Automatic Multi-Label Prompting: Simple and Interpretable Few-Shot Classification

Anonymous ACL submission

Abstract

Prompt-based learning (i.e., prompting) is an emerging paradigm for exploiting knowledge learned by a pretrained language model. In this paper, we propose Automatic Multi-Label Prompting (AMuLaP), a simple yet effective method to automatically select label mappings for few-shot text classification with prompting. Our method exploits one-to-many label mappings and a statistics-based algorithm to select label mappings given a prompt template. Our experiments demonstrate that AMuLaP achieves competitive performance on the GLUE benchmark without human effort or external resources.[1]

1 Introduction

Since the release of GPT-3 (Brown et al., 2020), several studies have focused on exploiting pretrained language models with only a few training examples (Brown et al., 2020; Gao et al., 2021; Shin et al., 2020). These works demonstrate the potential of using natural language prompts to encourage the model to recall similar patterns in its training corpus and thus make accurate predictions. This setting of few-shot learning is closer to how humans learn to solve a task, often without many examples as in a traditional deep learning paradigm. The use of prompts can strengthen the explicit connection between input and output, helping the model exploit the knowledge learned from pretraining in a better way. Furthermore, recent works (Schick and Schütze, 2021a,b; Gao et al., 2021) show that prompts can also help the model generalize better in fine-tuning.

Prompt-based learning (i.e., prompting) aims to use a template to convert the original input into a prompt-based input with some unfilled masked tokens, and then use the pretrained language model to fill these masked tokens, and finally the tokens filled into these slots are mapped to the corresponding labels as the final output. In prompting, the design of prompts often plays an important role. Many attempts have been made in this emerging direction of *prompt engineering* (Shin et al., 2020; Gao et al., 2021). Meanwhile, finding a good mapping from the original task labels to tokens (i.e., *label engineering*) is also critical to few-shot performance, as found in Schick et al. (2020); Gao et al. (2021). However, manually assigning the label mapping requires human expertise with trial and error. One may argue that the same effort can be used to label more supervised data for a conventional deep learning pipeline. Thus, an efficient automatic label mapping method is desirable.

In this paper, we aim to design a method that can automatically find a good label mapping to save human effort from label engineering. We propose **A**utomatic **Mu**lti-**La**bel **P**rompting (AMuLaP), a simple yet effective method to tackle the label selection problem for few-shot classification. AMuLaP is a parameter-free statistical technique that can identify the label patterns from a few-shot training set given a prompt template. AMuLaP exploits multiple labels to suppress the noise and inherently extend the training set for prompt-based fine-tuning. Compared with a hand-crafted label mapping and previous works on automatic label mapping (Schick et al., 2020; Gao et al., 2021), AMuLaP achieves competitive performance despite being simpler and does not require access to the weights of the backbone model, or finetune an external pretrained language model for searching label mapping. We conduct extensive experiments and demonstrate the effectiveness of our method under multiple settings. Moreover, we attempt to scale AMuLaP with different sizes of the training set and find AMuLaP to work surprisingly well even with one or two shots. To understand the few-shot performance on different datasets, we investigate the relation between accuracy and inter-

[1] We will make all code publicly available upon acceptance.

class token distribution divergence, shedding light on the explainability of few-shot text classification.

2 Related Work

Discrete Prompts The release of GPT-3 (Brown et al., 2020) has led to interest in *prompting*, a new way to leverage pretrained language models (PLM). Brown et al. (2020) proposes an intuitive in-context learning paradigm by concatenating a few input and output examples and feeding them to the language model and let the model autoregressively generate answers for new examples. Recent works (Petroni et al., 2019; Davison et al., 2019; Jiang et al., 2020) design prompts to probe the factual and commonsense knowledge encoded within a PLM. Recent works (Schick and Schütze, 2021a,b; Gao et al., 2021) demonstrate that even smaller PLMs have similar few-shot learning capacity. Le Scao and Rush (2021) analyzes the effect of prompting and concludes that a single prompt may be worth 100 training examples in fine-tuning.

Instead of manually designing prompts (i.e., prompt engineering), some recent studies also explore automatic prompt generation. PETAL (Schick et al., 2020) augments Pattern Exploiting Training (PET, Schick and Schütze, 2021a,b) with automatically identified label words; Gao et al. (2021) uses re-ranking to find the best label words by fine-tuning a RoBERTa model on the candidates searched by RoBERTa, and using an external generation model for data augmentation of prompt templates; AutoPrompt (Shin et al., 2020) uses a gradient-based search to determine both prompts and label words. However, these methods require parameter updates with gradient descent, which is infeasible without access to the model weights (e.g., GPT-3). PET and its variants also require a large unlabeled set and need to be fine-tuned multiple times. AutoPrompt uses discretization techniques to approximately map a continuous vector back to tokens in the vocabulary (i.e., "vocablization"). These searched prompts and labels are often uninterpretable by humans. Different from these prior studies, our proposed AMuLaP is a simple and interpretable method for few-shot prompting that can work well with and without access to model weights. Concurrently to our work, Hu et al. (2021) propose a method that exploits an external knowledge base to find label mapping.

