DynaMaR: Dynamic Prompt with Mask Token Representation

Xiaodi Sun[1,*], **Sunny Rajagopalan**[2,*], **Priyanka Nigam**[3], **Weiyi Lu**[3], **Yi Xu**[3]
Iman Keivanloo[3], **Belinda Zeng**[3], **Trishul Chilimbi**[3]
[1]Microsoft, [2]Google, [3]Amazon
[1]xiaodisun315@gmail.com, [2]sunny.rg@gmail.com
[3]{nigamp,weiyilu,yxaamzn,imanke,zengb,trishulc}@amazon.com

Abstract

Recent research has shown that large language models pretrained using unsupervised approaches can achieve significant performance improvement on many downstream tasks. Typically when adapting these language models to downstream tasks, like a classification or regression task, we employ a fine-tuning paradigm in which the sentence representation from the language model is input to a task-specific head; the model is then fine-tuned end-to-end. However, with the emergence of models like GPT-3, prompt-based fine-tuning has been proven to be a successful approach for few-shot tasks. Inspired by this work, we study discrete prompt technologies in practice. There are two issues that arise with the standard prompt approach. First, it can overfit on the prompt template. Second, it requires manual effort to formulate the downstream task as a language model problem. In this paper, we propose an improvement to prompt-based fine-tuning that addresses these two issues. We refer to our approach as DynaMaR – **Dyna**mic Prompt with **Ma**sk Token **R**epresentation. Results show that DynaMaR can achieve an average improvement of 10% in few-shot settings and improvement of 3.7% in data-rich settings over the standard fine-tuning approach on four e-commerce applications.

1 Introduction

Unsupervised pre-trained Language Models (LMs) such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) have achieved state-of-the-art performance on many natural language understanding tasks. In general, these models are fine-tuned for different tasks through the addition of a task-specific head on top of the [CLS] token representation (Scao and Rush, 2021).

An alternative method to applying LMs on downstream tasks is through discrete prompts. A discrete prompt is an additional text phrase inserted along with the original input text that encapsulates the task of interest. By adding the prompt, we convert the downstream task into a masked language (MLM) problem. For example, to classify the sentiment of a movie review, "I hate this movie.", we can append a prompt to the input to get "I hate this movie. It was [MASK]". The pre-trained language model is thus prompted to identify the sentiment of the input statement and classify the [MASK] token as "terrible" instead of "great" (Liu et al., 2021). In this paper, we call a function that includes a prompt and its position information a prompt template.

Prompt-based approaches have shown success in low-data regimes (Petroni et al., 2019; Schick and Schütze, 2021; Jiang et al., 2020; Gao et al., 2021; Lester et al., 2021). Prompt-based fine-tuning is beneficial in few-shot learning, because it provides extra task information to the model through the prompt text (Schick and Schütze, 2021). However, when we explore this technique in practice, two issues have arisen. First, the trained model can overfit on words or phrases within the prompt and on the position of the [MASK] token in the prompt (Zhong et al., 2021). For example, in movie review sentiment analysis, when we append the prompt, "Does the user like the movie? [MASK]", to a negative review, "This is a bad movie.", the trained model is inclined to predict the positive class, because the word "like" frequently appears in positive reviews and the masked language model has greater attention on the words/phrases that are closer to the mask token as shown in Figure 1. We call this issue prompt-related overfitting in this work.

We tackle prompt-related overfitting by introducing a dynamic prompt approach. In this approach, we create a prompt pool consisting of multiple prompt templates. To construct this pool, we generate a set of prompt candidates and filter by a similarity score we propose, called the pairwise prompt dissimilarity score (detailed in Section 3). We then

*Work done while at Amazon.

Proceedings of EMNLP 2022 Industry Track, pages 9–17
December 9–11, 2020.

