

# 大规模语言模型

陆俞因

2023. 05. 30



➤标准定义:

给定词典 $V$ ，语言模型能够计算出任意单词序列 $\{w_1, w_2, \dots, w_n\}$ 是一句话的概率  $p(w_1, w_2, \dots, w_n)$ ，其中， $p \geq 0$ 。

$$p = 0.98$$



今天上课的内容是大规模语言模型

$$p = 0.08$$



猫机器天气

➤目标：计算概率  $p(w_1, w_2, \dots, w_n)$

# N-gram Language Model

## N-gram语言模型



➤根据概率论中的链式法则(chain rule)将  $p$  展开

$$p(w_1, w_2, \dots, w_n) = p(w_1) \prod_{i=2}^n p(w_i | w_1, \dots, w_{i-1})$$

➤文本生成角度，定义语言模型：

给定一个短语（或句子），语言模型能够生成/预测接下来的一个词

# N-gram Language Model

## N-gram语言模型



➤为简化  $p(w_i | w_2, \dots, w_{i-1})$  的计算, 引入一阶马尔可夫假设(first-order Markov assumption)

每个词只依赖前一个词, 即  $p(w_i | w_2, \dots, w_{i-1}) \approx p(w_i | w_{i-1})$

$$\begin{aligned} p(w_1, w_2, \dots, w_n) \\ &= p(w_1) \prod_{i=2}^n p(w_i | w_1, \dots, w_{i-1}) \\ &\approx p(w_1) \prod_{i=2}^n p(w_i | w_{i-1}) \end{aligned}$$



# N-gram Language Model

## N-gram语言模型

➤也可以引入二阶马尔可夫假设(second-order Markov assumption)

每个词只依赖前两个词, 即  $p(w_i | w_2, \dots, w_{i-1}) \approx p(w_i | w_{i-2}, w_{i-1})$

$$\begin{aligned} p(w_1, w_2, \dots, w_n) \\ &= p(w_1) \prod_{i=2}^n p(w_i | w_1, \dots, w_{i-1}) \\ &\approx p(w_1) p(w_2 | w_1) \prod_{i=3}^n p(w_i | w_{i-2}, w_{i-1}) \end{aligned}$$

统计方法:

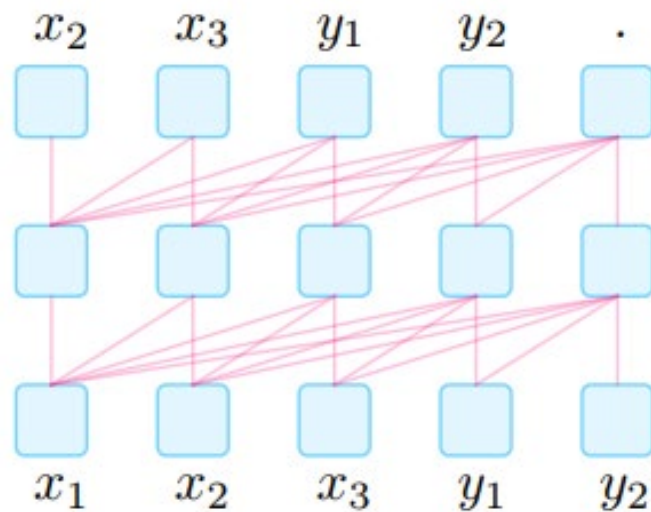
$$p(w_i | w_{i-2}, w_{i-1}) = \frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$$

# Neural Network Language Model

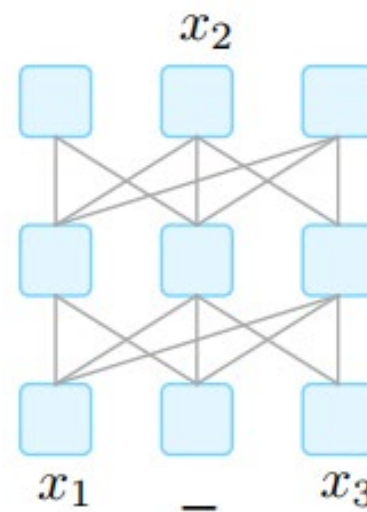
## 神经网络语言模型



➤通过神经网络预测下一个单词



(a) Left-to-right LM.



