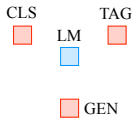
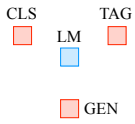
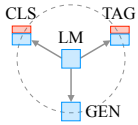
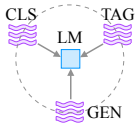
Paradigm	Engineering	Task Relation
a. Fully Supervised Learning (Non-Neural Network)	Features (e.g. word identity, part-of-speech, sentence length)	
b. Fully Supervised Learning (Neural Network)	Architecture (e.g. convolutional, recurrent, self-attentional)	
c. Pre-train, Fine-tune	Objective (e.g. masked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

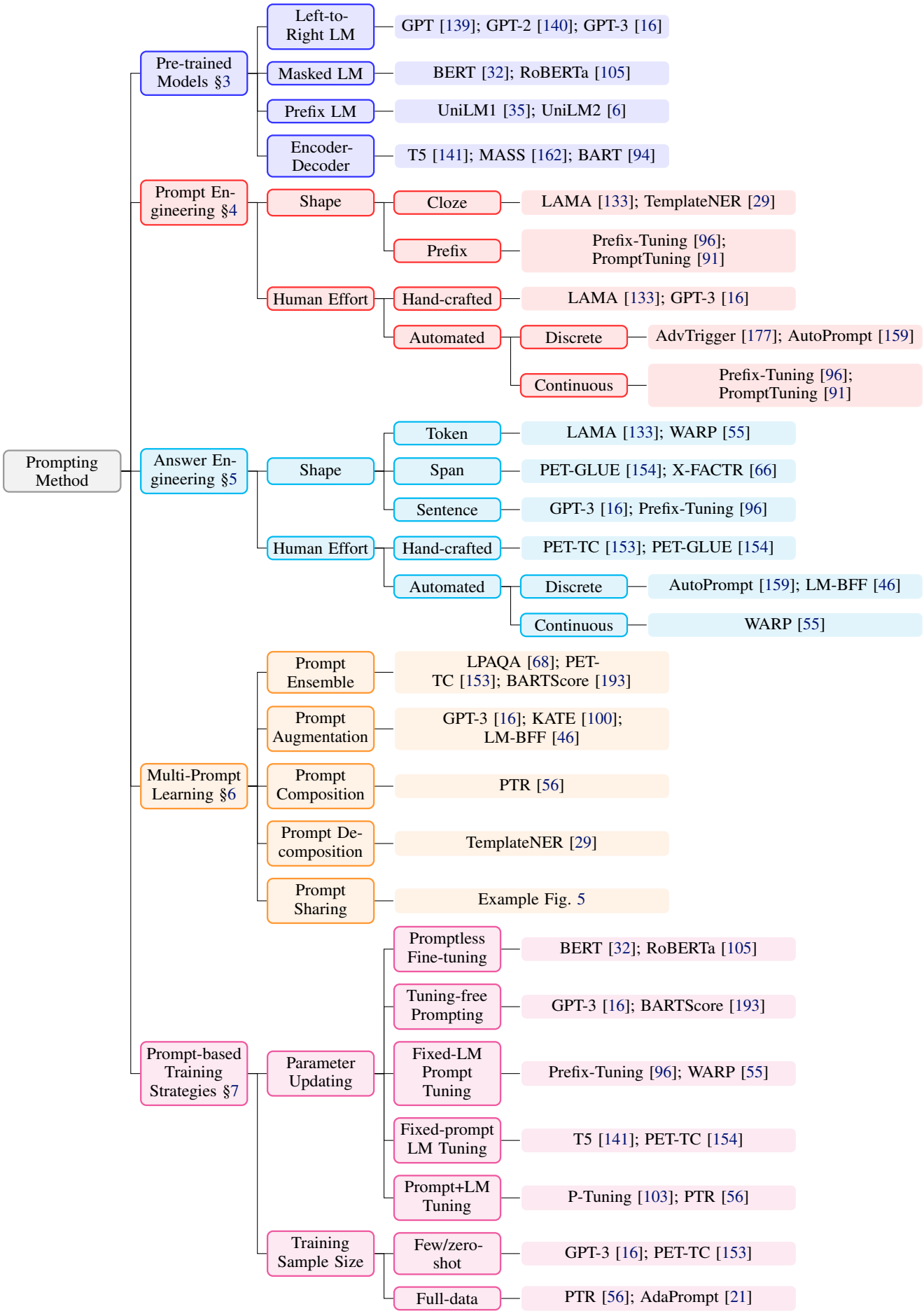


Figure 1: Typology of prompting methods.