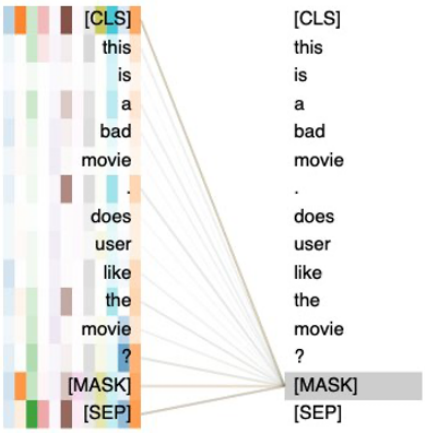

Figure 1: BERT Attention Distribution. The figure shows that the MLM model puts greater attention on the prompt than the original input.

introduce the dynamic component of the algorithm by randomly selecting a prompt template from the pool and applying to the input for each training step. For example, in the movie review sentiment analysis task, the trained model will randomly see either "does the user like the movie? [MASK]" or "does the user dislike the movie? [MASK]" appended to the original input. This prevents the model to overfit on spurious correlations between words in the prompt and the class label.

In addition, as previously mentioned, the standard prompt-based fine-tuning setup can be inefficient. It requires significant input and answer engineering to reformulate the downstream tasks as MLM problems (Liu et al., 2021). This process is time-consuming especially for tasks with large numbers of classes. Besides, another disadvantage of the standard setup is that it cannot be directly applied to regression problems, as they cannot be easily converted to MLM problems. To simplify this process, we fine-tune the model by feeding the mask token representation into a task-specific classifier/predictor head instead of the pre-trained MLM head to avoid the answer engineering process, as shown in Figure 2. We refer to our prompt-based approach with these two improvements as Dynamic Prompt with Mask Token Representation (DynaMaR). We apply DynaMaR to both few-shot and data-rich settings and, for the first time, show improvement gains across four tasks not only in few-shot settings but also in data-rich settings.

Our contributions include: (1) proposing DynaMaR, which can be applied without reformulating downstream tasks into language problems and is robust to prompt-related overfitting, (2) showing DynaMaR can achieve improvements in both few-shot and data-rich settings, (3) proposing a prompt dissimilarity score to evaluate the degree of dissimilarity between two prompt templates and to help construct a diverse dynamic prompt pool, (4) demonstrating that a larger dynamic prompt pool achieves better performance on downstream tasks.

2 Related Work

Our work can be divided into three components: language model fine-tuning, prompt generation, and the design of the prompt template.

Language Model Fine-tuning is the main focus of our work. Recently, a large amount of research has focused on improved language model finetuning methods (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. In contrast to these works, Gao et al. (2021) describe the concept of prompt-based fine-tuning for language models. We adapt and simplify the core ideas from this work to create a simple yet efficient prompt-based fine-tuning approach.

Prompt Generation is a key process in prompt-based fine-tuning. The choice of prompt significantly influences performance. The most natural way to generate prompts is through manual design. Petroni et al. (2019) employ manually generated prompts with ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019) models. They evaluate on the LAMA (LAnguage Model Analysis) benchmark (Bordes et al., 2013; Nickel et al., 2016) without fine-tuning and conclude that the model is able to recall knowledge learned from the pre-training tasks. While manually crafting prompts is intuitive, creating effective prompts through manual effort requires time, experience, and expertise. To address this issue, a number of automatic prompt searching methods have been proposed. For example, Jiang et al. (2020) propose a data mining-based method that searches for a prompt based on the shortest path between the original inputs and answers. They also propose paraphrasing-based methods that take a seed prompt and paraphrase it into several semantically similar expressions. Gao et al. (2021) treat prompt generation as a text generation task and utilize T5, a sequence-to-sequence pretrained model, in the template search process. They generate templates by specifying the position to insert a prompt

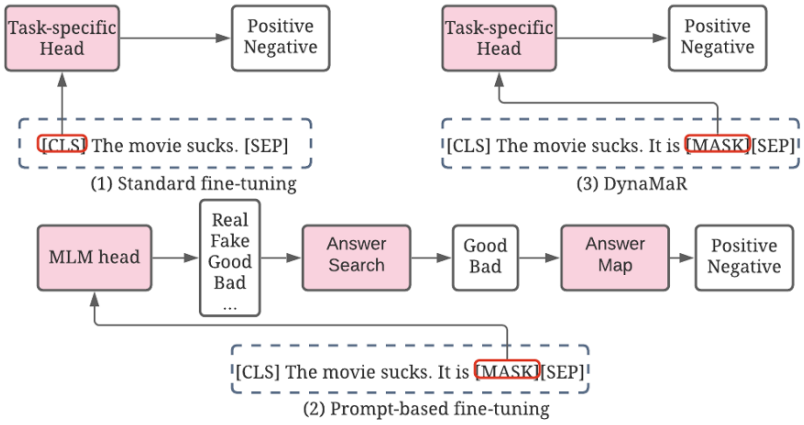

Figure 2: Fine-tuning approach demonstration.

