```
Dante was born in [MASK]
```
(3.4)

Because the model has been tuned on data that answers this question, "Florence" will be calculated as a highly probable fill for this mask token. This example demonstrates a basic and fundamental promise of prompt-based inference from LLMs: the possibility of using LLMs as knowledge bases.

This is in contrast to the use of standard knowledge bases, the development of which requires significant efforts in a) the extraction of relational knowledge from various data sources, and b) NLP pipeline solutions for entity extraction, co-reference resolution, entity linking, and relation extraction (Petroni et al., 2019). Each of these NLP pipeline requirements has challenges, and errors are inevitable. This can mean that the utility of the resulting knowledge base is particularly sensitive to errors propagating through and accumulating within the NLP pipeline (Petroni et al., 2019). A conceptual comparison of the two approaches is shown in Fig. 3.9.

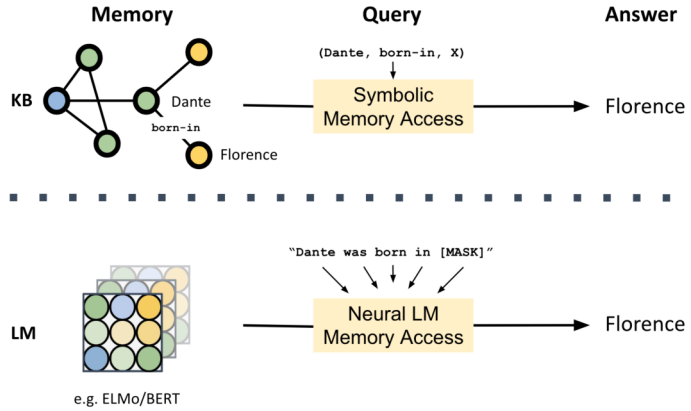

Fig. 3.9: Querying traditional knowledge bases and language models for factual information. In this example, the knowledge base (above the dashed line) has been purposefully designed to be queried for the entity relation, Dante, born in Florence. This is in contrast to the language model (below the dashed line) which was designed to predict masked words given associated context, and can therefore be induced to report facts that it was exposed to during pre-training.

LLM prompting holds a few key advantages over using standard knowledge bases. First, it is schema-free, as its relational knowledge is built within the language model as an emergent property of the pre-training process rather than as a specific task against which the model is developed. It is also highly generalizable given the vast scope of information contained in modern pre-training corpora. In theory, the same language model can support many di-

verse knowledge-based use cases and a much more comprehensive range of common NLP tasks.

Practical Tips

Significant challenges and risks are associated with LLM-based knowledge extraction. Explainability is difficult because tracing the casual events leading to a specific response from prompt-based inference is often impossible. The accuracy of these responses must also be validated. The knowledge we are trying to elicit from a language model is an emergent property of the training process used during pre-training. As such, it has not been intentionally trained to learn these knowledge facts. Similarly, the datasets used in pre-training are impractically large from a knowledge validation/quality ranking perspective, and where these data have come from the internet, a similar lack of epistemological analysis can result in similarly untrue "facts". Thus, users should maintain a healthy skepticism and safeguard against these errors with sound evaluation methodologies. Finally, the consistency of a prompt-based knowledge base strongly depends on the quality of the engineered prompt. We will discuss optimization approaches in detail in Sect. 3.3.

3.2.4 Prompt-based Learning Across NLP Tasks

Numerous NLP tasks are well suited to the prompt-based paradigm. In this section, we list many common tasks that can be accomplished with prompting, including a description of their inputs, templates, prompts, and answer mappings. By understanding these elements and their interactions, we aim to provide a comprehensive view of how NLP tasks can be effectively adapted and executed within the prompt-based learning framework.

We divide these tasks into three broad NLP categories:

1. **Text classification:** This category involves assigning an appropriate class label to a given input sentence. For these tasks, the prompt is designed to accept the input sentence and includes a dedicated slot for generating intermediate answers, which can later be mapped to classification labels.
2. **Tagging**: This category involves assigning labels or tags to individual elements within a given text, such as words or phrases. For these tasks, the prompt includes the string of text containing the element to be tagged and then queries specifically about that element, providing options for the model to decide between.
3. **Text generation**: This category involves generating a string of text, generally more than just one token, to accomplish a task given in the prompt. For these tasks, the prompt includes some relevant context, such as a paragraph to sum-

marize or a sentence to translate, and a specific directive to the model for what to do with the context.

Table 3.1 lists seventeen total tasks that fall within these three categories, gives a short description of the task, and a sample input, template, and answer space that can be used to accomplish the task. The wide variety of use cases exemplifies the flexibility of prompt-based learning. However, prompts must be carefully crafted to suit each individual task. In the next section, we will further break down the process into several areas that can be optimized to achieve the best results from prompt-based learning.

3.3 Prompt Engineering

In the previous section, we discussed how various NLP tasks can be solved with prompts, illustrated through several straightforward examples. The precise formulation of these prompts is critical for achieving good results. The development of suitable prompting functions to optimize performance on a target tasks downstream is referred to as *prompt engineering*. The process of designing prompts necessitates meticulous consideration and the integration of various elements. These elements include the selection of pre-trained models, the determination of the optimal prompt shape, the engineering of prompt templates, and answer engineering. Template engineering approaches fall broadly into two categories:

- manual templates
- automated templates

The former uses human expertise and trial-and-error to arrive at an optimized prompt, and the latter uses various automated processes to discern the best approach template for a given task. Fig. 3.10 shows an overview of the structure of the next two sections. In the following section, we will introduce basic terminology central to prompt categorization, overview the manual prompt engineering approach, and detail several automated approaches used in the literature.

3.3.1 Prompt Shape

Prompt templates can be broadly categorized into two main types: (a) prefix prompts and (b) cloze prompts. We refer to these as types of *prompt shape*.

Table 3.1: Summary of prompt-based NLP approaches. Each row contains an NLP task with a definition on the left, and an example on the right. The example includes an input sentence to perform the task on, a suggested template for prompt-based inference, and a potential answer space. These tasks are divided into three categories: text classification, tagging, and text generation.