Continuous Prompts In parallel with text-based discrete prompts, there is also a line of work focused on tuning only a fraction of parameters of an LM with the help of continuous prompts (i.e., soft prompts). Zhong et al. (2021) and Qin and Eisner (2021) propose continuous prompts for knowledge probing by tuning some trainable vectors in the input sequence while fixing the rest of the input. Li and Liang (2021) applies a similar method for natural language generation and achieves comparable performance to fine-tuning while updating only 0.1% of model parameters. Lester et al. (2021) reveals that prompt tuning is more competitive when scaled up and can achieve identical performance to conventional fine-tuning when the model is large enough. Guo et al. (2021) introduces Q-Learning to optimize the soft prompt. Notably, different from discrete prompting, these works often use all training data to update model weights. Different from these works, AMuLaP is a discrete prompting method that has better interpretability and works well in the few-shot setting.

3 Prompting for Few-Shot Classification

We follow the setup in LM-BFF (Gao et al., 2021) for few-shot text classification. Given a pretrained language model $\mathcal{L}$, a task $\mathcal{D}$ and its defined label space $\mathcal{Y}$, we have n training examples per class for the training set $\mathcal{D}_{train}$. As pointed out in Perez et al. (2021), using the *full* development set may be misleading to claim a few-shot setting. Thus, we use a *few-shot* development set with the same size as the training set (i.e., $|\mathcal{D}_{train}| = |\mathcal{D}_{dev}|$), to be consistent with Gao et al. (2021) and constitute a "true few-shot" setting (Perez et al., 2021).

For an input example x (a single sentence or a sentence pair), we first use a task-specific template $\mathcal{T}$ to convert it to x', a token sequence with a `[MASK]` token. We then map the original label space to a set of selected words from the vocabulary, denoted as $\mathcal{M} : \mathcal{Y} \rightarrow \mathcal{V}'$. Some examples of $\mathcal{T}$ and $\mathcal{M}$ are shown in Table 1. Note that since we focus on automatically finding the label mapping $\mathcal{M}$, we use the manual templates $\mathcal{T}$ from Gao et al. (2021) throughout this paper. Since $\mathcal{L}$ is trained to complete the `[MASK]` token in an input sequence, we can directly make zero-shot prediction of the probability of class $y \in \mathcal{Y}$ by the masked language modeling:

$$p\left(y|x\right) = p\left(\texttt{[MASK]} = \mathcal{M}\left(y\right) \mid x'\right). \quad (1)$$

Task	Template	Class	Manual (2021)	Selected Labels by AMuLaP
MNLI	<S_1> ? [MASK] , <S_2>	entailment neutral contradiction	Yes Maybe No	Yes, Indeed, Also, Currently Historically, Suddenly, Apparently, And No, However, Instead, Unfortunately
SST-2	<S_1> It was [MASK] .	positive negative	great terrible	great, perfect, fun, brilliant terrible, awful, disappointing, not
QNLI	<S_1> ? [MASK] , <S_2>	entailment not_entailment	Yes No	Yes, Historically, Overall, Indeed Well, First, However, Unfortunately
RTE	<S_1> ? [MASK] , <S_2>	entailment not_entailment	Yes No	Yes, Today, Specifically, Additionally However, Ironically, Also, Indeed
MRPC	<S_1> [MASK] , <S_2>	equivalent not_equivalent	Yes No	</s>, Currently, Additionally, Today However, Meanwhile, Overall, Finally
QQP	<S_1> [MASK] , <S_2>	equivalent not_equivalent	Yes No	Or, So, Specifically, Actually Also, And, Finally, Well
CoLA	<S_1> This is [MASK] .	grammatical not_grammatical	correct incorrect	why, true, her, amazing it, ridiculous, interesting, sad

Table 1: The manual and automatically selected labels by AMuLaP. The templates used for prompting are from Gao et al. (2021).

Alternately, one can further fine-tune $\mathcal{L}$ with supervised pairs $\{x', \mathcal{M}(y)\}$ to achieve even better performance.

4 Automatic Multi-Label Prompting

4.1 Exploiting Multiple Labels

Selecting one label word can be insufficient for some complicated tasks, as mentioned in Schick et al. (2020). We also argue that selecting only one label (especially automatically) may bring noise. This can be resolved by introducing multiple label words. Schick et al. (2020) use multiple label combinations for PET (Schick and Schütze, 2021a) and ensemble them afterwards. We instead use a straightforward sum to consider multiple label words when making predictions. This design has a similar advantage of exploiting multiple labels without training and ensembling multiple models.

Instead of a one-to-one mapping from the original label space $\mathcal{Y}$ to $\mathcal{V}$, we map each $y \in \mathcal{Y}$ to its label word set $\mathcal{S}(y)$ of k words. We denote the mapping function as $\mathcal{M}' : \mathcal{Y} \rightarrow \mathcal{V}^k$. For class $y \in \mathcal{Y}$, the predicted probability is calculated as:

$$p(y|x) = \sum_{v \in \mathcal{S}(y)} p\left(\texttt{[MASK]} = v \mid x'\right) \quad (2)$$

Then, we can simply make predictions by selecting the label with the largest likelihood.