template and then inputting samples into T5 to decode the templates. These automatic approaches achieve comparable performance to manually designed prompts. Besides, Logan IV et al. (2021) propose the null prompt method. Instead of generating prompts, they concatenate a [MASK] token with original inputs and it performs competitively to manually designed prompts. In our experiments, we utilize the prompt generation methods to create candidates for the dynamic prompt pool, while also including the null prompt approach as one of the baselines.

Prompt Template Design Factors are the factors that we take into consideration to create a metric that informs how prompts are selected for the dynamic prompt pool. Numerous previous works analyze prompt template design factors and the impact of prompt design on performance. Liu et al. (2021) summarize the factors that influence the application of prompt-related technologies in language models. Logan IV et al. (2021) note that the order in which the original input and the [MASK] token are concatenated is an important consideration. Zhong et al. (2021) propose to unify the prompts into a question-answering format. These previous works indicate that prompt construction impacts performance. To this end, we hypothesize that diversity in the set of prompt templates is an important factor in the performance of the model and propose a prompt dissimilarity score for measuring diversity.

3 Our Method: DynaMaR

In this section, we describe details of our approach, DynaMaR. Before explaining the training process, we define two concepts: the dynamic prompt pool and the inference prompt.

Dynamic Prompt Pool is a pool of prompt templates from which a prompt template will be randomly selected and applied to the input during training.

Inference Prompt is the prompt template used during inference. It is selected from the set of templates in the dynamic prompt pool. In general, it is the prompt template among those in the dynamic prompt pool that can achieve the highest performance in a fixed prompt setting.

We generate the candidates for the dynamic prompt pool and inference prompt through manual generation and paraphrasing-based methods proposed by Jiang et al. (2020). However, we do not include all candidates in the dynamic prompt pool. We want to ensure the prompts within a pool are sufficiently diverse so that the model will not overfit on any of them. Therefore, we introduce a prompt dissimilarity score to measure the level of dissimilarity between these candidates. We consider three factors in developing this metric: (1) prompt position, or whether to append or prepend the prompt to the input or even insert into the middle of pairwise inputs, (2) prompt wording or the prompt word selection, and (3) prompt format, or whether to create prompts in statement format or in the question-answering format proposed by Zhong et al. (2021). To define the prompt dissimilarity score, we first introduce the normalized Hamming

distance and the normalized Levenshtein distance.

Normalized Hamming Distance is equal to the number of different bits between two binary representations divided by the length of the binary representations (Norouzi et al., 2012). Let $HD(b_i, b_j)$ be the Hamming distance between binary representations b_i and b_j with equal length K. The equation of normalized Hamming distance $NHD(b_i, b_j)$ then follows:

$$HD(b_i, b_j) = \sum_{k=1}^{K} |b_{ik} - b_{jk}|, \quad (1)$$

$$NHD(b_i, b_j) = HD(b_i, b_j)/K. \quad (2)$$

Normalized Levenshtein Distance is equal to the minimum number of operations (substitution, insertion and deletion) required to transform a given string into another string divided by the length of the longer string and is calculated in a recursive fashion (Yujian and Bo, 2007). Let $LD(s_i, s_j)$ be the Levenshtein distance between string s_i and s_j. Let $|s_i|$ and $|s_j|$ be the length of prompt string s_i and s_j, respectively. Let $t(x)$ be a function that keeps a string of all but the first character of x. The equation of the normalized Levenshtein distance $NLD(s_i, s_j)$ follows:

