Making Pre-trained Language Models Better Few-shot Learners

Tianyu Gao[†*] **Adam Fisch**[‡*] **Danqi Chen**[†]
[†]Princeton University [‡]Massachusetts Institute of Technology
{tianyug,danqic}@cs.princeton.edu
fisch@csail.mit.edu

Abstract

The recent GPT-3 model (Brown et al., 2020) achieves remarkable few-shot performance solely by leveraging a natural-language prompt and a few task demonstrations as input context. Inspired by their findings, we study few-shot learning in a more practical scenario, where we use smaller language models for which fine-tuning is computationally efficient. We present LM-BFF—better few-shot fine-tuning of language models[1]—a suite of simple and complementary techniques for fine-tuning language models on a small number of annotated examples. Our approach includes (1) prompt-based fine-tuning together with a novel pipeline for automating prompt generation; and (2) a refined strategy for dynamically and selectively incorporating demonstrations into each context. Finally, we present a systematic evaluation for analyzing few-shot performance on a range of NLP tasks, including classification and regression. Our experiments demonstrate that our methods combine to dramatically outperform standard fine-tuning procedures in this low resource setting, achieving up to 30% absolute improvement, and 11% on average across all tasks. Our approach makes minimal assumptions on task resources and domain expertise, and hence constitutes a strong task-agnostic method for few-shot learning.[2]

1 Introduction

The GPT-3 model (Brown et al., 2020) has made waves in the NLP community by demonstrating astounding few-shot capabilities on myriad language understanding tasks. Given only a *natural language prompt* and a few *demonstrations* of the task, GPT-3 is able to make accurate predictions without updating any of the weights of its underlying language model. However, while remarkable, GPT-3 consists of 175B parameters, which makes it challenging to use in most real-wold applications.

In this work, we study a more practical scenario in which we only assume access to a moderately-sized language model such as BERT (Devlin et al., 2019) or RoBERTa (Liu et al., 2019), and a small number of examples (i.e., a *few-shot* setting), which we can use to fine-tune the weights of the language model. This setting is appealing as (1) such models can be trained on typical research hardware; (2) few-shot settings are realistic, as it is generally both easy to acquire a few annotations (e.g., 32 examples) and efficient to train on them; and (3) updating parameters typically leads to better performance. Inspired by GPT-3's findings, we propose several novel strategies for expanding its few-shot learning abilities to our setting, considering both classification and—for the first time—regression.

First, we follow the route of *prompt-based* prediction, first developed by the GPT series (Radford et al., 2018, 2019; Brown et al., 2020) for zero-shot prediction and recently studied by PET (Schick and Schütze, 2021a,b) for fine-tuning. Prompt-based prediction treats the downstream task as a (masked) language modeling problem, where the model directly generates a textual response (referred to as a *label word*) to a given prompt defined by a task-specific *template* (see Figure 1(c)). Finding the right prompts, however, is an art—requiring both domain expertise and an understanding of the language model's inner workings. Even if significant effort is invested, manual prompts are likely to be suboptimal. We address this issue by introducing automatic prompt generation, including a pruned brute-force search to identify the best working label words, and a novel decoding objective to automatically generate templates using the generative T5 model (Raffel et al., 2020)—all of which only require the few-shot training data. This allows us

[*]The first two authors contributed equally.
[1]Alternatively, language models' best friends forever.
[2]Our implementation is publicly available at https://github.com/princeton-nlp/LM-BFF.

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 3816–3830
August 1–6, 2021.

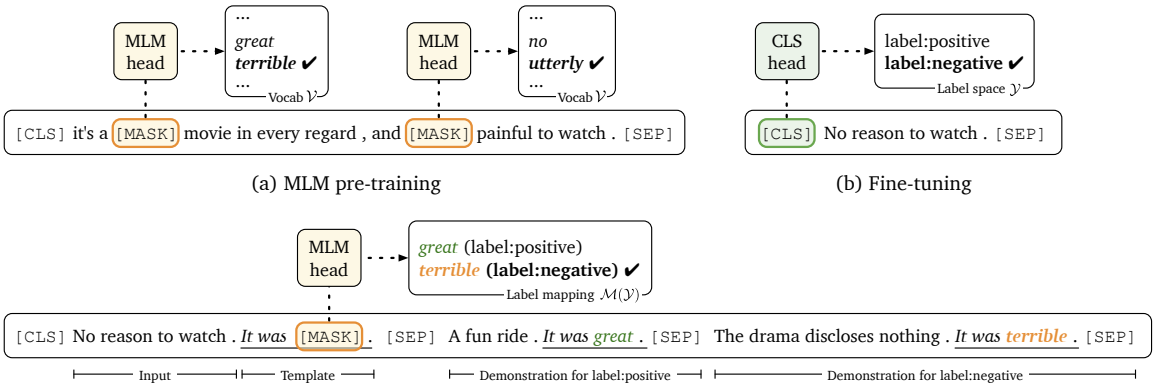

Figure 1: An illustration of (a) masked language model (MLM) pre-training, (b) standard fine-tuning, and (c) our proposed LM-BFF using prompt-based fine-tuning with demonstrations. The underlined text is the task-specific *template*, and colored words are *label words*.

to cheaply obtain effective prompts that match or outperform our manually chosen ones.

Second, we adopt the idea of incorporating demonstrations as additional context. GPT-3's naive "in-context learning" paradigm picks up to 32 randomly sampled examples, and concatenates them with the input. This method is not guaranteed to prioritize the most informative demonstrations, and mixing random examples from different classes together creates long contexts which can be hard to learn from. Additionally, the number of usable demonstrations is bounded by the model's maximum input length. We develop a more refined strategy, where, for each input, we randomly sample a *single* example at a time from *each* class to create multiple, minimal demonstration *sets*. We also devise a novel sampling strategy that pairs inputs with similar examples, thereby providing the model with more discriminative comparisons.

We present a systematic evaluation for analyzing few-shot performance on 8 single-sentence and 7 sentence-pair NLP tasks. We observe that given a small number of training examples, (1) prompt-based fine-tuning largely outperforms standard fine-tuning; (2) our automatic prompt search method matches or outperforms manual prompts; and (3) incorporating demonstrations is effective for fine-tuning, and boosts few-shot performance. Together, these simple-yet-effective methods contribute towards a dramatic improvement across the tasks we evaluate on, and we obtain gains up to 30% absolute improvement (11% on average) compared to standard fine-tuning. For instance, we find that a RoBERTa-large model achieves around 90% accuracy on most binary sentence classification tasks, while only relying on 32 training examples. We refer to our approach as LM-BFF, better few-shot fine-tuning of language models: a strong, task-agnostic method for few-shot learning.

2 Related Work

Language model prompting. The GPT series (Radford et al., 2018, 2019; Brown et al., 2020) fueled the development of prompt-based learning, and we follow many of its core concepts. We are also greatly inspired by the recent PET work (Schick and Schütze, 2021a,b), although they mainly focus on a semi-supervised setting where a large set of unlabeled examples are provided. We only use a few annotated examples as supervision, and also explore automatically generated prompts and fine-tuning with demonstrations. Furthermore, we deviate from their evaluation by providing a more rigorous framework, as we will discuss in §3. Finally, there is a large body of work on prompting for mining knowledge from pre-trained models (Trinh and Le, 2018; Petroni et al., 2019; Davison et al., 2019; Talmor et al., 2020, *inter alia*). Different from these works, we focus on leveraging prompting for fine-tuning on downstream tasks.

Automatic prompt search. Schick and Schütze (2021a) and Schick et al. (2020) explore ways of identifying label words automatically, however, none of these results lead to better performance compared to hand-picked ones. In contrast, our method searches over both templates and label words, and is able to match or outperform our manual prompts. Several other attempts have been made in addition—yet these approaches either op-

erate in limited domains, such as finding patterns to express specific relations (Jiang et al., 2020), or require a large number of examples for gradient-guided search (Shin et al., 2020; Zhong et al., 2021). Our approach aims to develop general-purpose search methods that rely only on a few annotations.

