Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing

Pengfei Liu
Carnegie Mellon University
pliu3@cs.cmu.edu

Weizhe Yuan
Carnegie Mellon University
weizhey@cs.cmu.edu

Jinlan Fu
National University of Singapore
jinlanjonna@gmail.com

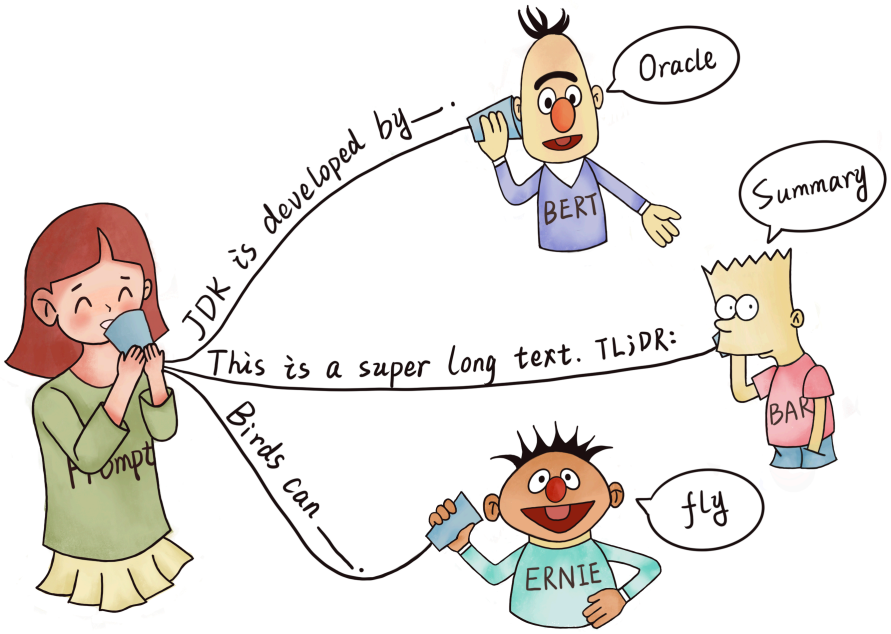
Zhengbao Jiang
Carnegie Mellon University
zhengbaj@cs.cmu.edu

Hiroaki Hayashi
Carnegie Mellon University
hiroakih@cs.cmu.edu

Graham Neubig
Carnegie Mellon University
gneubig@cs.cmu.edu

Abstract

This paper surveys and organizes research works in a new paradigm in natural language processing, which we dub "prompt-based learning". Unlike traditional supervised learning, which trains a model to take in an input $\boldsymbol{x}$ and predict an output $\boldsymbol{y}$ as $P(\boldsymbol{y}|\boldsymbol{x})$, prompt-based learning is based on language models that model the probability of text directly. To use these models to perform prediction tasks, the original input $\boldsymbol{x}$ is modified using a *template* into a textual string *prompt* $\boldsymbol{x}'$ that has some unfilled slots, and then the language model is used to probabilistically fill the unfilled information to obtain a final string $\hat{\boldsymbol{x}}$, from which the final output $\boldsymbol{y}$ can be derived. This framework is powerful and attractive for a number of reasons: it allows the language model to be *pre-trained* on massive amounts of raw text, and by defining a new prompting function the model is able to perform *few-shot* or even *zero-shot* learning, adapting to new scenarios with few or no labeled data. In this paper we introduce the basics of this promising paradigm, describe a unified set of mathematical notations that can cover a wide variety of existing work, and organize existing work along several dimensions, e.g. the choice of pre-trained models, prompts, and tuning strategies. To make the field more accessible to interested beginners, we not only make a systematic review of existing works and a highly structured typology of prompt-based concepts, but also release other resources, e.g., a website NLPedia–Pretrain including constantly-updated survey, and paperlist.



Contents

1 Two Sea Changes in NLP

Fully supervised learning, where a task-specific model is trained solely on a dataset of input-output examples for the target task, has long played a central role in many machine learning tasks (Kotsiantis et al., 2007), and natural language processing (NLP) was no exception. Because such fully supervised datasets are ever-insufficient for learning high-quality models, early NLP models relied heavily on *feature engineering* (Tab. 1 a.; e.g. Lafferty et al. (2001); Guyon et al. (2002); Och et al. (2004); Zhang and Nivre (2011)), where NLP researchers or engineers used their domain knowledge to define and extract salient features from raw data and provide models with the appropriate inductive bias to learn from this limited data. With the advent of neural network models for NLP, salient features were learned jointly with the training of the model itself (Collobert et al., 2011; Bengio et al., 2013), and hence focus shifted to *architecture engineering*, where inductive bias was rather provided through the design of a suitable network architecture conducive to learning such features (Tab. 1 b.; e.g. Hochreiter and Schmidhuber (1997); Kalchbrenner et al. (2014); Chung et al. (2014); Kim (2014); Bahdanau et al. (2014); Vaswani et al. (2017)).[1]

However, from 2017-2019 there was a sea change in the learning of NLP models, and this fully supervised paradigm is now playing an ever-shrinking role. Specifically, the standard shifted to the *pre-train and fine-tune* paradigm (Tab. 1 c.; e.g. Radford and Narasimhan (2018); Peters et al. (2018); Dong et al. (2019); Yang et al. (2019); Lewis et al. (2020a)). In this paradigm, a model with a fixed[2] architecture is *pre-trained* as a language model (LM), predicting the probability of observed textual data. Because the raw textual data necessary to train LMs is available in abundance, these LMs can be trained on large datasets, in the process learning robust general-purpose features of the language it is modeling. The above pre-trained LM will be then adapted to different downstream tasks by introducing additional parameters and *fine-tuning* them using task-specific objective functions. Within this paradigm, the focus turned mainly to *objective engineering*, designing the training objectives used at both the pre-training and fine-tuning stages. For example, Zhang et al. (2020a) show that introducing a loss function of predicting salient sentences from a document will lead to a better pre-trained model for text summarization. Notably, the main body of the pre-trained LM is generally (but not always; Peters et al. (2019)) fine-tuned as well to make it more suitable for solving the downstream task.

Now, as of this writing in 2021, we are in the middle of a second sea change, in which the "pre-train, fine-tune" procedure is replaced by one in which we dub "*pre-train, prompt, and predict*". In this paradigm, instead of adapting pre-trained LMs to downstream tasks via objective engineering, downstream tasks are reformulated to look more like those solved during the original LM training with the help of a textual *prompt*. For example, when recognizing the emotion of a social media post, "I missed the bus today.", we may continue with a prompt "I felt so ___", and ask the LM to fill the blank with an emotion-bearing word. Or if we choose the prompt "English: I missed the bus today. French: ___"), an LM may be able to fill in the blank with a French translation. In this way, by selecting the appropriate prompts we can manipulate the model behavior so that the pre-trained LM itself can be used to *predict* the desired output, sometimes even without any additional task-specific training (Tab. 1 d.; e.g. Radford et al. (2019); Petroni et al. (2019); Brown et al. (2020); Raffel et al. (2020); Schick and Schütze (2021b); Gao et al. (2021)). The advantage of this method is that, given a suite of appropriate prompts, a single LM trained in an entirely unsupervised fashion can be used to solve a great number of tasks (Brown et al., 2020; Sun et al., 2021). However, as with most conceptually enticing prospects, there is a catch – this method introduces the necessity for *prompt engineering*, finding the most appropriate prompt to allow a LM to solve the task at hand.

This survey attempts to organize the current state of knowledge in this rapidly developing field by providing an overview and formal definition of prompting methods (§2), and an overview of the pre-trained language models that use these prompts (§3). This is followed by in-depth discussion of prompting methods, from basics such as prompt engineering (§4) and answer engineering (§5) to more advanced concepts such as multi-prompt learning methods (§6) and prompt-aware training methods (§7). We then organize the various applications to which prompt-based learning methods have been applied, and discuss how they interact with the choice of prompting method (§8). Finally, we attempt to situate the current state of prompting methods in the research ecosystem, making connections to other research fields (§9), suggesting some current challenging problems that may be ripe for further research (§10), and performing a meta-analysis of current research trends (§11).

Finally, in order to help beginners who are interested in this field learn more effectively, we highlight some systematic resources about prompt learning (as well as pre-training) provided both within this survey and on companion websites:

-  : A website of prompt-based learning that contains: frequent updates to this survey, related slides, etc.
- Fig.1: A typology of important concepts for prompt-based learning.

[1] Even during this stage, there was some use of pre-trained models exemplified by word2vec (Mikolov et al., 2013b,a) and GloVe (Pennington et al., 2014), but they were used for only a limited portion of the final model parameters.

[2] This paradigm is less conducive to architectural exploration because (i) unsupervised pre-training allows models to learn with fewer structural priors, and (ii) as pre-training of models is time-consuming, experimenting with structural variants is costly.

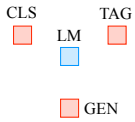
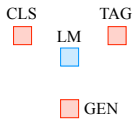
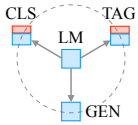
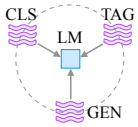
Paradigm	Engineering	Task Relation
a. Fully Supervised Learning (Non-Neural Network)	Features (e.g. word identity, part-of-speech, sentence length)	
b. Fully Supervised Learning (Neural Network)	Architecture (e.g. convolutional, recurrent, self-attentional)	
c. Pre-train, Fine-tune	Objective (e.g. masked language modeling, next sentence prediction)	
d. Pre-train, Prompt, Predict	Prompt (e.g. cloze, prefix)	

Table 1: Four paradigms in NLP. The "**engineering**" column represents the type of engineering to be done to build strong systems. The "**task relation**" column, shows the relationship between language models (LM) and other NLP tasks (CLS: classification, TAG: sequence tagging, GEN: text generation). : fully unsupervised training. : fully supervised training. : Supervised training combined with unsupervised training. indicates a textual prompt. Dashed lines suggest that different tasks can be connected by sharing parameters of pre-trained models. "LM→Task" represents *adapting LMs (objectives) to downstream tasks* while "Task→LM" denotes *adapting downstream tasks (formulations) to LMs*.

- Tab.7: A systematic and comprehensive comparison among different prompting methods.
- Tab.10: An organization of commonly-used prompts.
- Tab.12: A timeline of prompt-based research works.
- Tab.13: A systematic and comprehensive comparison among different pre-trained LMs.

2 A Formal Description of Prompting

2.1 Supervised Learning in NLP

In a traditional supervised learning system for NLP, we take an **input** $\boldsymbol{x}$, usually text, and predict an **output** $\boldsymbol{y}$ based on a model $P(\boldsymbol{y}|\boldsymbol{x};\theta)$. $\boldsymbol{y}$ could be a label, text, or other variety of output. In order to learn the parameters θ of this model, we use a dataset containing pairs of inputs and outputs, and train a model to predict this conditional probability. We will illustrate this with two stereotypical examples.

First, *text classification* takes an input text $\boldsymbol{x}$ and predicts a label y from a fixed label set $\mathcal{Y}$. To give an example, sentiment analysis (Pang et al., 2002; Socher et al., 2013) may take an input $\boldsymbol{x}$ ="I love this movie." and predict a label $y = ++$, out of a label set $\mathcal{Y} = \{++, +, \sim, -, --\}$.

Second, *conditional text generation* takes an input $\boldsymbol{x}$ and generates another text $\boldsymbol{y}$. One example is machine translation (Koehn, 2009), where the input is text in one language such as the Finnish $\boldsymbol{x}$ = "Hyvää huomenta." and the output is the English $\boldsymbol{y}$ = "Good morning"..