$$LD(s_i, s_j) = \begin{cases} |s_i|, & \text{if } |s_i| = 0; \\ |s_j|, & \text{if } |s_j| = 0; \\ LD(t(s_i), t(s_j)), & \text{if } |s_i| = |s_j|; \\ 1+ & \\ \min \begin{pmatrix} LD(t(s_i), s_j), \\ LD(s_i, t(s_j)), \\ LD(t(s_i), t(s_j)) \end{pmatrix}, & \text{otherwise.} \end{cases} \quad (3)$$

$$NLD(s_i, s_j) = \begin{cases} \frac{LD(s_i, s_j)}{|s_i|}, & \text{if } |s_j| \leq |s_i|, \\ \frac{LD(s_i, s_j)}{|s_j|}, & \text{if } |s_i| < |s_j|. \end{cases} \quad (4)$$

Suppose we generate N prompt templates. Let p_i and p_j be two prompt templates with s_i, s_j as prompt strings, respectively, where $i \neq j$ and $i, j \in \{1, 2, \ldots, N\}$. We treat the prompt position and format information as categorical variables and convert them into binary representations, b_i, b_j. Let $PDS(p_i, p_j)$ denote the prompt dissimilarity score between prompt templates p_i and p_j. The prompt dissimilarity score equation can be found below:

$$PDS(p_i, p_j) = NHD(b_i, b_j) + NLD(s_i, s_j). \quad (5)$$

In our experiment, we use 0.5 as the pairwise prompt dissimilarity score threshold. We add the prompt templates that have prompt dissimilarity score larger than the threshold to others to a dynamic prompt pool. During the training process, we randomly pick one prompt template from the pool for each training step and apply it to the original input. We treat the mask token representation from the modified input as the sentence embedding and train the model by directly feeding it into a task-specific predictor head.

4 Experiment

4.1 Data

In this experiment, we use four e-commerce proprietary datasets: (1) Variation Elimination (VE), (2) Music Match (MM), (3) Music Genre (MG), and (4) Price Prediction (PP). VE is a binary classification problem with pairwise-document inputs where the label identifies whether two items are the variations of the same product or not. For example, similar shirts (from the same producer and brand) in different sizes or colors are considered to be variations. MM is a binary classification problem with pairwise-document inputs that identifies whether two music tracks from different sources are the same or not. MG is a 303-way classification problem with single-document inputs that classifies music tracks to genres. PP is a regression problem with single-document inputs that aims to estimate the sales price based on the product information. It should be noted that the percentage of inputs with number of tokens larger than 512 in VE, MM, MG, PP are 90%, 75%, 82%, 1%, respectively.

For each task, we split the dataset into three parts: (1) train, (2) validation, and (3) test. We use the full training dataset for the data-rich settings. We also sample multiple few-shot training datasets for few-shot learning settings. In few-shot learning, each classification dataset contains roughly 20 samples for each class. For the regression task (i.e., PP), we randomly sample 1% of the full training dataset as a few-shot training dataset.

4.2 Model and Tokenizer Setup

For training the tokenizer, we collect an English product catalog dataset with text features including title, description, and detail bullet points. We train a 32K BPE vocabulary on this dataset using the SentencePiece library (Kudo and Richardson, 2018).

We create a 500M parameter transformer encoder-only model, with 38 hidden layers, 1024 embedding size, 16 attention heads, and maximum

sequence length of 512. We train the model using the LANS optimizer (Zheng et al., 2020) with a batch size of 8192 and a learning rate of 10^{-4} on the product catalog dataset.

4.3 Prompt Generation and Selection

To create the dynamic prompt pool for our tasks, we first generate 20 prompt templates for each task and select 5 out of them using the prompt dissimilarity score. Specifically, for each task, we first manually design 10 prompt templates. By treating prompt template generation as paraphrase generation task (Jiang et al., 2020), we use these 10 prompt templates as seeds to generate another 10 templates per task by leveraging the public T5 paraphrase generation model from Hugging Face[1]. Afterwards, we use the prompt dissimilarity score to select 5 prompt templates out of the 20 based on the method discussed at the end of Section 3. The selected prompt templates are used as each task's dynamic prompt pool. For inference, we evaluate each template in the dynamic prompt pool through the evaluation process discussed in Section 4.5, and select the prompt template that produces the best performance on each task. Table 5 shows the dynamic prompt templates as well as the inference prompt selected for each task.

