Task Prompt Vectors: Effective Initialization through Multi-Task Soft-Prompt Transfer

Anonymous ACL submission

Abstract

Prompt tuning is an efficient solution for training large language models (LLMs). However, current soft-prompt-based methods often sacrifice *multi-task modularity*, requiring the training process to be fully or partially repeated for each newly added task. While recent work on *task vectors* applied arithmetic operations on full model weights to achieve the desired multi-task performance, a similar approach for soft-prompts is still missing. To this end, we introduce **Task Prompt Vectors**, created by element-wise difference between weights of tuned soft-prompts and their random initialization. Experimental results on 12 NLU datasets show that task prompt vectors can be used in low-resource settings to effectively initialize prompt tuning on similar tasks. In addition, we show that task prompt vectors are independent of the random initialization of prompt tuning on 2 different language model architectures. This allows prompt arithmetics with the pre-trained vectors from different tasks. In this way, we provide a competitive alternative to state-of-the-art baselines by arithmetic addition of task prompt vectors from multiple tasks.

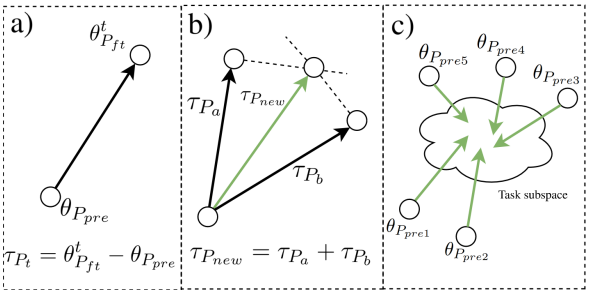

Figure 1: An illustration of task prompt vector and the combination via addition that we include in our work. (a) A task prompt vector is created by subtracting the soft-prompt initialization weights $\theta_{P_{pre}}$ from the soft-prompt weights after prompt tuning $\theta^t_{P_{ft}}$ (Section 3, eq. 2). (b) A combination via the addition of two task prompt τ_{P_a} and τ_{P_b} resulting in $\tau_{P_{new}}$ (Section 3, eq. 4). (c) Different task prompt vectors point into the same sub-space in the embedding space of PLM (Section 4.2). The circles represent different random initializations.

1 Introduction

Standard fine-tuning methods change the weights of a pre-trained language model (PLM) to increase its performance on a downstream task. As there is a trend of improving the overall results by increasing the number of parameters, the models require a vast amount of computational resources for training (e.g., GPT-3 (Brown et al., 2020) having 175 billion parameters). Besides their parameter hunger, large language models also require significant amounts of training data, which especially benefits well-resourced languages.

To address the problem of the increasing number of parameters, *Parameter-Efficient Fine-Tuning (PEFT)* methods (Lester et al., 2021; Houlsby et al., 2019; Hu et al., 2022) were introduced, capable of solving multiple problems even with small amounts of labeled data while training only a fraction of the model parameters (e.g., for RoBERTa base (Liu et al., 2019), prompt tuning (Lester et al., 2021) is training only 0.5% parameters, and LoRA (Hu et al., 2022) is training only 0.7% of parameters (Xu et al., 2023)). The key concept that makes such methods effective is their *task modularity*.

Some of the recent PEFT (Xu et al., 2023; Lester et al., 2021; Asai et al., 2022) methods focus on fine-tuning *soft-prompts*. Soft-prompts are trainable (parametrized) weights that are prepended to the input embeddings while training the model. *Prompt tuning* is one such efficient solution for soft-prompt-based tuning of large language models (LLMs).

The most of the soft-prompt-based methods lack sufficient multi-task modularity, requiring the training process to be fully or partially repeated for each newly added task (Vu et al., 2022; Wang et al., 2023). Other methods while keeping their relatively high modularity usually lack robustness and

their performance depends on the quality and the number of pre-trained soft-prompts (Asai et al., 2022). Moreover, creating a soft-prompt for multiple tasks may often reduce the overall multi-task performance and require further fine-tuning. Building upon the findings from *task vector arithmetics* (Ilharco et al., 2022), we utilize the efficiency and modularity of prompt tuning (Lester et al., 2021) to create **Task Prompt Vectors**. We thoroughly investigate the properties of task prompt vectors and demonstrate their functionality in combining pairs of task prompt vectors while evaluating their in-distribution performance and out-of-distribution performance in full and limited data scenarios.

Our main contributions and findings are [1]:

- We introduce the novel concept of **task prompt vectors** created from fine-tuned soft-prompts, as a method of weight interpolation that leverages findings from task vectors. In addition, we investigate vector arithmetics on such task prompt vectors, based on simple arithmetic operations as a method to reinforce PLMs to solve multi-task problems.

- We provide a comprehensive investigation of task prompt vector properties on 12 NLU datasets separated into 3 task types and demonstrate important properties of task prompt vectors. We show that their random initialization independence makes them robust and universally applicable, while their similarity across related problems provides a necessary base for efficient cross-task transfer.

- We show that task prompt vectors allow efficient prompt tuning initializations, by leveraging multi-task combinations of the pre-trained task prompt vectors using the task prompt vector arithmetics. Experimental results show that especially in zero- or few-shot settings, task-prompt-vector-based initialization can outperform or match SPoT (Soft-Prompt Transfer learning, Vu et al. (2022)) for specific tasks while maintaining high multi-task modularity.

[1]To support the replicability of our work, we provide a repository where we store all of our implementation and results: https://anonymous.4open.science/r/task-prompt-vectors-58CB

2 Related Work

Soft-prompt-based fine-tuning. After the introduction of prompt tuning (Lester et al., 2021) and prefix tuning (Li and Liang, 2021) many new soft-prompt-based methods (Gu et al., 2022; Liu et al., 2023; Shi and Lipani, 2024) were introduced. Some of these methods focus on task knowledge transfer (e.g., SPoT (Vu et al., 2022) or cross-model transfer (Su et al., 2022)) and task combinations (e.g., ATTEMPT (Asai et al., 2022), MPT (Wang et al., 2023), or BMTPT (Lee et al., 2023)). These can be classified as works on PEFT weight interpolations to increase the performance of prompt tuning in single or multi-task settings. However, they do not represent the tasks as vectors in the embedding space and usually require further training of the added parameters.

Model weights interpolation. Model weight interpolation (Frankle et al., 2020; Wortsman et al., 2022) is a widely discussed topic in the literature since it enables combining knowledge of different fine-tuned models without or with a small amount of training. Authors of tasks vectors (Ilharco et al., 2022) show, that it is possible to combine multiple task vectors created from fine-tuned models and still maintain the overall multi-task performance. Ortiz-Jimenez et al. (2024) focuses mostly on improving the work on task vectors, by showing that training models in their tangent space contributes to the weight disentanglement and increases the performance of full model task arithmetic. Another subcategory for weight interpolation can be model merging (Stoica et al., 2024; Matena and Raffel, 2022; Li et al., 2022; Davari and Belilovsky, 2023). In the work Ramé et al. (2023), the authors propose a strategy of merging multiple model weights pre-trained sets of auxiliary tasks as an initialization to multiple parallel fine-tunings to enhance out-of-distribution generalization. Most of these works on model weights interpolations usually focus only on the weights of the whole model or particular weights (e.g., classification heads, activation layers) of the pre-trained model.

There are also works on weight interpolation of PEFT methods in general (Zhang et al., 2023; Chronopoulou et al., 2023; Qin et al., 2021; Pfeiffer et al., 2021), but not many of them focus on interpolation using task vectors. In the work Klimaszewski et al. (2024) authors present a way of combining pre-trained adapters using task vector

arithmetics, but the method lacks the investigation of the dependency of their method on the random initialization of adapters, therefore it may require training of specific adapters from the same random initialization, which we provide in our work in the context of prompt tuning.

To the best of our knowledge, there is no research on task vectors in the context of soft-prompt-based fine-tuning. In this work, we address this drawback by building on the existing knowledge on prompt tuning and task vectors.

3 Task Prompt Vectors

Background. Prompt tuning, as introduced in Lester et al. (2021), casts tasks as text generation, modeling a probability $Pr(Y|X)$, where X is a sequence of input tokens and Y is a sequence of output tokens representing the class label. The classification $Pr_\theta(Y|X)$ is then parametrized by the model weights θ. Prompting adds extra information to the classification process by prepending a series of tokens (prompt) P to the input X, such that the model maximizes the probability of getting current Y in $Pr_\theta(Y|[P;X])$, while keeping the parameters θ frozen. Prompt tuning adds another parameter θ_P to the equation, which parametrizes the prompt. During the training, only θ_P is updated as the $\mathcal{L}_{PT} = -\sum_i logPr_{\theta,\theta_P}(Y_i|[P;X_i])$ (1) function is optimized.

As a method of adapting model weights without training, task vectors (Ilharco et al., 2022) compute the difference between pre-trained weights and fine-tuned weights on a specific task. The task vector is simply defined by the element-wise difference between the pre-trained weights of the whole model and the weights after fine-tuning. Task vectors can be then applied to any model weights θ of the same dimensionality (architecture) by element-wise addition. The representation of task vectors in the weight space of the model has the same properties as standard vectors, therefore it is possible to include them in arithmetic expressions like addition, negation, or combinations via the addition of two or more vectors together. We further build on the findings from Ilharco et al. (2022) and Lester et al. (2021) in the following parts of this section.