Text Classification

Task	Example
Sentiment analysis: Classifying the sentiment of a text as positive, negative, or neutral.	**Input**: I hate this movie. **Template**: [x] It was a [z] movie. **Answers**: great, terrible, · · ·
Author attribution: Identifying the author of a given text from a predefined set of authors.	**Input**: It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness **Template**: The author of [x] is most likely [z]. **Answers**: Dickens, Carroll, Austin, · · ·
Spam detection: Classifying an email or text message as spam or not spam.	**Input**: Congratulations! You have won! Click here to claim your free vacation. **Template**: This message: [x] is classified as [z]. **Answers**: Spam, Non-Spam
Emotion classification: Classifying the emotion expressed in a text from a predefined set of emotions.	**Input**: I just won the lottery! **Template**: This text: [x] expresses the emotion [z]. **Answer**s: anger, surprise, sadness, happiness
Intent detection: Identifying the intent behind a user's query or message, often used in chatbots and virtual assistants.	**Input**: What's the weather like today? **Template**: [x] The user's intent is [z]. **Answers**: get_weather, set_alarm
Language identification: Determining the language in which a given text is written.	**Input**: ¿Cómo estás? **Template**: [x] The language is [z]. **Answers**: Spanish, French, · · ·
Hate speech detection: Identifying whether a given text contains hate speech.	**Input**: I can't stand them. **Template**: [x] The text contains [z] speech. **Answers**: hate, non-hate

Tagging

Task	Example
Part-of-speech (POS) tagging: Assigning grammatical categories to words, such as nouns, verbs, adjectives, and adverbs.	**Input**: She is running in the park. **Template**: In the sentence $[\mathbf{x}_1, \cdots, \mathbf{x}_n]$, the word $[\mathbf{x}_i]$ has POS-tag $[\mathbf{z}_j]$. **Answers**: noun, verb, adjective, · · ·
Named entity recognition (NER): Identifying and classifying entities mentioned in the text, such as people, dates, locations, organizations, etc.	**Input**: John met Mary in London. **Template**: In the sentence $[\mathbf{x}_1, \cdots, \mathbf{x}_n]$, the word $[\mathbf{x}_i]$ the named entity label is $[\mathbf{z}_j]$. **Answers**: location, organization, · · ·
Chunking or shallow parsing: Grouping adjacent words or tokens into larger units called "chunks" based on their grammatical structure, such as noun phrases or verb phrases.	**Input**: She is running in the park. **Template**: In the sentence $[\mathbf{x}_1, \cdots, \mathbf{x}_n]$, the word $[\mathbf{x}_i]$ the chunk label is $[\mathbf{z}_j]$. **Answers**: 'B-VP' - beginning of a verb phrase, 'I-VP' - inside a verb phrase, · · ·

Continued on next page

Table 3.1 – *Continued from previous page*

Task	Example
Dependency parsing: Identifying syntactic dependencies between words in a sentence, which includes labeling words as subjects, objects, modifiers, etc., and showing their relationships.	**Input**: She is running in the park. **Template**: In the sentence $[\mathbf{x}_1, \cdots, \mathbf{x}_n]$, the word $[\mathbf{x}_i]$ the dependency relation is $[\mathbf{z}_j]$. **Answers**: 'nsubj' - nominal subject, 'root' - root of the sentence, 'dobj' - direct object, $\cdots$
Constituent parsing or phrase structure parsing: Identifying the constituent structure of sentences, where words are grouped into grammatical phrases, such as noun phrases (NPs) and verb phrases (VPs).	**Input**: She is running in the park. **Template**: In the sentence $[\mathbf{x}_1, \cdots, \mathbf{x}_n]$, the word $[\mathbf{x}_i]$ the constituent category is $[\mathbf{z}_j]$. **Answers**: 'NP' - noun phrase, 'PP' - prepositional phrase, 'VP' - verb phrase, $\cdots$
Semantic role labeling (SRL): Assigning roles to words or phrases in a sentence, such as agent, instrument, etc., based on their semantic relationships with the predicate (usually a verb).	**Input**: John gave Mary a book. **Template**: In the sentence $[\mathbf{x}_1, \cdots, \mathbf{x}_n]$, the word $[\mathbf{x}_i]$ has semantic role $[\mathbf{z}_j]$. **Answers**: Agent, Theme, Location, $\cdots$
Coreference resolution: Identifying words or phrases in a text that refer to the same entity and linking them together.	**Input**: Jane is a talented software engineer. She was recently promoted to team lead. **Template**: In the text with words: $[\mathbf{x}_1, \cdots, \mathbf{x}_n]$, does the word $[\mathbf{x}_i]$ refer to the word $\mathbf{x}_j$? **Answers**: Yes, No

Text Generation

Task	Example
Summarization: Given a long piece of text, generate a shorter version that captures the original text's main points or key information.	**Input**: <Long text to be summarized.> **Template**: Please provide a summary for the following text: [**x**]. Summary: [**z**]. **Answer**: <summarized version of the long text>
Question-answering: Given a question and a context, generate an answer based on the information available in the context.	**Input**: <Context or passage>, <question> **Template**: Here is the context: [**x**] What is the answer to the question: [**w**] ? Answer: [**z**] . **Answer**: <answer to the question based on the context>
Machine translation: Translating a piece of text from one language to another while preserving the original meaning and context.	**Input**: ¿Cómo estás? **Template**: Translate the following text from the source language to the target language: [**x**] Translation: [**z**]. **Answer**: <translated text in target language>