(b) Masked LM.

代表模型:

GPT-3

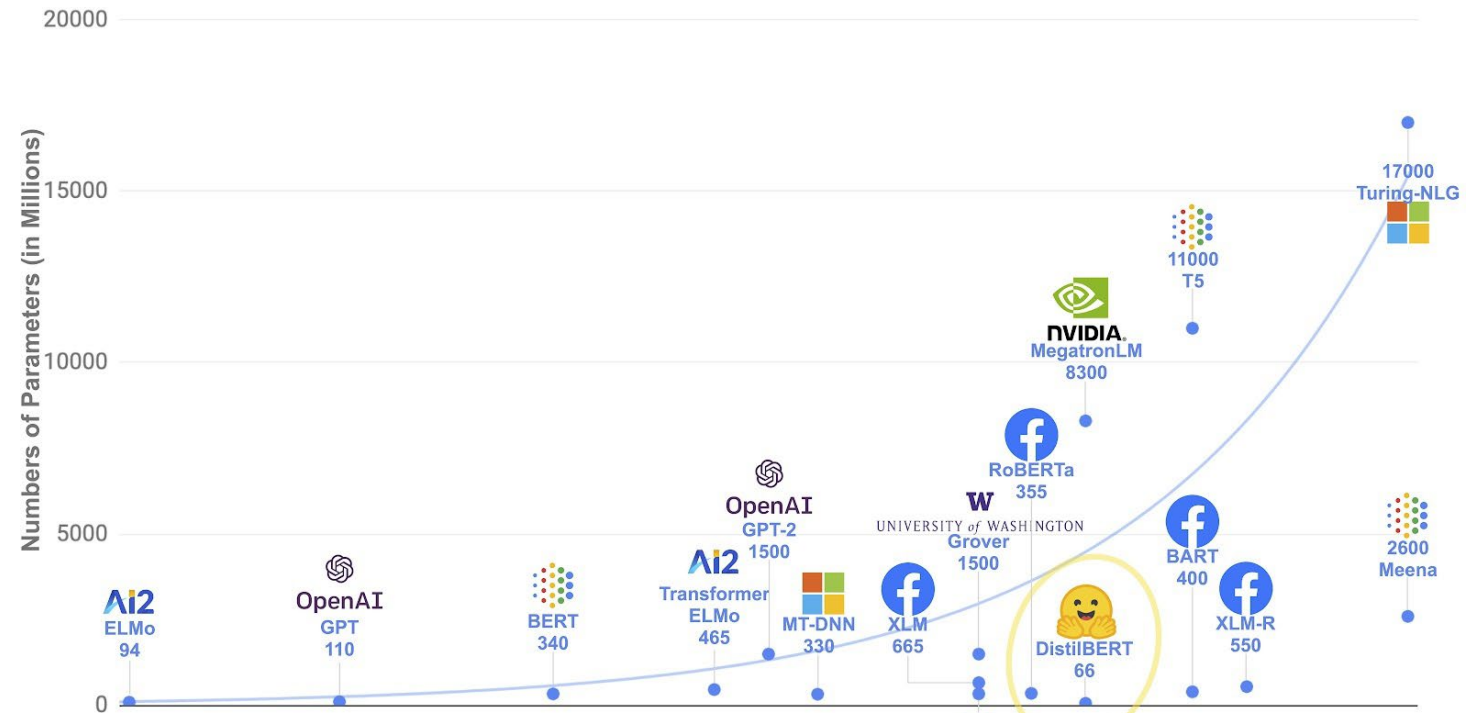
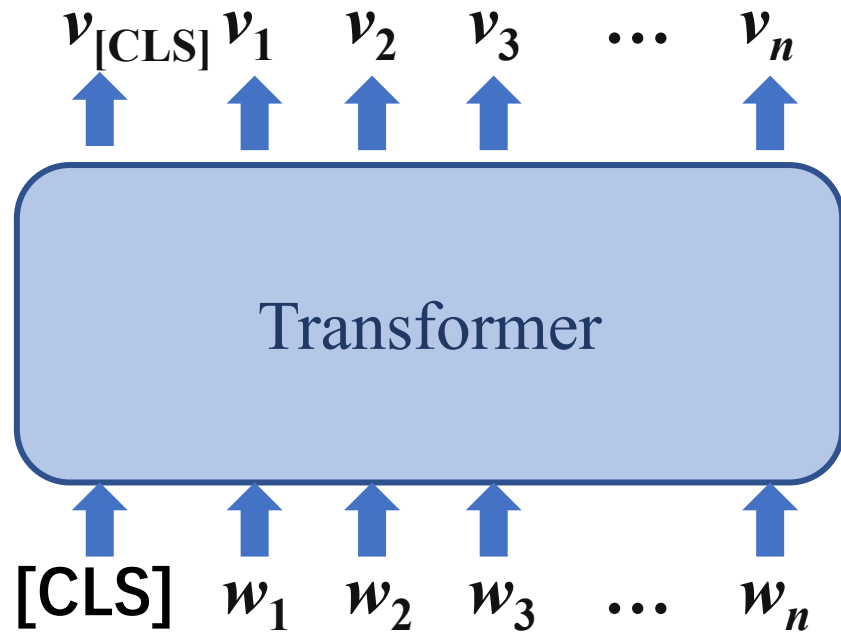
BERT、ERNIE



