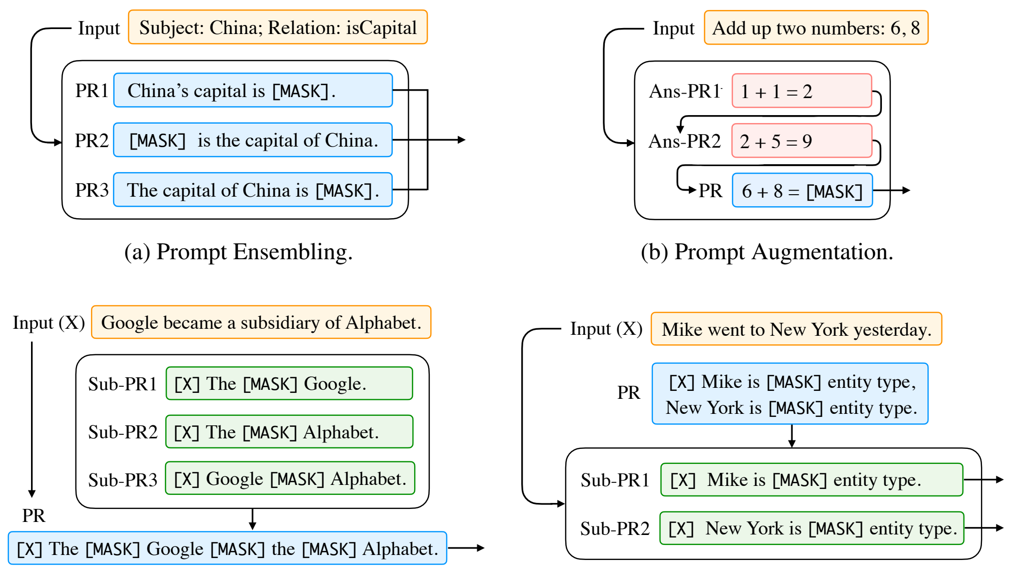
Input
Subject: China; Relation: isCapital
PR1
China's capital is [MASK].
PR2
[MASK] is the capital of China.
PR3
The capital of China is [MASK].
Input
Add up two numbers: 6, 8
Ans-PR1
1 + 1 = 2
Ans-PR2
2 + 5 = 9
PR
6 + 8 = [MASK]
Input (X)
Google became a subsidiary of Alphabet.
Sub-PR1
[X] The [MASK] Google.
Sub-PR2
[X] The [MASK] Alphabet.
Sub-PR3
[X] Google [MASK] Alphabet.
PR
[X] The [MASK] Google [MASK] the [MASK] Alphabet.
Input (X)
Mike went to New York yesterday.
PR
[X] Mike is [MASK] entity type, New York is [MASK] entity type.
Sub-PR1
[X] Mike is [MASK] entity type.
Sub-PR2
[X] New York is [MASK] entity type.

(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 $P(\boldsymbol{z}|\boldsymbol{x}) := \frac{1}{K}\sum_i^K P(\boldsymbol{z}|f_{\text{prompt},i}(\boldsymbol{x}))$ 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 $P(z_t|\boldsymbol{x}, z_{<t}) := \frac{1}{K}\sum_i^K P(z_t|f_{\text{prompt},i}(\boldsymbol{x}), z_{<t})$. 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.