Task prompt vector definition. Let $T_1, ..., T_t$ be a set of source tasks and $\theta_{P_1}, ..., \theta_{P_i}$be a set of random soft-prompt weights initializations. Intuitively, the random soft-prompt weights initializations are random points in the embedding space of the PLM. We then move each of these points (via prompt tuning) into a task sub-space where the optimization function from the equation 1 returns the (sufficiently) minimal value and we repeat this for each task $t \in T$. These points are further denoted as *task prompts* – soft-prompts fine-tuned by prompt tuning to a set of downstream tasks. The straight trajectory from the initial point to the task prompt is our *task prompt vector* (see Figure 1 part a)).

Let $\theta_{P_{pre}} \in \mathbb{R}^d$ be the weights of the soft-prompt randomly initialized from the embedding vocabulary of a PLM, and $\theta^t_{P_{ft}} \in \mathbb{R}^d$ be the weights of the soft-prompt P fine-tuned on a specific task t, using the standard prompt tuning formula. We formulate the task prompt vector τ_{P_t} for soft-prompt P and task as an element-wise difference:$\tau_{P_t} = \theta^t_{P_{ft}} - \theta_{P_{pre}}$ (2).

Applying a task prompt vector to the soft-prompt weights of the same size would follow: $\theta_{P_{new}} = \theta_P + \lambda\theta^t_{P_{ft}}$ (3). Where the rescaling term λ is a number from the same interval $0 < \lambda \leq 1$ and when $\lambda = 1$, then $\theta_{P_{new}} = \theta^t_{P_{ft}} = \theta_P$.

Vector arithmetics with task prompt vectors. The task prompt vectors for different tasks can be combined by simple vector addition, combining knowledge from different tasks. When we experiment with combinations, we refer to the arithmetic addition of two task prompt vectors (see Figure 1 part b)): $\tau_{P_{new}} = \tau_{P_a} + \tau_{P_b}$ (4).

This makes for efficient task adaptation as we perform no further training but only use vector addition in the next sections. Task prompt vector combinations can be also used for initializing a new task that is sufficiently similar to an already trained task. We investigate and discuss these possible use cases for task prompt vectors in upcoming sections.

4 Experiments

4.1 Experimental Setup and Implementation Details

We investigate the properties of task prompt vectors using the representative foundation **T5-base** (Raffel et al., 2020) model for all of our experiments and representative open autoregressive **LLaMa-3.1-8B-Instruct** (Dubey et al., 2024) model for origin dependency experiments in Section 4.2. Our investigation covers 3 types of classification problems and 12 corresponding datasets, namely **natural language inference** – *MNLI* (Williams et al., 2018), *QNLI* (Wang et al., 2018), *SciTail* (Khot

et al., 2018), *SNLI* (Bowman et al., 2015); **topic classification** – *DBPedia* (Auer et al., 2007), *TREC* (Li and Roth, 2002; Hovy et al., 2001), *AG News*, *Yahoo Answers* (Zhang et al., 2015); and **sentiment classification** – *SST2* (Socher et al., 2013), *Yelp Polarity*, *SST5*, *IMDB* (Maas et al., 2011).

For all experimental results, we report F1 macro, if not specified otherwise. The cosine similarity between vectors (task prompts or task prompt vectors) is measured using the flattened weights of each vector (which has a size of 100×768 parameters, resulting in a 76800-dimensional vector). We average our zero- and few-shot results across 3 different runs (i.e., different random initializations of soft-prompts) for ATTEMPT and multi-task SPoT baselines (mostly to save more computational resources) and across 10 different runs for all other experiments. To determine the statistical significance of our results we perform a two-sample Student's t-test (Student, 1908) with Bonferroni correction (Dunn, 1959) between the best result and the second best result. If the population sizes differ (e.g. 10 and 3 runs) we use Welch's t-test (Welch, 1947).

For the few-shot experiments (simulating limited labeled data scenarios), we randomly sub-sample from the data for the respective number of shots while keeping the class distribution. We consider shot and sample to be equivalent (i.e., for a 5-shot setting, we choose 5 samples overall, not 5 samples per class). A detailed description of our experimental setup can be found in Appendix A.

4.2 Investigating Task Prompt Vectors Properties

In this section, we aim to address the following research question (RQ):

RQ1: How universally can we apply task prompt vectors to a) different prompt initializations and b) different tasks?

There are two fundamental properties that are crucial for the effectiveness of task prompt vectors: 1) If such vectors should be applied universally, their dependence on the random initialization of prompt tuning should be low, since soft-prompts are usually initialized randomly, unlike PLM for task prompts in Ilharco et al. (2022). 2) The similarity of task prompt vectors between similar tasks should be large, in order to be able to combine task prompt vectors of similar tasks.

To evaluate these properties, we train a set of soft-prompts on specified source tasks for inference classification (*MNLI, QNLI*), topic classification (*DBPedia, TREC*), and sentiment classification (*SST2, Yelp Polarity*), resulting in a set of six soft-prompts that were trained from a single random initialization. We sample *10 random initializations* for T5-base model from which we create the task prompt vectors as described in equation 2. For LLaMa-3.1-8B-instruct we sample only 3 random initialization to preserve computational resources. We aggregate by averaging our results across random initializations in Table 1 and Figures 2, 3. We start with the evaluation of whether the task prompt vectors are independent of the random initialization and continue with the experiments to confirm whether the trained task prompts from prompt tuning end up in the same task sub-space of the PLM embedding space. This helps us determine whether the task prompt vectors point in the same space, similar to Figure 1 part c).

The performance of task prompt vectors is independent of the random initialization for the majority of observed tasks. We conduct experiments to evaluate the performance of applying task prompt vectors to different (mixed) random initializations. For each task and each random initialization, we apply the task prompt vector (according to the equation 3) to all of the other random initializations and evaluate performance for each task prompt vector-initialization pair on the test set of the particular dataset. The aggregated results in "Mixed init" rows in Table 1 differ only slightly in most observed tasks for both of the observed models, compared to the results of prompt tuning in the "Original init" rows. This indicates that task prompt vectors perform well irrespective of their initialization. The only exception is in the TREC task, where the performance decreases drastically only for the T5-base model. We suspect that this may be caused by the task being harder for the T5-base model to learn, which also confirms the higher standard deviation from the mean of prompt tuning performance. We can also see that in the case of the LLaMa-3.1-8B-Instruct model, there is no statistically significant difference between using the original initialization or different task prompt vector initialization and for the TREC and SST2 the average results slightly increased.

Task prompts and task prompt vectors maintain good performance even if they do not point

model	dataset	QNLI	MNLI	TREC	DBpedia	SST2	Yelp	avg
T5	Original init	93.3_{0}	$85.4_{0.1}$	$95.5_{1.7}$	99.1_{0}	$93.8_{0.3}$	97.2_{0}	$93.8_{0.4}$
	Mixed init	$93.2_{0.1}{}^{*}$	$85.3_{0.2}$	$26.5_{18.2}{}^{*}$	$99_{0.1}{}^{*}$	$93.2_{0.6}$	$97.1_{0.1}{}^{*}$	$82.4_{3.2}{}^{*}$
LLaMa	Original init	92.0_{0}	$89.7_{0.2}$	$95.8_{0.3}$	99.2_{0}	$95.9_{0.4}$	$98.6_{0.1}$	$95.2_{0.2}$
	Mixed init	$92.0_{0.1}$	$89.7_{0.2}$	$96.0_{0.3}$	99.2_{0}	$96.0_{0.5}$	$98.6_{0.1}$	$95.3_{0.2}$

Table 1: Comparison of exact match results across 10 random soft-prompt initializations for T5-base model and 3 initializations for LLaMa-3.1-8B model. The subscript represents the standard deviation from the average. The first row (Original init) represents the results of prompt tuning. The second row (Mixed init) represents the results of moving a specific initialization in the direction of a task prompt vector created from different (mixed) initializations.

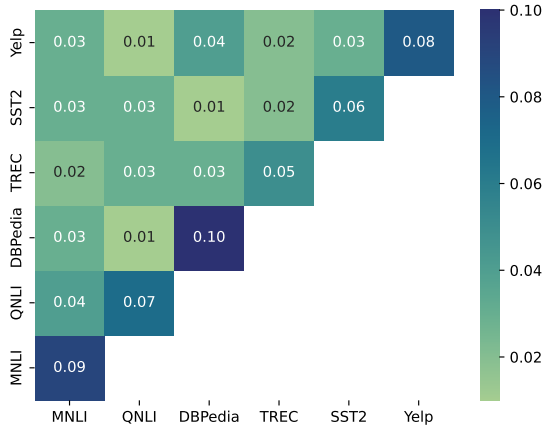

Figure 2: Comparison of average cosine similarities of *task prompts* fine-tuned on different tasks for T5-base model. The average is calculated across all combinations of 10 random initializations (i.e., row QNLI column MNLI was calculated as the average of all cosine similarities between MNLI and QNLI task prompts for all random initialization combinations omitting the combinations where cosine similarity is equal to 1). The diagonal represents the cosine similarities of the same tasks and it represents the maximum value of cosine similarity across different random initializations.