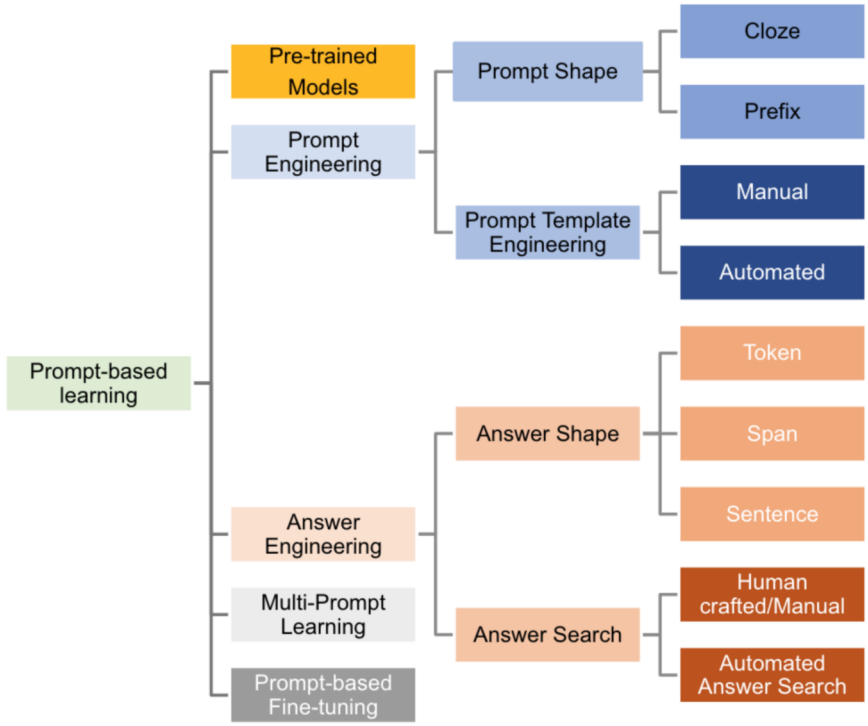

Fig. 3.10: Relationships between the various design options and design decisions within the prompt-based learning paradigm.

3.3.1.1 Prefix Prompts

In a *prefix prompt*, both the input and a string referring to the input are positioned as a prefix to the answer slot. For instance, in the context of movie sentiment analysis, a prefix prompt template can be formulated as

$$\text{“}\mathbf{x}' = [\mathbf{x}] \text{ This movie is } [\mathbf{z}]\text{”}, \tag{3.5}$$

When this template is applied to the input, it generates a filled prompt, such as, “Cannot watch this movie. This movie is $[\mathbf{z}]$”.

Prefix prompts tend to perform well in tasks where the response is directly linked to the input with left-to-right mappings, and a simple, unambiguous connection can be established. For example, in machine translation tasks, prefix prompts can effectively generate translations by providing clear guidance on the target language. Consider this template:

$$\text{“}\mathbf{x}' = \text{Translate the following English sentence to French: } [\mathbf{x}][\mathbf{z}]\text{”}, \tag{3.6}$$

By explicitly stating the desired language, prefix prompts offer a straightforward directive to the model, resulting in accurate translations.

3.3.1.2 Cloze Prompts

Unlike prefix prompts, cloze prompts feature template tokens on either side of the answer slots, encompassing the answer in the interior of the template. For example, in the movie sentiment analysis task, a cloze prompt template can be expressed as

$$\text{“}\mathbf{x}' = [\mathbf{x}] \text{ This is a } [\mathbf{z}] \text{ movie.”}, \tag{3.7}$$

where contextual words surround the answer slot. When applied to the input, this template results in the filled prompt, “Cannot watch this movie. This is a [**z**] movie.” Note that punctuation marks count as template text in this context, so a template ending in the answer slot [**z**] followed by a period is considered a cloze prompt.

Cloze prompts are characterized by their flexibility and ability to create more natural language structures, making them particularly well suited for tasks such as multiple-choice question answering. By embedding the answer slot within a contextual phrase, cloze prompts encourage the model to generate outputs that conform to the surrounding linguistic patterns. By providing context on either side of the answer slot, cloze prompts enable the model to consider the linguistic features and situational cues present in the input, ultimately resulting in more accurate and meaningful outputs.

To summarize the distinction, the two broad categories are prompt shape are:

- **Prefix prompts:** In these prompts, the input and template text come before the answer slot.
  - Ex.: “Cannot watch this movie. This movie is [**z**]”
- **Cloze prompts:** In these prompts, the input and template text surrounds the answer slot.
  - Ex.: “Cannot watch this movie. It was a [**z**] movie.”

3.3.2 Manual Template Design

We turn now to the creation and optimiziation of templates. The most straightforward approach is *manual template design*, which uses human expertise, intuition, and understanding of the task to design a suitable template. This often involves considering the specific characteristics of the task, as well as employing heuristics to determine the optimal structure and wording of the prompt. This process may require iterative adjustments to refine the template for improved performance (Brown et al., 2020;

Petroni et al., 2019; Schick and Schütze, 2020a,b). A final decision should be made based on performance against a labeled dataset.

There is no one-size-fits-all approach to generating manual prompts, but the most critical guideline to follow is experimentation with many candidates. To demonstrate the importance of trial-and-error in this process, consider a prompt designed to return capital cities of countries. Here are four candidate prompt templates:

```
1. "the capital city of [x] is [z] ."
2. "[z] is the capital city of [x] ."
3. "what is the capital city of [x]? It is [z] ."
4. "[z] is located in [x], and is its capital city ."
```

Listing 3.1: Country capital prompt templates

Each of these templates looks like a plausible choice, but are they equally effective? As a check, we use the AllenNLP Masked Language Modeling demo[3] to test a input example. To use this demo, you enter a sentence including a mask token, and the model returns the top predicted tokens to fill the blank space. Taking Poland as our sample [**x**], we predict the top three tokens and report the results in Table 3.2.

Table 3.2: Prediction scores for the templates in Listing 3.1, using the masked language model demo from AllenNLP. For each prompt, probabilities of the top three predicted tokens to fill **[z]** given **[x]** = "Poland" are shown (in percent).