Similarly, if we need to fine-tune $\mathcal{L}$ with supervised pairs, instead of optimizing the cross-entropy loss between the gold label and a single token, we optimize the loss between the sum of the output probabilities of $\mathcal{S}(y)$ and the gold label with a cross-entropy loss:

$$l = -\sum_{x \in \mathcal{D}_{train}} \sum_{y \in \mathcal{Y}} \left[\mathbb{1}\left[y = \hat{y}\right] \cdot \log p\left(y|x\right)\right] \quad (3)$$

where $\hat{y}$ is the ground truth label for the input x and $p(y|x)$ is defined in Equation 2.

4.2 Automatic Label Selection

Finding a good label mapping $\mathcal{M}$ is non-trivial, especially when $\mathcal{M}'$ maps an original label to a set of label words instead of one. Selecting a good label mapping often requires significant human effort, including domain knowledge and trial-and-error. Previously, Schick and Schütze (2021a,b) both use hand-crafted label mappings while Schick et al. (2020) explores automatic label mapping searching but it still requires manual pre-filtering and significantly underperforms the manual mapping. (Gao et al., 2021) exploits a large pretrained masked language model (RoBERTa, Liu et al., 2019) to construct a pruned set of label words and then determine the final mapping by fine-tuning on all of them and selecting the best one with $\mathcal{D}_{dev}$. We introduce a new selection algorithm for label mapping that achieves competitive results compared to previous efforts.

We aim to achieve two goals: **(1) Selecting the most likely label mapping based on the training set.** For example, in a sentiment classification task, we would like to see positive words in the label set

of the "positive" class while negative words in the label set of the "negative" class. A simple solution is to select the k most likely tokens predicted for the [MASK] token in the training examples of each class y. However, in practice, we would find common words in more than one label set. For example, if we simply take the 10 most likely tokens for the SST-2 dataset (Socher et al., 2013), we would find "good" in both positive and negative label sets, although it is ranked second place in the positive set and ninth in the negative set. Thus, we want to make sure that **(2) Each token only belongs to at most one label set where it has the highest probability.** To ensure this, we have to iterate over the vocabulary and check that for every token. Then, we can truncate the candidate sets of each class and select the k most likely tokens from each set. The time complexity of this algorithm is $O(k \cdot |\mathcal{V}| \cdot |\mathcal{Y}|)$.

Formally, we select $\mathcal{M}' : \mathcal{Y} \rightarrow \mathcal{V}^k$ by the following steps:

1. For each $y_i \in \mathcal{Y}$, we iterate through all training samples $x_j \in \mathcal{D}_{train}$ whose ground truth label $\hat{y}_j = y_i$. We use $\mathcal{L}$ to predict the token probability of the [MASK] token and take the average of the predicted probabilities of the n examples to be $\mathbf{z}_i$, where $\mathbf{z}_i$ is a vector over the whole vocabulary.

2. For each $y_i \in \mathcal{Y}$, initialize an empty candidate token set $\tilde{\mathcal{S}}(y_i)$.

3. For each $v \in \mathcal{V}$ where $\mathcal{V}$ is the vocabulary of the model $\mathcal{L}$, we retrieve v's probability value z_i^v from $\mathbf{z}_i$ of each class.

4. We assign v to the most likely token set of the m-th class $\tilde{\mathcal{S}}(y_m)$ where $m = \text{argmax}_i\, z_i^v$.

5. For $y_i \in \mathcal{Y}$, we choose the top-k tokens from $\tilde{\mathcal{S}}(y_i)$ with the largest probability z_i^v and obtain the truncated word set $\mathcal{S}(y_i)$.

5 Experiments

5.1 Experimental Setting

Datasets We evaluate seven classification tasks of the GLUE benchmark (Wang et al., 2019). Specifically, we test on Microsoft Research Paraphrase Matching (MRPC) (Dolan and Brockett, 2005), Quora Question Pairs (QQP)[2] for Paraphrase Similarity Matching; Stanford Sentiment Treebank (SST-2) (Socher et al., 2013) for Sentiment Classification; Multi-Genre Natural Language Inference Matched (MNLI-m), Multi-Genre Natural Language Inference Mismatched (MNLI-mm) (Williams et al., 2018), Question Natural Language Inference (QNLI) (Rajpurkar et al., 2016) and Recognizing Textual Entailment (RTE) (Wang et al., 2019) for the Natural Language Inference (NLI) task; The Corpus of Linguistic Acceptability (CoLA) (Warstadt et al., 2019) for Linguistic Acceptability. We use the manual templates in Gao et al. (2021), as listed in Table 1. The metrics for each dataset are indicated in Table 2.

Baselines We compare our method to various baselines: **(1) Majority**: always predict the majority class in the test set. **(2) GPT-3-style in-context learning** (Brown et al., 2020): present a few examples to the language model and make it directly predict the next token as the prediction. **(3) Manual prompts**: we use the human-designed prompts in Gao et al. (2021). **(4) PETAL-CE** (Schick et al., 2020): the variant of PETAL using the cross-entropy metric. **(5) PETAL-LR** (Schick et al., 2020): the variant of PETAL using the likelihood ratio metric. **(6) Auto-L** (Gao et al., 2021): the automatic label searching method with an external pretrained language model, RoBERTa-large (Liu et al., 2019). The detailed description can be found in Appendix A. Note that the results of this baseline is different from those reported in Table 3 of Gao et al. (2021) since they search for both templates and label mapping whereas we fix the templates and search for the label mapping alone, for the sake of fair comparison. We use the officially released code and same hyperparameters for this baseline.

