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).

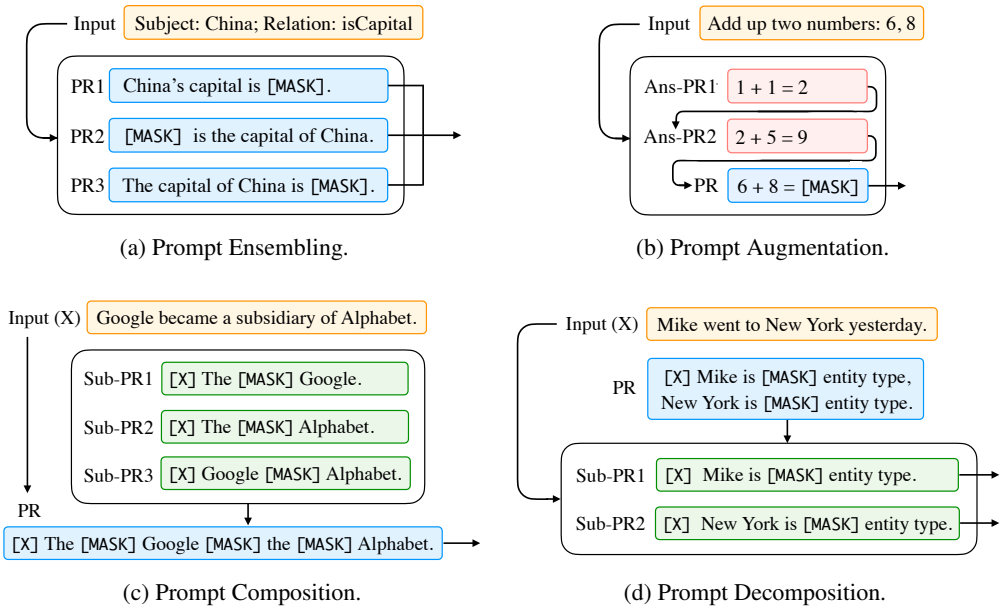
Answer Paraphrasing These methods start with an initial answer space $\mathcal{Z}'$, and then use paraphrasing to expand this answer space to broaden its coverage (Jiang et al., 2020b). Given a pair of answer and output $\langle \boldsymbol{z}', \boldsymbol{y} \rangle$, we define a function that generates 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.