2.2 Prompting Basics

The main issue with supervised learning is that in order to train a model $P(\boldsymbol{y}|\boldsymbol{x};\theta)$, it is necessary to have supervised data for the task, which for many tasks cannot be found in large amounts. Prompt-based learning methods for NLP attempt to circumvent this issue by instead learning an LM that models the probability $P(\boldsymbol{x};\theta)$ of text $\boldsymbol{x}$ itself (details in §3) and using this probability to predict $\boldsymbol{y}$, reducing or obviating the need for large supervised datasets. In this section we lay out a mathematical description of the most fundamental form of prompting, which encompasses many works on prompting and can be expanded to cover others as well. Specifically, basic prompting predicts the highest-scoring $\hat{\boldsymbol{y}}$ in three steps.

Name	Notation	Example	Description
Input	$\boldsymbol{x}$	I love this movie.	One or multiple texts
Output	$\boldsymbol{y}$	++ (very positive)	Output label or text
Prompting Function	$f_{\text{prompt}}(\boldsymbol{x})$	`[X]` Overall, it was a `[Z]` movie.	A function that converts the input into a specific form by inserting the input $\boldsymbol{x}$ and adding a slot `[Z]` where answer $\boldsymbol{z}$ may be filled later.
Prompt	$\boldsymbol{x}'$	I love this movie. Overall, it was a `[Z]` movie.	A text where `[X]` is instantiated by input $\boldsymbol{x}$ but answer slot `[Z]` is not.
Filled Prompt	$f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z})$	I love this movie. Overall, it was a bad movie.	A prompt where slot `[Z]` is filled with any answer.
Answered Prompt	$f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z}^*)$	I love this movie. Overall, it was a good movie.	A prompt where slot `[Z]` is filled with a true answer.
Answer	$\boldsymbol{z}$	"good", "fantastic", "boring"	A token, phrase, or sentence that fills `[Z]`

Table 2: Terminology and notation of prompting methods. $\boldsymbol{z}^*$ represents answers that correspond to true output $\boldsymbol{y}^*$.

2.2.1 Prompt Addition

In this step a *prompting function* $f_{\text{prompt}}(\cdot)$ is applied to modify the input text $\boldsymbol{x}$ into a *prompt* $\boldsymbol{x}' = f_{\text{prompt}}(\boldsymbol{x})$. In the majority of previous work (Kumar et al., 2016; McCann et al., 2018; Radford et al., 2019; Schick and Schütze, 2021a), this function consists of a two step process:

1. Apply a *template*, which is a textual string that has two slots: an *input slot* `[X]` for input $\boldsymbol{x}$ and an *answer slot* `[Z]` for an intermediate generated *answer* text $\boldsymbol{z}$ that will later be mapped into $\boldsymbol{y}$.
2. Fill slot `[X]` with the input text $\boldsymbol{x}$.

In the case of sentiment analysis where $\boldsymbol{x}$ ="I love this movie.", the template may take a form such as "`[X]` Overall, it was a `[Z]` movie.". Then, $\boldsymbol{x}'$ would become "I love this movie. Overall it was a `[Z]` movie." given the previous example. In the case of machine translation, the template may take a form such as "Finnish: `[X]` English: `[Z]`", where the text of the input and answer are connected together with headers indicating the language. We show more examples in Tab. 3

Notably, (1) the prompts above will have an empty slot to fill in for $\boldsymbol{z}$, either in the middle of the prompt or at the end. In the following text, we will refer to the first variety of prompt with a slot to fill in the middle of the text as a *cloze prompt*, and the second variety of prompt where the input text comes entirely before $\boldsymbol{z}$ as a *prefix prompt*. (2) In many cases these template words are not necessarily composed of natural language tokens; they could be virtual words (e.g. represented by numeric ids) which would be embedded in a continuous space later, and some prompting methods even generate continuous vectors directly (more in §4.3.2). (3) The number of `[X]` slots and the number of `[Z]` slots can be flexibly changed for the need of tasks at hand.

2.2.2 Answer Search

Next, we search for the highest-scoring text $\hat{\boldsymbol{z}}$ that maximizes the score of the LM. We first define $\mathcal{Z}$ as a set of permissible values for $\boldsymbol{z}$. $\mathcal{Z}$ could range from the entirety of the language in the case of generative tasks, or could be a small subset of the words in the language in the case of classification, such as defining $\mathcal{Z} = \{$"excellent", "good", "OK", "bad", "horrible"$\}$ to represent each of the classes in $\mathcal{Y} = \{$++, +, ~, -, --$\}$.

We then define a function $f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z})$ that fills in the location `[Z]` in prompt $\boldsymbol{x}'$ with the potential answer $\boldsymbol{z}$. We will call any prompt that has gone through this process as a *filled prompt*. Particularly, if the prompt is filled with a true answer, we will refer to it as an *answered prompt* (Tab. 2 shows an example). Finally, we search over the set of potential answers $\boldsymbol{z}$ by calculating the probability of their corresponding filled prompts using a pre-trained LM $P(\cdot; \theta)$

$$\hat{\boldsymbol{z}} = \underset{\boldsymbol{z} \in \mathcal{Z}}{\text{search}}\, P(f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z}); \theta). \tag{1}$$

This search function could be an *argmax* search that searches for the highest-scoring output, or *sampling* that randomly generates outputs following the probability distribution of the LM.

2.2.3 Answer Mapping

Finally, we would like to go from the highest-scoring *answer* $\hat{\boldsymbol{z}}$ to the highest-scoring *output* $\hat{\boldsymbol{y}}$. This is trivial in some cases, where the answer itself is the output (as in language generation tasks such as translation), but there

Type	Task	Input (`[X]`)	Template	Answer (`[Z]`)
Text CLS	Sentiment	I love this movie.	`[X]` The movie is `[Z]`.	great fantastic ...
	Topics	He prompted the LM.	`[X]` The text is about `[Z]`.	sports science ...
	Intention	What is taxi fare to Denver?	`[X]` The question is about `[Z]`.	quantity city ...
Text-span CLS	Aspect Sentiment	Poor service but good food.	`[X]` What about service? `[Z]`.	Bad Terrible ...
Text-pair CLS	NLI	`[X1]`: An old man with ... `[X2]`: A man walks ...	`[X1]`? `[Z]`, `[X2]`	Yes No ...
Tagging	NER	`[X1]`: Mike went to Paris. `[X2]`: Paris	`[X1][X2]` is a `[Z]` entity.	organization location ...
Text Generation	Summarization	Las Vegas police ...	`[X]` TL;DR: `[Z]`	The victim ... A woman ... ...
	Translation	Je vous aime.	French: `[X]` English: `[Z]`	I love you. I fancy you. ...

Table 3: Examples of *input*, *template*, and *answer* for different tasks. In the **Type** column, "CLS" is an abbreviation for "classification". In the **Task** column, "NLI" and "NER" are abbreviations for "natural language inference" (Bowman et al., 2015) and "named entity recognition" (Tjong Kim Sang and De Meulder, 2003) respectively.

are also other cases where multiple answers could result in the same output. For example, one may use multiple different sentiment-bearing words (e.g. "excellent", "fabulous", "wonderful") to represent a single class (e.g. "++"), in which case it is necessary to have a mapping between the searched answer and the output value.

2.3 Design Considerations for Prompting

Now that we have our basic mathematical formulation, we elaborate a few of the basic design considerations that go into a prompting method, which we will elaborate in the following sections:

- **Pre-trained Model Choice:** There are a wide variety of pre-trained LMs that could be used to calculate $P(\boldsymbol{x}; \theta)$. In §3 we give a primer on pre-trained LMs, specifically from the dimensions that are important for interpreting their utility in prompting methods.
- **Prompt Engineering:** Given that the prompt specifies the task, choosing a proper prompt has a large effect not only on the accuracy, but also on which task the model performs in the first place. In §4 we discuss methods to choose which prompt we should use as $f_{\text{prompt}}(\boldsymbol{x})$.
- **Answer Engineering:** Depending on the task, we may want to design $\mathcal{Z}$ differently, possibly along with the mapping function. In §5 we discuss different ways to do so.
- **Expanding the Paradigm:** As stated above, the above equations represent only the simplest of the various underlying frameworks that have been proposed to do this variety of prompting. In §6 we discuss ways to expand this underlying paradigm to further improve results or applicability.
- **Prompt-based Training Strategies:** There are also methods to train parameters, either of the prompt, the LM, or both. In §7, we summarize different strategies and detail their relative advantages.

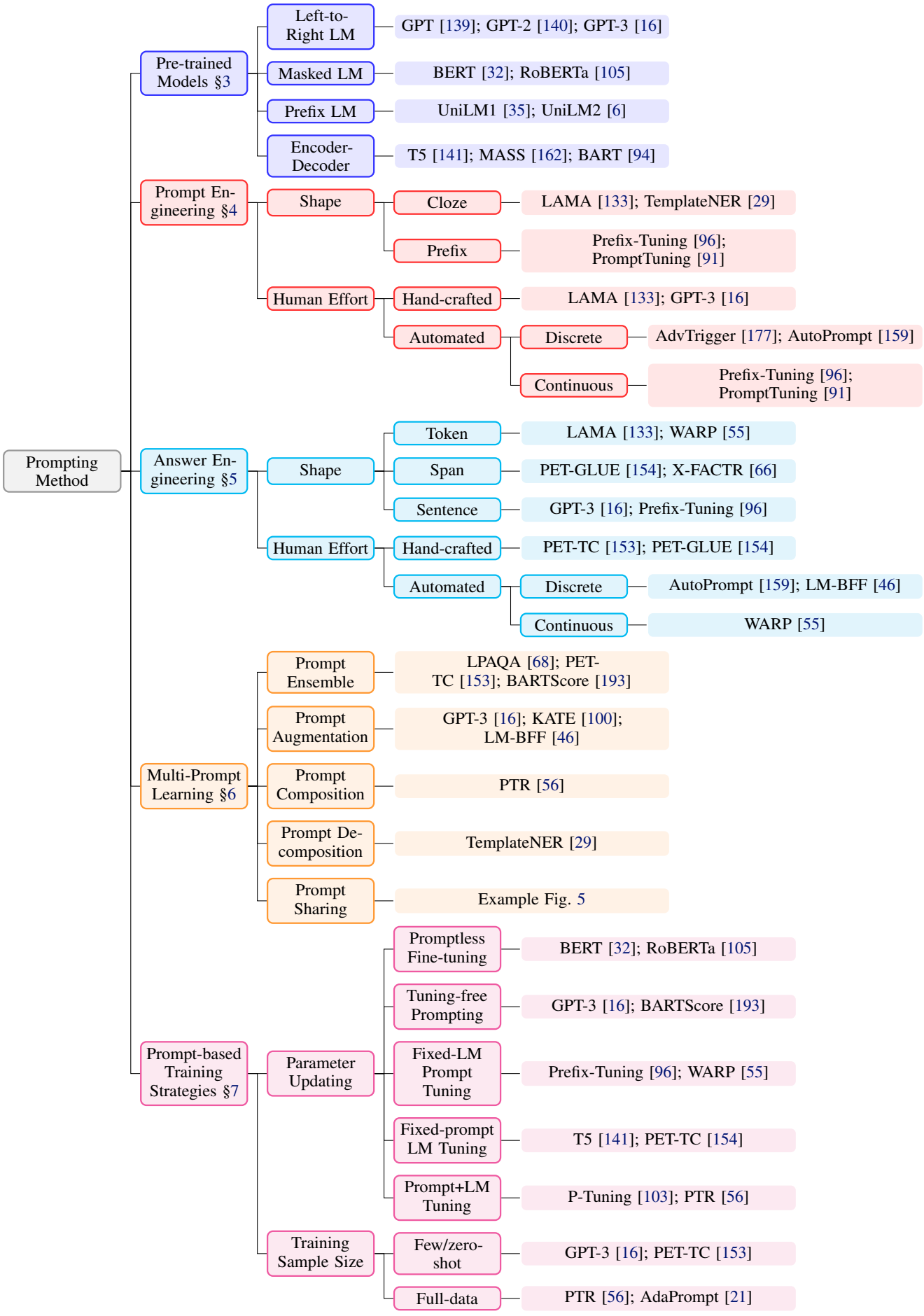


Figure 1: Typology of prompting methods.

