10	**0.353** ± 0.010	0.886 ± 0.007	1.134 ± 0.011

Table 1: The soft prompt test loss on the trained on models based on the amount of models the soft prompt was trained on. Even though the prompt length remains the same, while increasing the model size, no to little significant performance drops are shown when training on multiple models.

Different trained Soft Prompts To gain an understanding of multi-model training and the effects on soft prompts, we train soft prompts in 1, 2, 5, and 10 model configurations on each dataset. All of these configurations maintain the same initialization, training seed, and hyper-parameters while varying the model seeds (A different selection of the 25 MultiBERT models). To gain insights into the impact of initialization and training seeds we train soft prompts on 5 model configurations with different training and initialization seeds while keeping the model seeds fixed. This leaves us with 6 soft prompt categories: `1-model`, `2-model`, `5-model`, `5-model-init`, `5-model-train`, and `10-model`. The category `1-model` always has 25 soft prompts, one per model. For IMDb, we trained 5 soft prompts for each of the other categories, for the other datasets we trained 3 soft prompts per category. Within the categories `5-model`, `5-model-init`, `5-model-train` the first trained soft prompt is the same.

Evaluation All reported results are on the test set. We report the average across multiple runs within each soft prompt category. In addition, we report the standard deviation. Bold are the best results and those that are within 2 standard deviations of the best (following Raffel et al. (2020)). We round all reported values to 3 decimal points.

4.2 Multi-model training

Table 1 shows the difference in test loss between a single-model and a multi-model configuration is small. Even though each multi-model soft prompt has a smaller amount of tokens per model.

4.3 Soft Prompt Transfer

To evaluate the transferability of soft prompts, we transfer the soft prompts to each of the 25 MultiBERT models. As an example see Table 8.

Transfer to trained-on models performs well (in addition seen in Table 1), but the transfer to out-of-distribution models yields bad test losses. We conclude that soft prompt transfer does not work with our implementation of multi-model training.

4.4 Interpretability

Back translation Back translation generally does not produce human-readable output. One example of a hard prompt produced with this method can be found in Table 9. Even though the token "positive" is learned, which is a label for the IMDb dataset, the output is not interpretable and the method is not robust enough to allow for interpretability. Some soft prompts could have a more readable back translation, but an interpretability method should provide robust and easy to interpret results. With these results, we have confirmed previous literature on the problems with interpretability of back translation (Khashabi et al., 2022).

Similarity Table 2 shows the similarity between one soft prompt trained on MultiBERT 0, one trained on MultiBERT 1, and one trained on both. The cosine similarities are generally poor. The avg_sim seems to indicate a high similarity between **0** and **0&1**, which is not reflected by the other two cosine metrics.

Further, the euclidean per-token distance metric shows a high inner complexity, with only a marginally higher distance between the prompts. This raises the question of whether the soft prompts are nearly equally distributed in the vector space. The pairwise distance seems to disagree and put the soft prompt equally far away but without an inner complexity. The average distance is much smaller than all other distances.

The variability of these measures means they are not usable for interpretation. Effective interpretability metrics need clarity and robustness, qualities that are lacking with our current metrics.

4.5 Token splitting effect

Dimensionality Reduction We apply dimensionality reduction techniques to the soft prompts and their corresponding model embedding spaces. Specifically, we use t-SNE from scikit-learn (Pedregosa et al., 2011), with random state 1 and cosine metric. For UMAP we choose the umap library (McInnes et al., 2018), with n_neighbors 50 and cosine metric. We manually evaluate some visualizations of dimensionality reductions of soft prompts and the embedding spaces of the trained-on models. In Figure 1 a dimensionality reduction using t-SNE for `5_model_0` is shown.