Fine-tuning of language models. A number of recent studies have focused on better methods for fine-tuning language models (Howard and Ruder, 2018; Dodge et al., 2020; Lee et al., 2020; Zhang et al., 2021). These works mainly focus on optimization and regularization techniques to stabilize fine-tuning. Here we use standard optimization techniques, and instead mainly focus our efforts on better prompt-based fine-tuning in a more extreme few-shot setting. We anticipate that results of these studies are largely complementary to ours.

Few-shot learning. Broadly speaking, our setting is also connected to other few-shot learning paradigms in NLP, including (1) semi-supervised learning (Miyato et al., 2017; Xie et al., 2020; Chen et al., 2020), where a set of unlabeled examples are given; (2) meta-learning (Yu et al., 2018; Han et al., 2018; Bansal et al., 2020a,b; Bao et al., 2020), where a set of auxiliary tasks are given; and (3) intermediate training (Phang et al., 2018; Yin et al., 2020), where a related, intermediate task is given. We deviate from these settings by making minimal assumptions about available resources: we only assume a few annotated examples and a pre-trained language model. Our focus is on understanding how far we can push without any other advantages.

3 Problem Setup

Task formulation. In this work, we assume access to a pre-trained language model $\mathcal{L}$ that we wish to fine-tune on a task $\mathcal{D}$ with a label space $\mathcal{Y}$. For the task, we only assume K training examples *per class*[3] for the task's training set $\mathcal{D}_{\text{train}}$, such that the total number of examples is $K_{\text{tot}} = K \times |\mathcal{Y}|$, and $\mathcal{D}_{\text{train}} = \{(x_{\text{in}}^i, y^i)\}_{i=1}^{K_{\text{tot}}}$. Our goal is then to develop task-agnostic learning strategies that generalize well to an unseen test set $(x_{\text{in}}^{\text{test}}, y^{\text{test}}) \sim \mathcal{D}_{\text{test}}$. For model selection and hyper-parameter tuning, we assume a development set $\mathcal{D}_{\text{dev}}$, of the same size as the few-shot training set, i.e., $|\mathcal{D}_{\text{dev}}| = |\mathcal{D}_{\text{train}}|$. This distinction is important: using a larger development set confers a significant advantage (see our experiments in Appendix A), and subverts our initial goal of learning from limited data.[4] For all of the following experiments (unless specified otherwise), we take $\mathcal{L} =$ RoBERTa-large and $K = 16$.

Evaluation datasets. We conduct a systematic study across 8 single-sentence and 7 sentence-pair English tasks, including 8 tasks from the GLUE benchmark (Wang et al., 2019), SNLI (Bowman et al., 2015), and 6 other popular sentence classification tasks (SST-5, MR, CR, MPQA, Subj, TREC). All of the dataset details are provided in Appendix B. For *single-sentence* tasks, the goal is to make a prediction based on an input sentence $x_{\text{in}} = x_1$, such as whether a movie review is positive or not. For *sentence-pair* tasks, the goal is to take a pair of input sentences $x_{\text{in}} = (x_1, x_2)$ and predict the relationship between them. We also interchangeably refer to the inputs as $<S_1>$ or ($<S_1>$, $<S_2>$). Note that we mainly use SST-2 and SNLI for pilot experiments and model development, making it close to a true few-shot setting, at least for all the other datasets we evaluate on.

Evaluation protocol. Systematically evaluating few-shot performance can be tricky. It is well-known that fine-tuning on small datasets can suffer from instability (Dodge et al., 2020; Zhang et al., 2021), and results may change dramatically given a new split of data. To account for this, we measure average performance across 5 different randomly sampled $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{dev}}$ splits. This issue has also been discussed in Schick and Schütze (2021b)—they suggest using a fixed set of training examples. We argue that sampling multiple splits gives a more robust measure of performance, and a better estimate of the variance. We also observe that hyper-parameters can make a significant difference, thus we sweep multiple hyper-parameters for each data sample, and take the best setting as measured on the $\mathcal{D}_{\text{dev}}$ of that sample (see Appendix C.1).

4 Prompt-based Fine-tuning

Given a masked language model $\mathcal{L}$, we first convert input x_{in} to a token sequence $\tilde{x}$, and the language model $\mathcal{L}$ then maps $\tilde{x}$ to a sequence of hidden vectors $\{\mathbf{h}_k \in \mathbb{R}^d\}$. During standard fine-tuning, we usually take $\tilde{x}_{\text{single}} =$ `[CLS]` x_1 `[SEP]` or $\tilde{x}_{\text{pair}} =$ `[CLS]` x_1 `[SEP]` x_2 `[SEP]`. For down-

[3] For regression, we partition the data into two "classes" according to being above or below the median value.

[4] In contrast, Schick and Schütze (2021a,b) do not use a development set, and adopt a set of hyper-parameters based on practical considerations. This is akin to "shooting in the dark" on a setting that we show can have unintuitive outcomes.

Task	Template	Label words
SST-2	$<S_1>$ It was `[MASK]` .	positive: great, negative: terrible
SST-5	$<S_1>$ It was `[MASK]` .	v.positive: great, positive: good, neutral: okay, negative: bad, v.negative: terrible
MR	$<S_1>$ It was `[MASK]` .	positive: great, negative: terrible
CR	$<S_1>$ It was `[MASK]` .	positive: great, negative: terrible
Subj	$<S_1>$ This is `[MASK]` .	subjective: subjective, objective: objective
TREC	`[MASK]` : $<S_1>$	abbreviation: Expression, entity: Entity, description: Description human: Human, location: Location, numeric: Number
COLA	$<S_1>$ This is `[MASK]` .	grammatical: correct, not_grammatical: incorrect
MNLI	$<S_1>$? `[MASK]` , $<S_2>$	entailment: Yes, netural: Maybe, contradiction: No
SNLI	$<S_1>$? `[MASK]` , $<S_2>$	entailment: Yes, netural: Maybe, contradiction: No
QNLI	$<S_1>$? `[MASK]` , $<S_2>$	entailment: Yes, not_entailment: No
RTE	$<S_1>$? `[MASK]` , $<S_2>$	entailment: Yes, not_entailment: No
MRPC	$<S_1>$ `[MASK]` , $<S_2>$	equivalent: Yes, not_equivalent: No
QQP	$<S_1>$ `[MASK]` , $<S_2>$	equivalent: Yes, not_equivalent: No
STS-B	$<S_1>$ `[MASK]` , $<S_2>$	y_u: Yes, y_l: No

Table 1: Manual templates and label words that we used in our experiments. STS-B is a regression task (§4.2).

stream classification tasks with a label space $\mathcal{Y}$, we train a task-specific head, $\text{softmax}(\mathbf{W}_o\mathbf{h}_{\texttt{[CLS]}})$, by maximizing the log-probability of the correct label, where $\mathbf{h}_{\texttt{[CLS]}}$ is the hidden vector of `[CLS]`, and $\mathbf{W}_o \in \mathbb{R}^{|\mathcal{Y}|\times d}$ is a set of randomly initialized parameters introduced at the start of fine-tuning. Similarly, for a regression task, we can introduce $\mathbf{w}_o \in \mathbb{R}^d$ and optimize the mean squared error between $\mathbf{w}_o \cdot \mathbf{h}_{\texttt{[CLS]}}$ and the gold label. In either case, the number of new parameters can be substantial—for example, a simple binary classification task will introduce 2,048 new parameters for a RoBERTa-large model—making it challenging to learn from a small amount of annotated data (e.g., 32 examples).

An alternative approach to solving this problem is *prompt-based fine-tuning*, in which $\mathcal{L}$ is directly tasked with "auto-completing" natural language prompts. For instance, we can formulate a binary sentiment classification task using a prompt with input x_1 (e.g., "*No reason to watch it* .") as:

$$x_{\text{prompt}} = \texttt{[CLS]}\ x_1 \text{ It was } \texttt{[MASK]}\ .\ \texttt{[SEP]}$$

and let $\mathcal{L}$ decide whether it is more appropriate to fill in "*great*" (positive) or "*terrible*" (negative) for `[MASK]`. We now formalize this approach for classification and regression (§4.1 and §4.2), and discuss the importance of prompt selection (§4.3).