4.4 Fine-tuning (Ft) Methods

We compare DynaMaR with the following approaches:

- **Promptless Fine-tuning - CLS (PFt-CLS)** is our baseline approach where we fine-tune the model by feeding the [CLS] token representation into a predictor head.
- **Promptless Fine-tuning - Average Pooling (PFt-Avg)** fine-tunes the model by using the average of sequence output for prediction.
- **Null Prompt - Prefix (NP-Prefix)** prepends the [MASK] token to the original inputs and fine-tunes the model by feeding the [MASK] token representation into a predictor head. This approach avoids the issue where the model overfits the prompt template since it does not require a template.
- **Null Prompt - Suffix (NP-Suffix)** is the same as the above approach except that the [MASK] token is appended to the inputs instead of being prepended.
- **Fixed Prompt with Mask Token Representation (FiTeR)** utilizes a static prompt template in both the training and inference stages and fine-tunes the model by feeding the [MASK] token representation into a predictor head.

[1] https://huggingface.co/Vamsi/T5_Paraphrase_Paws

Ft Method	VE	MM	MG	PP	Avg
PFt-CLS	0	0	**0**	0	0
PFt-Avg	-1.5%	+7.2%	-3.7%	-8.8%	-1.7%
NP-Prefix	-1.0%	+4.1%	-2.0%	+2.6%	+0.9%
NP-Suffix	-2.6%	+0.2%	-1.6%	+6.7%	+0.7%
FiTeR	-0.7%	+13.9%	-1.1%	+7.3%	+4.9%
DPMR	**+0.8%**	**+15.8%**	-0.5%	**+23.8%**	**+10.0%**

Table 1: Few-shot Learning Performance Comparison.

Note that we use a task-specific predictor head in combination with all above approaches including the prompt-based fine-tuning methods, which typically use the pre-trained MLM head for prediction. The reason is that we have a regression task as one of our evaluation datasets, and as already discussed in Section 1, it is not straight forward to convert regression tasks into MLM tasks.

4.5 Model Training and Evaluation Setup

As mentioned in Section 1, we measure the performance in both few-shot and data-rich settings. For both VE and MM, we use Area Under the Precision-Recall Curve (PRAUC) as the evaluation metric. For MG, we use classification accuracy as the evaluation metric. For PP, we use Root Mean Square Error (RMSE) as the evaluation metric. We validate the performance every 2 training steps in the few-shot settings and every 100 steps in the data-rich settings. We use early stopping with a patience of 3 validation steps to select the best model for each task. We then evaluate the best models on the test datasets. For few-shot learning, we report the average performance across multiple few-shot datasets per task to reduce the variation in performance. In Table 1 and Table 2, we calculate and report the improvement percentage, which is the ratio of positive change as compared to PFt performance.

4.6 Results

Table 1 and 2 show the performance results for both few-shot and data-rich settings. In both settings, PFt-Avg shows degradation in average of

Ft Method	VE	MM	MG	PP	Avg
PFt-CLS	**0**	0	**0**	0	0
PFt-Avg	-0.1%	+1.2%	-1.0%	-11.0%	-2.7%
NP-Prefix	-0.1%	+1.0%	-0.4%	0	+0.1%
NP-Suffix	-0.3%	+1.7%	-0.7%	+2.2%	+0.7%
FiTeR	**0**	+1.5%	-0.2%	+3.3%	+1.2%
DPMR	**0**	**+2.9%**	-0.3%	**+12.1%**	**+3.7%**

Table 2: Data-rich Performance Comparison.

performance compared to PFt-CLS. This shows that average pooling generates worse sentence representations than does taking the [CLS] token representation.

In contrast, both null prompt approaches show improvement in average performance compared to PFt-CLS in both few-shot and data-rich settings. The improvement could be a result of aligning the format of the downstream tasks and that of the pretraining task. By changing the input format to be similar to that of the MLM task, we reduce the amount of data that are required to coach the model to learn the new task.