Task Setup We closely follow the setup in Gao et al. (2021). We sample n training examples and n development examples per class. We set $k = 16$ throughout all experiments. We use RoBERTa-large (Liu et al., 2019) as the backbone LM $\mathcal{L}$. For each reported result, we measure average performance across 5 different randomly sampled $\mathcal{D}_{train}$ and $\mathcal{D}_{dev}$ splits. Following Gao et al. (2021), the original development split of each dataset is used as the test set in our experiments. We also report the standard deviation for each result. To fairly compare with different baselines, we consider the following three settings:

- **Setting 1**: We only use $\mathcal{D}_{train}$ alone for both label selection and tuning k. The parameters of $\mathcal{L}$ are not updated. $\mathcal{D}_{dev}$ is not used. This setting is for fair comparison with *In-context learning*.

[2] https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs

	MNLI (acc)	MNLI-mm (acc)	SST-2 (acc)	QNLI (acc)	RTE (acc)	MRPC (F1)	QQP (F1)	CoLA (Matt.)	Avg.
Baselines									
Majority	32.7	33.0	50.9	49.5	52.7	81.2	0.0	0.0	37.5
Manual Label 0-shot (2021)	50.8	51.7	83.6	50.8	51.3	61.9	49.7	2.0	50.2
Full Fine-tuning	89.8	89.5	95.0	93.3	80.9	91.4	81.7	62.6	85.5
Setting 1: *$\mathcal{D}_{train}$ only; No parameter update.*									
In-context learning (2020)	**52.0** (0.7)	**53.4** (0.6)	84.8 (1.3)	**53.8** (0.4)	**60.4** (1.4)	45.7 (6.0)	36.1 (5.2)	-1.5 (2.4)	48.1 (2.3)
AMuLaP (ours)	50.8 (2.1)	52.3 (1.8)	**86.9** (1.6)	53.1 (2.8)	58.9 (7.9)	**56.3** (5.0)	**60.2** (2.7)	**2.3** (1.4)	**52.6** (3.2)
Setting 2: *$\mathcal{D}_{train}$ + $\mathcal{D}_{dev}$; No parameter update.*									
PETAL-CE (2020)	48.8 (2.6)	49.7 (2.3)	75.6 (7.2)	49.5 (0.0)	**63.5** (3.3)	28.9 (39.6)	59.2 (0.0)	1.3 (3.0)	47.1 (7.3)
PETAL-LR (2020)	38.6 (2.0)	38.4 (2.1)	85.3 (3.3)	53.3 (3.6)	54.7 (6.4)	28.0 (38.5)	55.6 (2.8)	1.5 (3.4)	44.4 (7.8)
Auto-L (2021)	41.6 (5.4)	42.3 (6.2)	84.3 (3.3)	57.9 (3.9)	61.9 (7.5)	**67.7** (7.9)	55.5 (5.0)	1.2 (4.8)	51.6 (5.5)
AMuLaP (ours)	50.8 (2.1)	52.2 (1.9)	87.0 (1.5)	53.5 (2.3)	59.1 (7.4)	56.7 (5.7)	**61.5** (1.7)	2.6 (1.8)	52.9 (3.1)
Auto-L + AMuLaP (ours)	**52.9** (3.0)	**54.2** (2.7)	**90.1** (0.4)	**57.9** (2.6)	59.9 (5.2)	66.0 (3.0)	59.4 (2.3)	**2.7** (5.7)	**55.4** (3.1)
Setting 3: *$\mathcal{D}_{train}$ + $\mathcal{D}_{dev}$; Prompt-based fine-tuning.*									
Fine-tuning	45.8 (6.4)	47.8 (6.8)	81.4 (3.8)	60.2 (6.5)	54.4 (3.9)	76.6 (2.5)	60.7 (4.3)	**33.9** (14.3)	57.6 (6.1)
Manual Label FT (2021)	68.3 (2.3)	70.5 (1.9)	92.7 (0.9)	64.5 (4.2)	69.1 (3.6)	74.5 (5.3)	65.5 (5.3)	9.3 (7.3)	64.3 (3.9)
PETAL-CE FT (2020)	57.5 (3.2)	57.7 (2.6)	92.6 (1.0)	50.5 (0.0)	68.6 (6.5)	32.1 (42.5)	66.7 (3.2)	3.8 (8.4)	53.7 (8.4)
PETAL-LR FT (2020)	64.0 (6.5)	65.9 (6.4)	92.9 (1.7)	65.5 (6.8)	63.3 (7.7)	77.7 (3.9)	65.7 (4.2)	11.9 (7.5)	63.4 (5.6)
Auto-L FT (2021)	64.8 (4.7)	67.3 (4.3)	**93.5** (0.5)	**69.8** (3.0)	67.4 (3.9)	76.2 (4.8)	66.4 (4.5)	23.2 (17.1)	66.1 (5.4)
AMuLaP FT (ours)	**70.6** (2.7)	**72.5** (2.4)	93.2 (0.7)	65.1 (5.9)	65.9 (6.3)	**79.3** (4.0)	**69.1** (2.5)	18.3 (9.4)	**66.8** (4.2)
Auto-L + AMuLaP FT (ours)	68.5 (2.2)	71.1 (2.3)	93.4 (1.0)	69.6 (1.1)	**69.4** (4.0)	75.5 (5.6)	66.4 (3.0)	14.2 (14.0)	66.0 (4.2)

Table 2: Experimental results under three settings with RoBERTa-large as $\mathcal{L}$. For few-shot settings, n is set to 16 per class. We report the average of 5 runs along with their standard deviation in the parentheses.