	Euclidean avg dist			Cosine avg sim		
	0	**1**	**0 & 1**	**0**	**1**	**0 & 1**
0	0.0	5.894	3.657	1.0	0.012	0.636
1	5.894	0.0	4.948	0.012	1.0	0.16
0 & 1	3.657	4.948	0.0	0.636	0.16	1.0

	Euclidean pairwise dist			Cosine pairwise sim		
	0	**1**	**0 & 1**	**0**	**1**	**0 & 1**
0	0.0	14.466	13.415	1.0	0.0	0.088
1	14.466	0.0	13.43	0.0	1.0	0.043
0 & 1	13.415	13.43	0.0	0.088	0.043	1.0

	Euclidean per token dist			Cosine per token sim		
	0	**1**	**0 & 1**	**0**	**1**	**0 & 1**
0	12.836	14.446	13.223	0.19	0.003	0.11
1	14.446	12.671	13.56	0.003	0.143	0.025
0 & 1	13.223	13.56	11.751	0.11	0.025	0.164

Table 2: All six similarity metrics (see section 3.1) for the comparison between one soft prompt trained on MultiBERT 0, one trained on MultiBERT 1, and one trained on both.

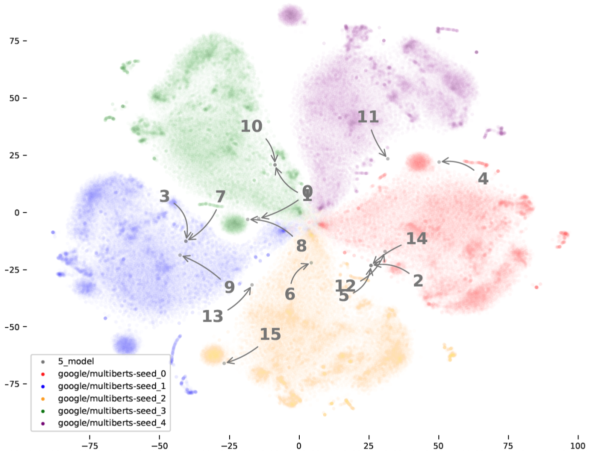

Figure 1: A dimensionality reduction of the embedding spaces of MultiBERTs $0, 1, 2, 3, 4$ and the prompt tokens of `5_model_0`. The dimensionality reduction was done using PCA and t-SNE.

Across all evaluated visualizations of dimensionality reductions, a clear separation of embedding spaces is visible. In addition, soft prompt tokens lie within the embedding spaces and are roughly

#Model	IMDb	emotion	mnli
2	0.631 ± 0.237	0.75 ± 0.042	1.145 ± 0.017
5	2.065 ± 0.684	1.384 ± 0.05	1.251 ± 0.032
10	**4.777** ± 0.387	**2.543** ± 0.006	**1.68** ± 0.089

Table 3: The test loss the important tokens achieve, when masked, grouped after the number of trained-on models and datasets. The more models share a prompt, with the same prompt length, the higher the importance of the individual tokens.

equally distributed between them. This is supported by Figure 2, showing a soft prompt lying only within the embedding spaces of trained-on models, not out-of-distribution embedding spaces. To find out if a token's spatial location within a model's embedding space is an indication of its significance to the model, we use token masking, selectively disabling individual soft prompt tokens to assess their impact on model performance.

Token masking Dropping out individual tokens in a soft prompt shows the importance of the token for a model (section 3.1). In Table 10 we provide an example token importance matrix. Each model relies on a distinct subset of tokens that are important, which is shown by a considerable increase in loss when these tokens are masked. Interestingly, for each token only one model is important and for each model roughly the same number of tokens are important. The observed effect is strong since the difference between the worst loss for each token and the rest is high. We also find that unimportant tokens, when masked, had a negligible effect on loss, confirming the existence of model-specific token dependencies.

Scaling the token splitting effect As defined in section 3.1 an increased loss leads to higher importance, we report the average loss for the important tokens across all runs Table 3.

While increasing the model count, but keeping the prompt length the same (16), the token-splitting effect gets stronger. If fewer tokens are available each token has to be more important for the model.

Token location vs token performance To evaluate if the important tokens for a model lie within the embedding space of that model, we report the alignment between the token importance, identified through token masking, and the token position, determined via the k-nearest neighbor vote. We report the alignment, using cosine similarity and euclidean distance as the metric for determining the nearest neighbors, over all datasets Table 13.

Our alignment analysis shows modest values. Yet when viewed as a classification task to match tokens to their respective models our method exceeds the random baseline significantly, particularly as the number of models increased. The alignment between a token's position and its significance to the model's performance increases as the token splitting effect increases.