3 Pre-trained Language Models

Given the large impact that pre-trained LMs have had on NLP in the pre-train and fine-tune paradigm, there are already a number of high-quality surveys that interested readers where interested readers can learn more (Raffel et al., 2020; Qiu et al., 2020; Xu et al., 2021; Doddapaneni et al., 2021). Nonetheless, in this chapter we present a systematic view of various pre-trained LMs which (i) organizes them along various axes in a more systematic way, (ii) particularly focuses on aspects salient to prompting methods. Below, we will detail them through the lens of *main training objective*, *type of text noising*, *auxiliary training objective*, *attention mask*, *typical architecture*, and *preferred application scenarios*. We describe each of these objectives below, and also summarize a number of pre-trained LMs along each of these axes in Tab. 13 in the appendix.

3.1 Training Objectives

The main training objective of a pre-trained LM almost invariably consists of some sort of objective predicting the probability of text $\boldsymbol{x}$.

Standard Language Model (SLM) objectives do precisely this, training the model to optimize the probability $P(\boldsymbol{x})$ of text from a training corpus (Radford et al., 2019). In these cases, the text is generally predicted in an *autoregressive* fashion, predicting the tokens in the sequence one at a time. This is usually done from left to right (as detailed below), but can be done in other orders as well.

A popular alternative to standard LM objectives are *denoising* objectives, which apply some noising function $\tilde{\boldsymbol{x}} = f_{\text{noise}}(\boldsymbol{x})$ to the input sentence (details in the following subsection), then try to predict the original input sentence given this noised text $P(\boldsymbol{x}|\tilde{\boldsymbol{x}})$. There are two common flavors of these objectives:

Corrupted Text Reconstruction (CTR) These objectives restore the processed text to its uncorrupted state by calculating loss over *only* the noised parts of the input sentence.

Full Text Reconstruction (FTR) These objectives reconstruct the text by calculating the loss over the *entirety* of the input texts whether it has been noised or not (Lewis et al., 2020a).

The main training objective of the pre-trained LMs plays an important role in determining its applicability to particular prompting tasks. For example, left-to-right autoregressive LMs may be particularly suitable for prefix prompts, whereas reconstruction objectives may be more suitable for cloze prompts. In addition, models trained with standard LM and FTR objectives may be more suitable for tasks regarding text generation, whereas other tasks such as classification can be formulated using models trained with any of these objectives.

In addition to the main training objectives above, a number of *auxiliary objectives* have been engineered to further improve models' ability to perform certain varieties of downstream tasks. We list some commonly-used auxiliary objectives in Appendix A.2.

3.2 Noising Functions

In training objectives based on reconstruction, the specific type of corruption applied to obtain the noised text $\tilde{\boldsymbol{x}}$ has an effect on the efficacy of the learning algorithm. In addition, prior knowledge can be incorporated by controlling the type of noise, e.g. the noise could focus on entities of a sentence, which allows us to learn a pre-trained model with particularly high predictive performance for entities. In the following, we introduce several types of noising functions, and give detailed examples in Tab. 4.

Operation	Element	Original Text	Corrupted Text
Mask	one token	Jane will move to New York .	Jane will `[Z]` to New York .
	two tokens	Jane will move to New York .	Jane will `[Z]` `[Z]` New York .
	one entity	Jane will move to New York .	Jane will move to `[Z]` .
Replace	one token	Jane will move to New York .	Jane will move `[X]` New York .
	two tokens	Jane will move to New York .	Jane will move `[X]` `[Y]` York .
	one entity	Jane will move to New York .	Jane will move to `[X]` .
Delete	one token	Jane ~~will~~ move to New York .	Jane move to New York .
	two token	Jane ~~will move~~ to New York .	Jane to New York .
Permute	token	Jane will move to New York .	New York . Jane will move to
Rotate	none	Jane will move to New York .	to New York . Jane will move
Concatenation	two languages	Jane will move to New York .	Jane will move to New York . `[/s]` 简将搬到纽约。

Table 4: Detailed examples for different noising operations.

Masking (e.g. Devlin et al. (2019)) The text will be masked in different levels, replacing a token or multi-token span with a special token such as `[MASK]`. Notably, masking can either be random from some distribution or specifically designed to introduce prior knowledge, such as the above-mentioned example of masking entities to encourage the model to be good at predicting entities.

Replacement (e.g. Raffel et al. (2020)) Replacement is similar to masking, except that the token or multi-token span is not replaced with a `[MASK]` but rather another token or piece of information (e.g., an image region (Su et al., 2020)).

Deletion (e.g. Lewis et al. (2020a)) Tokens or multi-token spans will be deleted from a text without the addition of `[MASK]` or any other token. This operation is usually used together with the FTR loss.

Permutation (e.g. Liu et al. (2020a)) The text is first divided into different spans (tokens, sub-sentential spans, or sentences), and then these spans are be permuted into a new text.

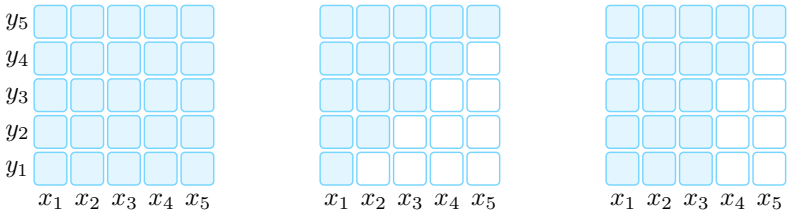
3.3 Directionality of Representations

A final important factor that should be considered in understanding pre-trained LMs and the difference between them is the directionality of the calculation of representations. In general, there are two widely used ways to calculate such representations:

Left-to-Right The representation of each word is calculated based on the word itself and all previous words in the sentence. For example, if we have a sentence "This is a good movie", the representation of the word "good" would be calculated based on previous words. This variety of factorization is particularly widely used when calculating standard LM objectives or when calculating the output side of an FTR objective, as we discuss in more detail below.

Bidirectional The representation of each word is calculated based on all words in the sentence, including words to the left of the current word. In the example above, "good" would be influenced by all words in the sentence, even the following "movie".

In addition to the two most common directionalities above, it is also possible to mix the two strategies together in a single model (Dong et al., 2019; Bao et al., 2020), or perform conditioning of the representations in a randomly permuted order (Yang et al., 2019), although these strategies are less widely used. Notably, when implementing these strategies within a neural model, this conditioning is generally implemented through *attention masking*, which masks out the values in an attentional model (Bahdanau et al., 2014), such as the popular Transformer architecture (Vaswani et al., 2017). Some examples of such attention masks are shown in Figure 2.



(a) Full. (b) Diagonal. (c) Mixture.

Figure 2: Three popular attention mask patterns, where the subscript t indicates the t-th timestep. A shaded box at (i, j) indicates that the attention mechanism is allowed to attend to the input element i at output time step j. A white box indicates that the attention mechanism is not allowed to attend to the corresponding i and j combination.

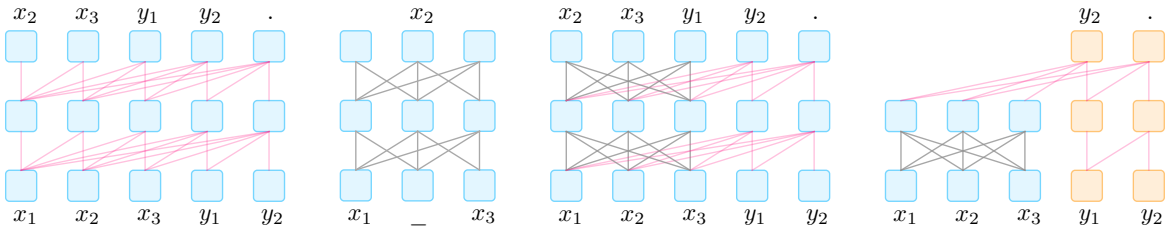
3.4 Typical Pre-training Methods

With the above concepts in mind, we introduce four popular pre-training methods, resulting from diverse combinations of objective, noising function, and directionality. These are described below, and summarized in Fig. 3 and Tab. 5.

3.4.1 Left-to-Right Language Model

Left-to-right LMs (L2R LMs), a variety of *auto-regressive LM*, predict the upcoming words or assign a probability $P(\boldsymbol{x})$ to a sequence of words $\boldsymbol{x} = x_1, \cdots, x_n$ (Jurafsky and Martin, 2021). The probability is commonly broken down using the chain rule in a left-to-right fashion: $P(\boldsymbol{x}) = P(x_1) \times \cdots P(x_n|x_1 \cdots x_{n-1})$.[3]

[3]Similarly, a right-to-left LM can predict preceding words based on the future context, such as $P(x_i|x_{i+1}, \cdots, x_n)$.



(a) Left-to-right LM. (b) Masked LM. (c) Prefix LM. (d) Encoder-Decoder.

Figure 3: Typical paradigms of pre-trained LMs.

Example & Applicable Scenario

Left-to-right LMs have been standard since their proposal by Markov in 1913 (Markov, 2006), and have been used continuously since then in both count-based (Goodman, 2001) and neural forms (Bengio et al., 2003; Mikolov et al., 2010; Radford and Narasimhan, 2018). Representative examples of modern pre-trained left-to-right LMs include GPT-3 (Brown et al., 2020), and GPT-Neo (Black et al., 2021).
L2R pre-trained LMs are also the popular backbone that many prompting methods adopt (Radford et al., 2019; Brown et al., 2020) . One practical reason for this is that many such models are large (PanGu-α (Zeng et al., 2021), Ernie-3 (Sun et al., 2021)) and ponderous to train, or not even available publicly. Thus using these models in the pre-train and fine-tune regimen is often not possible.

LMs	$\boldsymbol{x}$			$\boldsymbol{y}$			Application
	Mask	Noise	Main Obj.	Mask	Noise	Main Obj.	
L2R	Diagonal	None	SLM	-	-	-	NLU & NLG
Mask	Full	Mask	CTR	-	-	-	NLU
Prefix	Full	Any	CTR	Diagonal	None	SLM	NLU & NLG
En-De	Full	Any	None†	Diagonal	None	FTR/CRT	NLU & NLG

Table 5: Typical architectures for pre-trained LMs. $\boldsymbol{x}$ and $\boldsymbol{y}$ represent text to be encoded and decoded, respectively. **SLM**: Standard language model. **CTR**: Corrupted text reconstruction. **FTR**: Full text reconstruction. †: Encoder-decoder architectures usually apply objective functions to the decoder only.

3.4.2 Masked Language Models

While autoregressive language models provide a powerful tool for modeling the probability of text, they also have disadvantages such as requiring representations be calculated from left-to-right. When the focus is shifted to generating the optimal representations for down-stream tasks such as classification, many other options become possible, and often preferable. One popular bidirectional objective function used widely in representation learning is the *masked language model* (MLM; Devlin et al. (2019)), which aims to predict masked text pieces based on surrounded context. For example, $P(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ represents the probability of the word x_i given the surrounding context.