- **Setting 2**: We use $\mathcal{D}_{train}$ for label selection and an additional $\mathcal{D}_{dev}$ for k tuning. The parameters of $\mathcal{L}$ are not updated. This setting is for fair comparison with Auto-L (Gao et al., 2021) and PETAL (Schick et al., 2020).

- **Setting 3**: We use $\mathcal{D}_{train}$ and $\mathcal{D}_{dev}$ in the same way as Setting 2 but fine-tune the parameters of the language model $\mathcal{L}$. This setting is for fair comparison with conventional fine-tuning, prompt-based fine-tuning with manual prompts, Auto-L (Gao et al., 2021) and PETAL (Schick et al., 2020).

Implementation Details We implement AMuLaP based on Hugging Face Transformers (Wolf et al., 2020). When selecting k, if there are multiple k with identical performance (which happens occasionally given there are only 16 examples for each class in $\mathcal{D}_{dev}$), we always choose the largest k. For Settings 1 and 2, we search k over $\{1, 2, 4, \ldots, 1024\}$. Note that for settings that do not update the parameters of $\mathcal{L}$, search over k is fast, as we only need to run the model once and cache the distribution of the `[MASK]` token. For prompt-based fine-tuning (Setting 3), where we fine-tune the model $\mathcal{L}$, we search k in a smaller space $\{1, 2, 4, 8, 16\}$ due to the increased computational overhead. Following (Gao et al., 2021), we grid search the learning rate from {1e-5, 2e-5, 5e-5} and batch size from {2, 4, 8}.

5.2 Experimental Results

We demonstrate experimental results under three settings in Table 2. Under Setting 1, AMuLaP outperforms GPT-3-style in-context learning by 4.5 in terms of the average score and outperforms zero-shot inference with manually designed labels by 2.4. Under Setting 2, compared to variants of PETAL (Schick et al., 2020), AMuLaP has an advantage of 5.8 and 8.5 in terms of the average score over CE and LR, respectively. Notably, AMuLaP even outperforms Auto-L by 1.3 without using any external model or data. Additionally, we attempt to replace the predicted token distribution of AMuLaP with the validation score of all fine-tuned assignments (Gao et al., 2021).[3] With the help of many trials in automatic search, AMuLaP outperforms Auto-L by a considerable margin of 3.8 in terms of the average score, verifying the versatility of our multi-label mechanism and label selection algorithm. Under Setting 3, AMuLaP FT outperforms all baselines including Auto-L. Generally speaking, methods with parameter update (Setting 3) have

[3]The validation scores of all fine-tuned assignments are obtained on $\mathcal{D}_{dev}$, as described in Gao et al. (2021). No external data used. All of these we use are from `https://github.com/princeton-nlp/LM-BFF/tree/main/auto_label_mapping`.

Class	PETAL-CE (Schick et al., 2020)	PETAL-LR (Schick et al., 2020)
positive	amazing, great, brilliant, perfect, fun, wonderful, beautiful, fantastic, awesome, not	superb, fearless, acclaimed, addictive, visionary, immersive, irresistible, timely, unforgettable, gripping
negative	not, awful, fun, funny, terrible, great, amazing, hilarious, awesome, good	annoying, insulting, meaningless, lame, shitty, humiliating, childish, stupid, embarrassing, irritating
Class	**Auto-L** (Gao et al., 2021)	**AMuLaP** (ours)
positive	exquisite, perfection, effective, fabulous, intense inspiring, spectacular, sublime, astounding, thrilling	great, perfect, fun, brilliant, amazing, good, wonderful, beautiful, excellent, fantastic
negative	embarrassing, boring, frustrating, ridiculous, awkward silly, nothing, disgusting, ugly, confusing	terrible, awful, disappointing, not, horrible, obvious, funny, inevitable, bad, boring

Table 3: Most likely label mapping for the SST-2 dataset obtained by PETAL (Schick et al., 2020), Auto-L (Gao et al., 2021) and our AMuLaP. Suitable labels annotated by the human annotator are underlined.