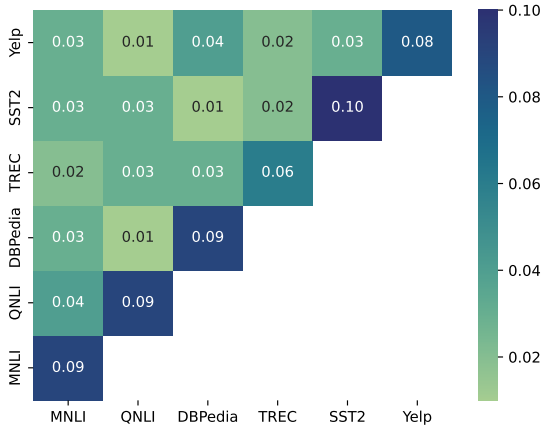

Figure 3: Comparison of average cosine similarities of *task prompt vectors*. The averages are calculated equivalently to Figure 2 but with task prompt vectors created from different task prompts.

to the exact same location in the task subspace. To see whether the trained task prompts end up in the same task sub-space, we evaluate cosine similarity across multiple random initializations. We train multiple task prompts for 10 different random initializations and each source task (60 task prompts in total) and compute the cosine similarity from trained task prompts for each combination of random initializations and for each combination of tasks. We then average this cosine similarity for each task combination across all random initialization combinations. If task prompts initialized from different random initializations are pointing to different points in the task sub-space, we should also witness this phenomenon with their corresponding task prompt vectors. Therefore, we repeat this process for task prompt vectors.

Figures 2 and 3 show the comparison of cosine similarities between task prompts and task prompt vectors from different tasks averaged over all random initialization combinations. We can see from the low cosine similarities in both tables, that the task prompts and task prompt vectors do not end up in the same direction when initialized from different points in the embedding space. The highest cosine similarities on the diagonal represent the highest cosine similarity, which serves as a baseline for comparison with the cross-task cosine similarities. We can see in Table 1 row 1, that the downstream performance of prompt tuning on the source tasks across 10 different random initializations has a low standard deviation from the average. This means that the task prompts after prompt tunings end up in a subspace with sufficient task performance, without necessarily pointing to the same spot in the task subspace. Cosine similarities that we have used to create the aggregated figures can be seen in the Appendix B in Figures 6 and 7. We have also evaluated cosine similarities of task prompt vectors created from 3 different random initializations for

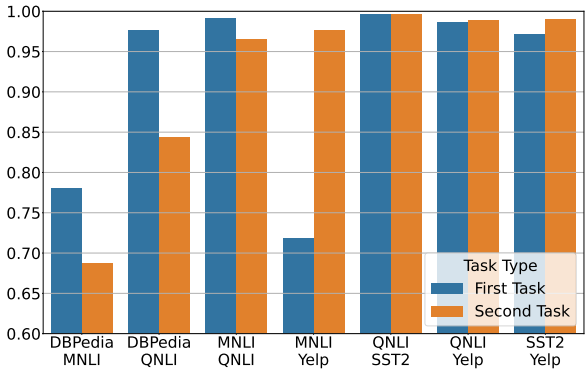

Figure 4: Comparison of relative exact match performance of combinations of task prompt vectors across averaged across 10 different random initializations. The results are relative to the original single-task performance (1 is the performance of single-task prompt tuning).

LLaMa-3.1-8B-Instruct in Appendix B in Figures 9 and 10.

Task prompt vectors from similar problems are more similar. Additionally, we evaluate the similarity of different task prompt vectors across different tasks. Figure 3 shows the cosine similarity between task prompt vectors for different tasks. We can see that certain pairs of tasks are more similar than others, what can be shared properties of these tasks, such as the same number of classes, same labels, or solving the same problem. Problem similarity can be seen in DBPedia–TREC and MNLI–QNLI task prompt vectors, and the similarity in the number of classes can be seen in the MNLI task prompt vector which tends to have higher cosine similarity with task prompt vectors for tasks with more classes (e.g., DBPedia, TREC).

4.3 Combination of Task Prompt Vectors via Addition for Multi-Task Transfer

This section addresses the following research question: **RQ2: Can we combine multiple task prompt vectors and maintain multi-task performance on the source tasks?**

To answer this research question, we investigate the method of combination via addition on 15 task pair combinations from the set of NLU datasets (*MNLI, QNLI, DBPedia, TREC, SST2, Yelp Polarity*). We also evaluate combinations of task prompt vectors in a simulated limited data environment by providing 0 to 100 training examples before evaluation on the test set.

Combinations of task prompt vector pairs maintain single-task performance on specific observed tasks. To evaluate how the combinations of task prompt vectors maintain their single-task performance, we conduct experiments of creating pair combinations from all of the source tasks (according to equation 4). We aggregate the best-performing combinations in Figure 4. The full results from the experiment can be found in Appendix C in Figure 8. We can see from the results that most of the binary classification tasks retain their single-task performance on both of the tasks, which implies that task prompt vectors can be used for solving multi-task problems. In some cases, the single-task performance was kept only for a single source task, which leads us to the conclusion that certain combinations of task prompt vectors may be more suitable than others.

Additionally, we chose two target tasks for inference classification (*SciTail, SNLI*), topic classification (*AG News, Yahoo Answers*), and sentiment classification (*SST5, IMDB*) and we keep the same set of source tasks. Results for SciTail, AG News and IMDB are in Table 2; the full table with extended experiments is in Appendix D in Table 4.

Task prompt vector combinations can initialize zero-shot and few-shot learning. We compare initialization with randomly initialized soft-prompt, soft-prompt trained on single and multiple source tasks (this is an equivalent of soft-prompt transfer presented in Vu et al. (2022)), the multi-task ATTEMPT (Asai et al., 2022) method, and a combination of task prompt vectors of both of the source tasks. From the 0-shot and 100-shot results (Table 2), we can see that our combination of task prompt vectors can outperform the initialization with a single-task source soft-prompt on SciTail and IMDB datasets and the multi-task source soft-prompt only in the case of SciTail task. In some cases, the combination of task prompt vectors matched the SPoT baseline like in the case of AG News. This may be because the combination of DBPedia and TREC does not retain much information about TREC, which could benefit the overall result.

We can also see that the ATTEMPT method is significantly underperforming when using a smaller set of pre-trained source soft-prompts. Another observation is that ATTEMPT performs better on the AG News task. This may be caused by using the original implementation of ATTEMPT, where

SciTail (NLI)			AG News (Topic)			IMDB (Sentiment)		
Source tasks	F1		Source tasks	F1		Source tasks	F1	
	0 shots	100 shots		0 shots	100 shots		0 shots	100 shots
Random	$54.9_{6.6}$	$75.6_{0.5}$	Random	0_{0}	$50.4_{11.2}$	Random	$77.2_{9.6}$	$89.4_{0.4}$
MNLI (SPoT)	$\underline{70.4_{0.4}}$	$\underline{87.8_{0.9}}$	DBPedia (SPoT)	0_{0}	$\mathbf{83.4_{0.6}}^{*}$	SST2 (SPoT)	$88_{0.6}$	$90.2_{0.3}$
QNLI (SPoT)	$57.7_{13.1}$	$77.7_{1.3}$	TREC (SPoT)	0_{0}	$65.7_{5.6}$	Yelp (SPoT)	$90_{0.3}$	$90.3_{0.2}$
QNLI + MNLI (SPoT)	$70.4_{1.2}$	$87.7_{0.6}$	DBPedia + TREC (SPoT)	0_{0}	$82.1_{0.9}$	SST2 + Yelp (SPoT)	$\mathbf{90.8_{0.2}}$	$\mathbf{90.8_{0.2}}$
QNLI + MNLI (ATTEMPT)	$63.8_{4.2}$	83.6_{3}	DBPedia + TREC (ATTEMPT)	$\mathbf{11.5_{1.7}}$	$20.7_{2.8}$	SST2 + Yelp (ATTEMPT)	79.2_{6}	$89.4_{0.8}$
QNLI + MNLI (ours)	$\mathbf{71.5_{0.8}}^{*}$	$\mathbf{88.1_{0.9}}$	DBPedia + TREC (ours)	0_{0}	$\underline{83_{0.9}}$	SST2 + Yelp (ours)	$\underline{90.1_{0.5}}$	$\underline{90.4_{0.2}}$

Table 2: Test results of training T5-base model with random, single- and multi-task soft-prompt transfer (SPoT), multi-task ATTEMPT, and our task prompt vectors on 0-shot and 100-shots of data. We show the initialization with different combinations for natural language inference classification, topic classification, and sentiment classification. The subscript represents the standard deviation from the average. The best results are bold, while the second-best results are underlined. The * denotes statistical significance.

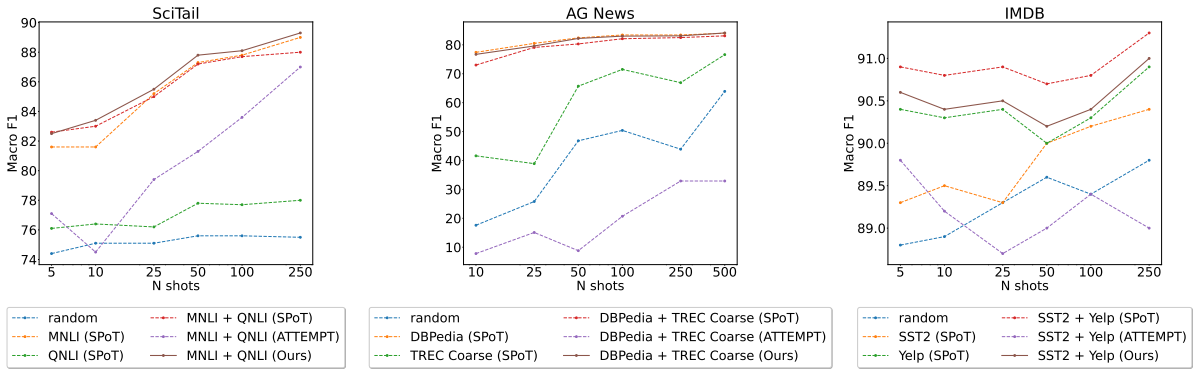

Figure 5: Test results of training T5-base model with random, single, and multi-task soft-prompt transfer (SPoT), multi-task ATTEMPT, and our task prompt vectors combination on increasing numbers of shots of data. We can see that for SciTail and IMDB tasks, a combination of task prompt vectors outperforms single task transfer.