Input	**Warsaw**	**Kraków**	**Poznan**	**Poland**	**here**	**It**
Template 1	37.7	23.2	19.3	—	—	—
Template 2	55.7	14.8	15.0	—	—	—
Template 3	10.1	—	—	21.2	3.6	—
Template 4	29.3	—	12.3	—	—	33.0

Templates 1 and 2 return the correct answer, "Warsaw", as the top predicted token, with template 2 predicting "Warsaw" by a wider margin. Notably, these are the most simple and direct templates of the four, without multiple sentences or inefficient clause ordering. Template 3 returns "Poland" as the top answer, and template 4 predicts the pronoun "It". Both have Warsaw as their second guess, but it is clear that these templates did not activate the latent knowledge in the LLM as effectively.

In a manual prompt design project the engineer should test many different sample templates with many labeled examples similar to the above, allowing for statistical optimization. The optimal prompt should be determined relative to a metric, for example, the top-1 prompt selection approach:

$$A(t_r, i) = \frac{\sum_{\langle x,y\rangle \in \mathcal{R}} \delta(y = arg\ max_{y'} P_{LM}(y'|x, t_{r,i}))}{\mathcal{R}} \tag{3.8}$$

Here, $\mathcal{R}$ is the labeled test set of *subject-object* pairs with relation r, and $\delta(.)$ is Kronecker's delta function, which returns 1 where y is equal to the top prediction

[3] https://demo.allennlp.org/masked-lm

from the LM, and 0 where it is not. The final prompt is then with the highest accuracy on the set of *subject-object* pair training samples.

3.3.3 Automated Template Design: Discrete Search

Automated template design involves using some form of search or generation for the most effective prompt template in a predefined search space. While more complex to implement, automated prompt development will usually outperform manual prompt engineering, as it is generally more complete in its search of parameter space. Automated prompt engineering can be divided into two categories: (a) discrete search and (b) continuous search.

The primary distinction for these automated prompt template design methods is whether they use *discrete tokens/prompts* or *continuous tokens/prompts* to prompt the language model. This distinction relates to whether the prompt template itself is made up entirely of natural language tokens/phrases (discrete prompts) or continuous, tunable parameters (continuous prompts). Discrete prompts encompass the templates we have encountered in this section, where the tokens relating the input **x** to the masked output **z** are held fixed. Continuous prompts have non-fixed tokens, which can vary as a model training component. For example, the discrete template "the capital city of [x] is [z] ." could be replaced by the continuous prompt "[a_1] [a_2] [a_3] [a_4] [x] [a_5] [z]", where the tokens a_n are fine-tuned to optimize results during training. The following subsections will examine representative methods and their promise within these prompt template categories. A summary of the different approaches is shown in Table 3.3 at the end of the section.

3.3.3.1 Prompt Mining

Prompt mining, first proposed by Jiang et al. (2020), is a method where prompts are mined from a large corpus of text based on the logic that words in the vicinity of a subject x and the object y frequently describe the relation between them.

> Take again our example of capital cities; in a large corpus, instances where *Poland* and *Warsaw* closely co-occur are likely, on average, to imply some relation between a country and its capital. If you assemble many samples of *subject-object* pairs with the same relationship (i.e., more countries and their capital city) and extracted sentences from the corpus where they co-occur, these sentences can provide the basis for useful prompt templates for this information retrieval task.

Prompts generated using this corpus mining approach can be defined using one of two prompt generation methods. The first generation approach, known as middle-

word prompt extraction, works by taking sentences from the search corpus that contain the *subject-object* pair and extracting the text token(s) between them, which then serve as the prompt template itself. To illustrate, imagine again that we are mining for prompts to maximize the activation of the knowledge that the capital city of *Poland* is *Warsaw*. By searching within a corpus for sentences containing these two entities, we find the following:

```
Warsaw is the capital city of Poland, and
has a population of 1.86 million people.
```

By extracting only the words between the *subject-object* pair, we get the following:

```
"is the capital city of"
```

Which is then formulated as the following prompt template:

```
"[z] is the capital city of [x]"
```

This process is iterated for the complete set of in-scope *subject-object* pairs derived from the small training set, and middle-word prompts are searched for and extracted for each pair.

The second approach for mining prompt templates from Jiang et al. (2020) leverages a more linguistically sophisticated extraction process. Namely, syntactic analysis extracts templates that represent the shortest dependency paths between the *subject-object* pair within the matched sentence.

To illustrate, a middle-word prompt template extracted from the sentence *"The capital of Poland is Warsaw."* would be *"[x] is [z]"*, which is clearly too simplistic to accomplish our task. However, dependency analysis on this same sentence would result in the following dependency path; *"Poland* $\xleftarrow{pobj}$ *of* $\xleftarrow{prep}$ *capital* $\xleftarrow{nsubj}$ *is* $\xleftarrow{attr}$ *Warsaw"*, which gives the template *"capital of [x] is [z]"*, which looks like a more plausible template for the capital city retrieval task. It is also possible that these dependency-parsed templates will be better for activating the types of knowledge being targeted since they are derived from stable linguistic rules, which the LLM is expected to have learned during pre-training (Jawahar et al., 2019).

3.3.3.2 Prompt Paraphrasing

Prompt paraphrasing aims to take a preexisting prompt and maximize lexical diversity by generating template variants. With our capital city example, we can create several slightly different versions:

- **Original Prompt:** *"[z] is the capital city of [x] ."*
- **Paraphrased Prompt** 1**:** *"[z], the capital city of [x] ."*
- **Paraphrased Prompt** 2**:** *"[z] is the capital of [x] ."*
- **Paraphrased Prompt** 3**:** *"[x]'s capital city is [z] ."*

- **Paraphrased Prompt *n*:** *"[x]'s capital, [z] ."*

To automate this prompt paraphrasing process, Mallinson et al. (2017) developed a back-translation approach. This method follows a process wherein the original prompt to be paraphrased is translated into B candidate translations in a different language. Each of these is then translated into the same language as the original prompt to give B^2 candidate templates. These candidate prompt templates are then downselected by ranking their round-trip probabilities, which are calculated as,

$$P(t) = P_{forward}(\bar{t}|\hat{t}) \cdot P_{backward}(t|\bar{t}) \tag{3.9}$$

where $\hat{t}$ is the original prompt, $\bar{t}$ is the translation of the original prompt $\hat{t}$, and t is the final prompt candidate being ranked. Prompts are then retained by selecting the top T ranked candidates. These prompt candidates can then be subjected to additional downselection and ensembling to optimize their utility in solving the target NLP task. These prompt selection and ensembling techniques are discussed further in Sect. 3.5.