	MNLI (acc)	MNLI-mm (acc)	SST-2 (acc)	QNLI (acc)	RTE (acc)	MRPC (F1)	QQP (F1)	CoLA (Matt.)	Avg.
Setting 2: *$\mathcal{D}_{train}$ + $\mathcal{D}_{dev}$; No parameter update.*									
AMuLaP	**50.8** (2.1)	**52.2** (1.9)	87.0 (1.5)	53.5 (2.3)	**59.1** (7.4)	**56.7** (5.7)	**61.5** (1.7)	**2.6** (1.8)	**52.9** (3.1)
w/o dedup.	45.4 (2.7)	46.5 (2.5)	**87.9** (1.0)	**53.8** (3.0)	54.6 (6.0)	66.7 (12.3)	57.2 (2.1)	2.5 (4.2)	51.8 (4.2)
$k = 1$	46.5 (2.7)	48.4 (2.6)	68.8 (12.0)	51.9 (1.6)	58.8 (12.7)	55.0 (4.8)	59.2 (0.0)	5.6 (2.1)	49.3 (4.8)
Setting 3: *$\mathcal{D}_{train}$ + $\mathcal{D}_{dev}$; Prompt-based fine-tuning.*									
AMuLaP FT	**70.6** (2.7)	**72.5** (2.4)	**93.2** (0.7)	65.1 (5.9)	**65.9** (6.3)	79.3 (4.0)	**69.1** (2.5)	18.3 (9.4)	**66.8** (4.2)
w/o dedup.	56.9 (5.4)	58.2 (5.2)	92.8 (0.9)	50.6 (0.4)	57.1 (10.8)	79.2 (3.6)	55.0 (26.0)	5.6 (7.1)	56.9 (7.4)
$k = 1$	67.7 (4.1)	69.8 (3.8)	92.6 (1.0)	**65.9** (5.2)	63.1 (8.0)	**80.2** (3.8)	66.7 (3.2)	19.3 (15.5)	65.7 (5.6)
random $\mathcal{M}'$	58.8 (6.2)	61.1 (6.2)	92.1 (2.1)	62.1 (7.1)	57.0 (11.2)	74.7 (9.2)	60.8 (5.8)	**31.0** (13.9)	62.2 (7.7)
random $\mathcal{M}'$ ($k = 1$)	52.6 (7.8)	55.4 (8.3)	89.0 (4.9)	65.2 (4.5)	55.2 (6.2)	73.4 (10.6)	60.7 (3.7)	17.3 (14.7)	58.6 (7.6)

Table 4: Experimental results for the ablation study. We report the average of 5 runs along with their standard deviation in the parentheses.

better performance than those that do not require access to parameters. On all tasks except CoLA, AMuLaP outperforms direct few-shot fine-tuning, suggesting that prompting is a promising method for exploiting large pretrained LMs.

6 Analysis

6.1 Case Study

As shown in Table 3, we list the 10 most likely label mappings output by PETAL (Schick et al., 2020), Auto-L (Gao et al., 2021) and AMuLaP for the SST-2 dataset, respectively. We shuffle the labels from each model and ask a human annotator to annotate whether they are suitable mappings. PETAL-CE suffers from incorrect mappings for "negative" while PETAL-LR occasionally outputs vague labels. AMuLaP achieves interpretability that is competitive to automatic labels obtained by a fine-tuned pretrained language model, measured by the human agreement ratio. Although AMuLaP outputs three labels that are rated not suitable by the human annotator, it should be noted that all three tokens are ranked low in the candidate set. Thus, introducing top-k truncation can resolve the problem. Additionally, we would like to highlight that AMuLaP mainly collects common words while other methods prefer rare words. This may explain why AMuLaP works well, especially for the non-finetuning settings.

6.2 Ablation Study

As shown in Table 4, we evaluate the effect of each design choice on the GLUE benchmark. For both non-finetuning and prompt-based fine-tuning settings, our deduplication algorithm can effectively improve the overall performance by 1.1 and 9.9 in terms of the GLUE average score, respectively. Notably, deduplication is especially important for prompt-based fine-tuning since if the same label maps to two classes, optimization would be difficult due to the contradiction of supervision signals. Also, our multi-label strategy is shown to be effective at improving the average GLUE scores by 3.6 and 1.1 for non-finetuning and fine-tuning settings, respectively. Moreover, a random label mapping often leads to lower performance than a la-

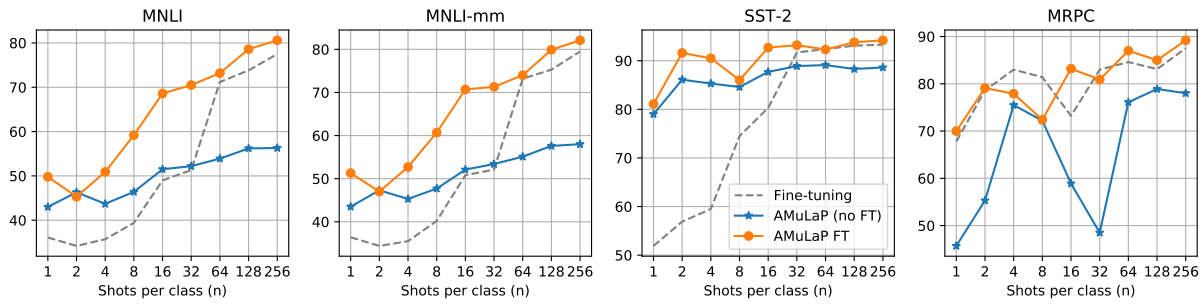

Figure 1: Comparison of AMuLaP, AMuLaP FT and fine-tuning on MNLI, SST and MRPC with different n for the training set and the development set.