4.1 Classification

Let $\mathcal{M}: \mathcal{Y} \to \mathcal{V}$ be a mapping from the task label space to individual words[5] in the vocabulary $\mathcal{V}$ of $\mathcal{L}$. Then for each x_{in}, let the manipulation $x_{\text{prompt}} = \mathcal{T}(x_{\text{in}})$ be a *masked language modeling* (MLM) input which contains one `[MASK]` token. In this way, we can treat our task as an MLM, and model the probability of predicting class $y \in \mathcal{Y}$ as:

$$\begin{aligned} p(y \mid x_{\text{in}}) &= p\left(\texttt{[MASK]} = \mathcal{M}(y) \mid x_{\text{prompt}}\right) \\ &= \frac{\exp\left(\mathbf{w}_{\mathcal{M}(y)} \cdot \mathbf{h}_{\texttt{[MASK]}}\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\mathbf{w}_{\mathcal{M}(y')} \cdot \mathbf{h}_{\texttt{[MASK]}}\right)}, \end{aligned} \quad (1)$$

where $\mathbf{h}_{\texttt{[MASK]}}$ is the hidden vector of `[MASK]` and $\mathbf{w}_v$ denotes the pre-softmax vector corresponding to $v \in \mathcal{V}$. When supervised examples $\{(x_{\text{in}}, y)\}$ are available, $\mathcal{L}$ can be fine-tuned to minimize the cross-entropy loss. It is important to note that this approach re-uses the pre-trained weights $\mathbf{w}_v$ and does not introduce any new parameters. It also reduces the gap between pre-training and fine-tuning, making it more effective in few-shot scenarios.

4.2 Regression

We assume the same basic setup as in classification, but treat the label space $\mathcal{Y}$ as a bounded interval $[v_l, v_u]$. Inspired by Mettes et al. (2019), we model the problem as an interpolation between two opposing poles, $\{y_l, y_u\}$, with values v_l and v_u respectively. For instance, we can formulate our previous sentiment analysis task as a regression problem in the range $[0, 1]$, where we slide between "*terrible*" ($v_l = 0$) and "*great*" ($v_u = 1$). In this way, we can express y as a *mixture model*:

$$y = v_l \cdot p(y_l \mid x_{\text{in}}) + v_u \cdot p(y_u \mid x_{\text{in}}), \quad (2)$$

where $p(y_u \mid x_{\text{in}})$ is the probability of y_u, and $p(y_l \mid x_{\text{in}}) = 1 - p(y_u \mid x_{\text{in}})$. Then we define

[5] More generally, we can consider a one-to-many mapping $\mathcal{M}: \mathcal{Y} \to 2^{|\mathcal{Y}|}$ in which we map labels to sets of words. However, we did not find significant gains in our experiments.

Template	Label words	Accuracy
SST-2 (positive/negative)		mean (std)
$<S_1>$ It was [MASK] .	great/terrible	**92.7 (0.9)**
$<S_1>$ It was [MASK] .	good/bad	92.5 (1.0)
$<S_1>$ It was [MASK] .	cat/dog	91.5 (1.4)
$<S_1>$ It was [MASK] .	dog/cat	86.2 (5.4)
$<S_1>$ It was [MASK] .	terrible/great	83.2 (6.9)
Fine-tuning	-	81.4 (3.8)
SNLI (entailment/neutral/contradiction)		mean (std)
$<S_1>$? [MASK] , $<S_2>$	Yes/Maybe/No	**77.2 (3.7)**
$<S_1>$. [MASK] , $<S_2>$	Yes/Maybe/No	76.2 (3.3)
$<S_1>$? [MASK] $<S_2>$	Yes/Maybe/No	74.9 (3.0)
$<S_1>$ $<S_2>$ [MASK]	Yes/Maybe/No	65.8 (2.4)
$<S_2>$? [MASK] , $<S_1>$	Yes/Maybe/No	62.9 (4.1)
$<S_1>$? [MASK] , $<S_2>$	Maybe/No/Yes	60.6 (4.8)
Fine-tuning	-	48.4 (4.8)

Table 2: The impact of templates and label words on prompt-based fine-tuning ($K = 16$).

$\mathcal{M}: \{y_l, y_u\} \rightarrow \mathcal{V}$, and model $p(y_u \mid x_{\text{in}})$ the same as Eq. (1). We fine-tune $\mathcal{L}$ to minimize the KL-divergence between the inferred $p(y_u \mid x_{\text{in}})$ and the observed mixture weight, $(y - v_l)/(v_u - v_l)$.

4.3 Manual prompts: the good and the bad

The key challenge is to construct the template $\mathcal{T}$ and label words $\mathcal{M}(\mathcal{Y})$—we refer to these two together as a *prompt* $\mathcal{P}$. Previous works (Schick and Schütze, 2021a,b) hand-craft both the templates and label words, which usually requires domain expertise and trial-and-error. Table 1 summarizes manual templates and label words chosen for each dataset in our experiments. These templates and label words were designed by intuition, and by considering formats used in previous literature.

To better understand what constitutes a good template or label word, we conduct a pilot study on SST-2 and SNLI. Table 2 shows that different prompts can lead to substantial differences in final accuracy. Specifically, when a template is fixed, the better the label words match the "semantic classes", the better the final accuracy is (*great*/*terrible* > *good*/*bad* > *cat*/*dog*). In extreme cases where we swap plausible label words (e.g., *terrible*/*great*), we achieve the worst overall performance.[6] Furthermore, with the same set of label words, even a small change in the template can make a difference. For example, for SNLI, if we put [MASK] at the end, or swap sentence order, we observe a >10% drop. The above evidence clearly underlines the importance of selecting good templates and label words. Searching for prompts, however, is hard, as the search space can be very large—especially for the template. Even worse, we only have a few examples to use to guide our search, which can easily overfit. We will address these issues next.

[6]It is unclear, however, why RoBERTa thinks that "cat" is more positive than "dog". The authors tend to disagree.

5 Automatic Prompt Generation

We now explore principled ways of automating the search process for label words (§5.1) and templates (§5.2). Our goals are to reduce the human involvement required to design prompts, and to find more optimal settings than those that we manually choose. Here, we assume a classification task, but the process for regression is analogous.

5.1 Automatic selection of label words

We first study how to construct a label word mapping $\mathcal{M}$ that maximizes accuracy on $\mathcal{D}_{\text{dev}}$ after fine-tuning, given a fixed template $\mathcal{T}$. Naively searching all possible assignments, however, is (1) generally intractable, as the search space is exponential in the number of classes; and (2) prone to overfitting, as we will tend to uncover spurious correlations given only a few annotations. As a simple solution, for each class $c \in \mathcal{Y}$, we construct a pruned set $\mathcal{V}^c \subset \mathcal{V}$ of the top k vocabulary words based on their conditional likelihood using the initial $\mathcal{L}$. That is, let $\mathcal{D}^c_{\text{train}} \subset \mathcal{D}_{\text{train}}$ be the subset of all examples of class c. We take $\mathcal{V}^c$ as

$$\underset{v \in \mathcal{V}}{\text{Top-}k} \left\{ \sum_{x_{\text{in}} \in \mathcal{D}^c_{\text{train}}} \log P_{\mathcal{L}}\Big(\texttt{[MASK]} = v \mid \mathcal{T}(x_{\text{in}}) \Big) \right\}, \quad (3)$$

where $P_{\mathcal{L}}$ denotes the output probability distribution of $\mathcal{L}$. To further narrow down the search space, we find the top n assignments over the pruned space that maximize zero-shot accuracy on $\mathcal{D}_{\text{train}}$ (both n and k are hyper-parameters, see Appendix C.2). Then we fine-tune all top n assignments, and re-rank to find the best one using $\mathcal{D}_{\text{dev}}$. This approach is similar to the automatic verbalizer search methods in Schick and Schütze (2021a); Schick et al. (2020), except that we use a much simpler search process (brute-force) and also apply re-ranking—which we find to be quite helpful.