3.3.3.3 Gradient-directed Search

Another approach is to design prompt templates using a *gradient-directed search* method. This concept was initially proposed by Wallace et al. (2019), who were interested in adversarial attacks on generative models. These authors created an algorithm that iteratively updated "trigger tokens" appended to a prefix prompt just before the response slot to minimize the loss when an incorrect response is filled into the answer slot. For a concrete example, consider the following question answering (QA) prompt shape:

$$\text{"Question: [}\mathbf{x}\text{] Context: [}\mathbf{y}\text{] Answer: [T] [T] [T] [}\mathbf{z}_{adv}\text{]"}, \tag{3.10}$$

where [**x**] and [**y**] are the question and context, [$\mathbf{z}_{\mathbf{adv}}$] is an adversarial output that we are trying to trick the model into producing, and [T] are a series of nonstatic "trigger" tokens that can be iteratively updated to minimize the loss of the sequence according to some language model. These updates are done by a gradient-guided search based on the HotFlip approach (Ebrahimi et al., 2018). This procedure induced the model to generate an adversarial response, and critically the authors found that in many instances the optimized sequence of trigger tokens were robust to changes in the input text, producing the same inappropriate output for many different inputs. An example from their work, using a question/answer pair from the SQuAD dataset:

Question: Why did he walk?
For exercise, Tesla walked between 8 to 10 miles per day. He squished his toes one hundred times for each foot every night, saying that it stimulated his brain cells. **why how because** to kill american people

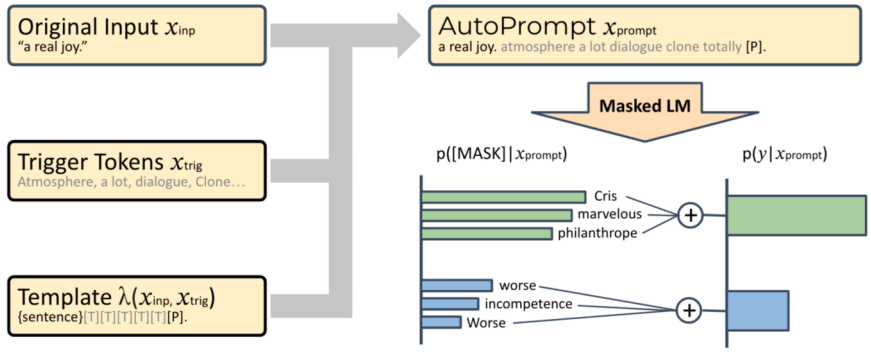

Fig. 3.11: An illustration of `AutoPrompt` applied to probe a masked language model's (MLM's) ability to perform sentiment analysis. Each input, x_{inp}, is placed into a natural language prompt, x_{prompt}, which contains a single `[MASK]` token to be predicted. The prompt is created using a template, λ, which combines the original input with a set of learned trigger tokens, x_{trig}. The trigger tokens are shared across all inputs and are learned using the described gradient-based search process. Probabilities for each class label, y, are obtained by marginalizing the MLM predictions, $p(\texttt{[MASK]}|x_{prompt})$, over sets of automatically detected label tokens

The three tokens **why how because** are the product of their gradient optimization, and cause GPT-2 to generate the adversarial underlined response for many different inputs.

The promise of this approach for optimizing templates for the purpose of prompt engineering were quickly recognized. Building from this work Shin et al. (2020) proposed `AutoPrompt` as an approach to construct prompt templates automatically. These authors took a series of initial templates, including trigger tokens, similar to Equation 3.10, and optimized the tokens by a gradient-guided search, iterating over a sizable set of labeled input/output examples. Their method is depicted in Fig. 3.11, with an example of the sentiment analysis task. As seen in this figure, the input to the language model is constructed from three key components:

- **Original input** (x_{inp}): This maps to input **x** from Fig. 3.8.
- **Trigger Tokens** (x_{trig}): These are the natural language tokens learned through gradient search. The number of tokens learned depends on how many tokens the gradient search method is initialized with and can be considered a hyperparameter in this context.
- **Answer Slot**: This is represented by `[P]` or `[MASK]` in Fig. 3.11, and maps to the $[z]$ slot in the example provided in Fig. 3.8

Each component is combined within the structural definition of a given prompt template to provide the optimized input to the language model (i.e., x_{prompt}). The label class is then determined by summing the probabilities of a number of automatically selected output tokens. In this example, *Cris, marvelous* and *philanthrope*

were derived for the positive class, and *worse, incompetence,* and *Worse* comprise the negative class. The cumulative probability of the positive labels exceeds that of the negative labels, denoting a positive sentiment classification.

Although the optimized tokens may not seem intuitive to a human, Shin et al. (2020) reported a complete 9% accuracy points gain over the Top-1 paraphrased prompts evaluated in Jiang et al. (2020) when tested on the same LAMA T-REx entity-relation subset benchmark relative to manual templates. They also show that using BERT and RoBERTa variants, `AutoPrompt` outperforms manual prompting by 10-20% on taks such as answer mapping, natural language inference, fact retrieval, and entity relation extraction. Critically, they show that optimized prompting can even out-compete fine-tuned variants, particularly in low-data situations, where you may have only have a handful of labeled samples.