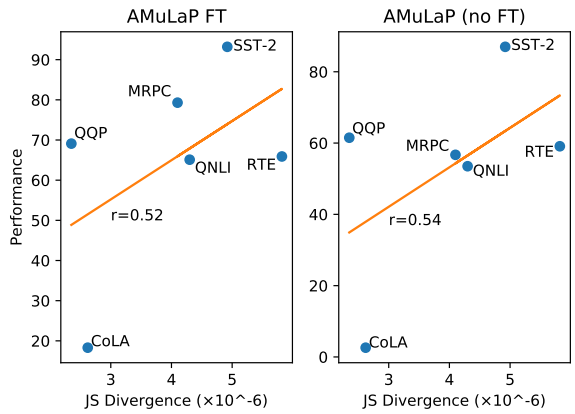

Figure 2: Correlation between inter-class JS divergence and performance for two-class tasks.

bel mapping selected based on the training set. An interesting exception is that for CoLA, the random mapping outperforms all label selection methods in Table 2 (both manual and automatic) and is close to the fine-tuning baseline. We will discuss this more in Section 6.4.

6.3 Scaling Few-Shot Learning

Le Scao and Rush (2021) explore the scaling law of PET (Schick and Schütze, 2021a) when using more examples for training. Similarly, in this section, we aim to test how AMuLaP scales to different training set sizes n. Figure 1 illustrates how standard fine-tuning and our AMuLaP with non-finetuning and fine-tuning compare as n increases. For MNLI and SST-2 task, AMuLaP outperforms standard fine-tuning when we use no more than 16 training examples for non-finetuning and fine-tuning setting. When using more than 16 training examples, AMuLaP under fine-tuning setting still outperforms standard fine-tuning. For an easier task like SST-2, although only 32 training examples are used, the performance of our AMuLaP with non-finetuning and fine-tuning is close to saturation and can be comparable to standard fine-tuning on the entire dataset. For a harder task like MNLI, although the performance of AMuLaP under non-finetuning setting gradually becomes saturated as n increases, AMuLaP under fine-tuning settings continues to improve as n increases and continues to outperform the standard fine-tuning. For MRPC, although the performance of our AMuLaP and standard fine-tuning fluctuate as n increases, in general, AMuLaP with fine-tuning can still achieve comparable performance to standard fine-tuning. In addition, the results demonstrate the effectiveness of AMuLaP especially for extreme few-shot settings. With only one example, AMuLaP achieves decent performance while standard fine-tuning is close to random.

6.4 Understanding Few-Shot Performance

As shown in Table 1, AMuLaP seems to be able to find good labels for some datasets while failing on others. Intuitively, this phenomenon should correspond to classification performance. To quantitatively understand the relation between the quality of found labels and the final performance, we design a meta-experiment. For every two-class dataset (all GLUE datasets except three-class MNLI), we calculate the JS divergence between the average predicted token probabilities $\mathbf{z}_0$ and $\mathbf{z}_1$. This metric measures how different the language model $\mathcal{L}$ considers the examples from two classes. This can be regarded as the "confidence" of $\mathcal{L}$ to distinguish between the two classes. If the model can easily distinguish examples from one class with the other, we would expect the divergence to be large, and vice versa.

We illustrate the relation between inter-class JS divergence and the performance of AMuLaP on each dataset in Figure 2. The correlation coeffi-

cients r between the two variables are 0.52 and 0.54 (0.14 and 0.19 if ignoring CoLA) for AMuLaP with and without prompt-based fine-tuning, respectively. This observation suggests that the performance can degrade if the model cannot distinguish examples of different classes and thus fail to find suitable labels. As we analyze, AMuLaP fails on CoLA since the backbone model (RoBERTa) is trained on corpora that contain noisy and possibly ungrammatical text (e.g., OpenWebText) (Liu et al., 2019). Thus, it is naturally tolerant to grammatical errors that are key to distinguishing between the two classes in CoLA, the dataset for linguistic acceptability. As shown in Table 4, a random mapping can outperform all automatic and manual mappings by a large margin. This finding reveals that some tasks may be naturally unsuitable for prompting, which warrants further investigation.

7 Discussion

Why Does AMuLaP Work? Schick et al. (2020) argues that one single label sometimes cannot represent all examples in a class, and thus multiple labels are needed. However, we find this explanation insufficient for understanding the mechanism behind the improved performance with multiple labels. Under a few-shot setting, the limited number of training examples n and complex training procedure of the backbone model $\mathcal{L}$ can often bring noise to both automatic label selection and inference. One example is the meaningless `</s>` (end-of-sequence marker) label found by AMuLaP, as shown in Table 1. This is due to the format processing in the pretraining of $\mathcal{L}$. Allowing multiple labels can resolve mishaps like this and thus improve the final performance.