Method	Modularity	Multi-task performance	Source prompt independence
SPoT	✗	✓	✓
ATTEMPT	✓	✓	✗
Task Prompt Vectors	✓	✓	✓

Table 3: Comparison of multi-task properties for compared methods. Task prompt vectors maintain high task modularity and multi-task performance, and are independent of the quality or number of pre-trained source soft-prompts.

authors, instead of using textual labels (i.e., "entailment", "not entailment"), used textual numbers as labels (i.e., "0", "1"), which made the model predict numbers instead of specific words.

While matching the results of full multi-task soft-prompt transfer (SPoT) training initialization of prompt tuning using task prompt vector combinations also retains high task modularity, which means that we can add new tasks without the necessity of training. Only in the case of the IMDB task does the SPoT baseline fine-tuned on both datasets perform better. However, it requires the training process to be fully repeated for each new task, resulting in higher computational costs. Table 3 compares attributes beneficial for multi-task training for SPoT, ATTEMPT, and task prompt vector methods. We can see that the SPoT method has low multi-task modularity because we need to retrain the source soft-prompt every time we change the set of source tasks. ATTEMPT, while having sufficient task modularity, depends heavily on the quality and number of source soft-prompts. Task prompt vectors have both of these attributes and also retain sufficient multi-task performance.

4.4 Additional Results: Few-Shot Comparison

In this section, we study how increasing the number of demonstration data affects the performance of prompt tuning on a target task initialized by a combination of task prompt vectors of similar source tasks. We keep the same experiment setup as in the previous section and further evaluate the soft-prompt initialization on 5, 10, 25, 100, 250, 500 shots. We also assessed the topic classification tasks for 500 shots since we started with 10 shots

due to our sampling method.

From the results in Figure 5, we can see that the performance of the combination of task prompt vectors for SciTail and IMDB target tasks outperforms using a single-task initialization for multiple shots. We can also see that our method outperforms the multi-task initialization for the SciTail dataset across all shots of data. Comparing the results from Figure 4 and Figure 5, if we choose a combination of tasks that maintains a significant amount of the source task performance (MNLI + QNLI and SST2 + Yelp), the few-shot performance of the task prompt vector combination tend to be higher than single-task transfer. The full results across more shots and more target tasks can be found in Appendix D in Figure 11.

5 Discussion

In Section 4.2, we showed that task prompts and their corresponding task prompt vectors are close to orthogonal, by comparing their cosine similarities across multiple initializations in Figures 2 and 3. Despite their near-orthogonality, **task prompt vectors created from one initialization and applied to a different one maintain their performance** for the majority of the observed tasks.

In Section 4.3, we showed that combinations of certain task prompts maintain their source single-task performance (in Figure 4) and that the combinations of task prompt vectors can be used for initialization of prompt tuning (in Table 2) in simulated low resource setting on the set of target tasks. From the results in Figure 5, we can see that, for the SciTail and IMDB datasets, our task prompt vector initialization maintains its higher performance compared to the single-task soft-prompt transfer even with the higher number of samples.

Theoretical implications and analysis. It lies beyond the scope of our work to further deliver theoretical proofs for diverse properties of task prompt vectors, which we will leave for future work. However, we still discuss the implications that arise from the contributions of task prompt vectors.

Based on the observations from Section 4.2, we can re-use pre-trained task prompt vectors for different tasks and use them in downstream scenarios. Since task prompt vectors are independent of their initialization, we can also re-use pre-trained task prompt vectors shared on the internet (e.g., on a designated vector hub). Another implication that we can derive from these findings is that the **subspace with optimal values in the soft-prompt space has probably a convex shape**. This may be indicated by the fact that task prompts trained from different random initializations for the same task do not point to the same direction (based on Figures 2 and 3), but still achieve identical results.

Section 4.3 implied that we can use different combinations of task prompt vectors to gain even zero-shot multi-task behavior (in Figure 4). **We can combine multiple task prompt vectors** and maintain multi-task performance on the source tasks, **but the right task combinations need to be found** (e.g., by evaluating on held-out validation sets). We can see that it is possible to use linear combinations even though the soft-prompts space is non-linear. The rationale behind this could be that **task prompt vectors are linear approximations of how the soft-prompts change** during the training. Another possibility may be that the task prompt vectors are sparse, and a combination of 2 sparse task prompt vectors creates a vector that contains more information about both tasks. These findings can be further useful for **machine unlearning tasks**, where one could also include subtraction.

6 Conclusion

In our work, we introduce and investigate task prompt vectors as a method of multi-task transfer from prompt tuning. We show that the task prompt vectors are not dependent on random initialization and that the performance across different random initializations does not change significantly in the majority of observed source tasks. We show that in some tasks, the combination via arithmetic addition maintains the single-task performance. Finally, we show that certain combinations of task prompt vectors can be a better option for initialization for certain tasks while maintaining higher multi-task modularity than other methods.

In the future, we would like to evaluate the cross-model performance of task prompt vectors. We think that further experiments with generation tasks may be another interesting extension. Moreover, task prompt vector arithmetic has the highest potential for improving the unlearning in PLMs by negating the task prompt vectors for the tasks we want to unlearn. Such an option is enabled by introducing task prompt vectors, which would not be possible with the existing state-of-the-art methods.

Limitations

The experiments in our work utilize only classification datasets in the English language with various characteristics (problem type, dataset size, number of classes), mostly because of the large amount of publicly available data covering a variety of NLU problems. At the same time, to direct our focus primarily on the evaluation of task prompt vectors, we utilize only monolingual models in the scope of our work.

Even though there are many other PLMs capable of conditional generation that beat T5 models in performance on various benchmarks, we focus our experiments on the T5-base model as it is commonly used as a representative model in many PEFT methods. We also utilize LLaMa-3.1-8B-Instruct model only for the initialization dependency experiments.

Finally, we focus on 3 common NLU problems (natural language inference, topic classification, and sentiment classification) that are commonly incorporated in NLU benchmarks and do not consider other NLU problems (e.g., question answering, slot tagging, acceptability classification, text generation). However, we find that our set of 3 common NLU problems each covering 4 different tasks, is enough to evaluate the properties of task prompt vectors. Including generation tasks would further introduce a problem with the evaluation of the generated text.

Ethical Considerations and Impact Statement

The experiments in this paper were conducted with publicly available datasets MNLI, QNLI, SciTail, SNLI, DBPedia, TREC, AG News, Yahoo Answers, SST2, Yelp Polarity, SST5, and IMDB, citing the original authors. MNLI, QNLI, and SST2 are part of the GLUE benchmark. As we were not able to determine the license for all used datasets, we have opted to use them as in a limited form as possible, adhering to the terms of use of the GLUE benchmark for all of the mentioned datasets. As the datasets are commonly used in other related works, and were published in scientific works that went through an established review process, we do not check for the presence of any offensive content as it was already removed by the authors of these publicly available datasets. In addition, we do not utilize any personally identifiable information or offensive content and we do not perform crowdsourcing in any form for data annotation. To our knowledge, we are not aware of any potential ethical harms or negative societal impacts of our work, apart from the ones related to the field of Machine Learning (i.e., use of computational resources that are consuming energy and producing heat with indirect CO2 emission production). We follow the license terms for the T5-base model we use – all models and datasets allow their use as part of the research. As we perform conditional generation transform into the classification problem (generating only labels), we minimize the problem of generating offensive or biased content.

Impact Statement: CO2 Emissions Related to Experiments The experiments in this paper require a significant amount of GPU computing resources as we train and evaluate 1 model over multiple random initializations (10) for different methods (4) and datasets (12). Overall the experiments including evaluations (which did not require training, but still used GPU resources for inference) and preliminary experiments (which are not reported in the scope of our work) were conducted using a private infrastructure, which has a carbon efficiency of 0.432 $kgCO_2eq/kWh$. Approximately, 1000 hours of computation performed on hardware of type A100 PCIe 40GB (TDP of 250W). Total emissions are estimated to be 105.24 $kgCO_2eq$ of which 0 percent were directly offset. These estimations were conducted using the CodeCarbon (Courty et al., 2024) python module. Whenever possible we tried to reduce the computational costs. Because our method is built upon the prompt tuning PEFT method, we always trained only a small part of the model parameters (76800 parameters, which is around 0.2% of the T5-base model parameters), and training the model fully will probably require more GPU hours and create more CO2 emissions.

References

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

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The*

Semantic Web, pages 722–735, Berlin, Heidelberg. Springer Berlin Heidelberg.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

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

Alexandra Chronopoulou, Jonas Pfeiffer, Joshua Maynez, Xinyi Wang, Sebastian Ruder, and Priyanka Agrawal. 2023. Language and task arithmetic with parameter-efficient layers for zero-shot summarization. *arXiv preprint arXiv:2311.09344*.