	IMDb	
#Model	masked	shortend
2	**2.162** ± 1.756	2.821 ± 2.041
5	**1.1** ± 0.523	**1.416** ± 0.714
10	**1.972** ± 0.399	4.405 ± 0.861
	emotion	
#Model	masked	shortend
2	**2.393** ± 0.603	**2.237** ± 1.287
5	5.425 ± 0.512	5.350 ± 2.482
10	6.826 ± 0.341	6.830 ± 2.704
	mnli	
#Model	masked	shortend
2	**6.302** ± 0.883	**6.597** ± 4.603
5	**7.88** ± 0.266	**8.056** ± 3.992
10	10.15 ± 0.429	10.938 ± 2.976

Table 4: Test loss of the soft prompts, when the unimportant tokens are masked or removed. Masking inflicts a performance drop, which is quite high depending on the dataset. Prompt compression performs similarly to prompt masking.

Prompt masking and compression Token masking reveals that there are only a few important tokens per model, allowing us to mask or remove unimportant tokens. Table 4 shows that while masking or compressing unimportant tokens increases the model loss, masked soft prompts perform better than untrained soft prompts (See performance from untrained prompts in Table 8).

As a comparison, we train a soft prompt with a prompt length of 2 on each MultiBERT model with the IMDb dataset. They can be compared to the IMDb `10_model` soft prompts from Table 4 since they have roughly 2 unmasked tokens[7]. The shorter single model prompts achieve an average

[7]If we use a prompt length of 16, train on 10 models, and assume equal distribution of tokens each model gets 1.6 tokens.

test loss of **0.43** $\pm$ 0.085, which is significantly better than the 1.972 $\pm$ 0.399 achieved by the masked `10_model` soft prompts.

Still the masking heuristic performs better than a random masking. We randomly mask all except 2 tokens for the IMDb `10_model` soft prompts and achieve a test loss of 9.598 $\pm$ 2.865 averaged over 3 different seeds, which is significantly worse than masking with our heuristic.

However, this approach is not without variability. Some soft prompts exhibit model-specific performance differences, likely due to unequal token allocation. As an example see Table 14. For some experiments, one model had a favorable token assignment which leads to good performance, while an important token was taken from another model. Prompt masking and compression are feasible, yet the process of selecting which tokens to compress is complex and requires careful consideration.

Future efforts could allow tokens to be shared across models, which, despite marginally extending prompt lengths, could result in an improved token distribution and thus better outcomes.

Effects on transferability The transferability of soft prompts, as shown in subsection 4.3, is hindered by the model-specific nature of the token splitting effect. This effect explains the inability of soft prompts to generalize across out-of-distribution models, which will have none of the crucial tokens within their embedding spaces (see Figure 2). Consequently, soft prompts trained on multiple models do not display generalizability but rather specialize within the embedding spaces of the trained-on models. This specialization hinders their application to models beyond the original training distribution, which diverges from the generalization capabilities shown in previous research like Zou et al. (2023).

4.6 Factors influencing the token splitting effect

Influence of training and initialization seed The training and initialization seed does not significantly impact the token importance or the alignment within models (see Table 5). The results are similar to the `5_model` results. The seeds do have a substantial impact on the token assignment. When comparing token masking assignments from the same models with different training and initialization seeds (see Table 10; with different training seed Table 12; and with different initialization seed Table 11) the tokens are assigned to different models and the soft prompts perform differently the models. There seems to be no predictability of which tokens get chosen from which model.

Dimension	IMDb	emotion	mnli
Models	**2.065** $\pm$ 0.684	**1.384** $\pm$ 0.05	1.251 $\pm$ 0.032
Train seed	**1.605** $\pm$ 0.192	**1.438** $\pm$ 0.071	**1.289** $\pm$ 0.011
Init seed	**1.645** $\pm$ 0.254	**1.574** $\pm$ 0.306	**1.274** $\pm$ 0.02

Table 5: The test loss the important tokens achieve, when masked, categorized after soft prompt category and dataset. Choosing different models, different initializations, or training seeds, has no impact on the strength of the token-splitting effect.

#Epochs	Normal loss	Important token loss
2	1.153 $\pm$ 0.015	**1.298** $\pm$ 0.039
5	1.108 $\pm$ 0.009	1.251 $\pm$ 0.032
10	1.078 $\pm$ 0.004	1.259 $\pm$ 0.011
20	**1.052** $\pm$ 0.005	**1.363** $\pm$ 0.048

Table 6: The test loss the important tokens achieve, when masked, grouped after a number of epochs. These are only the soft prompts trained on mnli. The more epochs a model trains the better its performance and the higher the important token loss.