Moreover, when selecting multiple labels in fine-tuning, it is equivalent to training on an augmented training set, as multiple labels increase the overall size of the supervision pairs $(x, \hat{y})$. To verify this guess, we test the fine-tuning performance of a random mapping with different labels selected. We find that for random mapping, more labels (i.e., a larger k) often leads to better performance. This suggests our guess may be correct. However, we do not observe significant improvement when continuing increasing k with labels selected by AMuLaP. As we analyze, increasing k harms the overall quality of selected labels and thus overrides the benefit of a larger k. In general, we do not observe a clear law for choosing the best k for AMuLaP. As mentioned before, k can influence both the overall quality of labels (in both ways) and the training procedure (for fine-tuning). Thus, for the optimal performance, we find it essential to search k with a development set.

Limitations and Future Directions In this paper, we only focus on the selection of the label mapping with a fixed prompt template. There is more to explore when considering the prompt template at the same time. Similar to our paper, previous works (Schick et al., 2020; Gao et al., 2021) separately search for a prompt template $\mathcal{T}$ and the label mapping $\mathcal{M}$. However, these two variables are closely related and greedily search for the best template $\mathcal{T}$ then the best mapping under $\mathcal{T}$ may be suboptimal. Jointly searching for $\mathcal{T}$ and $\mathcal{M}$ could be a promising direction for future research.

More broadly, we would like to point out some limitation and contradictions within current few-shot prompting techniques. There is a natural contradiction between performance and access to the model weights. Brown et al. (2020) highlights few-shot prompting as a way to mitigate their decision to not release the model weights. However, as shown in our Table 2, with the same backbone model $\mathcal{L}$, GPT-3-style in-context learning and other methods that do not access the model weights generally underperform those with access to the model weights by a large margin. Also, in-context learning cannot handle more training examples due to the maximum length limit of the model while AMuLaP without fine-tuning gets saturated quickly, as shown in Figure 1.

In addition, complicated prompting techniques are not practically useful for real-world scenarios. For most techniques, the required effort for finding good templates and label mappings, and sometimes training models outweighs the cost of simply labeling more training examples. As shown in Figure 1, 64 examples per class are enough to bring the performance of standard fine-tuning to the same level of prompting. Although recent works on automatic selection of prompts and label mappings are making meaningful contribution to the practicability of few-shot learning, we believe more work should be done to simplify the learning procedure and eliminate human effort while achieving good performance.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Joe Davison, Joshua Feldman, and Alexander M. Rush. 2019. Commonsense knowledge mining from pretrained models. In *EMNLP-IJCNLP*, pages 1173–1178. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *IWP@IJCNLP*.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. In *ACL-IJCNLP*. Association for Computational Linguistics.

Han Guo, Bowen Tan, Zhengzhong Liu, Eric P Xing, and Zhiting Hu. 2021. Text generation with efficient (soft) q-learning. *arXiv preprint arXiv:2106.07704*.

Shengding Hu, Ning Ding, Huadong Wang, Zhiyuan Liu, Juanzi Li, and Maosong Sun. 2021. Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification. *arXiv preprint arXiv:2108.02035*.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know. *Trans. Assoc. Comput. Linguistics*, 8:423–438.

Teven Le Scao and Alexander M. Rush. 2021. How many data points is a prompt worth? In *NAACL-HLT*, pages 2627–2636. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

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*.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. 2021. True few-shot learning with language models. *arXiv preprint arXiv:2105.11447*.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. 2019. Language models as knowledge bases? In *EMNLP-IJCNLP*, pages 2463–2473. Association for Computational Linguistics.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying lms with mixtures of soft prompts. In *NAACL-HLT*, pages 5203–5212. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*.

Timo Schick, Helmut Schmid, and Hinrich Schütze. 2020. Automatically identifying words that can serve as labels for few-shot text classification. In *COLING*, pages 5569–5578. International Committee on Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021a. Exploiting cloze-questions for few-shot text classification and natural language inference. In *EACL*, pages 255–269. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021b. It's not just size that matters: Small language models are also few-shot learners. In *NAACL-HLT*, pages 2339–2352. Association for Computational Linguistics.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *EMNLP*, pages 4222–4235. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

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

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *TACL*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*.

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 *EMNLP (Demos)*, pages 38–45. Association for Computational Linguistics.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual probing is [MASK]: learning vs. learning to recall. In *NAACL-HLT*, pages 5017–5033. Association for Computational Linguistics.

A Automatic Label Selection (Auto-L) in LM-BFF

Gao et al. (2021) proposed a method to automatically construct a label word mapping $\mathcal{M}$ given a fixed template $\mathcal{T}$. They construct a pruned label word set $\mathcal{V}^c \in \mathcal{V}$ of the top k words based on their conditional likehood using the pretrained language model $\mathcal{L}$ for each class $c \in \mathcal{Y}$. They take $\mathcal{V}^c$ as

$$\underset{v \in \mathcal{V}}{\text{Top-}k} \left\{ \sum_{x \in \mathcal{D}^c_{\text{train}}} \log p\left(\texttt{[MASK]} = v \mid \mathcal{T}\left(x\right)\right) \right\}$$

where $\mathcal{D}^c_{\text{train}} \subset \mathcal{D}_{\text{train}}$ denotes the subset of all examples of class c. They find the top n assignments over the pruned space that maximize zero-shot accuracy on $\mathcal{D}_{\text{train}}$ to further narrow the search space. Then they fine-tune n assignments and re-rank to find the best label words mapping on $\mathcal{D}_{\text{dev}}$.