Example & Applicable Scenario

Representative pre-trained models using MLMs include: BERT (Devlin et al., 2019), ERNIE (Zhang et al., 2019; Sun et al., 2019b) and many variants. In prompting methods, MLMs are generally most suitable for natural language understanding or analysis tasks (e.g., text classification, natural language inference , and extractive question answering). These tasks are often relatively easy to be reformulated into cloze problems, which are consistent with the training objectives of the MLM. Additionally, MLMs have been a pre-trained model of choice when exploring methods that combine prompting with fine-tuning, elaborated further in §7.

3.4.3 Prefix and Encoder-Decoder

For conditional text generation tasks such as translation and summarization where an input text $\boldsymbol{x} = x_1, \cdots, x_n$ is given and the goal is to generate target text $\boldsymbol{y}$, we need a pre-trained model that is both capable of encoding the input text and generating the output text. There are two popular architectures for this purpose that share a common

thread of (1) using an encoder with fully-connected mask to encode the source $\boldsymbol{x}$ first and then (2) decode the target $\boldsymbol{y}$ auto-regressively (from the left to right).

Prefix Language Model The prefix LM is a left-to-right LM that decodes $\boldsymbol{y}$ conditioned on a prefixed sequence $\boldsymbol{x}$, which is encoded by the *same* model parameters but with a fully-connected mask. Notably, to encourage the prefix LM to learn better representations of the input, a corrupted text reconstruction objective is usually applied over $\boldsymbol{x}$, in addition to a standard conditional language modeling objective over $\boldsymbol{y}$.

Encoder-decoder The encoder-decoder model is a model that uses a left-to-right LM to decode $\boldsymbol{y}$ conditioned on a *separate* encoder for text $\boldsymbol{x}$ with a fully-connected mask; the parameters of the encoder and decoder are not shared. Similarly to the prefix LM, diverse types of noising can be applied to the input $\boldsymbol{x}$.

Example & Applicable Scenario

Prefix LMs have been explored in UniLM 1-2 (Dong et al., 2019; Bao et al., 2020) and ERNIE-M (Ouyang et al., 2020) while encoder-decoder models are widely used in pre-trained models such as T5 (Raffel et al., 2020), BART (Lewis et al., 2020a), MASS (Song et al., 2019) and their variants.
Pre-trained models with prefix LMs and encoder-decoder paradigms can be naturally used to text generation tasks with (Dou et al., 2021) or without (Yuan et al., 2021a; Liu and Liu, 2021) prompting using input texts. However, recent studies reveal that other non-generation tasks, such as information extraction (Cui et al., 2021), question answering (Khashabi et al., 2020) , and text generation evaluation (Yuan et al., 2021b) can be reformulated a generation problems by providing appropriate prompts. Therefore, prompting methods (i) broaden the applicability of these generation-oriented pre-trained models. For example, pre-trained models like BART are less used in NER while prompting methods make BART applicable, and (ii) breaks the difficulty of unified modelling among different tasks (Khashabi et al., 2020).

4 Prompt Engineering

Prompt engineering is the process of creating a prompting function $f_{\text{prompt}}(\boldsymbol{x})$ that results in the most effective performance on the downstream task. In many previous works, this has involved *prompt template engineering*, where a human engineer or algorithm searches for the best template for each task the model is expected to perform. As shown in the "Prompt Engineering" section of Fig.1, one must first consider the *prompt shape*, and then decide whether to take a *manual* or *automated* approach to create prompts of the desired shape, as detailed below.

4.1 Prompt Shape

As noted above, there are two main varieties of prompts: *cloze prompts* (Petroni et al., 2019; Cui et al., 2021), which fill in the blanks of a textual string, and *prefix prompts* (Li and Liang, 2021; Lester et al., 2021), which continue a string prefix. Which one is chosen will depend both on the task and the model that is being used to solve the task. In general, for tasks regarding generation, or tasks being solved using a standard auto-regressive LM, prefix prompts tend to be more conducive, as they mesh well with the left-to-right nature of the model. For tasks that are solved using masked LMs, cloze prompts are a good fit, as they very closely match the form of the pre-training task. Full text reconstruction models are more versatile, and can be used with either cloze or prefix prompts. Finally, for some tasks regarding multiple inputs such as *text pair classification*, prompt templates must contain space for two inputs, `[X1]` and `[X2]`, or more.

4.2 Manual Template Engineering

Perhaps the most natural way to create prompts is to manually create intuitive templates based on human introspection. For example, the seminal LAMA dataset (Petroni et al., 2019) provides manually created cloze templates to probe knowledge in LMs. Brown et al. (2020) create manually crafted prefix prompts to handle a wide variety of tasks, including question answering, translation, and probing tasks for common sense reasoning. Schick and Schütze (2020, 2021a,b) use pre-defined templates in a few-shot learning setting on text classification and conditional text generation tasks.

4.3 Automated Template Learning

While the strategy of manually crafting templates is intuitive and does allow solving various tasks with some degree of accuracy, there are also several issues with this approach: (1) creating and experimenting with these prompts is an art that takes time and experience, particularly for some complicated tasks such as semantic parsing (Shin et al., 2021); (2) even experienced prompt designers may fail to manually discover optimal prompts (Jiang et al., 2020c).

To address these problems, a number of methods have been proposed to automate the template design process. In particular, the automatically induced prompts can be further separated into *discrete prompts*, where the prompt is an

actual text string, and *continuous prompts*, where the prompt is instead described directly in the embedding space of the underlying LM.

One other orthogonal design consideration is whether the prompting function $f_{\text{prompt}}(\boldsymbol{x})$ is *static*, using essentially the same prompt template for each input, or *dynamic*, generating a custom template for each input. Both static and dynamic strategies have been used for different varieties of discrete and continuous prompts, as we will mention below.

4.3.1 Discrete Prompts

Works on discovering *discrete prompts* (a.k.a *hard prompts*) automatically search for templates described in a discrete space, usually corresponding to natural language phrases. We detail several methods that have been proposed for this below:

D1: Prompt Mining Jiang et al. (2020c)'s MINE approach is a mining-based method to automatically find templates given a set of training inputs $\boldsymbol{x}$ and outputs $\boldsymbol{y}$. This method scrapes a large text corpus (e.g. Wikipedia) for strings containing $\boldsymbol{x}$ and $\boldsymbol{y}$, and finds either the *middle words* or *dependency paths* between the inputs and outputs. Frequent middle words or dependency paths can serve as a template as in "`[X]` middle words `[Z]`".

D2: Prompt Paraphrasing Paraphrasing-based approaches take in an existing seed prompt (e.g. manually constructed or mined), and paraphrases it into a set of other candidate prompts, then selects the one that achieves the highest training accuracy on the target task. This paraphrasing can be done in a number of ways, including using round-trip translation of the prompt into another language then back (Jiang et al., 2020c), using replacement of phrases from a thesaurus (Yuan et al., 2021b), or using a neural prompt rewriter specifically optimized to improve accuracy of systems using the prompt (Haviv et al., 2021). Notably, Haviv et al. (2021) perform paraphrasing *after* the input $\boldsymbol{x}$ is input into the prompt template, allowing a different paraphrase to be generated for each individual input.

D3: Gradient-based Search Wallace et al. (2019a) applied a gradient-based search over actual tokens to find short sequences that can trigger the underlying pre-trained LM to generate the desired target prediction. This search is done in an iterative fashion, stepping through tokens in the prompt . Built upon this method, Shin et al. (2020) automatically search for template tokens using downstream application training samples and demonstrates strong performance in prompting scenarios.

D4: Prompt Generation Other works treat the generation of prompts as a text generation task and use standard natural language generation models to perform this task. For example, Gao et al. (2021) introduce the seq2seq pre-trained model T5 into the template search process. Since T5 has been pre-trained on a task of filling in missing spans, they use T5 to generate template tokens by (1) specifying the position to insert template tokens within a template[4] (2) provide training samples for T5 to decode template tokens. Ben-David et al. (2021) propose a domain adaptation algorithm that trains T5 to generate unique domain relevant features (DRFs; a set of keywords that characterize domain information) for each input. Then those DRFs can be concatenated with the input to form a template and be further used by downstream tasks.

D5: Prompt Scoring Davison et al. (2019) investigate the task of knowledge base completion and design a template for an input (head-relation-tail triple) using LMs. They first hand-craft a set of templates as potential candidates, and fill the input and answer slots to form a filled prompt. They then use a unidirectional LM to score those filled prompts, selecting the one with the highest LM probability. This will result in custom template for each individual input.

4.3.2 Continuous Prompts

Because the purpose of prompt construction is to find a method that allows an LM to effectively perform a task, rather than being for human consumption, it is not necessary to limit the prompt to human-interpretable natural language. Because of this, there are also methods that examine *continuous prompts* (a.k.a. *soft prompts*) that perform prompting directly in the embedding space of the model. Specifically, continuous prompts remove two constraints: (1) relax the constraint that the embeddings of template words be the embeddings of natural language (e.g., English) words. (2) Remove the restriction that the template is parameterized by the pre-trained LM's parameters. Instead, templates have their own parameters that can be tuned based on training data from the downstream task. We highlight several representative methods below.

[4]The number of template tokens do not need to be pre-specified since T5 can decode multiple tokens at a masked position.

C1: Prefix Tuning Prefix Tuning (Li and Liang, 2021) is a method that prepends a sequence of continuous task-specific vectors to the input, while keeping the LM parameters frozen. Mathematically, this consists of optimizing over the following log-likelihood objective given a trainable prefix matrix M_ϕ and a fixed pre-trained LM parameterized by θ.

$$\max_{\phi} \log P(\boldsymbol{y}|\boldsymbol{x}; \theta; \phi) = \max_{\phi} \sum_{y_i} \log P(y_i|h_{<i}; \theta; \phi) \tag{2}$$

In Eq. 2, $h_{<i} = [h_{<i}^{(1)}; \cdots; h_{<i}^{(n)}]$ is the concatenation of all neural network layers at time step i. It is copied from M_ϕ directly if the corresponding time step is within the prefix (h_i is $M_\phi[i]$), otherwise it is computed using the pre-trained LM.

Experimentally, Li and Liang (2021) observe that such continuous prefix-based learning is more sensitive to different initialization in low-data settings than the use of discrete prompts with real words. Similarly, Lester et al. (2021) prepend the input sequence with special tokens to form a template and tune the embeddings of these tokens directly. Compared to Li and Liang (2021)'s method, this adds fewer parameters as it doesn't introduce additional tunable parameters within each network layer. Tsimpoukelli et al. (2021) train a vision encoder that encodes an image into a sequence of embeddings that can be used to prompt a frozen auto-regressive LM to generate the appropriate caption. They show that the resulting model can perform few-shot learning for vision-language tasks such as visual question answering etc. Different from the above two works, the prefix used in (Tsimpoukelli et al., 2021) is sample-dependent, namely a representation of input images, instead of a task embedding.

C2: Tuning Initialized with Discrete Prompts There are also methods that initialize the search for a continuous prompt using a prompt that has already been created or discovered using discrete prompt search methods. For example, Zhong et al. (2021b) first define a template using a discrete search method such as AUTOPROMPT (Shin et al., 2020)'s, initialize virtual tokens based on this discovered prompt, then fine-tune the embeddings to increase task accuracy. This work found that initializing with manual templates can provide a better starting point for the search process. Qin and Eisner (2021) propose to learn a mixture of soft templates for each input where the weights and parameters for each template are jointly learned using training samples. The initial set of templates they use are either manually crafted ones or those obtained using the "prompt mining" method. Similarly, Hambardzumyan et al. (2021) introduce the use of a continuous template whose shape follows a manual prompt template.