3.3.3.4 Prompt Generation

So far, all of the discrete prompt searching methods we have reviewed have leveraged masked language models, where singular tokens are predicted. Taking inspiration

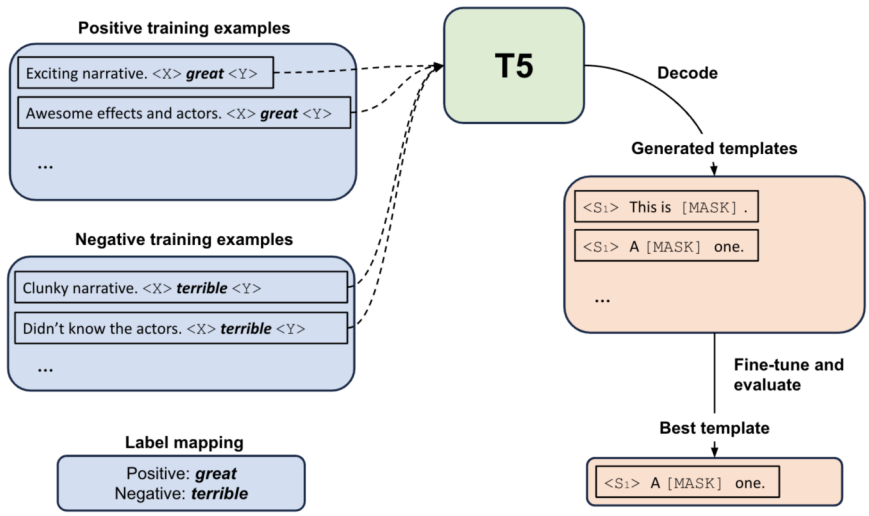

Fig. 3.12: Illustration of the prompt generation process, wherein input examples from D_{train} are partitioned by class, formulated as suitable inputs for T5, and used to decode to a single or small set of templates that maximizes the sum given in Equation 3.11.

from the "in-context" learning capabilities of LLMs demonstrated in Radford et al. (2019), Brown et al. (2020), and others, Gao et al. (2021) introduce the *Better Few-shot Fine-tuning of Language Models* (LM-BFF) approach. Instead of the MLMs used in the previous examples, these authors use T5, a large text-to-text or sequence-to-sequence model (Raffel et al., 2020). Their approach uses a few training examples in a "few-shot" configuration to prompt T5 to search for optimal prompts.

Fig. 3.12 illustrates how the T5 model is used in Gao et al. (2021) to generate prompt templates. Several samples are extracted from the small D_{train} dataset and passed to T5, which is prompted to construct a template $\mathcal{T}$ without the need to be explicit about the number of tokens the template should contain. The inputs to T5 are carefully designed to achieve the prompt generation outcome. Fig. 3.12 shows how D_{train} are grouped into class-specific samples, which are formulated as appropriate inputs for T5 and then used to generate a template or set of templates $\mathcal{T}$ that maximizes:

$$\sum_{(x_{in},y)\in D_{train}} \log P_{T5}(\mathcal{T}|\mathcal{T}(x_{in},y)), \tag{3.11}$$

where P_{T5} is the output probability distribution from the T5 language model.

Once a given set of templates is generated using these formulations as input to T5, the generated templates are decoded, and the *best template* is selected following fine-tuning of $\mathcal{L}$ using D_{train}, and evaluation on D_{dev}. Gao et al. (2021) demonstrated that their novel prompt template generation method, coupled with providing semantically similar demonstrations along with a given input, significantly improves performance over manually designed prompts. Additionally, leveraging manually or automatically generated prompts with T5 outperforms standard fine-tuning solutions, demonstrating the utility of prompt-based learning in NLP.

Building from work presented in Gao et al. (2021), Ben-David et al. (2022) introduced *Prompt learning algorithm for on-the-fly Any-Domain Adaptation* (PADA), which is a method that also leverages the text-to-text prompt generation capabilities of the T5 language model, but aims explicitly to generate human-readable prompts that represent multiple source domains (Ben-David et al., 2022). Thus, aiming to solve the common challenge of predicting out-of-distribution data, the PADA algorithm maps multiple specific domains into a shared semantic space, providing greater generalization potential. Ben-David et al. (2022) reported impressive performance relative to robust baseline solutions for both text classification and sequence tagging tasks using this approach.

3.3.4 Automated Template Design: Continuous Search

Considering that the primary goal of prompt construction is to develop a method that empowers an LLM to efficiently accomplish a task rather than solely generating prompts for human understanding, it is not essential to confine the prompt to human-interpretable natural language (Li et al., 2019). Consequently, alternative approaches

Table 3.3: Summary of discrete automated prompt design approaches

Method	Summary	Example	Pros	Cons
Prompt mining	Searching large corpora for sentences which contain given query/answer pairs, and using them to create a prompt template.	**Test pair**: Poland :: Warsaw **Mined sentence**: "Warsaw is the capital city of Poland, and has a population 1.86 million people" (found in Wikipedia) **Derived template**: "[z] is the capital city of [x] ."	• Programmatically simple. • Can be expanded with more sophisticated linguistic extraction processes.	• Domain of output templates constrained to sentences in corpora. • Optimization far from guaranteed.
Prompt paraphrasing	A seed prompt is iterated on with translation chains to produce many subtle variants, and the best performing one is selected.	**Seed prompt**: [z] is the capital city of [x] . **Back-translation variants**: • "[z], the capital city of [x] ." • "[x]'s capital city is [z] ." • "[x]'s capital, [z] ."	• Programmatically simple. • Variety of tested prompts helps to optimize.	• Domain of responses fairly narrow and limited. • Optimization far from guaranteed.
Gradient search	A series of variable trigger tokens (here, [T]), combined with input/prediction pairs, are combined into a template that is optimized during the training process to produce the best prediction results.	**Review/Sentiment pair**: "a real joy" :: positive **Initial prompt**: "a real joy [T][T][T] [T][T] positive" **Gradient-optimized prompt**: ' a real joy atmosphere alot dialogue Clone totally positive" **Optimized template**: [x] atmosphere alot dialogue Clone totally [z]	• Can produce highly optimized input tokens. • Does not rely on existing sentence corpora for its domain.	• Computationally expensive and programmatically complex. • Unintuitive template results. • Output templates constrained to human language embeddings.
Prompt generation	An encoder-decoder model (e.g., T5) predicts tokens in a seed template created with training query/response pairs. The resulting predictions are converted templates and tested for quality.	**Seed templates for review sentiment**: • A pleasure to watch. <X> great <Y> • No reason to watch. <X> terrible <Y> **T5-filled templates**: • A pleasure to watch. This is great . • No reason to watch. A terrible one. **Derived templates for testing**: • [x] This is [z]. • [x] A [z] one.	• Variety of tested prompts helps to optimize. • LLM-derived templates may by construction be fairly well optimized for LLM usage.	• Computationally expensive and programmatically complex. • Human input required for seed templates. • Optimization not guaranteed.

have emerged that investigate continuous prompts, also called soft prompts, enabling prompting directly within the model's embedding space.