Influence of training duration The soft prompts perform worst on the mnli dataset. We evaluate if this is due to too little training by training the soft prompt for 2, 5, 10, and 20 epochs (see Table 6). Extended training periods result in a notable reduction in loss across models and an increase in the significance of important tokes. It should be noted, that the importance of the soft prompts trained on 2 models is relatively high due to a higher baseline loss in these configurations, which is a limitation of our current importance metric as discussed in section 5. We find that the token-splitting effect is stronger with increased training.

5 Limitations

Our work is limited by the exclusive use of the MultiBERT models for our experiments. This homogeneity in architecture and size means our findings regarding the token splitting effect and the overall performance of soft prompts may not extend to models with different architecture or those significantly larger or smaller in scale.

Our results are further limited by the focus on three datasets. The diversity of natural language processing tasks suggests that our insights may not apply across different data domains or task types. This shows the importance of extending future research to validate this on a broader range of datasets and tasks.

To reduce the carbon footprint of our research we limited ourselves to testing only on 3 to 5 different randomness configurations per soft prompt class. Increasing the number of runs would reduce the influence of the seeds and allow for more general statements.

We did not investigate the balance between prompt length and available input space. While our experiments used a fixed prompt length, task-specific characteristics might need adjustments of this parameter, which should be explored further.

The importance metric as defined in section 3.1 is simple but has flaws. One example can be seen in Table 11, where model 0 achieves a loss of 0.348 when token 9 is masked. This is even slightly better than the performance without token masking, still, the token is assigned to the model since the initial loss of that model is much higher than the rest. The metric only works if the soft prompt has a similar loss on all models. One possible change of this metric would be to include the performance of the soft prompt on the model without masking:

$$\text{importance}(t_i, m) = \frac{\text{Loss}_m\left([t_{<i}, z, t_{>i}]; \text{input}\right)}{\text{Loss}_m\left(p; \text{input}\right)}$$

Where p is the prompt, t_i the prompt token at index i, m the model, and z a zero vector of the soft prompt dimension.

6 Conclusion

In our research, we extend the discourse on the characteristics of soft prompts within the context of pre-trained language models. Based on a series of experiments we analyze the dynamics of soft prompts and their relation with the embedding spaces of pre-trained language models.

6.1 Main Findings

We have confirmed previous literature in the lack of interpretability given by back translation and shown the difficulties in relying on similarity metrics.

The main contribution of this work is the identification of the Token Splitting Effect. Specific tokens hold critical importance across different models and lie within their respective embedding spaces. The models "split up" the tokens roughly equally and if the important tokens are masked, performance decreases rapidly. The effect is stronger when increasing the number of models involved in training while maintaining a constant prompt length, especially among models that show strong task performance. Changes in the training and initialization seed result in similar strengths of the effect, but a different token distribution.

We also confirm the challenges in transferring soft prompts across models as laid out in previous literature. The soft prompt does not learn general behavior through multi-model training, but specialized behavior on the trained-on models. Prompt masking inflicts performance drops and performs worse than training smaller prompts, but it works better than random masking. Prompt compression performs similarly to token masking but results in shorter prompts.

6.2 Future Work

Future work should broaden the evaluation of the Token Splitting Effect to include a wider range of model architectures. This would help determine whether the effect is consistent across different models and identify any potential limitations. Additionally, exploring how changing prompt lengths according to the number of models affects this phenomenon could provide insights into optimizing soft prompts more effectively.

The use of soft prompts in creating adversarial scenarios also presents an interesting area for further research. Building on studies such as Khashabi et al. (2022) and Bailey et al. (2023), it would be valuable to examine if the Token Splitting Effect can be utilized to generate tokens that improve model performance in specific contexts while simultaneously presenting challenges or lead to adversarial behavior in others.

Additionally, the concept of Prompt Mixing, which involves incorporating key tokens from one model's soft prompt into another, can be further explored. This strategy could potentially improve model performance by using the strengths of various prompts. Testing the viability of this approach could open up new approaches for using soft prompts more flexibly and effectively in language models.