5.2 Automatic generation of templates

Next, we study how to generate a diverse set of templates $\{\mathcal{T}\}$ automatically from a fixed set of label words $\mathcal{M}(\mathcal{Y})$. To address this challenging problem, we propose to use T5 (Raffel et al., 2020),

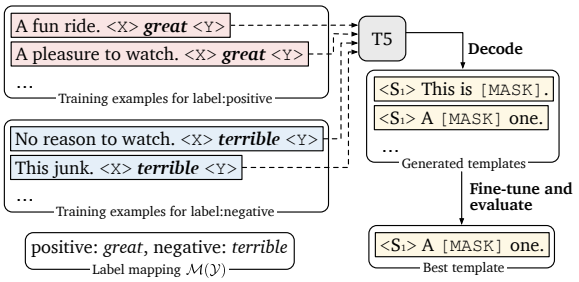

Figure 2: Our approach for template generation.

a large pre-trained text-to-text Transformer. T5 is pre-trained to fill in missing spans (replaced by T5 mask tokens, e.g., <X> or <Y>) in its input. For example, given the input "*Thank you <X> me to your party <Y> week*", T5 is trained to generate "*<X> for inviting <Y> last <Z>*", meaning that "*for inviting*" is the replacement for <X> and "*last*" is the replacement for <Y>. This is well suited for prompt generation: we can simply take input sentences from $\mathcal{D}_{\text{train}}$ and let the T5 model construct the template $\mathcal{T}$, without having to specify a predefined number of tokens for it.

Given an input example $(x_{\text{in}}, y) \in \mathcal{D}_{\text{train}}$, we consider the following simple conversions, denoted as $\mathcal{T}_{\text{g}}(x_{\text{in}}, y)$, for formulating the T5 model inputs:[7]

$$\begin{aligned} \texttt{<}S_1\texttt{>} &\longrightarrow \texttt{<X>}\ \mathcal{M}(y)\ \texttt{<Y>}\ \texttt{<}S_1\texttt{>}, \\ \texttt{<}S_1\texttt{>} &\longrightarrow \texttt{<}S_1\texttt{>}\ \texttt{<X>}\ \mathcal{M}(y)\ \texttt{<Y>}, \\ \texttt{<}S_1\texttt{>}, \texttt{<}S_2\texttt{>} &\longrightarrow \texttt{<}S_1\texttt{>}\ \texttt{<X>}\ \mathcal{M}(y)\ \texttt{<Y>}\ \texttt{<}S_2\texttt{>}. \end{aligned}$$

As shown in Figure 2, we rely on the T5 model to fill in the placeholders. When decoding, our goal here is to find an output that can work well for *all* examples in $\mathcal{D}_{\text{train}}$, i.e., the output template $\mathcal{T}$ that maximizes $\sum_{(x_{\text{in}},y)\in\mathcal{D}_{\text{train}}} \log P_{\text{T5}}(\mathcal{T} \mid \mathcal{T}_{\text{g}}(x_{\text{in}}, y))$, where P_{T5} denotes the output probability distribution of T5. It can be decomposed according to:

$$\sum_{j=1}^{|\mathcal{T}|} \sum_{(x_{\text{in}},y)\in\mathcal{D}_{\text{train}}} \log P_{\text{T5}}\big(t_j \mid t_1, ..., t_{j-1}, \mathcal{T}_{\text{g}}(x_{\text{in}}, y)\big), \quad (4)$$

where $(t_1, \ldots, t_{|\mathcal{T}|})$ are the template tokens.

We use beam search to decode multiple template candidates. Concretely, we use a wide beam width (e.g., 100) to cheaply obtain a large set of diverse templates. We then fine-tune each generated template on $\mathcal{D}_{\text{train}}$ and use $\mathcal{D}_{\text{dev}}$ to either pick the single template with the best performance (Table 3), or the top k templates to use as an ensemble (Table 4). Though it might appear to be expensive to fine-tune the model on each individual template, this is fast in practice due to the small size of $\mathcal{D}_{\text{train}}$, and is also fully automated: making it easy to use, compared to manually tuning prompts for each dataset.

[7] We consider putting the label word both before and after the input sentence for single-sentence tasks. However, we find that it is always better to put the label words in the middle (between the two sentences) for sentence-pair tasks.

6 Fine-tuning with Demonstrations

In this section, we study whether we can leverage demonstrations when *fine-tuning* medium-sized LMs, and find better ways to exploit them.

6.1 Training examples as demonstrations

GPT-3's naive approach to in-context learning simply involves concatenating the input with up to 32 examples randomly drawn from the training set. This approach is suboptimal as (1) the number of available demonstrations is bounded by the model's maximum input length;[8] and (2) mixing numerous random examples from different classes together creates extremely long contexts which can be hard to leverage, especially for a smaller model. To address these issues, we propose a simpler solution: at each training step, we randomly sample *one*[9] example $(x_{\text{in}}^{(c)}, y^{(c)}) \in \mathcal{D}_{\text{train}}$ from each class, convert it into $\mathcal{T}(x_{\text{in}}^{(c)})$ with [MASK] replaced by $\mathcal{M}(y^{(c)})$—we denote this as $\tilde{\mathcal{T}}(x_{\text{in}}^{(c)}, y^{(c)})$—and then concatenate them with x_{in} (Figure 1(c)):

$$\mathcal{T}(x_{\text{in}}) \oplus \tilde{\mathcal{T}}(x_{\text{in}}^{(1)}, y^{(1)}) \oplus \cdots \oplus \tilde{\mathcal{T}}(x_{\text{in}}^{(|\mathcal{Y}|)}, y^{(|\mathcal{Y}|)}).$$

Here $\oplus$ denotes concatenation of input sequences. During both training and inference we sample multiple demonstration sets for each x_{in}. Note that both x_{in} and demonstration examples are sampled from the same set $\mathcal{D}_{\text{train}}$ during training. At testing time, we still sample demonstration sets from $\mathcal{D}_{\text{train}}$ and ensemble predictions across all sets.

6.2 Sampling similar demonstrations

We observe that controlling the construction of the demonstration examples $\{(x_{\text{in}}^{(c)}, y^{(c)})\}$ is crucial for good final performance. For example, if the set of contrastive demonstrations $x_{\text{in}}^{(c)}$ are all dramatically different—from each other, or from the query x_{in}—then it becomes challenging for the language model to decipher meaningful patterns. As a result, the model may simply ignore

[8] GPT-3 uses a context size of 2,048 while most smaller language models (e.g., RoBERTa) have a context size of 512.

[9] We also explored sampling multiple examples per class, but did not observe any improvements.