C3: Hard-Soft Prompt Hybrid Tuning Instead of using a purely learnable prompt template, these methods insert some tunable embeddings into a hard prompt template. Liu et al. (2021b) propose "P-tuning", where continuous prompts are learned by inserting trainable variables into the embedded input. To account for interaction between prompt tokens, they represent prompt embeddings as the output of a BiLSTM (Graves et al., 2013). P-tuning also introduces the use of task-related anchor tokens (such as "capital" in relation extraction) within the template for further improvement. These anchor tokens are not tuned during training. Han et al. (2021) propose prompt tuning with rules (PTR), which uses manually crafted sub-templates to compose a complete template using logic rules. To enhance the representation ability of the resulting template, they also insert several virtual tokens whose embeddings can be tuned together with the pre-trained LMs parameters using training samples. The template tokens in PTR contain both actual tokens and virtual tokens. Experiment results demonstrate the effectiveness of this prompt design method in relation classification tasks.

5 Answer Engineering

In contrast to prompt engineering, which designs appropriate inputs for prompting methods, *answer engineering* aims to search for an answer space $\mathcal{Z}$ and a map to the original output $\mathcal{Y}$ that results in an effective predictive model. Fig.1's "Answer Engineering" section illustrates two dimensions that must be considered when performing answer engineering: deciding the *answer shape* and choosing an *answer design method*.

5.1 Answer Shape

The shape of an answer characterizes its granularity. Some common choices include:

- **Tokens:** One of the tokens in the pre-trained LM's vocabulary, or a subset of the vocabulary.
- **Span:** A short multi-token span. These are usually used together with cloze prompts.
- **Sentence:** A sentence or document. These are commonly used with prefix prompts.

In practice, how to choose the shape of acceptable answers depends on the task we want to perform. Token or text-span answer spaces are widely used in classification tasks (e.g. sentiment classification; Yin et al. (2019)), but also other tasks such as relation extraction (Petroni et al., 2019) or named entity recognition (Cui et al., 2021). Longer phrasal or sentential answers are often used in language generation tasks (Radford et al., 2019), but also

used in other tasks such as multiple-choice question answering (where the scores of multiple phrases are compared against each-other; Khashabi et al. (2020)).

5.2 Answer Space Design Methods

The next question to answer is how to design the appropriate answer space $\mathcal{Z}$, as well as the mapping to the output space $\mathcal{Y}$ if the answers are not used as the final outputs.

5.2.1 Manual Design

In manual design, the space of potential answers $\mathcal{Z}$ and its mapping to $\mathcal{Y}$ are crafted manually by an interested system or benchmark designer. There are a number of strategies that can be taken to perform this design.

Unconstrained Spaces In many cases, the answer space $\mathcal{Z}$ is the space of all tokens (Petroni et al., 2019), fixed-length spans (Jiang et al., 2020a), or token sequences (Radford et al., 2019). In these cases, it is most common to directly map answer $\boldsymbol{z}$ to the final output $\boldsymbol{y}$ using the identity mapping.

Constrained Spaces However, there are also cases where the space of possible outputs is constrained. This is often performed for tasks with a limited label space such as text classification or entity recognition, or multiple-choice question answering. To give some examples, Yin et al. (2019) manually design lists of words relating to relevant topics ("health", "finance", "politics", "sports", etc.), emotions ("anger", "joy", "sadness", "fear", etc.), or other aspects of the input text to be classified. Cui et al. (2021) manually design lists such as "person", "location", etc. for NER tasks. In these cases, it is necessary to have a mapping between the answer $\mathcal{Z}$ and the underlying class $\mathcal{Y}$.

With regards to multiple-choice question answering, it is common to use an LM to calculate the probability of an output among multiple choices, with Zweig et al. (2012) being an early example.

5.2.2 Discrete Answer Search

As with manually created prompts, it is possible that manually created answers are sub-optimal for getting the LM to achieve ideal prediction performance. Because of this, there is some work on automatic answer search, albeit less than that on searching for ideal prompts. These work on both discrete answer spaces (this section) and continuous answer spaces (the following).

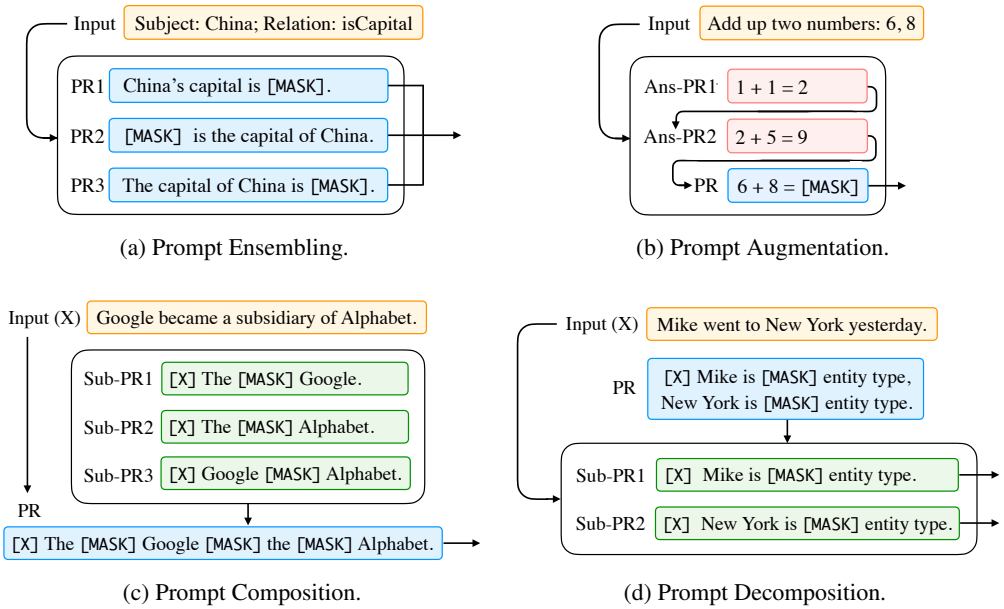
Answer Paraphrasing These methods start with an initial answer space $\mathcal{Z}'$, and then use paraphrasing to expand this answer space to broaden its coverage (Jiang et al., 2020b). Given a pair of answer and output $\langle \boldsymbol{z}', \boldsymbol{y} \rangle$, we define a function that generates a paraphrased set of answers $\text{para}(\boldsymbol{z}')$. The probability of the final output is then defined as the marginal probability *all* of the answers in this paraphrase set $P(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{z} \in \text{para}(\boldsymbol{z}')} P(\boldsymbol{z}|\boldsymbol{x})$. This paraphrasing can be performed using any method, but Jiang et al. (2020b) specifically use a back-translation method, first translating into another language then back to generate a list of multiple paraphrased answers.

Prune-then-Search In these methods, first, an initial pruned answer space of several plausible answers $\mathcal{Z}'$ is generated, and then an algorithm further searches over this pruned space to select a final set of answers. Note that in some of the papers introduced below, they define a function from label $\boldsymbol{y}$ to a single answer token $\boldsymbol{z}$, which is often called a *verbalizer* (Schick and Schütze, 2021a). Schick and Schütze (2021a); Schick et al. (2020) find tokens containing at least two alphabetic characters that are frequent in a large unlabeled dataset. In the search step, they iteratively compute a word's suitability as a representative answer $\boldsymbol{z}$ for a label $\boldsymbol{y}$ by maximizing the likelihood of the label over training data. Shin et al. (2020) learn a logistic classifier using the contextualized representation of the `[Z]` token as input. In the search step, they select the top-k tokens that achieve the highest probability score using the learned logistic classifier in the first step. Those selected tokens will form the answer. Gao et al. (2021) first construct a pruned search space $\mathcal{Z}'$ by selecting top-k vocabulary words based on their generation probability at the `[Z]` position determined by training samples. Then the search space is further pruned down by only selecting a subset of words within $\mathcal{Z}'$ based on their zero-shot accuracy on the training samples. (2) In the search step, they fine-tune the LM with fixed templates together with every answer mapping using training data and select the best label word as the answer based on the accuracy on the development set.

Label Decomposition When performing relation extraction, Chen et al. (2021b) automatically decompose each relation label into its constituent words and use them as an answer. For example, for the relation `per:city_of_death`, the decomposed label words would be {`person`, `city`, `death`}. The probability of the answer span will be calculated as the sum of each token's probability.

5.2.3 Continuous Answer Search

Very few works explore the possibility of using soft answer tokens which can be optimized through gradient descent. Hambardzumyan et al. (2021) assign a virtual token for each class label and optimize the token embedding for each



(a) Prompt Ensembling.

(b) Prompt Augmentation.

(c) Prompt Composition.

(d) Prompt Decomposition.

Figure 4: Different multi-prompt learning strategies. We use different colors to differentiate different components as follows. " " for input text, " " for prompt, " " for answered prompt. " " for sub-prompt. We use the following abbreviations. "PR" for prompt, "Ans-PR" for answered prompt, "Sub-PR" for sub-prompt.

class together with prompt token embeddings. Since the answer tokens are optimized directly in the embedding space, they do not make use of the embeddings learned by the LM and instead learn an embedding from scratch for each label.

6 Multi-Prompt Learning

The prompt engineering methods we discussed so far focused mainly on constructing a *single* prompt for an input. However, a significant body of research has demonstrated that the use of multiple prompts can further improve the efficacy of prompting methods, and we will call these methods *multi-prompt learning* methods. In practice, there are several ways to extend the single prompt learning to the use multiple prompts, which have a variety of motivations. We summarize representative methods in the "Multi-prompt Learning" section of Fig.1 as well as Fig.4.

6.1 Prompt Ensembling

Prompt ensembling is the process of using multiple *unanswered* prompts for an input at inference time to make predictions. An example is shown in Fig. 4-(a). The multiple prompts can either be discrete prompts or continuous prompts.[5] This sort of prompt ensembling can (1) leverage the complementary advantages of different prompts, (2) alleviate the cost of prompt engineering, since choosing one best-performing prompt is challenging, (3) stabilize performance on downstream tasks.

Prompt ensembling is connected to ensembling methods that are used to combine together multiple systems, which have a long history in machine learning (Ting and Witten, 1997; Zhou et al., 2002; Duh et al., 2011). Current research also borrows ideas from these works to derive effective ways for prompt ensembling, as described below.

Uniform averaging The most intuitive way to combine the predictions when using multiple prompts is to take the average of probabilities from different prompts. Concretely, this indicates that $P(\boldsymbol{z}|\boldsymbol{x}) \coloneqq \frac{1}{K}\sum_i^K P(\boldsymbol{z}|f_{\text{prompt},i}(\boldsymbol{x}))$ where $f_{\text{prompt},i}(\cdot)$ is the ith prompt in the prompt ensemble. Jiang et al. (2020c) first filter their prompts by selecting K prompts that achieve the highest accuracy on the training set, and then use the average log probabilities obtained from the top K prompts to calculate the probability for a single token at `[Z]` position when performing factual probing tasks. Schick and Schütze (2021a) also try a simple average when using an ensemble model to annotate an unlabeled dataset. When performing text generation evaluation, Yuan et al. (2021b) formulates this task as a text generation problem and take the average of the final generation scores obtained using different prompts.