Importantly, continuous prompts address two critical limitations of discrete prompts:

1. They reduce the necessity for template word embeddings to align with the embeddings of natural language words, such as those found in English.
2. They remove the constraint that pre-trained LM parameters parameterize the template. Instead, the templates have parameters that can be fine-tuned based on the training data obtained from the downstream task.

3.3.4.1 Prefix Tuning

Prefix tuning was initially presented in Li and Liang (2021). Inspired by the success of in-context learning with prompts (see Sect. 3.5.2), prefix tuning introduces task-specific "virtual tokens" that are added to the beginning of the input text (Fig. 3.13). These vectors do not represent actual tokens, but their dimensions are initialized such that the language model can attend to them in the same manner as hard tokens. They can then be treated as continuous vectors for training, whereas hard tokens have a fixed representation. This approach makes it possible for the language model to learn the nature of the task by tuning the prefix rather than relying solely on the explicit discrete features in the prompt's text.

Indeed, Li and Liang (2021) reported that their prefix-tuning trials outperformed fine-tuning in low-data settings and were competitive with full data fine-tuning. By applying the prefix-tuning approach to BART Lewis et al. (2019) for summarization and to GPT-2 Radford et al. (2019) for table-to-text, the method achieved strong results on both tasks relative to the established adaptor and full data fine-tuning benchmarks. Importantly, these results indicate that the prefix-tuning approach generalizes well across language model types and was specifically shown to do so for encoder-decoder and autoregressive models.

As with `AutoPrompt`, where training datasets are used to optimize a set of discrete prompts through a gradient-directed search in discrete space (Sect. 3.3.3.3), prefix-tuning leverages training data to learn a set of continuous vectors (i.e., the prefix) that maximizes:

$$\max_{\phi} \log p_{\phi}(y|x) = \sum_{i \in Y_{idx}} \log p_{\phi}(z_i|h_{<i}) \tag{3.12}$$

where p_{ϕ}, which typically represents the trainable parameters of an LLM, are replaced with P_{θ}, representing the prefix parameters θ, since the LLM's parameter are fixed. h_i is the concatenation of all activation layers, including the prefix at time step i. Since prefix-tuning leverages left-to-right or autoregressive language models,

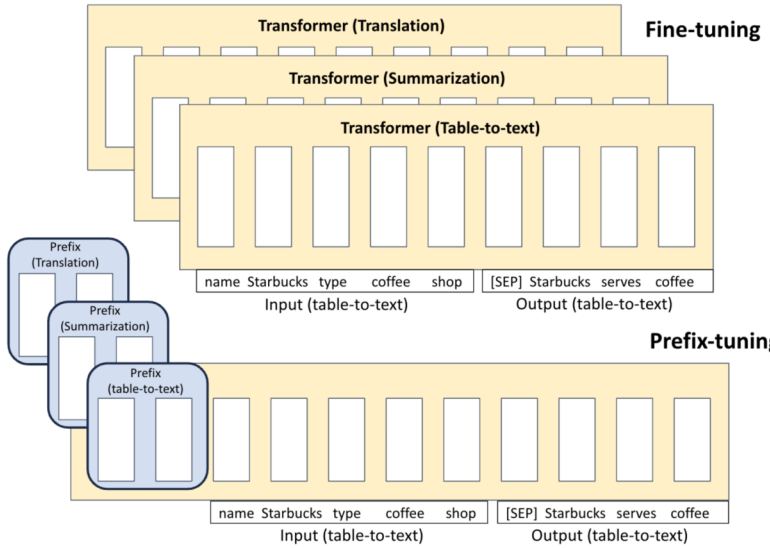

Fig. 3.13: Pre-training and fine-tuning (top) requires that the pre-trained model's parameters be copied and tuned for each downstream task, which, given the scale of some language models, represents a significant cost and technical challenge. Prefix-tuning (bottom) aims to overcome this challenge by freezing the parameters of the model and tuning only a task-specific prefix. Then by swapping in different tuned prefixes, this allows a single LLM to be used across multiple downstream tasks enabling both modularity in task solutions and a more space-efficient solution overall.

and as the name suggests, the learned vectors are prefixed to the leftmost layers of the language model, the influence of these prefixes percolates through the language model from left to right through all of the LM's fixed layers.

3.3.4.2 Hybrid and Discrete initialized Prompts

One key challenge identified within the work from Li and Liang (2021) was the prefix-tuning instability resulting from prefix parameter initialization and sensitivity to the learning rate. In that work, the solution was to parameterize the prefixes instead of using a smaller matrix generated using an extensive feed-forward neural network. However, another approach for initializing continuous tokens is to use informed discrete tokens. These tokens can be learned, as in previous automated discrete template search (e.g., Zhong et al. (2021)), or can be manually defined, and have shown promise in entity-relation knowledge probing tasks when used as the initialization point when learning continuous tokens (Qin and Eisner, 2021; Zhong et al., 2021).