	SST-2 (acc)	SST-5 (acc)	MR (acc)	CR (acc)	MPQA (acc)	Subj (acc)	TREC (acc)	CoLA (Matt.)
Majority†	*50.9*	*23.1*	*50.0*	*50.0*	*50.0*	*50.0*	*18.8*	*0.0*
Prompt-based zero-shot‡	83.6	35.0	80.8	79.5	67.6	51.4	32.0	2.0
"GPT-3" in-context learning	84.8 (1.3)	30.6 (0.9)	80.5 (1.7)	87.4 (0.8)	63.8 (2.1)	53.6 (1.0)	26.2 (2.4)	-1.5 (2.4)
Fine-tuning	81.4 (3.8)	43.9 (2.0)	76.9 (5.9)	75.8 (3.2)	72.0 (3.8)	90.8 (1.8)	88.8 (2.1)	**33.9** (14.3)
Prompt-based FT (man)	92.7 (0.9)	47.4 (2.5)	87.0 (1.2)	90.3 (1.0)	84.7 (2.2)	91.2 (1.1)	84.8 (5.1)	9.3 (7.3)
+ demonstrations	92.6 (0.5)	**50.6** (1.4)	86.6 (2.2)	90.2 (1.2)	**87.0** (1.1)	**92.3** (0.8)	87.5 (3.2)	18.7 (8.8)
Prompt-based FT (auto)	92.3 (1.0)	49.2 (1.6)	85.5 (2.8)	89.0 (1.4)	85.8 (1.9)	91.2 (1.1)	88.2 (2.0)	14.0 (14.1)
+ demonstrations	**93.0** (0.6)	49.5 (1.7)	**87.7** (1.4)	**91.0** (0.9)	86.5 (2.6)	91.4 (1.8)	**89.4** (1.7)	21.8 (15.9)
Fine-tuning (full)†	*95.0*	*58.7*	*90.8*	*89.4*	*87.8*	*97.0*	*97.4*	*62.6*
	MNLI (acc)	**MNLI-mm** (acc)	**SNLI** (acc)	**QNLI** (acc)	**RTE** (acc)	**MRPC** (F1)	**QQP** (F1)	**STS-B** (Pear.)
Majority†	*32.7*	*33.0*	*33.8*	*49.5*	*52.7*	*81.2*	*0.0*	-
Prompt-based zero-shot‡	50.8	51.7	49.5	50.8	51.3	61.9	49.7	-3.2
"GPT-3" in-context learning	52.0 (0.7)	53.4 (0.6)	47.1 (0.6)	53.8 (0.4)	60.4 (1.4)	45.7 (6.0)	36.1 (5.2)	14.3 (2.8)
Fine-tuning	45.8 (6.4)	47.8 (6.8)	48.4 (4.8)	60.2 (6.5)	54.4 (3.9)	76.6 (2.5)	60.7 (4.3)	53.5 (8.5)
Prompt-based FT (man)	68.3 (2.3)	70.5 (1.9)	77.2 (3.7)	64.5 (4.2)	69.1 (3.6)	74.5 (5.3)	65.5 (5.3)	71.0 (7.0)
+ demonstrations	**70.7** (1.3)	**72.0** (1.2)	**79.7** (1.5)	**69.2** (1.9)	68.7 (2.3)	77.8 (2.0)	**69.8** (1.8)	73.5 (5.1)
Prompt-based FT (auto)	68.3 (2.5)	70.1 (2.6)	77.1 (2.1)	68.3 (7.4)	**73.9** (2.2)	76.2 (2.3)	67.0 (3.0)	75.0 (3.3)
+ demonstrations	70.0 (3.6)	**72.0** (3.1)	77.5 (3.5)	68.5 (5.4)	71.1 (5.3)	**78.1** (3.4)	67.7 (5.8)	**76.4** (6.2)
Fine-tuning (full)†	*89.8*	*89.5*	*92.6*	*93.3*	*80.9*	*91.4*	*81.7*	*91.9*

Table 3: Our main results using RoBERTa-large. †: full training set is used (see dataset sizes in Table B.1); ‡: no training examples are used; otherwise we use $K = 16$ (per class) for few-shot experiments. We report mean (and standard deviation) performance over 5 different splits (§3). Majority: majority class; FT: fine-tuning; man: manual prompt (Table 1); auto: automatically searched templates (§5.2); "GPT-3" in-context learning: using the in-context learning proposed in Brown et al. (2020) with RoBERTa-large (no parameter updates).

the context, or even get confused by the additional examples. To address this issue, we devise a simple strategy in which we only sample examples that are semantically close to x_{in}. Specifically, we use a pre-trained SBERT (Reimers and Gurevych, 2019) model to obtain embeddings for all input sentences (for sentence-pair tasks, we use the concatenation of the two sentences). Here we just feed the raw sentences without the templates into SBERT. For each query x_{in} and each label $c \in \mathcal{Y}$, we sort all training instances with the label $x \in \mathcal{D}^c_{\text{train}}$ by their similarity score to the query $\cos(\mathbf{e}(x_{\text{in}}), \mathbf{e}(x))$, and only sample from the top $r = 50\%$ instances for each class to use as demonstrations.

7 Experiments

We present our main results, and address several research questions pertaining to our LM-BFF approach. Implementation details are in Appendix C.

7.1 Main results

We use a RoBERTa-large model and set $K = 16$ in our experiments. A comparison of using RoBERTa vs BERT can be found in Appendix D. For automatic prompt search, in our main table we report automatic template search only (which consistently performs the best, see Table 5). To put our results in perspective, we compare to a number of baselines, namely (1) standard fine-tuning in our few-shot setting; (2) standard fine-tuning using the full training set; (3) simply taking the most frequent class (measured on the full training set); (4) prompt-based zero-shot prediction where we take our manual prompts and use $\mathcal{L}$ "out-of-the-box" without using any training examples; and (5) "GPT-3" in-context learning, where we use the same prompt-based zero-shot setting, but augment the context with randomly sampled 32 demonstrations (and still use RoBERTa-large, not GPT-3).

Single-prompt results. Table 3 shows our main results using a single prompt, either from our manually designed ones (Table 1) , or the best generated ones. First, prompt-based zero-shot prediction achieves much better performance than the majority class, showing the pre-encoded knowledge in RoBERTa. Also, "GPT-3" in-context learning does not always improve over zero-shot prediction, likely because smaller language models are not expressive enough to use off-the-shelf like GPT-3.

Prompt-based Fine-tuning	MNLI	RTE
Our single manual $\mathcal{P}$	68.3 (2.3)	69.1 (3.6)
$\mathcal{P}_{\text{PET}}$	71.9 (1.5)	69.2 (4.0)
$\mathcal{P}_{\text{ours}}$, $\lvert\mathcal{P}_{\text{ours}}\rvert = \lvert\mathcal{P}_{\text{PET}}\rvert$	70.4 (3.1)	73.0 (3.2)
+ demonstrations	74.0 (1.9)	71.9 (4.6)
$\mathcal{P}_{\text{ours}}$, $\lvert\mathcal{P}_{\text{ours}}\rvert = 20$	72.7 (2.5)	**73.1** (3.3)
+ demonstrations	**75.4** (1.6)	72.3 (4.5)

Table 4: Ensemble models using manual prompts from PET (Schick and Schütze, 2021a,b) and our automatic templates. PET uses 4 prompts for MNLI and 5 for RTE. We also use an equal number of templates in $\lvert\mathcal{P}_{\text{ours}}\rvert = \lvert\mathcal{P}_{\text{PET}}\rvert$ for a fair comparison.

	SST-2	SNLI	TREC	MRPC
Manual	**92.7**	**77.2**	84.8	74.5
Auto T	92.3	77.1	88.2	76.2
Auto L	91.5	75.6	87.0	**77.2**
Auto T + L	92.1	77.0	**89.2**	74.0

Table 5: Comparison between manual prompts and different automatic prompt generation methods: auto-generated templates (Auto T), auto-generated label words (Auto L), and their combination (Auto T + L).

Second, prompt-based fine-tuning can greatly outperform standard fine-tuning, both when using a manual prompt or a generated one. CoLA is one interesting exception, as the input may be a non-grammatical sentence which is out of the distribution of $\mathcal{L}$. Generally, our automatically searched templates can achieve comparable or even higher results than manual ones, especially for tasks in which constructing strong manual templates is less intuitive (e.g., TREC, QNLI and MRPC).

Finally, using demonstrations in context leads to consistent gains in a majority of tasks. In summary, our combined solution—fine-tuning with automatically searched templates and sampled demonstration sets—achieves a 30% gain on SNLI compared to standard fine-tuning, and 11% gain on average.

Ensemble results. An advantage of automatic prompt search is that we can generate as many prompts as we want, train individual models, and create large ensembles. PET (Schick and Schütze, 2021a,b) also ensembles multiple models trained with manual prompts.[10] In Table 4, we make a direct comparison of our searched prompts and PET's manual prompts on MNLI and RTE (two datasets that we evaluate in common).[11] As the results show, an ensemble with multiple templates always improves performance. An ensemble of the same number of automatic templates achieves comparable or better performance than the ensemble of PET's manual prompts. Increasing the number of automatic templates brings further gains.