Benoit Courty, Victor Schmidt, Sasha Luccioni, Goyal-Kamal, MarionCoutarel, Boris Feld, Jérémy Lecourt, LiamConnell, Amine Saboni, Inimaz, supatomic, Mathilde Léval, Luis Blanche, Alexis Cruveiller, ouminasara, Franklin Zhao, Aditya Joshi, Alexis Bogroff, Hugues de Lavoreille, Niko Laskaris, Edoardo Abati, Douglas Blank, Ziyao Wang, Armin Catovic, Marc Alencon, Michał Stęchły, Christian Bauer, Lucas-Otavio, JPW, and MinervaBooks. 2024. mlco2/codecarbon: v2.4.1.

MohammadReza Davari and Eugene Belilovsky. 2023. Model Breadcrumbs: Scaling Multi-Task Model Merging with Sparse Masks. ArXiv:2312.06795 [cs].

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Olive Jean Dunn. 1959. Confidence intervals for the means of dependent, normally distributed variables. *Journal of the American Statistical Association*, 54(287):613–621.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. 2020. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR.

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

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

Eduard Hovy, Laurie Gerber, Ulf Hermjakob, Chin-Yew Lin, and Deepak Ravichandran. 2001. Toward semantics-based answer pinpointing. In *Proceedings of the First International Conference on Human Language Technology Research*.

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*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2022. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*.

Tushar Khot, Ashish Sabharwal, and Peter Clark. 2018. Scitail: A textual entailment dataset from science question answering. In *AAAI Conference on Artificial Intelligence*.

Mateusz Klimaszewski, Piotr Andruszkiewicz, and Alexandra Birch. 2024. No train but gain: Language arithmetic for training-free language adapters enhancement. *arXiv preprint arXiv:2404.15737*.

Haeju Lee, Minchan Jeong, Se-Young Yun, and Kee-Eung Kim. 2023. Bayesian multi-task transfer learning for soft prompt tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4942–4958, Singapore. 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.

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.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. 2022. Branch-Train-Merge: Embarrassingly Parallel Training of Expert Language Models. ArXiv:2208.03306 [cs].

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.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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

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.

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

Michael Matena and Colin Raffel. 2022. Merging Models with Fisher-Weighted Averaging. ArXiv:2111.09832 [cs].

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. 2024. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems*, 36.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

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

Yujia Qin, Xiaozhi Wang, Yusheng Su, Yankai Lin, Ning Ding, Jing Yi, Weize Chen, Zhiyuan Liu, Juanzi Li, Lei Hou, et al. 2021. Exploring universal intrinsic task subspace via prompt tuning. *arXiv preprint arXiv:2110.07867*.

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.

Alexandre Ramé, Kartik Ahuja, Jianyu Zhang, Matthieu Cord, Léon Bottou, and David Lopez-Paz. 2023. Model ratatouille: Recycling diverse models for out-of-distribution generalization. In *International Conference on Machine Learning*, pages 28656–28679. PMLR.

Zhengxiang Shi and Aldo Lipani. 2024. DePT: Decomposed prompt tuning for parameter-efficient fine-tuning. In *The Twelfth International Conference on Learning Representations*.

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

George Stoica, Daniel Bolya, Jakob Bjorner, Pratik Ramesh, Taylor Hearn, and Judy Hoffman. 2024. ZipIt! Merging Models from Different Tasks without Training. ArXiv:2305.03053 [cs].

Student. 1908. The probable error of a mean. *Biometrika*, pages 1–25.

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.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl.

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

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

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. 2023. Multitask prompt tuning enables parameter-efficient transfer learning. In *The Eleventh International Conference on Learning Representations*.

Bernard L Welch. 1947. The generalization of 'student's'problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35.

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.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. 2022. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7959–7971.

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. *arXiv preprint arXiv:2312.12148*.

Jinghan Zhang, Junteng Liu, Junxian He, et al. 2023. Composing parameter-efficient modules with arithmetic operation. *Advances in Neural Information Processing Systems*, 36:12589–12610.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28.

A Experimental setup: Further Details

Implementation details. For implementing all of our experiments, we utilize *Python 3.11.8* with the *PyTorch* (Paszke et al., 2019) framework and Huggingface modules (*transformers* (Wolf et al., 2020) for model loading and training, *peft* (Mangrulkar et al., 2022) for PEFT methods initialization, *datasets* (Lhoest et al., 2021) for data loading, and *evaluate* for evaluation). We create a single data structure for task prompt vectors, that is capable of the arithmetic operations with soft-prompts.

Data splits. We take 1000 samples from the train set and use it as a validation set and make the test set from the original validation set for datasets that contain over 10000 samples. For datasets with less or equal to 10000 samples we do not modify the training set, and split the validation set in 2 halves for validation and test sets. We keep the same random seed for subsampling and splitting for all of our experiments.

Hyperparamters setings. We provide all of our configurations in the config directory of our repository. We set soft-prompt length to 100 tokens, learning rate to 0.3, and lower the weight decay of the AdamW optimizer (Loshchilov and Hutter, 2019) to 1×10^{-5} for T5-base model. We also utilize the Seq2SeqTrainer class from the Huggingface transformers Python module. We train all models on all data sets for 10 epochs, except for TREC, where we train for 50 epochs due to the tendency of models to underfit here. We set different hyperparameters for prompt tuning and the hyperparameters for zero- or few-shot evaluation do not differ much from the hyperparameters for prompt tuning. We train for 1000 update steps while keeping a batch size of 2 for 5, 10, 25 shots, 8 for 50, 100, 250 shots, and 16 for 500, 750, 1000 shots. In general, we chose the maximum token length for labels by searching the dataset for the maximum token length (in our configs, we set default *max_target_lenght* to 128 if the dataset requires to generate sentences), for the inputs we pad the token sequences to 256 tokens

with the *max_target_lenght* parameter. We use a learning rate of 0.3 for the *AdamW* optimizer, with weight decay of 1×10^{-5} and 500 warmup steps for 10 epochs (with an exception for the TREC dataset) with the batch size of 32. We evaluate, log and save after each 100 training steps and keep only the best model at the end of the training. In our configs, we set a number of tokens to 50, but in reality, Hugging Face *peft* library doubles the number for encoder-decoder models like T5. When combining task prompt vectors, we evaluate their performance on the individual source tasks that formed the task combination and found the best rescaling factor λ via held-out validation sets.

For training the soft-prompts with the LLaMa-3.1-8B-Instruct model we utilized similar hyperparameter settings as for the T5-base model with the exception of using the cosine function for the learning rate scheduler and the usage of the SFTTrainer class from the Huggingface trl Python module (von Werra et al., 2020) for training.

Since we are utilizing the T5-base for conditional generation, we are computing exact match instead of accuracy for classification. Because we are generating labels also for classification tasks, the exact match is equivalent to accuracy in the sequence classification task. In the scope of our work, we refer to a single dataset as a task.

For the training of multi-task ATTEMPT, we have used hyperparameters and a training environment based on the original implementation. Full hyperparameter settings can be found in the repository[2] of our replication study of ATTEMPT in the configs directory (files **attempt_tvp*.toml**).

B Additional results: Task Prompt Vectors and Task Prompt Cosine Similarities

In this section, we provide more detailed and de-aggregated results from Section 4.2. Figure 6 shows the comparison of cosine similarities across different random initializations of task prompts from prompt tuning. We can see that for all task combinations, the highest cosine similarity is for the equal random initializations. Additionally, when comparing different tasks and different random initializations the cosine similarities are the lowest, which only confirms our finding from Section 4.2.

We repeat the same process of comparing cosine similarities across different random initializations for task prompt vectors in Figure 7. Similarly to task prompts, the highest cosine similarity is for the equal random initializations. We can see that for task prompt vectors the cosine similarities between different random initializations are higher than compared to task prompts in Figure 6. Similarly to our findings in 4.2, we can that certain task combinations have higher cosine similarities than others. For both of these figures, we can see that task prompts and task prompt vectors from different initializations usually end up at different points in the task sub-space.