Weighted averaging Simple uniform averaging of results from multiple prompts is easy to implement, but can also be suboptimal given that some prompts are more performant than others. To account for this, some works also

[5]Multiple continuous prompts are typically learned by using different initializations or different random seeds.

explore to use of weighted averages for prompt ensembling where each prompt is associated with a weight. The weights are typically pre-specified based on prompt performance or optimized using a training set. For example, Jiang et al. (2020c) learn the weight for each prompt by maximizing the probability of the target output over training data. Qin and Eisner (2021) use the same approach except that the weight for each prompt is optimized together with soft prompt parameters. Besides, Qin and Eisner (2021) also introduce a data-dependent weighting strategy where the probability of the input appearing in that prompt is considered in weighting different prompts as well. Schick and Schütze (2021a,b) set the weight for each prompt proportional to the accuracy on the training set before training.

Majority voting For classification tasks, majority voting can also be used to combine the results from different prompts (Lester et al., 2021; Hambardzumyan et al., 2021).

Knowledge distillation An ensemble of deep learning models can typically improve the performance, and this superior performance can be distilled into a single model using knowledge distillation (Allen-Zhu and Li, 2020). To incorporate this idea, Schick and Schütze (2021a,b, 2020) train a separate model for each manually-created template-answer pair, and use the ensemble of them to annotate an unlabeled dataset. Then the final model is trained to distill the knowledge from the annotated dataset. Gao et al. (2021) use a similar ensemble method on their automatically generated templates.

Prompt ensembling for text generation There is relatively little work on prompt ensembling for generation tasks (i.e. tasks where the answers is a string of tokens instead of a single one). A simple way to perform ensembling in this case is to use standard methods that generate the output based on the ensembled probability of the next word in the answer sequence $P(z_t|\boldsymbol{x}, z_{<t}) \coloneqq \frac{1}{K}\sum_i^K P(z_t|f_{\text{prompt},i}(\boldsymbol{x}), z_{<t})$. In contrast, Schick and Schütze (2020) train a separate model for each prompt $f_{\text{prompt},i}(\boldsymbol{x})$, and thus storing each of these fine-tuned LMs in memory is infeasible. Instead, they first decode generations using each model and then score each generation by averaging their generation probability across all models.

6.2 Prompt Augmentation

Prompt augmentation, also sometimes called *demonstration learning* (Gao et al., 2021), provides a few additional *answered prompts* that can be used to demonstrate how the LM should provide the answer to the actual prompt instantiated with the input $\boldsymbol{x}$. For example, instead of just providing a prompt of "China's capital is `[Z]` .", the prompt can be prefaced by a few examples such as "Great Britain's capital is London . Japan's capital is Tokyo . China's capital is `[Z]` ." Another example of performing addition of two numbers can be found in Fig. 4-(b). These few-shot demonstrations take advantage of the ability of strong language models to learn repetitive patterns (Brown et al., 2020).

Although the idea of prompt augmentation is simple, there are several aspects that make it challenging: (1) *Sample Selection:* how to choose the most effective examples? (2) *Sample Ordering:* How to order the chosen examples with the prompt?

Sample Selection Researchers have found that the choice of examples used in this few-shot scenario can result in very different performance, ranging from near state-of-the-art accuracy on some tasks to near random guess (Lu et al., 2021). To address this issue, Gao et al. (2021); Liu et al. (2021a) utilize sentence embeddings to sample examples that are close to the input in this embedding space. To measure the generalization capability of pre-trained LMs to perform new tasks based on instructions, Mishra et al. (2021) provide both positive samples and negative samples that highlight things to avoid.

Sample Ordering Lu et al. (2021) found that the order of answered prompts provided to the model plays an important role in model performance, and propose entropy-based methods to score different candidate permutations. Kumar and Talukdar (2021) search for a good permutation of training examples as augmented prompts and learn a separator token between the prompts for further gains in performance.

Prompt augmentation is closely related to retrieval-based methods that provide more textual context to the model to improve performance (Guu et al., 2018), a method which has also been shown to be effective in prompt-based learning (Petroni et al., 2020). However, the key difference lies in the fact that prompt augmentation also leverages the template and answer, while larger context learning does not.

6.3 Prompt Composition

For those composable tasks, which can be composed based on more fundamental subtasks, we can also perform *prompt composition*, using multiple sub-prompts, each for one subtask, and then defining a composite prompt based on those sub-prompts. This process is illustrated in Fig. 4-(c). For example, in the relation extraction task, which aims to extract the relation of two entities, we can break down the task into several subtasks including identifying the characteristics of entities and classifying the relationships between entities. Based on this intuition, Han et al.

(2021) first use multiple manually created sub-prompts for entity recognition and relation classification and then compose them into a complete prompt based on logic rules for relation extraction.

6.4 Prompt Decomposition

For tasks where multiple predictions should be performed for one sample (e.g., sequence labeling), directly defining a holistic prompt with regards to the entire input text $\boldsymbol{x}$ is challenging. One intuitive method to address this problem is to break down the holistic prompt into different sub-prompts, and then answer each sub-prompt separately. Fig.4-(d) illustrates this idea with an example from the named entity recognition task, which aims to identify all named entities in an input sentence. In this case, the input will first be converted into a set of text spans, and the model can then be prompted to predict the entity type (including "Not an Entity") for each span. It is not easy to predict all the span types at the same time due to the large number of spans, so different prompts for each span can be created and predicted separately. This sort of *prompt decomposition* for named entity recognition has been explored by Cui et al. (2021) where they apply the approach we discussed here.

7 Training Strategies for Prompting Methods

With the methods in the above sections, it is now clear how to obtain an appropriate prompt (or prompts) and corresponding answers. Now we discuss about methods that explicitly train models in concert with prompting methods, as outlined in the "Training Strategies" section of Fig.1.

7.1 Training Settings

In many cases, prompting methods can be used without *any* explicit training of the LM for the down-stream task, simply taking an LM that has been trained to predict the probability of text $P(\boldsymbol{x})$ and applying it as-is to fill the cloze or prefix prompts defined to specify the task. This is traditionally called the *zero-shot* setting, as there is zero training data for the task of interest.

However, there are also methods that use training data to train the model in concert with prompting methods. These consist of either *full-data learning*, where a reasonably large number of training examples are used to train the model, or *few-shot learning* where a very small number of examples are used to train the model. Prompting methods are particularly useful in the latter case, as there are generally not enough training examples to fully specify the desired behavior, and thus using a prompt to push the model in the right direction is particularly effective.

One thing to note is that for many of the prompt engineering methods described in §4, although annotated training samples are not explicitly used in the training of the downstream task model, they *are* often used in the construction or validation of the prompts that the downstream task will use. As noted by Perez et al. (2021), this is arguably not true zero-shot learning with respect to the downstream task.

7.2 Parameter Update Methods

In prompt-based downstream task learning, there are usually two types of parameters, namely those from (1) pre-trained models and (2) prompts. Which part of parameters should be updated is one important design decision, which can lead to different levels of applicability in different scenarios. We summarize five tuning strategies (as shown in Tab. 6) based on (i) whether the parameters of the underlying LM are tuned, (ii) whether there are additional prompt-related parameters, (iii) if there are additional prompt-related parameters, whether those parameters are tuned.

Strategy	LM Params	Prompt Params		Example
		Additional	Tuned	
Promptless Fine-tuning	Tuned	-		ELMo [130], BERT [32], BART [94]
Tuning-free Prompting	Frozen	✗	✗	GPT-3 [16], AutoPrompt [159], LAMA [133]
Fixed-LM Prompt Tuning	Frozen	✓	Tuned	Prefix-Tuning [96], Prompt-Tuning [91]
Fixed-prompt LM Tuning	Tuned	✗	✗	PET-TC [153], PET-Gen [152], LM-BFF [46]
Prompt+LM Fine-tuning	Tuned	✓	Tuned	PADA [8], P-Tuning [103], PTR [56]

Table 6: Characteristics of different tuning strategies. "Additional" represents if there are additional parameters beyond LM parameters while "Tuned" denotes if parameters are updated.

7.2.1 Promptless Fine-tuning

As mentioned in the introduction, the *pre-train and fine-tune* strategy has been widely used in NLP since before the popularization of prompting methods. Here we refer to pre-training and fine-tuning *without* prompts as *promptless fine-tuning*, to contrast with the prompt-based learning methods introduced in the following sections. In this strategy, given a dataset of a task, all (or some (Howard and Ruder, 2018; Peters et al., 2019)) of the parameters of the pre-trained LM will be updated via gradients induced from downstream training samples. Typical examples of pre-trained models tuned in this way include BERT [32] and RoBERTa [105]. This is a simple, powerful, and widely-used method, but it may overfit or not learn stably on small datasets (Dodge et al., 2020). Models are also prone to *catastrophic forgetting*, where the LM loses its ability to do things that it was able to do before fine-tuning (McCloskey and Cohen, 1989).

- **Advantages**: Simplicity, no need for prompt design. Tuning all the LM parameters allows the model to fit to larger training datasets.
- **Disadvantages**: LMs may overfit or not learn stably on smaller datasets.

7.2.2 Tuning-free Prompting

Tuning-free prompting directly generates the answers without changing the parameters of the pre-trained LMs based only on a prompt, as described in the simplest incarnation of prompting in §2. These can be optionally augmenting input with answered prompts as described in §6.2, and this combination of tuning-free prompting and prompt augmentation is also referred to as *in-context learning* (Brown et al., 2020). Typical examples of tuning-free prompting include LAMA [133] and GPT-3 [16].

- **Advantages**: Efficiency, there is no parameter update process. No catastrophic forgetting, as LM parameters remain fixed. Applicable in zero-shot settings.
- **Disadvantages**: Because prompts are the only method that provide the task specification, heavy engineering is necessary to achieve high accuracy. In particular in the in-context learning setting, providing many answered prompts can be slow at test time, and thus cannot easily use large training datasets.

7.2.3 Fixed-LM Prompt Tuning

In the scenario where additional prompt-relevant parameters are introduced besides parameters of the pre-trained model, *fixed-LM prompt tuning* updates only the prompts' parameters using the supervision signal obtained from the downstream training samples, while keeping the entire pre-trained LM unchanged. Typical examples are Prefix-Tuning [96] and WARP [55].

- **Advantages**: Similarly to tuning-free prompting, it can retain knowledge in LMs and is suitable in few-shot scenarios. Often superior accuracy to tuning-free prompting.
- **Disadvantages**: Not applicable in zero-shot scenarios. While effective in few-shot scenarios, representation power is limited in large-data settings. Prompt engineering through choice of hyperparameters or seed prompts is necessary. Prompts are usually not human-interpretable or manipulable.

7.2.4 Fixed-prompt LM Tuning

Fixed-prompt LM tuning tunes the parameters of the LM, as in the standard pre-train and fine-tune paradigm, but additionally uses prompts with fixed parameters to specify the model behavior. This potentially leads to improvements, particularly in few-shot scenarios.

The most natural way to do so is to provide a discrete textual template that is applied to every training and test example. Typical examples include PET-TC [153], PET-Gen [152], LM-BFF [46]. Logan IV et al. (2021) more recently observe that the prompt engineering can be reduced by allowing for a combination of answer engineering and partial LM fine-tuning. For example, they define a very simple template, *null prompt*, where the input and mask are directly concatenated "`[X][Z]`" without any template words, and find this achieves competitive accuracy.

- **Advantages**: Prompt or answer engineering more completely specify the task, allowing for more efficient learning, particularly in few-shot scenarios.
- **Disadvantages**: Prompt or answer engineering are still required, although perhaps not as much as without prompting. LMs fine-tuned on one downstream task may not be effective on another one.

7.2.5 Prompt+LM Tuning

In this setting, there are prompt-relevant parameters, which can be fine-tuned together with the all or some of the parameters of the pre-trained models. Representative examples include PADA [8], P-Tuning [103]. Notably, this setting is very similar to the standard pre-train and fine-tune paradigm, but the addition of the prompt can provide additional bootstrapping at the start of model training.