[10]They then use unlabeled data and distillation to get a single model, which is outside of our scope.

SST-2	(positive/negative)
Auto T	$\mathcal{M}(\mathcal{Y})$ = {great, terrible} #1. <S_1> A `[MASK]` one . #2. <S_1> A `[MASK]` piece . #3. <S_1> All in all `[MASK]` .
Auto L	$\mathcal{T}(x_{\text{in}})$ = <S_1> It was `[MASK]` . #1. irresistible/pathetic #2. wonderful/bad #3. delicious/bad
SNLI	(entailment/neutral/contradiction)
Auto T	$\mathcal{M}(\mathcal{Y})$ = {Yes, Maybe, No} #1. <S_1> . `[MASK]` , no , <S_2> #2. <S_1> . `[MASK]` , in this case <S_2> #3. <S_1> . `[MASK]` this time <S_2>
Auto L	$\mathcal{T}(x_{\text{in}})$ = <S_1> ? `[MASK]` , <S_2> #1. Alright/Watch/Except #2. Hi/Watch/Worse #3. Regardless/Fortunately/Unless

Table 6: Examples of our automatically generated templates (Auto T) and label words (Auto L).

7.2 Analysis of generated prompts

Table 5 gives the results of using manual vs automatic prompts. For automatic prompts, we compare template search (Auto T), label word search (Auto L), and a joint variant (Auto T + L) in which we start from manual label words, apply Auto T, and then Auto L. In most cases, Auto T achieves comparable or higher performance than manual ones, and is consistently the best variant. Auto L outperforms manual prompts on TREC and MRPC—but is considerably worse on SNLI. Auto T + L is often better than Auto L, but only sometimes better than Auto T. Table 6 shows examples from Auto T and Auto L (A full list in Appendix E). Auto T templates generally fit the context and label words well, but can contain biased peculiarities (e.g., "{*Yes/No*}, *no*" in SNLI). For Auto L words, things are mixed: while most look intuitively reasonable, there are also some mysterious abnormalities (e.g., "*Hi*" for the "entailment" class in SNLI).

[11]In the PET NLI templates, the hypothesis is put before the premise, which we actually found to be suboptimal. In our experiments, we swap the two and get better results.

	SST-2	SNLI	TREC	MRPC
Prompt-based FT	**92.7**	77.2	84.8	74.5
Uniform sampling	92.3	78.8	85.6	70.9
+ RoBERTa sel.	**92.7**	79.5	83.4	76.6
+ SBERT sel.	92.6	**79.7**	**87.5**	**77.8**

Table 7: Impact of demonstration sampling strategies. Uniform sampling randomly samples demonstrations, while selective (sel.) sampling only takes top sentences measured by the sentence encoders (§6).

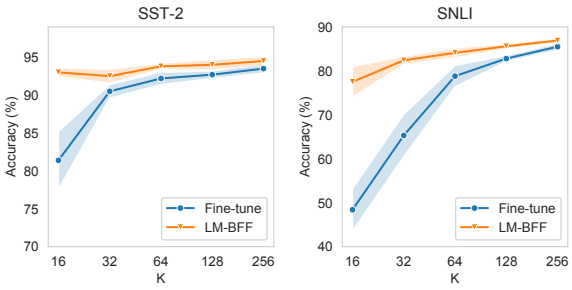

Figure 3: Standard fine-tuning vs our LM-BFF as a function of K (# instances per class). For lower K, our method consistently outperforms standard fine-tuning.

7.3 Analysis of demonstration sampling

Table 7 compares the performance of demonstrations using uniform sampling to selective sampling by SBERT. We acknowledge that SBERT is trained on SNLI and MNLI datasets, thus we also tried a simple sentence encoder using mean pooling of hidden representations from RoBERTa-large. We find that in either case, using selective sampling outperforms uniform sampling, highlighting the importance of sampling similar examples for incorporating demonstrations in context.

7.4 Sample efficiency

Figure 3 illustrates how standard fine-tuning and our LM-BFF compare as K increases. For a simple task such as SST-2 (also see MR, CR and MPQA in Table 3), despite using only 32 total examples, LM-BFF has already nearly saturated its performance and is comparable to standard fine-tuning over the entire dataset. On the harder task of SNLI, LM-BFF continues to improve as K increases while still maintaining a performance gap over standard fine-tuning, until the two converge around $K = 256$.

8 Discussion

Reformulating NLP tasks as MLM has exciting implications for few-shot learning, but also has limitations. First, while LM-BFF greatly outperforms standard fine-tuning, Table 3 shows that, overall, the performance still substantially lags behind fine-tuning with thousands of examples, especially for harder tasks. Additionally, just like standard fine-tuning, our results also suffer from high variance. As described in §2, several recent studies have tried to counter instability in few-shot fine-tuning and we expect these methods to also help here.

With respect to automatic prompt generation, despite its effectiveness, we still find it practically challenging to expand the search space, or generalize well based on only approximately 32 examples. This is partly due to our lingering reliance on *some* manual design—either manual templates (for label word search) or manual label words (for template search), which allows us to get our search off the ground, but does also bias it towards areas of the search space that we might have already imagined.

Finally, it is important to clarify that LM-BFF favors certain tasks which (1) can be naturally posed as a "fill-in-the-blank" problem; (2) have relatively short input sequences; and (3) do not contain many output classes. Issues (2) and (3) might be ameliorated with longer-context language models (e.g., Beltagy et al., 2020). For tasks that are not straightforward to formulate in prompting, such as structured prediction, issue (1) is more fundamental. We leave it as an open question for future work.

9 Conclusion

In this paper we presented LM-BFF, a set of simple but effective techniques for fine-tuning language models using only a few examples. Our approach proposes to (1) use prompt-based fine-tuning with automatically searched prompts; and (2) include selected task demonstrations (training examples) as part of the input context. We show that our method outperforms vanilla fine-tuning by up to 30% (and 11% on average). We concluded by discussing the limitations of our approach, and posed open questions for future study.

Acknowledgements

We thank the members of Princeton, MIT, Tsinghua NLP groups and the anonymous reviewers for their valuable feedback. TG is supported by a Graduate Fellowship at Princeton University and AF is supported by an NSF Graduate Research Fellowship. This research is also partly supported by a Google Research Scholar Award.

References

Trapit Bansal, Rishikesh Jha, and Andrew McCallum. 2020a. Learning to few-shot learn across diverse natural language classification tasks. In *International Conference on Computational Linguistics (COLING)*.

Trapit Bansal, Rishikesh Jha, Tsendsuren Munkhdalai, and Andrew McCallum. 2020b. Self-supervised meta-learning for few-shot natural language classification tasks. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Yujia Bao, Menghua Wu, Shiyu Chang, and Regina Barzilay. 2020. Few-shot text classification with distributional signatures. In *International Conference on Learning Representations (ICLR)*.

Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document Transformer. *arXiv:2004.05150*.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth PASCAL recognizing textual entailment challenge. In *TAC*.

Samuel Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *the 11th International Workshop on Semantic Evaluation (SemEval-2017)*.

Jiaao Chen, Zichao Yang, and Diyi Yang. 2020. MixText: Linguistically-informed interpolation of hidden space for semi-supervised text classification. In *Association for Computational Linguistics (ACL)*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*.

Joe Davison, Joshua Feldman, and Alexander M Rush. 2019. Commonsense knowledge mining from pretrained models. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional Transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *the Third International Workshop on Paraphrasing (IWP2005)*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*.

Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2018. Fewrel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Association for Computational Linguistics (ACL)*.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *ACM SIGKDD international conference on Knowledge discovery and data mining*.

Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association of Computational Linguistics (TACL)*.

Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. 2020. Mixout: Effective regularization to finetune large-scale pretrained language models. In *International Conference on Learning Representations (ICLR)*.