References

Luke Bailey, Gustaf Ahdritz, Anat Kleiman, Siddharth Swaroop, Finale Doshi-Velez, and Weiwei Pan. 2023. Soft prompting might be a bug, not a feature. In *Workshop on Challenges in Deployable Generative AI at International Conference on Machine Learning (ICML)*, Honolulu, Hawaii.

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, NAACL-HLT 2019, minneapolis, MN, USA, june 2-7, 2019, volume 1 (long and short papers)*, pages 4171–4186. Association for Computational Linguistics.

Konstantin Dobler, Maximilian Schall, and Oliver Zimmermann. 2023. nlp-research-template.

Evelyn Fix and Joseph Lawson Hodges. 1989. Discriminatory analysis. Nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247. Publisher: JSTOR.

Daniel Khashabi, Xinxi Lyu, Sewon Min, Lianhui Qin, Kyle Richardson, Sean Welleck, Hannaneh Hajishirzi, Tushar Khot, Ashish Sabharwal, Sameer Singh, and Yejin Choi. 2022. Prompt Waywardness: The Curious Case of Discretized Interpretation of Continuous Prompts. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3631–3643, Seattle, United States. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Brian Lester, Joshua Yurtsever, Siamak Shakeri, and Noah Constant. 2022. Reducing Retraining by Recycling Parameter-Efficient Prompts. Publisher: arXiv Version Number: 1.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 conference on empirical methods in natural language processing: System demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. ArXiv: 2109.02846 [cs.CL].

Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ArXiv:2107.13586 [cs].

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

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. ArXiv:1711.05101 [cs, math].

Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in neural information processing systems 30*, pages 4765–4774. Curran Associates, Inc.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr.

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

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Guanghui Qin and Jason Eisner. 2021. Learning How to Ask: Querying LMs with Mixtures of Soft Prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212, Online. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskeve. 2018. Improving Language Understanding by Generative Pre-Training.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen. 2018. CARER: Contextualized affect representations for emotion recognition. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 3687–3697, Brussels, Belgium. Association for Computational Linguistics.

Thibault Sellam, Steve Yadlowsky, Ian Tenney, Jason Wei, Naomi Saphra, Alexander Nicholas D'Amour, Tal Linzen, Jasmijn Bastings, Iulia Raluca Turc, Jacob Eisenstein, Dipanjan Das, and Ellie Pavlick. 2022. The MultiBERTs: BERT Reproductions for Robustness Analysis. Publication Title: ICLR 2022.

Yusheng Su, Xiaozhi Wang, Yujia Qin, Chi-Min Chan, Yankai Lin, Huadong Wang, Kaiyue Wen, Zhiyuan Liu, Peng Li, Juanzi Li, Lei Hou, Maosong Sun, and Jie Zhou. 2022. On Transferability of Prompt Tuning for Natural Language Processing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3949–3969, Seattle, United States. Association for Computational Linguistics.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou', and Daniel Cer. 2022. SPoT: Better Frozen Model Adaptation through Soft Prompt Transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5039–5059, Dublin, Ireland. Association for Computational Linguistics.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

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. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: System demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2024. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems*, 36.

Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. Publisher: arXiv Version Number: 1.

Parameter	IMDb	emotion	mnli
batch size	168	180	304
learning rate	0.811	0.128	0.266
warmup period	0.04	0.04	0.04
weight decay	0.173	0.173	0.173
beta1	0.803	0.803	0.803
beta2	0.939	0.939	0.938

Table 7: The tuned hyper-parameters per dataset. Note only batch size and learning rate are tuned per model.

A Computational budget

We use the 25 MultiBERT models (Sellam et al., 2022) which are trained based on the architecture of BERT Base and thus have 110 million parameters (Devlin et al., 2019). The soft prompts have a prompt length of 16 with an embedding dimension of 768, resulting in 12,288 tunable parameters.

Most training runs and all evaluations are performed on an NVIDIA GeForce RTX 3090 with one GPU and 24 CPUs. We logged about 5 days of training runs with wandb on this node. We trained for 5 hours on an NVIDIA RTX A6000 with one GPU and 24 CPUs. An evaluation requires about 20 seconds of testing (or validation since the dataset sizes are the same) on the RTX 3090. For the token splitting effect, we need one test epoch per model per prompt token. This results in 160 test epochs for every 10 model configuration and a run time of about one hour. We estimate to have spent around 3 days of compute for the evaluation. This results in 8 compute days on the RTX 3090 and 5 hours on the NVIDIA RTX A6000.