[2] https://anonymous.4open.science/r/ATTEMPT-C5C6

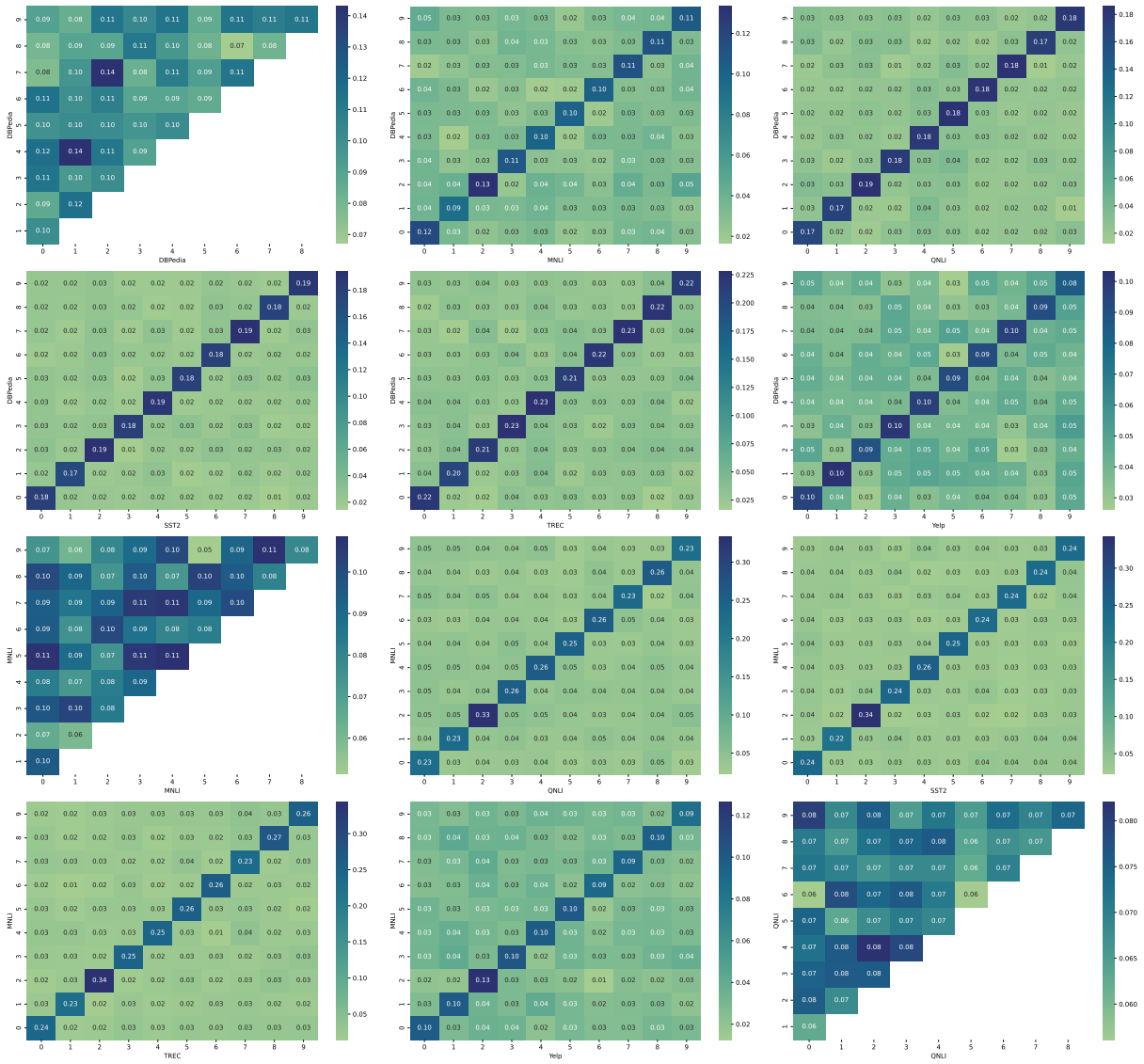

Figure 6: Comparisons of cosine similarities of *task prompts* fine-tuned on different tasks for T5-base model. Each heatmap represents a different task combination. We calculate the cosine similarities for all combinations of 10 random initializations omitting the combinations of random initializations where cosine similarity is equal to 1 (single-task comparisons). Each heatmap is represented as a single field in Figure 2 by averaging all values. The x and y axes represent the number of random initializations.

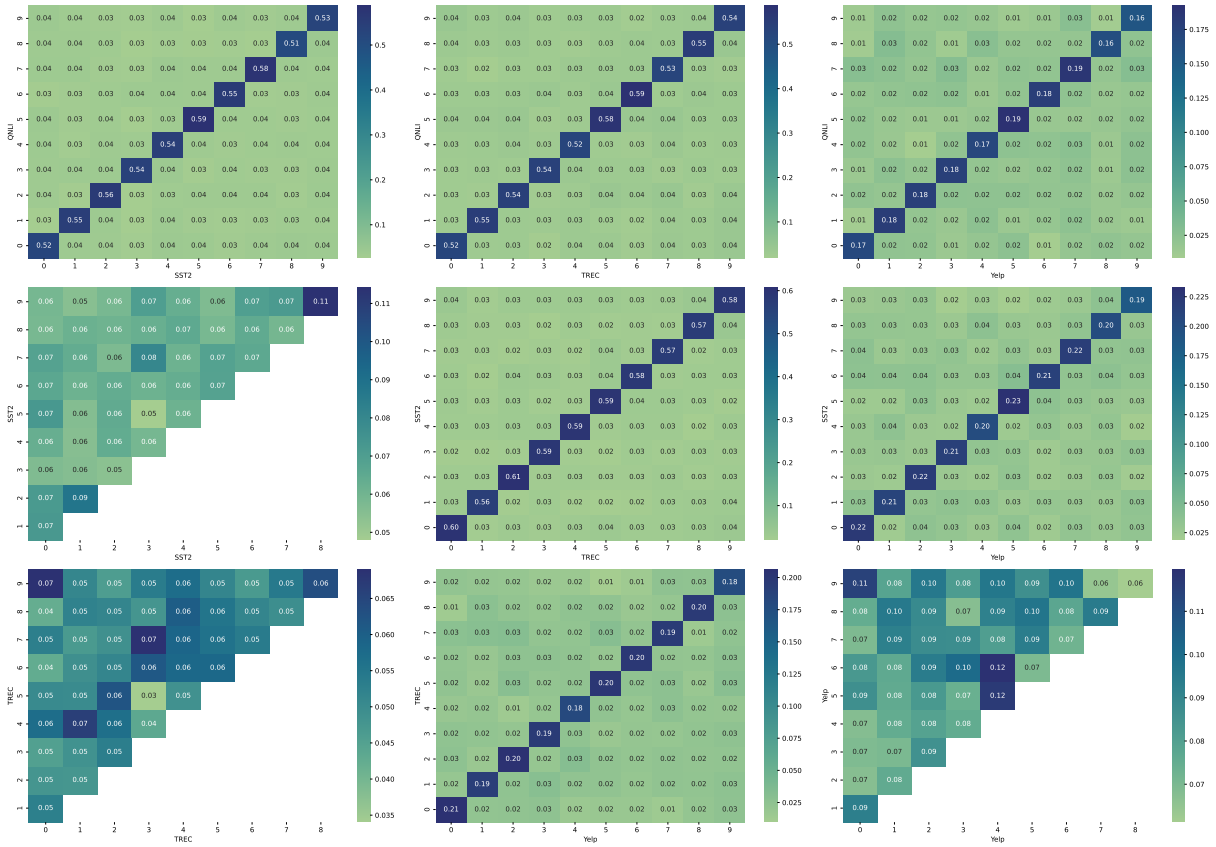

Figure 6 (cont.): Continuation of Figure 6 for additional tasks.

C Additional results: Combinations of Task Prompt Vectors

This section provides extended experiments to the results in Figure 4 in Section 4.3. Figure 8 shows the relative performance of all task combinations of task prompt vectors. Usually, tasks that solve the same NLU problem retain the most source task performance on both tasks except for the combination of DBPedia and TREC task prompt vectors, where the TREC performance is lower. In general, the performance of combinations with the TREC usually ends up in favor of the other task from the task pair.

D Additional results: Few-Shot Experiments

Here we provide extended results of zero- and few-shot experiments on additional target tasks that extend the results from Section 4.3. Table 4 extends the comparison of 0- and 100-shot results with SNLI, Yahoo Answers, and SST5 tasks. We can see that combinations of source task prompt vectors do not outperform the SPoT baseline in these specific tasks, but rather almost match the results.

Figure 11 extends the comparison in Figure 5 in Section 4.4 and shows how the performance on different initializations differs across all observed shots of data and on additional SNLI, Yahoo Answers, and SST5 target tasks. We can see that in the case of the SST5 task, the SST2 initialization performs the best. We think that the reason for this may also be the similarity of SST5 and SST2 and that the combination of source tasks does not retain enough information to match the SST5 baseline.