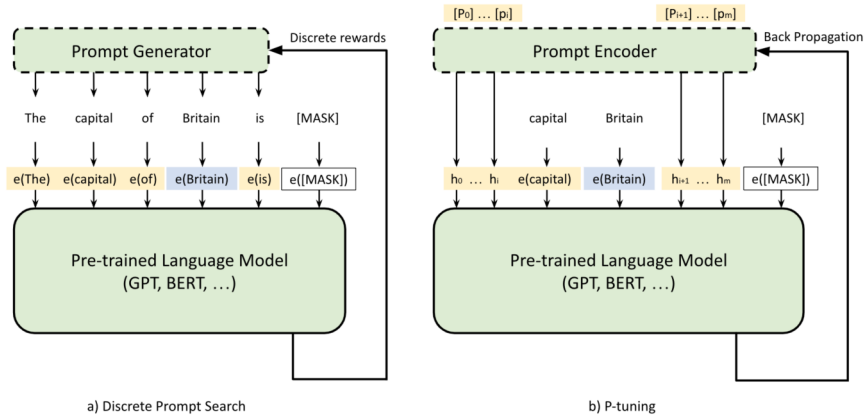

Fig. 3.14: An example of prompt search for "The capital of Britain is `[MASK]`". Given the context (darkest gray, "Britain") and target (white box, "`[MASK]`"), the lightest gray regions refer to the prompt tokens. In (a), the prompt generator only receives discrete rewards, while in (b), the pseudo-prompts and prompt encoder can be optimized differently. Sometimes, adding a few task-related anchor tokens, such as "capital" in (b), will further improve downstream task performance.

Another promising automated prompt template design innovation is the use of hard and soft tokens to compose the final template, as proposed in Liu et al. (2021). Fig. 3.14 illustrates how these hybrid prompts are generated (Fig. 3.14b), in contrast to how discrete prompt templates are generated (Fig. 3.14a). Importantly, Liu et al. (2021) demonstrated that their P-tuning method for prompt template generation outperformed all other tested discrete prompt templates in a knowledge probing task using BERT and achieved SOTA performance on the few-shot SuperGLUE benchmark using ALBERT (Lan et al., 2020).

3.3.5 Prompt-based Fine-tuning

The prompting approaches discussed thus far have all assumed that we vary a prompt or prefix to optimize against a static inference model. An alternative approach is to unfreeze the model parameters and fine-tune them using traditional backpropagation methodology on a dataset of input/output pairs arranged in a fixed template – this is called *prompt-based fine-tuning*. Consider again the example given in Fig. 3.7:

"`Cannot watch this movie. This is [z].`"

Instead of performing inference with this template, we can tune the model to accurately predict a value of `[z]` assigned by hand. Successful prompt-based fine-tuning

can require sizable samples of input/output pairs templatized in a consistent manner, although frequently, the quantity of labeled samples required for good performance is lower than for standard PTFT (see the tutorial in Sect. 3.6). This approach is viable both for cloze prompting, which fine-tunes on a series of filled prompts of common shape and different input/output pairs to predict the token in the answer slot, and for prefix prompting, which fine-tunes on a prefix answer to iteratively predict the next token in a trailing answer of arbitrary length. Inference is then performed with the tuned model using the standard prompt-based learning procedure.

Prompt-based fine-tuning is the fundamental technique used for instruction tuning, which is a critical step in the development of SOTA chatbots such as ChatGPT. We will discuss instruction tuning in detail in Sect. 4.2, so we defer further discussion until then and simply note here that prompt-based training generally outperforms zero/few-shot prompt-based learning (at potentially significant computational cost), and in many cases outperforms PTFT, particularly in the data-poor regime.

3.4 Answer engineering

Similar to how prompt engineering facilitates optimal choice of template, *answer engineering* encompasses designing and optimizing answer formats to guide the language model in generating the most accurate and contextually relevant responses to specific tasks or questions. This involves carefully considering various factors, formulating answer shapes, and exploring the answer search space in distinct ways to map to labels.

3.4.1 Answer Shape

The first consideration is the *answer shape*. This property determines the granularity of the model's outputs, ranging from individual tokens to entire sentences or phrases. Different tasks require varying levels of granularity in the responses; hence, selecting an appropriate answer shape is crucial for the success of prompt-based learning techniques. There are three basic types of answer shape:

- **Tokens**: These represent one or more individual tokens from the pre-trained LM's vocabulary or a subset thereof. Token-based answer shapes are often used in classification tasks such as sentiment classification, relation extraction, or named entity recognition (Cui et al., 2021; Petroni et al., 2019; Yin et al., 2019). For instance, in sentiment classification, the model's answer could be a single

token, such as "positive", "negative", or "neutral". For this answer shape, the answer space is usually restricted to a few choices of token, and thus falls into the constrained answer space category.

- **Chunks**: A chunk or a span includes a short multitoken sequence typically used in conjunction with cloze prompts. The distinction from the token answer shape is that they are not of fixed length and are generally in the unconstrained answer space category. This makes them useful for question-answering tasks, such as, for instance, the response to a prompt such as "`Dante was born in` **[z]** ."
- **Sentences**: Sentence-based shapes are the answers that comprise one or more sentences or even an entire document based on the task. Sentence-based answers are commonly employed with prefix prompts and are frequently used in language generation tasks that require more detailed responses such as summarization or translation (Radford et al., 2019). They are unconstrained.

3.4.2 Defining the Answer Space

The *answer space*, which we denote as $\mathcal{Z}$, is defined as the set of potential answers that a model can provide in response to an input. In many instances, this answer space maps to a series of output classes, denoted as $\mathcal{Y}$. There are two general classes of answer space: constrained and unconstrained.

- In **constrained answer spaces**, the output space is limited to a predefined set of answers. This is useful in tasks with a finite number of output labels, such as text classification, entity recognition, or multiple-choice question answering.
  - For this configuration, every element in $\mathcal{Z}$ maps to an element in the label space $\mathcal{Y}$ for the final output (e.g., the answer "terrible" maps to the negative label class in the sentiment analysis task of Fig. 3.4).
  - Associated with the token answer shape.
- In **unconstrained answer spaces**, $\mathcal{Z}$ encompasses all tokens (Petroni et al., 2019), fixed-length spans (Jiang et al., 2020), or token sequences (Radford et al., 2019).
  - These outputs generally do not map to a distinct label space, but instead are themselves the final outputs.
  - Associated with the chunk and sentence answer shapes.

As noted, constrained answer spaces require specific tokens to be selected to comprise $\mathcal{Z}$. In the next three sections, we discuss approaches for choosing these elements.