B Tables

Seed	initial soft prompt	trained soft prompt
0	11.77	**0.311**
1	10.142	**0.307**
2	10.129	**0.309**
3	9.847	**0.313**
4	10.484	**0.299**
5	11.567	11.47
6	10.628	10.624
7	9.812	9.694
8	11.337	11.378
9	11.970	11.959
10	10.877	10.836
11	11.701	11.587
12	10.35	10.364
13	10.305	10.128
14	11.046	11.111
15	10.650	10.557
16	9.784	9.821
17	11.257	11.596
18	11.03	11.032
19	11.301	11.312
20	10.857	10.836
21	11.150	10.79
22	11.942	12.333
23	12.219	12.041
24	9.991	10.034

Table 8: The test loss of the soft prompt `5_multibert_0` on the different MultiBERT models (distinguished by the seed id). We report the loss of the random initialization and the trained version of `5_multibert_0`. The soft prompt is trained with the IMDb dataset on the models $0, 1, 2, 3, 4$. The soft prompt achieves far better performance on the trained-on models. On out-of-distribution models, one can not distinguish between the initial and trained performance of the soft prompt.

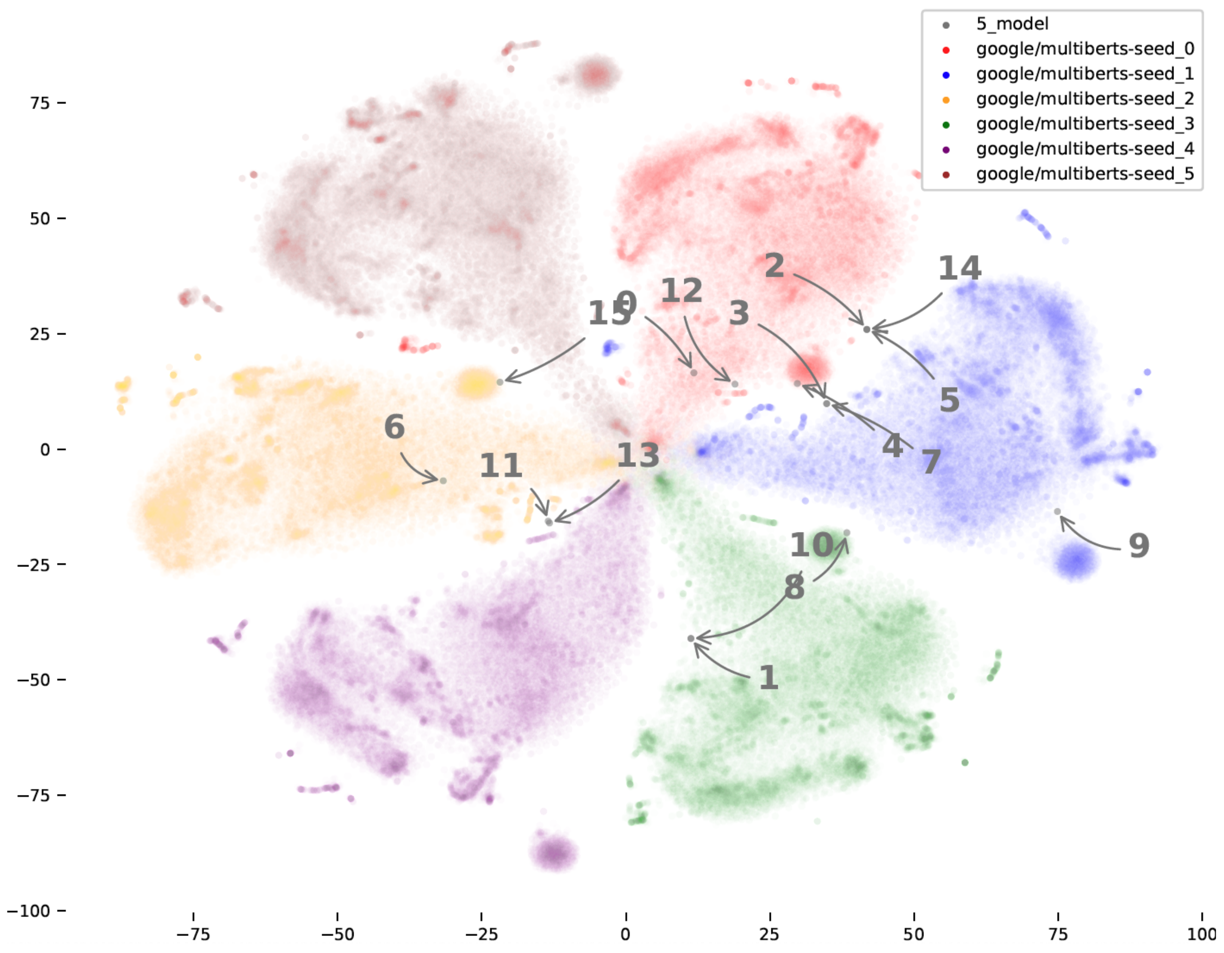

Figure 2: A dimensionality reduction of the embedding spaces of MultiBERTs $0, 1, 2, 3, 4, 5$ and the prompt tokens of `5_model_0`. Note that `5_model_0` was only trained on the first five models while MultiBERT 5 is an out-of-distribution model. The dimensionality reduction was done using PCA and t-SNE.

Token	model 0	model 1	model 2	model 3	model 4
t_0	belinda	did	[unused66]	[unused138]	you
t_1	[nd]	church	settling	[unused680]	es
t_2	positive	1805	taken	room	[nd]
t_3	nuremberg	[nd]	sight	appearances	barrister
t_4	[unused787]	[unused186]	chambers	must	our
t_5	[unused942]	most	[nd]	[unused261]	lan