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*.

Pascal Mettes, Elise van der Pol, and Cees Snoek. 2019. Hyperspherical prototype networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2017. Adversarial training methods for semi-supervised text classification. In *International Conference on Learning Representations (ICLR)*.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Association for Computational Linguistics (ACL)*.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Association for Computational Linguistics (ACL)*.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Empirical Methods in Natural Language Processing (EMNLP)*.

Jason Phang, Thibault Févry, and Samuel R Bowman. 2018. Sentence encoders on STILTs: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. Technical report, OpenAI.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI.

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. *The Journal of Machine Learning Research (JMLR)*, 21(140).

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Timo Schick, Helmut Schmid, and Hinrich Schütze. 2020. Automatically identifying words that can serve as labels for few-shot text classification. In *International Conference on Computational Linguistics (COLING)*.

Timo Schick and Hinrich Schütze. 2021a. Exploiting cloze questions for few-shot text classification and natural language inference. In *European Chapter of the Association for Computational Linguistics (EACL)*.

Timo Schick and Hinrich Schütze. 2021b. It's not just size that matters: Small language models are also few-shot learners. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Automatic prompt construction for masked language models. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Alon Talmor, Yanai Elazar, Yoav Goldberg, and Jonathan Berant. 2020. oLMpics-on what language model pre-training captures. *Transactions of the Association of Computational Linguistics (TACL)*, 8.

Trieu H Trinh and Quoc V Le. 2018. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*.

Ellen M Voorhees and Dawn M Tice. 2000. Building a question answering test collection. In *the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*.

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 *International Conference on Learning Representations (ICLR)*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association of Computational Linguistics (TACL)*, 7.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39(2-3).

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. 2020. Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems (NeurIPS)*, 33.

Wenpeng Yin, Nazneen Fatema Rajani, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020. Universal natural language processing with limited annotations: Try few-shot textual entailment as a start. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang,

and Bowen Zhou. 2018. Diverse few-shot text classification with multiple metrics. In *North American Chapter of the Association for Computational Linguistics (NAACL)*.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2021. Revisiting few-sample BERT fine-tuning. In *International Conference on Learning Representations (ICLR)*.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual probing is [MASK]: Learning vs. learning to recall. In *North American Association for Computational Linguistics (NAACL)*.

A Impact of Development Sets

Table A.1 shows how the size of the development sets can affect the final performance of the model. For "No $\mathcal{D}_{\text{dev}}$", we take the same hyper-parameters from Schick and Schütze (2021a,b): batch size = 16, learning rate = 1e-5 and training steps = 250. We also experiment with a variant that we sample a development set of 10 times larger than the training set. We can see that using larger development sets leads to better performance, and this is why we stick to $|\mathcal{D}_{\text{train}}| = |\mathcal{D}_{\text{dev}}|$ in our few-shot setting.

Fine-tuning	SST-2	SNLI	TREC	MRPC
No $\mathcal{D}_{\text{dev}}$	79.5	49.2	83.9	77.8
$\lvert\mathcal{D}_{\text{dev}}\rvert = \lvert\mathcal{D}_{\text{train}}\rvert$	81.4	48.4	88.8	76.6
$\lvert\mathcal{D}_{\text{dev}}\rvert = 10\lvert\mathcal{D}_{\text{train}}\rvert$	83.5	52.0	89.4	79.6
Prompt-based FT	**SST-2**	**SNLI**	**TREC**	**MRPC**
No $\mathcal{D}_{\text{dev}}$	92.1	75.3	84.8	70.2
$\lvert\mathcal{D}_{\text{dev}}\rvert = \lvert\mathcal{D}_{\text{train}}\rvert$	92.7	77.2	84.8	74.5
$\lvert\mathcal{D}_{\text{dev}}\rvert = 10\lvert\mathcal{D}_{\text{train}}\rvert$	93.0	79.7	89.3	80.9

Table A.1: Impact of different sizes of development sets. Standard deviations are omitted here to save space. For No $|\mathcal{D}_{\text{dev}}|$, we use the same set of hyper-parameters as Schick and Schütze (2021a,b).

B Datasets

For SNLI (Bowman et al., 2015) and datasets from GLUE (Wang et al., 2019), including SST-2 (Socher et al., 2013), CoLA (Warstadt et al., 2019), MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016), RTE (Dagan et al., 2005; Bar Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), MRPC (Dolan and Brockett, 2005), QQP[12] and STS-B (Cer et al., 2017), we follow Zhang et al. (2021) and use their original development sets for testing. For datasets which require a cross-validation evaluation—MR (Pang and Lee, 2005), CR (Hu and Liu, 2004), MPQA (Wiebe et al., 2005), Subj (Pang and Lee, 2004)—we simply randomly sample 2,000 examples as the testing set and leave them out from training. For SST-5 (Socher et al., 2013) and TREC (Voorhees and Tice, 2000), we use their official test sets. We show dataset statistics in Table B.1.

C Experimental Details

C.1 Hyper-parameter selection

For grid search, we take learning rates from {1e-5, 2e-5, 5e-5} and batch sizes from {2, 4, 8}. These numbers are picked by pilot experiments on the SST-2 and SNLI datasets. We also use early stopping to avoid overfitting. For each trial, we train the model for 1,000 steps, validate the performance every 100 steps, and take the best checkpoint.

C.2 Prompt-based fine-tuning

Table 1 shows all the manual templates and label words we use in experiment. For automatically template generation, we take the T5-3B[13] model, which is the largest publicly available one that can fit on a single GPU. For automatically searching label words, we set k to 100 for all tasks except SST-5 and TREC. For SST-5 we set a smaller $k = 30$, as it is a 5-way classification task. For TREC, we observe that filtering $\mathcal{V}^c$ using conditional likelihood alone is still noisy, thus we set $k = 1000$, and then re-rank $\mathcal{V}^c$ by the nearest neighbors of the original manual label words and take the top 30 per class. We set n to 100 in all experiments. Due to the large number of trials in automatic search, we take a fixed set of hyper-parameters in this part: batch size of 8 and learning rate of 1e-5.

Since the idea of prompt-based fine-tuning is to make the input and output distribution close to the pre-training, the implementation details are crucial. For templates, we put extra space before sentences if it is not at the beginning of the input. Also, we lowercase the first letter of the sentence if it is concatenated with a prefix (e.g., $<S_2>$ in Table 1). Also if one sentence is appended any punctuation (e.g., $<S_1>$ in Table 1), then the last character of the original sentence is discarded. Finally, we prepend a space for label words in $\mathcal{M}(\mathcal{Y})$. For example, we use "_great" instead of "great" in the RoBERTa vocabulary, where "_" stands for space.

C.3 Fine-tuning with demonstrations

When using demonstrations, we sample 16 different sets of demonstrations for each input and average the predicted log probability for each class during inference. We find that further increasing the number of samples does not bring substantial improvement. Additional, we have tried different aggregation methods like taking the result with the maximum confidence and we did not find a meaningful improvement. For selective demonstrations, we take `roberta-large-nli-stsb`

[12] https://www.quora.com/q/quoradata/

[13] We take the T5 1.0 checkpoint, which is trained on both unsupervised and downstream task data. We compared it to T5 1.1 (without downstream task data) and did not find a significant difference in generated templates.