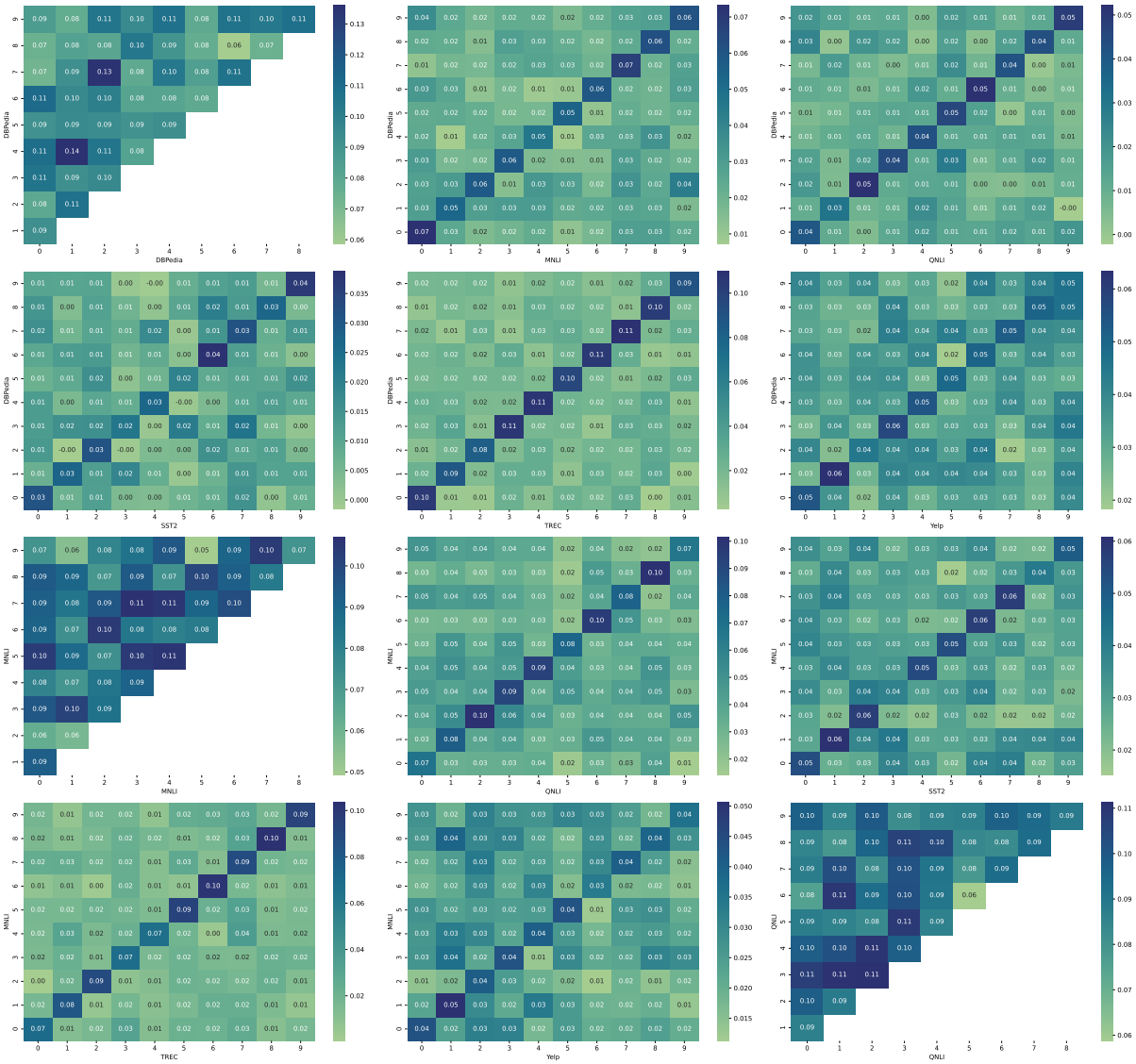

Figure 7: Comparisons of average cosine similarities of *task prompt vectors* for T5-base model. The averages are calculated similarly to Figure 6 but with task prompt vectors created from different task prompts. Each heatmap represents a different task combination. We calculate the cosine similarities for all combinations of 10 random initializations omitting the combinations of random initializations where cosine similarity is equal to 1 (single-task comparisons). Each heatmap is represented as a single field in Figure 3 by averaging all values. The x and y axes represent the number of random initializations.

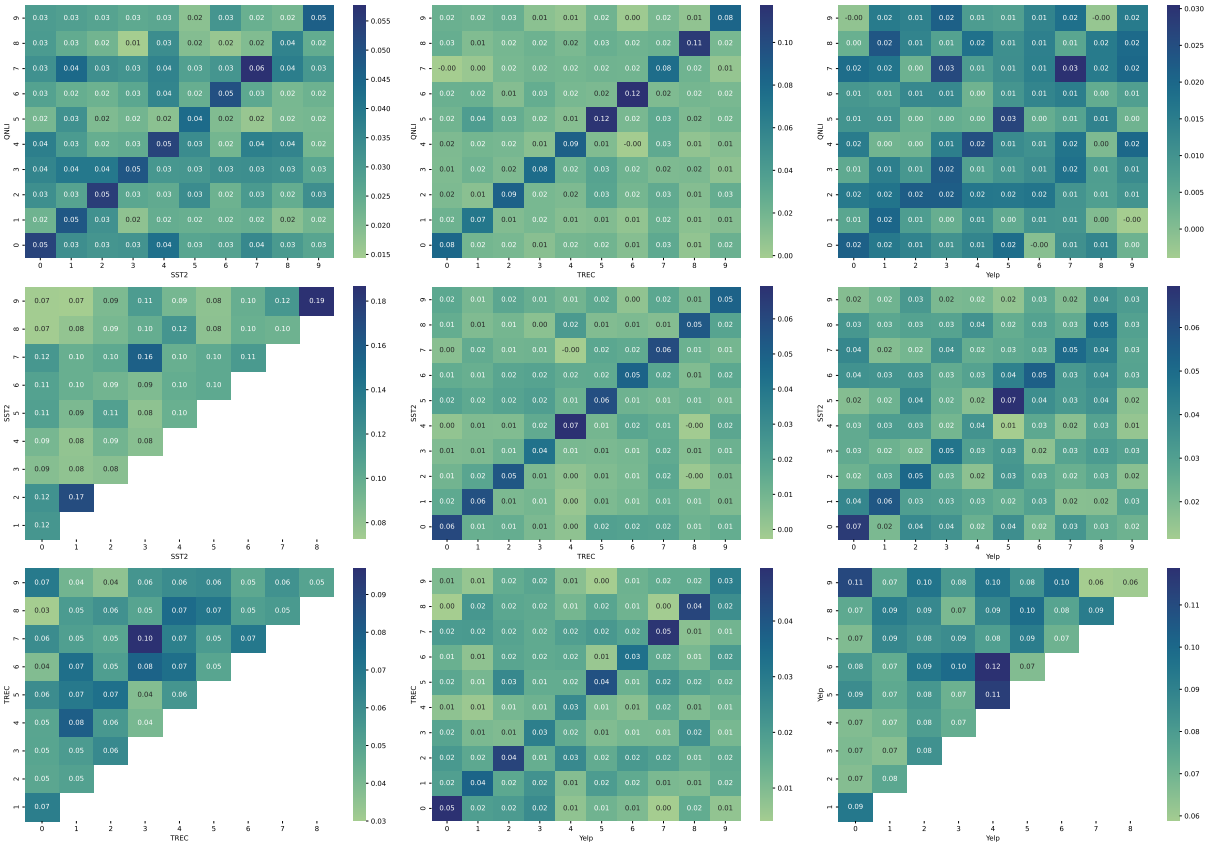

Figure 7 (cont.): Continuation of Figure 7 for additional tasks.

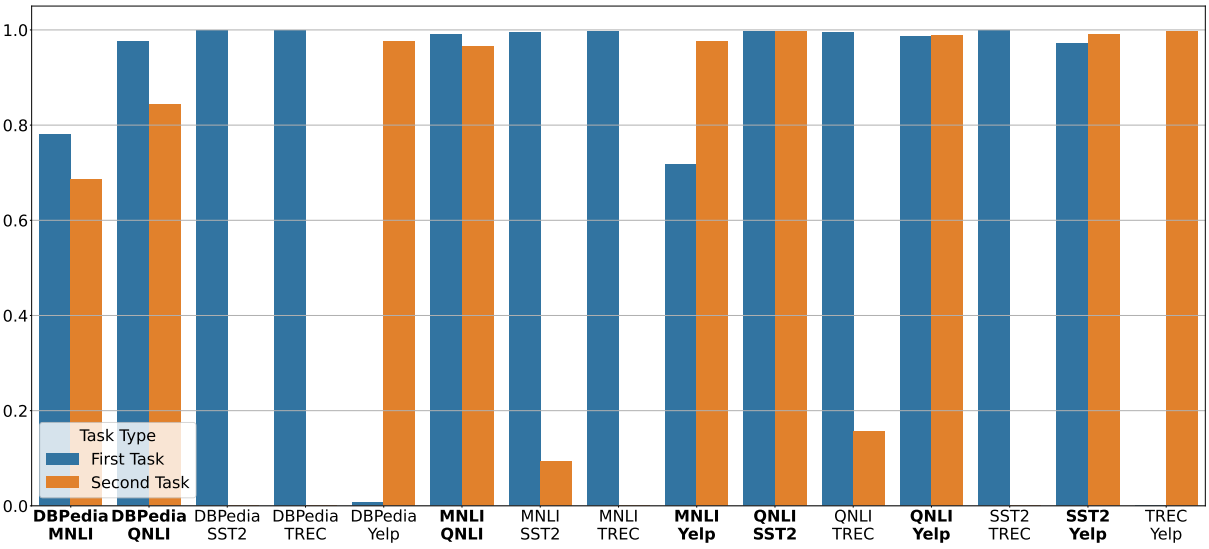

Figure 8: Comparison of relative exact match performance of combinations of task prompt vectors across averaged across 10 different random initializations and all task combinations. The results are relative to the original single-task performance (1 is the performance of single-task prompt tuning). The task combinations in bold are the combinations that achieved over 50% of single-task performance on both of the tasks.