t_6	sunday	formation	bacterium	[nd]	seafood
t_7	wastewater	towards	ussr	disgrace	[nd]
t_8	[unused397]	1989	marketed	[unused486]	nigel
t_9	specializing	[SEP]	guarantee	deck	without
t_{10}	itunes	jealous	cod	®	regards
t_{11}	periodical	triple	[nd]	steele	[nd]
t_{12}	slippery	criterion	hero	man	suspects
t_{13}	[unused201]	sit	enough	robin	[unused209]
t_{14}	nguyen	stages	simon	3	whitman
t_{15}	plateau	tell	[unused785]	hat	marshall

Table 9: The back translation fro soft prompt `5_multibert_0` (trained on IMDb and MulitBERTs $0, 1, 2, 3, 4$). [nd] is left for tokens not displayable. The results are not humanly readable and can not be used for interpretation.

Masked token	model 0	model 1	model 2	model 3	model 4
t_0	0.314	**1.064**	0.316	0.315	0.305
t_1	0.322	0.309	0.312	**0.651**	0.302
t_2	0.308	0.306	0.308	0.316	**1.272**
t_3	0.307	**1.779**	0.309	0.315	0.301
t_4	**1.975**	0.307	0.310	0.315	0.299
t_5	**0.342**	0.307	0.310	0.316	0.300
t_6	0.305	0.307	**2.732**	0.313	0.299
t_7	0.306	**1.891**	0.309	0.313	0.300
t_8	0.312	0.307	0.310	**0.876**	0.300
t_9	0.305	**5.025**	0.310	0.317	0.300
t_{10}	0.304	0.306	0.309	**3.715**	0.300
t_{11}	0.308	0.305	0.310	0.311	**3.568**
t_{12}	0.305	0.307	0.310	0.311	**0.369**
t_{13}	0.311	0.305	**0.369**	0.313	**0.362**
t_{14}	**0.388**	0.305	0.310	0.313	0.301
t_{15}	0.304	0.305	**3.310**	0.315	0.301

Table 10: The test loss for the soft prompt `5_multibert_0`, when each token is masked separately and evaluated on each model. It is clear to which model each token belongs since the test loss (importance) is high for only one model.