Category	Dataset	$\lvert\mathcal{Y}\rvert$	L	#Train	#Test	Type	Labels (classification tasks)
single-sentence	SST-2	2	19	6,920	872	sentiment	positive, negative
	SST-5	5	18	8,544	2,210	sentiment	v. pos., positive, neutral, negative, v. neg.
	MR	2	20	8,662	2,000	sentiment	positive, negative
	CR	2	19	1,775	2,000	sentiment	positive, negative
	MPQA	2	3	8,606	2,000	opinion polarity	positive, negative
	Subj	2	23	8,000	2,000	subjectivity	subjective, objective
	TREC	6	10	5,452	500	question cls.	abbr., entity, description, human, loc., num.
	CoLA	2	8	8,551	1,042	acceptability	grammatical, not_grammatical
sentence-pair	MNLI	3	22/11	392,702	9,815	NLI	entailment, neutral, contradiction
	SNLI	3	14/8	549,367	9,842	NLI	entailment, neutral, contradiction
	QNLI	2	11/30	104,743	5,463	NLI	entailment, not_entailment
	RTE	2	49/10	2,490	277	NLI	entailment, not_entailment
	MRPC	2	22/21	3,668	408	paraphrase	equivalent, not_equivalent
	QQP	2	12/12	363,846	40,431	paraphrase	equivalent, not_equivalent
	STS-B	$\mathcal{R}$	11/11	5,749	1,500	sent. similarity	-

Table B.1: The datasets evaluated in this work. $\lvert\mathcal{Y}\rvert$: # of classes for classification tasks (with one exception: STS-B is a real-valued regression task over the interval $[0, 5]$). L: average # of words in input sentence(s). Note that we only sample $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{dev}}$ of $K \times \lvert\mathcal{Y}\rvert$ examples from the original training set in our few-shot experiments (§3).

BERT-large	SST-2	SNLI	TREC	MRPC
Fine-tuning	79.5	51.4	80.3	**74.4**
Prompt-based FT	85.6	59.2	79.0	66.8
+ demo (1-seg)	**87.5**	50.4	77.2	68.5
+ demo (2-seg)	86.1	**61.3**	77.9	73.2
+ demo (n-seg)	86.4	58.6	**79.6**	71.0
RoBERTa-large	**SST-2**	**SNLI**	**TREC**	**MRPC**
Fine-tuning	81.4	48.4	**88.8**	76.6
Prompt-based FT	**92.7**	77.2	84.8	74.5
+ demonstrations	92.6	**79.7**	87.5	**77.8**

Table D.1: A comparison of BERT-large vs RoBERTa-large. We use manual prompts in these experiments.

`mean-tokens`[14] from Reimers and Gurevych (2019) as our sentence embedding model.

D Comparisons of BERT vs RoBERTa

Table D.1 compares the results of BERT-large (uncased) and RoBERTa-large in our settings. Pre-trained BERT provides two segment embeddings (A/B) for different parts of input. The common practice, when fine-tuning BERT, is that using only segment A for single-sentence tasks, and using segment A/B for the two sentences in sentence-pair tasks. In our case of incorporating demonstrations, however, we have more than two sentences. Thus we explore the following different strategies for segments: (1) using the A segment for all sentences (1-seg); (2) using the A segment for the original input and the B segment for the demonstrations (2-seg); (3) using different segment embeddings for each sentence (n-seg), e.g., for SNLI, we use different segments for each premise and hypothesis in both the original input and the demonstrations, which leads to a total number of 8 segment embeddings. This introduces new segment embeddings (randomly initialized and learned during fine-tuning) as the pre-trained BERT only has two.

Table D.1 shows that prompt-based fine-tuning with demonstrations also works for BERT, and 2-seg works the best when incorporating demonstrations. Still, we take RoBERTa-large as our main model, for RoBERTa performs much better than BERT and RoBERTa saves the trouble to tune the usage of segment embeddings.

E Generated Prompts

We demonstrate the top 3 automatically generated templates and label words for all tasks in Table E.1. In general, most automatic templates are reasonable and grammatically correct. For the label words, the generated results look intuitive for most single sentence tasks. For other tasks, the automatic ones can be counterintuitive in some cases. It is still unclear why the language model picks these words and sometimes they actually work well. We leave this for future study.

[14] https://github.com/UKPLab/sentence-transformers

Task	Auto template	Auto label words
SST-2	(positive/negative)	
	<S_1> A `[MASK]` one .	irresistible/pathetic
	<S_1> A `[MASK]` piece .	wonderful/bad
	<S_1> All in all `[MASK]` .	delicious/bad
SST-5	(very positive/positive/neutral/negative/very negative)	
	<S_1> The movie is `[MASK]` .	wonderful/remarkable/hilarious/better/awful
	<S_1> The music is `[MASK]` .	wonderful/perfect/hilarious/better/awful
	<S_1> But it is `[MASK]` .	unforgettable/extraordinary/good/better/terrible
MR	(positive/negative)	
	It was `[MASK]` ! <S_1>	epic/terrible
	<S_1> It's `[MASK]` .	epic/awful
	<S_1> A `[MASK]` piece of work .	exquisite/horrible
CR	(positive/negative)	
	<S_1> It's `[MASK]` !	fantastic/horrible
	<S_1> The quality is `[MASK]` .	neat/pointless
	<S_1> That is `[MASK]` .	magnificent/unacceptable
MPQA	(positive/negative)	
	<S_1> is `[MASK]` .	important/close
	<S_1>, `[MASK]` !	needed/bad
	<S_1>. `[MASK]` .	unexpected/shocking
Subj	(subjective/objective)	
	<S_1> It's all `[MASK]` .	everywhere/tragic
	<S_1> It's `[MASK]` .	everywhere/horrifying
	<S_1> Is it `[MASK]` ?	something/surreal
TREC	(abbreviation/entity/description/human/location/numeric)	
	Q: `[MASK]` : <S_1>	Application/Advisor/Discussion/Culture/Assignment/Minute
	<S_1> Why `[MASK]`?	Production/AE/Context/Artist/Assignment/Minute
	<S_1> Answer: `[MASK]` .	Personality/Advisor/Conclusion/Hum/Assignment/Minute
CoLA	(grammatical/not_grammatical)	
	<S_1> You are `[MASK]` .	one/proof
	It is `[MASK]` . <S_1>	wrong/sad
	I am `[MASK]` . <S_1>	misleading/disappointing
MNLI	(entailment/neutral/contradiction)	
	<S_1> . `[MASK]` , you are right , <S_2>	Fine/Plus/Otherwise
	<S_1> . `[MASK]` you're right <S_2>	There/Plus/Otherwise
	<S_1> . `[MASK]` ! <S_2>	Meaning/Plus/Otherwise
SNLI	(entailment/neutral/contradiction)	
	<S_1> . `[MASK]` , no , <S_2>	Alright/Watch/Except
	<S_1> . `[MASK]` , in this case <S_2>	Hi/Watch/Worse
	<S_1> . `[MASK]` this time <S_2>	Regardless/Fortunately/Unless
QNLI	(entailment/not_entailment)	
	<S_1> ? `[MASK]` . Yes , <S_2>	Okay/Nonetheless
	<S_1> ? `[MASK]` . It is known that <S_2>	Notably/Yet
	<S_1> ? `[MASK]` , however , <S_2>	Specifically/Notably
RTE	(entailment/not_entailment)	
	<S_1> . `[MASK]` , I believe <S_2>	Clearly/Yet
	<S_1> . `[MASK]` , I think that <S_2>	Accordingly/meanwhile
	<S_1> . `[MASK]` , I think <S_2>	So/Meanwhile
MRPC	(equivalent/not_equivalent)	
	<S_1> . `[MASK]` ! <S_2>	Rather/Alas
	<S_1> . `[MASK]` . This is the first time <S_2>	At/Thus
	<S_1> . `[MASK]` . That's right . <S_2>	Instead/Moreover
QQP	(equivalent/not_equivalent)	
	<S_1> ? `[MASK]` , but <S_2>	Me/Since
	<S_1> ? `[MASK]` , please , <S_2>	Um/Best
	<S_1> ? `[MASK]` , I want to know <S_2>	Ironically/Beyond
STS-B	(y_u/y_l)	
	<S_1> . `[MASK]` sir <S_2>	Note/Next
	<S_1> . `[MASK]` , it is not . <S_2>	Yesterday/meanwhile
	<S_1> . `[MASK]` . It is <S_2>	Yeah/meanwhile

Table E.1: Top 3 automatically generated templates and label words for all tasks based on one split of $K = 16$ training examples. Note that automatic template results are based on manual label words and automatic label word results are based on manual templates provided in Table 1.