SciTail (NLI)			AG News (Classification)			IMDB (Sentiment)		
Source tasks	F1		Source tasks	F1		Source tasks	F1	
	0 shots	100 shots		0 shots	100 shots		0 shots	100 shots
Random	$54.9_{6.6}$	$75.6_{0.5}$	Random	0_{0}	$50.4_{11.2}$	Random	$77.2_{9.6}$	$89.4_{0.4}$
MNLI (SPoT)	$\underline{70.4_{0.4}}$	$\underline{87.8_{0.9}}$	DBPedia (SPoT)	0_{0}	$\mathbf{83.4_{0.6}}^{*}$	SST2 (SPoT)	$88_{0.6}$	$90.2_{0.3}$
QNLI (SPoT)	$57.7_{13.1}$	$77.7_{1.3}$	TREC (SPoT)	0_{0}	$65.7_{5.6}$	Yelp (SPoT)	$90_{0.3}$	$90.3_{0.2}$
QNLI + MNLI (SPoT)	$70.4_{1.2}$	$87.7_{0.6}$	DBPedia + TREC (SPoT)	0_{0}	$82.1_{0.9}$	SST2 + Yelp (SPoT)	$\mathbf{90.8_{0.2}}$	$\mathbf{90.8_{0.2}}$
QNLI + MNLI (ATTEMPT)	$63.8_{4.2}$	83.6_{3}	DBPedia + TREC (ATTEMPT)	$\mathbf{11.5_{1.7}}$	$20.7_{2.8}$	SST2 + Yelp (ATTEMPT)	79.2_{6}	$89.4_{0.8}$
QNLI + MNLI (ours)	$\mathbf{71.5_{0.8}}^{*}$	$\mathbf{88.1_{0.9}}$	DBPedia + TREC (ours)	0_{0}	$\underline{83_{0.9}}$	SST2 + Yelp (ours)	$\underline{90.1_{0.5}}$	$\underline{90.4_{0.2}}$
SNLI (NLI)			**Yahoo Answers (Classification)**			**SST5 (Sentiment)**		
Source tasks	F1		Source tasks	F1		Source tasks	F1	
	0 shots	100 shots		0 shots	100 shots		0 shots	100 shots
Random	$46.5_{1.5}$	$47.6_{1.9}$	Random	0_{0}	$27.6_{10.6}$	Random	0_{0}	$83.2_{5.8}$
MNLI (SPoT)	$\underline{79.5_{0.3}}$	$\underline{80.8_{0.4}}$	DBPedia (SPoT)	0_{0}	$\mathbf{61.3_{1.1}}^{*}$	SST2 (SPoT)	$\mathbf{94_{0.3}}^{*}$	$\mathbf{93.9_{0.3}}^{*}$
QNLI (SPoT)	$47.1_{0.3}$	$49.1_{0.9}$	TREC (SPoT)	0_{0}	$36.5_{8.7}$	Yelp (SPoT)	$88.6_{0.8}$	$90.6_{0.5}$
QNLI + MNLI (SPoT)	$\mathbf{79.6_{0.2}}^{*}$	$\mathbf{81_{0.4}}^{*}$	DBPedia + TREC (SPoT)	0_{0}	60.7_{2}	SST2 + Yelp (SPoT)	$\underline{93.7_{0.5}}$	$\underline{93.8_{0.5}}$
QNLI + MNLI (ATTEMPT)	$78.5_{0.5}$	$79.6_{1.6}$	DBPedia + TREC (ATTEMPT)	$\mathbf{0.1_{0}}$	$8.1_{5.6}$	SST2 + Yelp (ATTEMPT)	$16.4_{4.5}$	37.8_{7}
QNLI + MNLI (ours)	$79.2_{1.4}$	$80.3_{0.3}$	DBPedia + TREC (ours)	0_{0}	$\underline{61.1_{0.9}}$	SST2 + Yelp (ours)	$89.9_{0.8}$	$91.5_{0.5}$

Table 4: Test results of training T5-base model with random, single- and multi-task soft-prompt transfer (SPoT), multi-task ATTEMPT, and our task prompt vectors on 0-shot and 100-shots of data for all of our observed source and target tasks. We show the initialization with different combinations for natural language inference classification, topic classification, and sentiment classification. The subscript represents the standard deviation from the average. The best results are bold, while the second-best results are underlined. The * in the superscript represents that the results are statistically significant from the second-best result, by two-sample Student's t-test (Student, 1908) or Welch's t-test (Welch, 1947).

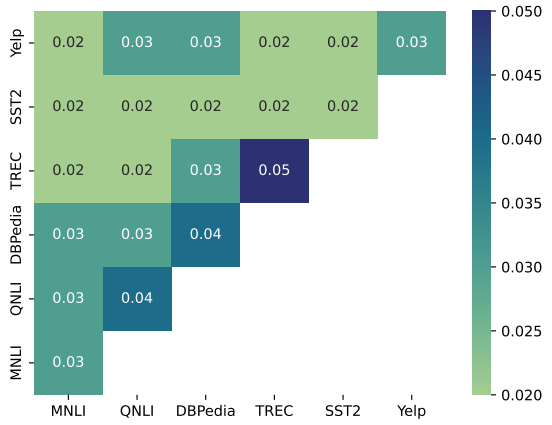

Figure 9: Comparison of average cosine similarities of *task prompts* fine-tuned on different tasks for LLaMa-3.1-8B-Instruct model. The average is calculated across all combinations of 3 random initializations (i.e., row QNLI column MNLI was calculated as the average of all cosine similarities between MNLI and QNLI task prompts for all random initialization combinations omitting the combinations where cosine similarity is equal to 1). The diagonal represents the cosine similarities of the same tasks and it represents the maximum value of cosine similarity across different random initializations.

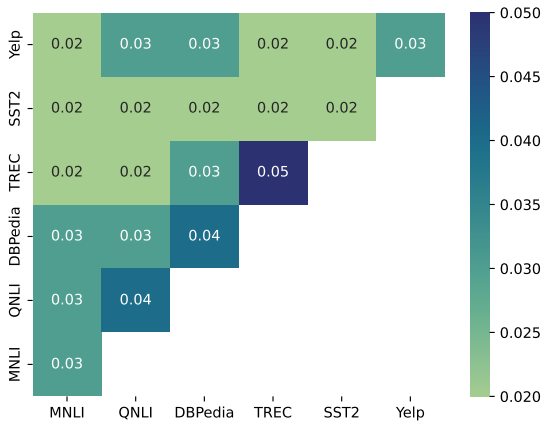

Figure 10: Comparison of average cosine similarities of *task prompt vectors*. The averages are calculated equivalently to Figure 9 but with task prompt vectors created from different task prompts.

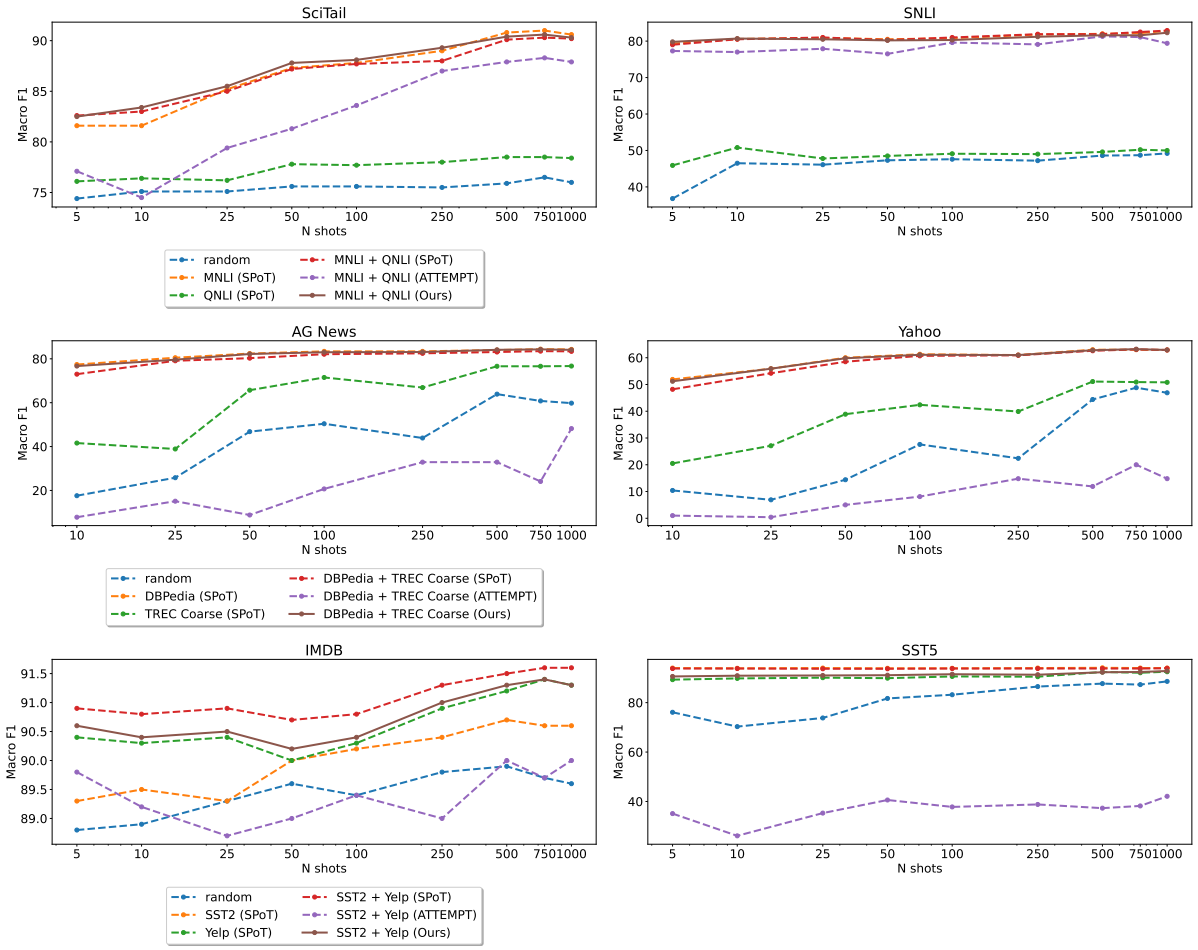

Figure 11: Test results of training T5-base model with random, single- and multi-task soft-prompt transfer (SPoT), multi-task ATTEMPT, and our task prompt vectors combination on increasing numbers of shots of data averaged over 10 different random initializations for all source and target tasks.