Masked token	model 0	model 1	model 2	model 3	model 4
t_0	**1.173**	0.317	0.305	0.297	0.303
t_1	0.370	**3.172**	0.300	0.296	0.295
t_2	0.347	**0.715**	0.363	0.295	0.296
t_3	0.347	0.311	0.299	0.298	**2.026**
t_4	0.340	0.311	**1.594**	0.526	0.296
t_5	0.344	0.310	0.308	**0.419**	0.295
t_6	**0.374**	0.310	0.300	0.300	0.296
t_7	**0.393**	0.313	0.300	0.299	0.296
t_8	0.344	0.311	**1.977**	0.298	0.296
t_9	**0.348**	0.312	0.300	0.299	0.327
t_{10}	0.355	**0.715**	0.302	0.299	0.332
t_{11}	0.352	0.311	0.303	0.298	**1.881**
t_{12}	0.348	0.311	0.303	**3.467**	0.296
t_{13}	0.338	0.312	**0.390**	0.301	0.296
t_{14}	0.354	0.310	**0.357**	0.300	0.340
t_{15}	0.343	**1.267**	0.300	0.298	0.295
None	0.35	0.31	0.299	0.297	0.297

Table 11: The test loss for the soft prompt `5_multibert_init_1`, when each token is masked separately and evaluated on each model. It is clear to which model each token belongs since the test loss (importance) is high for only one model.

Masked token	model 0	model 1	model 2	model 3	model 4
t_0	0.306	0.321	**3.216**	0.287	0.308
t_1	0.311	0.314	0.325	**3.296**	0.301
t_2	0.302	0.309	0.316	0.283	**0.434**
t_3	0.296	**1.819**	0.316	0.324	0.300
t_4	0.294	0.308	0.319	0.284	**1.690**
t_5	0.295	0.309	**4.638**	0.284	0.300
t_6	0.292	**2.290**	0.318	0.284	0.299
t_7	0.301	**1.931**	0.318	0.284	0.299
t_8	0.296	0.308	0.317	0.309	**0.379**
t_9	**0.341**	0.308	0.317	0.284	0.299
t_{10}	0.299	0.309	0.339	**0.972**	0.378
t_{11}	0.309	0.309	0.319	**0.646**	0.300
t_{12}	0.300	0.309	0.320	0.284	**5.632**
t_{13}	**0.319**	0.308	0.318	0.284	0.300
t_{14}	**0.483**	0.308	0.319	0.285	0.299
t_{15}	0.311	0.307	**0.321**	0.284	0.300
None	0.298	0.307	0.315	0.284	0.3

Table 12: The test loss for the soft prompt `5_multibert_train_1`, when each token is masked separately and evaluated on each model. It is clear to which model each token belongs since the test loss (importance) is high for only one model.

		IMDb		emotion		mnli	
#Model	random	cos	euc	cos	euc	cos	euc
2	0.5	0.575 $\pm$ 0.16	0.55 $\pm$ 0.17	**0.625** $\pm$ 0.051	0.625 $\pm$ 0.088	0.625 $\pm$ 0.184	0.604 $\pm$ 0.193
5	0.2	**0.713** $\pm$ 0.064	**0.725** $\pm$ 0.075	0.375 $\pm$ 0.135	0.438 $\pm$ 0.135	0.354 $\pm$ 0.078	0.333 $\pm$ 0.078
10	0.1	**0.65** $\pm$ 0.135	**0.638** $\pm$ 0.127	**0.417** $\pm$ 0.029	**0.417** $\pm$ 0.029	**0.271** $\pm$ 0.029	0.188 $\pm$ 0.051

Table 13: Alignment of the token position and performance based on the nearest neighbor vote and token masking. The random column is the score a random classifier would achieve when trying to predict to which model a token should belong. Bold are all results, which beat the random classifier significantly.

Soft prompt	model 0	model 1	model 2	model 3	model 4
`5_model_0`	0.317	0.332	1.736	0.33	0.944
`5_model_1`	4.587	0.329	0.349	0.320	6.095
`5_model_2`	0.372	0.362	3.107	0.321	0.614
`5_model_3`	1.583	0.433	4.867	3.958	0.336
`5_model_4`	0.452	0.472	0.648	2.019	0.513

Table 14: The test loss for prompt compression for the 5 model configurations trained on IMDb. Note they are all trained on different models, which means model 0 is not equal to MultiBERT seed 0, but rather the first model they were trained on. We see a high fluctuation in the test losses. Sometimes the losses are similar to single model training, but sometimes much worse.