- **Advantages**: This is the most expressive method, likely suitable for high-data settings.
- **Disadvantages**: Requires training and storing all parameters of the models. May overfit to small datasets.

8 Applications

In previous sections, we examined prompting methods from the point of view of the mechanism of the method itself. In this section, we rather organize prompting methods from the point of view of which applications they have been applied to. We list these applications in Tab. 7-8 and summarize them in the following sections.

8.1 Knowledge Probing

Factual Probing *Factual probing* (a.k.a. fact retrieval) is one of the earliest scenarios with respect to which prompting methods were applied. The motivation of exploring this task is to quantify how much factual knowledge the pre-trained LM's internal representations bear. In this task, parameters of pre-trained models are usually fixed, and knowledge is retrieved by transforming the original input into a cloze prompt as defined in §2.2, which can be manually crafted or automatically discovered. Relevant datasets including `LAMA` (Petroni et al., 2019) and `X-FACTR` (Jiang et al., 2020a). Since the answers are pre-defined, fact retrieval only focuses on finding effective templates and analyzing the results of different models using these templates. Both discrete template search (Petroni et al., 2019, 2020; Jiang et al., 2020c,a; Haviv et al., 2021; Shin et al., 2020; Perez et al., 2021) and continuous template learning (Qin and Eisner, 2021; Liu et al., 2021b; Zhong et al., 2021b) have been explored within this context, as well as prompt ensemble learning (Jiang et al., 2020c; Qin and Eisner, 2021).

Linguistic Probing Besides factual knowledge, large-scale pre-training also allows LMs to handle linguistic phenomena such as analogies (Brown et al., 2020), negations (Ettinger, 2020), semantic role sensitivity (Ettinger, 2020), semantic similarity (Sun et al., 2021), cant understanding (Sun et al., 2021), and rare word understanding (Schick and Schütze, 2020). The above knowledge can also be elicited by presenting *linguistic probing* tasks in the form of natural language sentences that are to be completed by the LM.

8.2 Classification-based Tasks

Prompt-based learning has been widely explored in classification-based tasks where prompt templates can be constructed relatively easily, such as text classification (Yin et al., 2019) and natural language inference (Schick and Schütze, 2021a). The key to prompting for classification-based tasks is reformulating it as an appropriate prompt. For example, Yin et al. (2019) use a prompt such as "the topic of this document is `[Z]`.", which is then fed into mask pre-trained LMs for slot filling.

Text Classification For *text classification* tasks, most previous work has used cloze prompts, and both prompt engineering (Gao et al., 2021; Hambardzumyan et al., 2021; Lester et al., 2021) and answer engineering (Schick and Schütze, 2021a; Schick et al., 2020; Gao et al., 2021) have been explored extensively. Most existing works explore the efficacy of prompt learning for text classification in the context of *few-shot* setting with "*fixed-prompt LM Tuning*" strategies (defined in §7.2.4).

Natural Language Inference (NLI) NLI aims to predict the relationship (e.g., `entailment`) of two given sentences. Similar to text classification tasks, for *natural language inference* tasks, cloze prompts are commonly used (Schick and Schütze, 2021a). Regarding prompt engineering, researchers mainly focus on the template search in the few-shot learning setting and the answer space $\mathcal{Z}$ is usually manually pre-selected from the vocabulary.

Work	Task	PLM	Setting	Prompt Engineering			Answer Engineering			Tuning	Mul-Pr
				Shape	Man	Auto	Shape	Man	Auto		
LMComm [173]	CR	L2R	Zero	Clo	✓	-	Sp	✓	-	TFP	-
GPT-2 [140]	CR,QA SUM,MT	GPT-2	Zero,Few	Clo,Pre	✓	-	Tok,Sp,Sen	✓	-	TFP	PA
WNLaMPro [150]	LCP	BERT	Zero	Clo	✓	-	Tok	✓	-	TFP	-
LMDiagnose [39]	CR,LCP	BERT	Zero	Clo	✓	-	Tok	✓	-	TFP	-
AdvTrigger [177]	GCG	GPT-2	Full	Pre	-	Disc	Sen	✓	-	TFP	-
CohRank [31]	CKM	BERT	Zero	Clo	✓	-	Tok,Sp	✓	-	TFP	-
LAMA [133]	FP	Conv,Trans ELMo,BERT	Zero	Clo	✓	-	Tok	✓	-	TFP	-
CTRL [75]	GCG	CTRL	Full	Pre	✓	-	Sen	✓	-	LMT	-
T5 [141]	TC,SUM QA,MT	T5	Full	Pre	✓	-	Tok,Sp,Sen	✓	-	LMT	-
Neg & Mis [74]	FP	Trans,ELMo BERT	Zero	Clo	✓	-	Tok	✓	-	TFP	-
LPAQA [68]	FP	BERT,ERNIE	Full	Clo	✓	Disc	Tok	✓	-	TFP	PE
ZSC [135]	TC	GPT-2	Full	Pre	✓	-	Tok,Sp	✓	-	LMT	-
PET-TC [153]	TC	RoBERTa,XLM-R	Few	Pre	✓	-	Tok	✓	Disc	LMT	PE
ContxFP [132]	FP	BERT,RoBERTa	Zero	Clo	✓	Disc	Tok	✓	-	TFP	-
UnifiedQA [76]	QA	T5,BART	Full	Prefix	✓	-	Tok,Sp,Sen	✓	-	LMT	-
RAG [95]	QA,GCG,TC	BART	Full	Pre	-	Disc	Tok,Sp,Sen	✓	-	LMPT	PE
GPT-3 [16]	QA,MT,GCG CR,TC,LCP MR,SR,AR	GPT-3	Zero,Few	Clo,Pre	✓	-	Tok,Sp,Sen	✓	-	TFP	PA
CommS2S [187]	CR	T5	Full	Pre	✓	-	Tok	✓	-	LMT	-
PET-SGLUE [154]	TC	ALBERT	Few	Clo	✓	-	Tok,Sp	✓	-	LMT	PE
ToxicityPrompts [47]	GCG	GPT-1,GPT-2 GPT-3,CTRL	Zero	Pre	✓	-	N/A			TFP	-
WhyLM [147]	Theory	GPT-2	Full	Pre	✓	-	Tok	✓	-	PT	-
X-FACTR [66]	FP	mBERT,BERT XLM,XLM-R	Zero	Clo	✓	-	Tok,Sp	✓	-	TFP	-
Petal [149]	TC	RoBERTa	Few	Clo	✓	-	Tok	-	Disc	LMT	PE
AutoPrompt [159]	TC,FP,IE	BERT,RoBERTa	Full	Clo	-	Disc	Tok	-	Disc	TFP	-
CTRLsum [59]	SUM	BART	Full	Pre	✓	-	Sen	✓	-	LMT	-
PET-Gen [152]	SUM	PEGASUS	Few	Pre	✓	-	Sen	✓	-	LMT	PE
LM-BFF [46]	TC	RoBERTa	Few	Clo	-	Disc	Tok	-	Disc	LMT	PE,PA
WARP [55]	TC	RoBERTa	Few,Full	Clo,Pre	✓	Cont	Tok	✓	Cont	PT	PE
Prefix-Tuning [96]	D2T,SUM	GPT-2,BART	Full	Pre	-	Cont	Sen	✓	-	PT	-
KATE [100]	TC,D2T,QA	GPT-3	Few	Pre	✓	-	Tok,Sp,Sen	✓	-	TFP	PA
PromptProg [145]	MT,MR AR,QA	GPT-3	Zero,Few	Pre	✓	-	Tok,Sp,Sen	✓	-	TFP	PA
ContxCalibrate [201]	TC,FP,IE	GPT-2,GPT-3	Few	Pre	✓	-	Tok,Sp	✓	-	TFP	PA
PADA [8]	TC,TAG	T5	Full	Pre	-	Disc	N/A			LMPT	-
SD [155]	GCG	GPT-2	Zero	Pre	✓	-	N/A			TFP	-
BERTese [58]	FP	BERT	Full	Clo	✓	Disc	Tok	✓	-	TFP	-
Prompt2Data [148]	TC	RoBERTa	Full	Clo	✓	-	Tok,Sp	✓	-	LMT	-
P-Tuning [103]	FP,TC	GPT-2,BERT ALBERT	Few,Full	Clo,Pre	✓	Cont	Tok,Sp	✓	-	TFP,LMPT	-
GLM [37]	TC	GLM	Full	Clo	✓	-	Tok,Sp	✓	-	LMT	-

Table 7: An organization of works on prompting (Part 1). See the caption of Tab. 8 for a detailed description for all the abbreviations used in this table.

Work	Task	PLM	Setting	Prompt Engineering			Answer Engineering			Tuning	Mul-Pr
				Shape	Man	Auto	Shape	Man	Auto		
ADAPET [170]	TC	ALBERT	Few	Clo	✓	-	Tok,Sp	✓	-	LMT	-
Meta [202]	TC	T5	Full	Pre	✓	-	Tok	✓	-	LMT	-
OptiPrompt [203]	FP	BERT	Full	Clo	✓	Cont	Tok	✓	-	PT	-
Soft [137]	FP	BERT,BART RoBERTa	Full	Clo	✓	Cont	Tok	✓	-	PT	PE
DINO [151]	GCG	GPT-2	Zero	Pre	✓	-	N/A			TFP	-
AdaPrompt [21]	IE	BERT	Few,Full	Clo	✓	-	Tok	-	Disc	LMT	-
PMI_{DC} [62]	GCG,QA,TC	GPT-2,GPT-3	Zero	Pre	✓	-	Tok,Sp,Sen	✓	-	TFP	-
Prompt-Tuning [91]	TC	T5	Full	Pre	-	Cont	Tok,Sp	✓	-	PT	PE
Natural-Instr [120]	GCG	GPT-3,BART	Few,Full	Pre	✓	-	Tok,Sp,Sen	✓	-	TFP,LMT	PA
OrderEntropy [111]	TC	GPT-2,GPT-3	Few	Pre	✓	-	Tok	✓	-	TFP	PA
FewshotSemp [158]	SEMP	GPT-3	Few	Pre	✓	-	Sen	✓	-	TFP	PA
PanGu-α [194]	QA,CR,TC SUM,GCG	PanGu-α	Zero,Few	Clo,Pre	✓	-	Tok,Sp,Sen	✓	-	TFP	PA
TrueFewshot [129]	TC,FP	GPT-2,GPT-3 ALBERT	Few	Clo,Pre	✓	Disc	Tok,Sp	✓	-	TFP,LMT	-
PTR [56]	IE	RoBERTa	Full	Clo	✓	Cont	Tok,Sp	✓	-	LMPT	PC
TemplateNER [29]	TAG	BART	Few,Full	Clo,Pre	✓	-	Tok	✓	-	LMT	PD
PERO [83]	TC,FP	BERT,RoBERTa	Few	Pre	✓	-	Tok	✓	-	TFP	PA
PromptAnalysis [181]	Theory	BERT	Full	Clo	-	Cont	N/A			PT	-
CPM-2 [198]	QA,MR,SUM TC,GCG,MT	CPM-2	Full	Pre	-	Cont	Tok,Sp,Sent	✓	-	PT,LMPT	-
BARTScore [193]	EVALG	BART	Zero	Pre	✓	Disc	Sen	✓	-	TFP	PE
NullPrompt [109]	TC	RoBERTa,ALBERT	Few	Pre	✓	-	Tok	✓	-	LMPT	-
Frozen [174]	VQA,VFP,MG	GPT-like	Full	Pre	-	Cont	Sp (Visual)	✓	-	PT	PA
ERNIE-B3 [167]	TC,LCP,NLI CR,QA,SUM GCG	ERNIE-B3	Zero	Clo,Pre	✓	-	Tok,Sp,Sen	✓	-	TFP	-
Codex [20]	CodeGen	GPT	Zero,Few Full	Pre	✓	-	Span	✓	Disc	TFP,LMT	PA
HTLM [1]	TC,SUM	BART	Zero,Few Full	Clo	✓	Disc	Tok,Sp,Sen	✓	-	LMT	PA
FLEX [15]	TC	T5	Zero,Few	Pre	✓	-	Tok,Sp	✓	-	LMT	-