Also, there is a difference in the performance of NP-suffix and NP-prefix. This is likely due to the positional differences of the [MASK] token in the two methods. For example, suppose we want to perform sentiment analysis on a sentence like "I love the movie". Prepending or appending the [MASK] token would result in different distances between [MASK] and the word "love", which holds the key information for classification. Such positional differences could lead to different performance even though the two methods are very similar in spirit.

Another observation is that FiTer shows higher improvement in average of performance compared to null prompt approaches. Recall that FiTer introduces task information through the prompt templates, while the null prompt approaches do not, which supposedly addresses the issue where the model overfits the prompt templates. Hence, the results show that the benefits of adding the extra task information outweigh the possible performance loss caused by the prompt-related overfitting issue.

Finally, DynaMar outperforms FiTer on all tasks in both setting, with the only exception being MG in the data-rich setting. This indicates that increasing the diversity of prompt templates used during training will improve model generalization. We also observe that DynaMar does not show significant improvement over PFt-CLS on both MG and VE. This is because both tasks contain a large number of documents with length longer than 512, as mentioned in Section 4.1. As a result of this, we need to truncate more of the original inputs for these tasks in order to insert prompts, which can lead to information loss. Thus, DynaMar is less efficient in problems with long documents.

4.7 Analysis

Larger dynamic prompt pool, better performance. The size of the dynamic prompt pool influences the performance. We compare the average improvement percentage across four tasks with the size of dynamic prompt pool = 1, 3, 5 (prompt information can be found in Appendix A). From Figure 3, we can see that performance improves as the dynamic prompt pool is made larger.

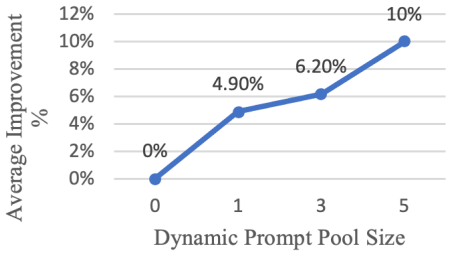

Figure 3: Pool Size vs Improvement Percentage.

4.8 Limitations and Future Directions

As mentioned in Section 4.6, our method does not show substantial improvement on tasks involving long documents. Besides, the threshold of prompt disimilarity score can be treated as a parameter. This work lack of a study on the effect of this threshold. In addition, we focus on e-commerce related English classification/regression tasks in this work, the performance of our method in other nature language processing use cases remains unexplored. As a next step, we will conduct additional studies on these three topics.

5 Conclusion

In this work, we discuss methods for generating prompts and propose a way to select prompt templates to include in the dynamic prompt pool. Also, we show that using the mask representation of a prompt either equals or improves upon the performance of standard fine-tuning on four e-commerce applications in both few-shot and data-rich settings. In addition, we find DynaMaR outperforms the

fixed prompt approach in both settings. Furthermore, we show that a larger dynamic prompt pool leads to improved model performance when employing DynaMaR.

References

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

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

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

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

Zhengbao Jiang, Frank F. Xu, J. Araki, and Graham Neubig. 2020. How can we know what language models know? In *Association for Computational Linguistics (ACL)*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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)*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. In *ArXiv*.

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. In *ArXiv*.

Robert L Logan IV, Ivana Balavzevi'c, Eric Wallace, Fabio Petroni, Sameer Singh, and Sebastian Riedel. 2021. Cutting down on prompts and parameters: Simple few-shot learning with language models. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Maximilian Nickel, Kevin P. Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A review of relational machine learning for knowledge graphs. In *Proceedings of the IEEE*.

Mohammad Norouzi, David J. Fleet, and Ruslan Salakhutdinov. 2012. Hamming distance metric learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. 2019. Language models as knowledge bases? In *Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Teven Le Scao and Alexander M. Rush. 2021. How many data points is a prompt worth? In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

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

Li Yujian and Liu Bo. 2007. A normalized levenshtein distance metric. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

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)*.

Shuai Zheng, Haibin Lin, Sheng Zha, and Mu Li. 2020. Accelerated large batch optimization of bert pretraining in 54 minutes. In *ArXiv*.

Ruiqi Zhong, Kristy Lee, Zheng Zhang, and Dan Klein. 2021. Adapting language models for zero-shot learning by meta-tuning on dataset and prompt collections. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

A Dynamic Prompt Pool with Different Sizes

We need to define two prompt-related parameters while using DynaMaR: the dynamic prompt pool and the inference prompt. The list of prompts in the pool and the inference prompt selected for dynamic prompt pool sizes of 1, 3, and 5 can be found in Table 3, Table 4, and Table 5, respectively.

Task	Inference Prompt	Dynamic Prompt Pool
VE	$f(x_1, x_2)$ = x_1 and x_2 are [MASK] product	$f(x_1, x_2)$ = x_1 and x_2 are [MASK] product
MM	$f(x_1, x_2)$ = x_1 and x_2 are [MASK] music	$f(x_1, x_2)$ = x_1 and x_2 are [MASK] music
MG	$f(x)$ = Genre: [MASK] x	$f(x)$ = Genre: [MASK] x
PP	$f(x)$ = x The price is [MASK]	$f(x)$ = x The price is [MASK]

Table 3: Dynamic Prompt Pool Size = 1.

Task	Inference Prompt	Dynamic Prompt Pool
VE	$f(x_1, x_2)$ = x_1 and x_2 are [MASK] product	(1) $f(x_1, x_2)$ = x_1 x_2. Are they the same product? [MASK] (2) $f(x_1, x_2)$ = x_1 and x_2 are [MASK] product (3) $f(x_1, x_2)$ = x_1 x_2. They are [MASK]
MM	$f(x_1, x_2)$ = x_1 and x_2 are [MASK] music	(1) $f(x_1, x_2)$ = x_1 x_2. Are they the same song? [MASK] (2) $f(x_1, x_2)$ = x_1 and x_2 are [MASK] music (3) $f(x_1, x_2)$ = x_1 is as [MASK] as x_2
MG	$f(x)$ = Genre: [MASK] x	(1) $f(x)$ = Genre: [MASK] x (2) $f(x)$ = Music Genre: [MASK] x (3) $f(x)$ = x what is genre of the music? [MASK]
PP	$f(x)$ = x The price is [MASK]	(1) $f(x)$ = Price: [MASK] x (2) $f(x)$ = x it cost [MASK] dollars (3) $f(x)$ = x what is price of the product? [MASK]

Table 4: Dynamic Prompt Pool Size = 3.

Task	Inference Prompt	Dynamic Prompt Pool
VE	$f(x_1, x_2)$ = x_1 and x_2 are [MASK] product	(1) $f(x_1, x_2)$ = x_1 x_2. Are they the same product? [MASK] (2) $f(x_1, x_2)$ = x_1 and x_2 are [MASK] product (3) $f(x_1, x_2)$ = x_1 x_2. They are [MASK] (4) $f(x_1, x_2)$ = Are x_1 and x_2 the same product? [MASK] (5) $f(x_1, x_2)$ = x_1 is as [MASK] as x_2
MM	$f(x_1, x_2)$ = x_1 and x_2 are [MASK] music	(1) $f(x_1, x_2)$ = x_1 x_2. Are they the same song? [MASK] (2) $f(x_1, x_2)$ = x_1 and x_2 are [MASK] music (3) $f(x_1, x_2)$ = x_1 x_2. They are [MASK] music (4) $f(x_1, x_2)$ = Are x_1 and x_2 the same music? [MASK] (5) $f(x_1, x_2)$ = x_1 is as [MASK] as x_2
MG	$f(x)$ = Genre: [MASK] x	(1) $f(x)$ = Genre: [MASK] x (2) $f(x)$ = Music Genre: [MASK] x (3) $f(x)$ = x This is a [MASK] music (4) $f(x)$ = Type: [MASK] x (5) $f(x)$ = x what is genre of the music? [MASK]
PP	$f(x)$ = x The price is [MASK]	(1) $f(x)$ = Price: [MASK] x (2) $f(x)$ = x Price: [MASK] (3) $f(x)$ = x it cost [MASK] dollars (4) $f(x)$ = x The price is [MASK] (5) $f(x)$ = x what is price of the product? [MASK]

Table 5: Dynamic Prompt Pool Size = 5.