Table 8: An organization of works on prompting (Part 2). The **Task** column lists the tasks that are performed in corresponding papers. We use the following abbreviations. **CR**: Commonsense Reasoning. **QA**: Question Answering. **SUM**: Summarization. **MT**: Machine Translation. **LCP**: Linguistic Capacity Probing. **GCG**: General Conditional Generation. **CKM**: Commonsense Knowledge Mining. **FP**: Fact Probing. **TC**: Text Classification. **MR**: Mathematical Reasoning. **SR**: Symbolic Reasoning. **AR**: Analogical Reasoning. **Theory**: Theoretical Analysis. **IE**: Information Extraction. **D2T**: Data-to-text. **TAG**: Sequence Tagging. **SEMP**: Semantic Parsing. **EVALG**: Evaluation of Text Generation. **VQA**: Visual Question Answering. **VFP**: Visual Fact Probing. **MG**: Multimodal Grounding. **CodeGen**: Code generation. The **PLM** column lists all the pre-trained LMs that have been used in corresponding papers for downstream tasks. **GPT-like** is an autoregressive language model which makes small modifications to the original GPT-2 architecture. For other pre-trained LMs, please refer to §3 for more information. **Setting** column lists the settings for prompt-based learning, can be zero-shot learning (**Zero**), few-shot learning (**Few**), fully supervised learning (**Full**). Under **Prompt Engineering**, **Shape** denotes the shape of the template (**Clo** for cloze and **Pre** for prefix), **Man** denotes whether human effort is needed, **Auto** denotes data-driven search methods (**Disc** for discrete search, **Cont** for continuous search). Under **Answer Engineering**, **Shape** indicates the shape of the answer (**Tok** for token-level, **Sp** for span-level, **Sen** for sentence- or document-level), and **Man** and **Auto** are the same as above. The **Tuning** column lists tuning strategies (§7). **TFP**: Tuning-free Prompting. **LMT**: Fixed-prompt LM Tuning. **PT**: Fixed-LM Prompt Tuning. **LMPT**: LM+Prompt Tuning. The **Mul-Pr** column lists multi-prompt learning methods. **PA**: Prompt Augmentation. **PE**: Prompt Ensembling. **PC**: Prompt Composition. **PD**: Prompt Decomposition.

8.3 Information Extraction

Unlike classification tasks where cloze questions can often be intuitively constructed, for *information extraction* tasks constructing prompts often requires more finesse.

Relation Extraction *Relation extraction* is a task of predicting the relation between two entities in a sentence. Chen et al. (2021b) first explored the application of *fixed-prompt LM Tuning* in relation extraction and discuss two major challenges that hinder the direct inheritance of prompting methodology from classification tasks: (1) The larger label space (e.g. 80 in relation extraction v.s 2 in binary sentiment classification) results in more difficulty in answer engineering. (2) In relation extraction, different tokens in the input sentence may be more or less important (e.g. entity mentions are more likely to participate in a relation), which, however, can not be easily reflected in the prompt templates for classification since the original prompt template regards each word equally. To address the above problems, Chen et al. (2021b) propose an adaptive answer selection method to address the issue (1) and task-oriented prompt template construction for the issue (2), where they use special markers (e.g. `[E]`) to highlight the entity mentions in the template. Similarly, Han et al. (2021) incorporate entity type information via multiple prompt composition techniques (illustrated in Fig. 4).

Semantic Parsing *Semantic parsing* is a task of generating a structured meaning representation given a natural language input. Shin et al. (2021) explore the task of few-shot semantic parsing using LMs by (1) framing the semantic parsing task as a paraphrasing task (Berant and Liang, 2014) and (2) constraining the decoding process by only allowing output valid according to a grammar. They experiment with the *in-context learning* setting described in §7.2.2, choosing answered prompts that are semantically close to a given test example (determined by the conditional generation probability of generating a test sample given another training example). The results demonstrate the effectiveness of the paraphrasing reformulation for semantic parsing tasks using pre-trained LMs.

Named Entity Recognition *Named entity recognition* (NER) is a task of identifying named entities (e.g., person name, location) in a given sentence. The difficulty of prompt-based learning's application to tagging tasks, exemplified as NER, is that, unlike classification, (1) each unit to be predicted is a token or span instead of the whole input text, (2) there is a latent relationship between the token labels in the sample context. Overall, the application of prompt-based learning in tagging task has not been fully explored. Cui et al. (2021) recently propose a template-based NER model using BART, which enumerates text spans and considers the generation probability of each type within manually crafted templates. For example, given an input "Mike went to New York yesterday", to determine what type of entity "Mike" is, they use the template "Mike is a `[Z]` entity", and the answer space $\mathcal{Z}$ consists of values such as "person" or "organization".

8.4 "Reasoning" in NLP

There is still a debate[6] about if deep neural networks are capable of performing "reasoning" or just memorizing patterns based on large training data (Arpit et al., 2017; Niven and Kao, 2019). As such, there have been a number of attempts to probe models' reasoning ability by defining benchmark tasks that span different scenarios. We detail below how prompting methods have been used in these tasks.

Commonsense Reasoning There are a number of benchmark datasets testing commonsense reasoning in NLP systems (Huang et al., 2019; Rajani et al., 2019; Lin et al., 2020; Ponti et al., 2020). Some commonly attempted tasks involve solving Winograd Schemas (Levesque et al., 2012), which require the model to identify the antecedent of an ambiguous pronoun within context, or involve completing a sentence given multiple choices. For the former, an example could be "The trophy doesn't fit into the brown suitcase because it is too large." And the task for the model is to infer whether "it" refers to the trophy or the "suitcase". By replacing "it" with its potential candidates in the original sentences and calculating the probability of the different choices, pre-trained LMs can perform quite well by choosing the choice that achieves the highest probability (Trinh and Le, 2018). For the latter, an example could be "Eleanor offered to fix her visitor some coffee. Then she realized she didn't have a clean `[Z]`.". The candidate choices are "cup", "bowl" and "spoon". The task for the pre-trained LM is to choose the one from the three candidates that most conforms to common sense. For these kinds of tasks, we can also score the generation probability of each candidate and choose the one with the highest probability (Ettinger, 2020).

Mathematical Reasoning Mathematical reasoning is the ability to solve mathematical problems, e.g. arithmetic addition, function evaluation. Within the context of pre-trained LMs, researchers have found that pre-trained embeddings and LMs can perform simple operations such as addition and subtraction when the number of digits is small, but fail when the numbers are larger (Naik et al., 2019; Wallace et al., 2019b; Brown et al., 2020). Reynolds and McDonell (2021) explore more complex mathematical (e.g. $f(\boldsymbol{x}) = \boldsymbol{x} * \boldsymbol{x}$, what is $f(f(3))$?) reasoning problems and improve LM performance through serializing reasoning for the question.

[6] e.g. https://medium.com/reconstruct-inc/the-golden-age-of-computer-vision-338da3e471d1

8.5 Question Answering

Question answering (QA) aims to answer a given input question, often based on a context document. QA can take a variety of formats, such as extractive QA (which identifies content from the context document containing the answer; e.g. SQuAD (Rajpurkar et al., 2016)), multiple-choice QA (where the model has to pick from several choices; e.g. RACE (Lai et al., 2017)), and free-form QA (where the model can return an arbitrary textual string as a response; e.g. NarrativeQA (Kočiský et al., 2018)). Generally, these different formats have been handled using different modeling frameworks. One benefit of solving QA problems with LMs, potentially using prompting methods, is that different formats of QA tasks can be solved within the same framework. For example, Khashabi et al. (2020) reformulate many QA tasks as a text generation problem by fine-tuning seq2seq-based pre-trained models (e.g. T5) and appropriate prompts from the context and questions. Jiang et al. (2020b) take a closer look at such prompt-based QA systems using sequence to sequence pre-trained models (T5, BART, GPT2) and observe that probabilities from these pre-trained models on QA tasks are not very predictive of whether the model is correct or not.

8.6 Text Generation

Text generation is a family of tasks that involve generating text, usually conditioned on some other piece of information. Prompting methods can be easily applied to these tasks by using *prefix prompts* together with autoregressive pre-trained LMs. Radford et al. (2019) demonstrated impressive ability of such models to perform generation tasks such as text summarization and machine translation using prompts such as "translate to french, `[X]`, `[Z]`". Brown et al. (2020) perform *in-context learning* (§7.2.2) for text generation, creating a prompt with manual templates and augmenting the input with multiple *answered prompts*. Schick and Schütze (2020) explore *fixed-prompt LM tuning* (§7.2.4) for few-shot text summarization with manually crafted templates. (Li and Liang, 2021) investigate *fixed-LM prompt tuning* (§7.2.3) for text summarization and data-to-text generation in few-shot settings, where learnable prefix tokens are prepended to the input while parameters in pre-trained models are kept frozen. Dou et al. (2021) explored the *prompt+LM tuning* strategy (§7.2.5) on text summarization task, where learnable prefix prompts are used and initialized by different types of guidance signals, which can then be updated together with parameters of pre-trained LMs.

8.7 Automatic Evaluation of Text Generation

Yuan et al. (2021b) have demonstrated that prompt learning can be used for automated evaluation of generated texts. Specifically, they conceptualize the evaluation of generated text as a text generation problem, modeled using a pre-trained sequence-to-sequence, and then use *prefix prompts* that bring the evaluation task closer to the pre-training task. They experimentally find that simply adding the phrase "such as" to the translated text when using pre-trained models can lead to a significant improvement in correlation on German-English machine translation (MT) evaluation.

8.8 Multi-modal Learning

Tsimpoukelli et al. (2021) shift the application of prompt learning from text-based NLP to the *multi-modal* setting (vision and language). Generally, they adopt the *fixed-LM prompt tuning* strategy together with *prompt augmentation* techniques. They specifically represent each image as a sequence of continuous embeddings, and a pre-trained LM whose parameters are frozen is prompted with this prefix to generate texts such as image captions. Empirical results show few-shot learning ability: with the help of a few demonstrations (answered prompts), system can rapidly learn words for new objects and novel visual categories.

8.9 Meta-Applications

There are also a number of applications of prompting techniques that are not NLP tasks in and of themselves, but are useful elements of training strong models for any application.

Domain Adaptation Domain adaptation is the practice of adapting a model from one domain (e.g. news text) to another (e.g. social media text). Ben-David et al. (2021) use self-generated *domain related features* (DRFs) to augment the original text input and perform sequence tagging as a sequence-to-sequence problem using a seq2seq pre-trained model.

Debiasing Schick et al. (2021) found that LMs can perform self-diagnosis and self-debiasing based on biased or debiased instructions. For example, to self-diagnosis whether the generated text contains violent information, we can use the following template "The following text contains violence. `[X][Z]`". Then we fill `[X]` with the input text and look at the generation probability at `[Z]`, if the probability of "Yes" is greater than "No", then we would assume the given text contains violence, and vice versa. To perform debiasing when generating text, we first compute the probability of the next word $P(x_t|\boldsymbol{x}_{<t};\theta)$ given the original input. Then we compute the probability