# Large Language Model (LLM)

## 大规模语言模型

➤通常采用Transformer作为网络架构

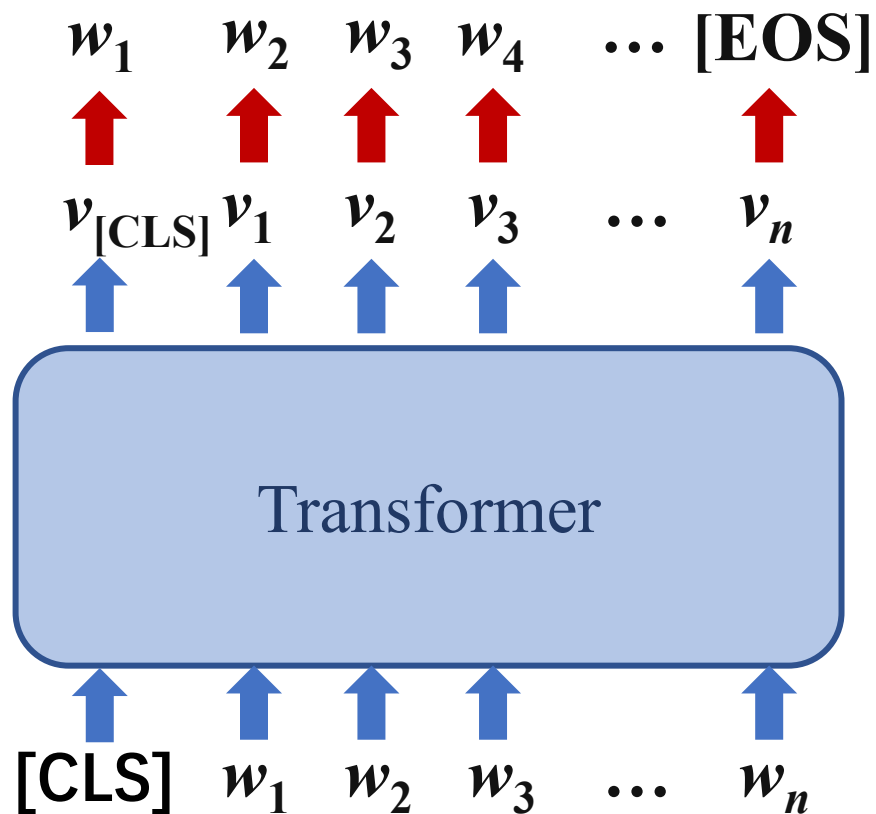


参数量

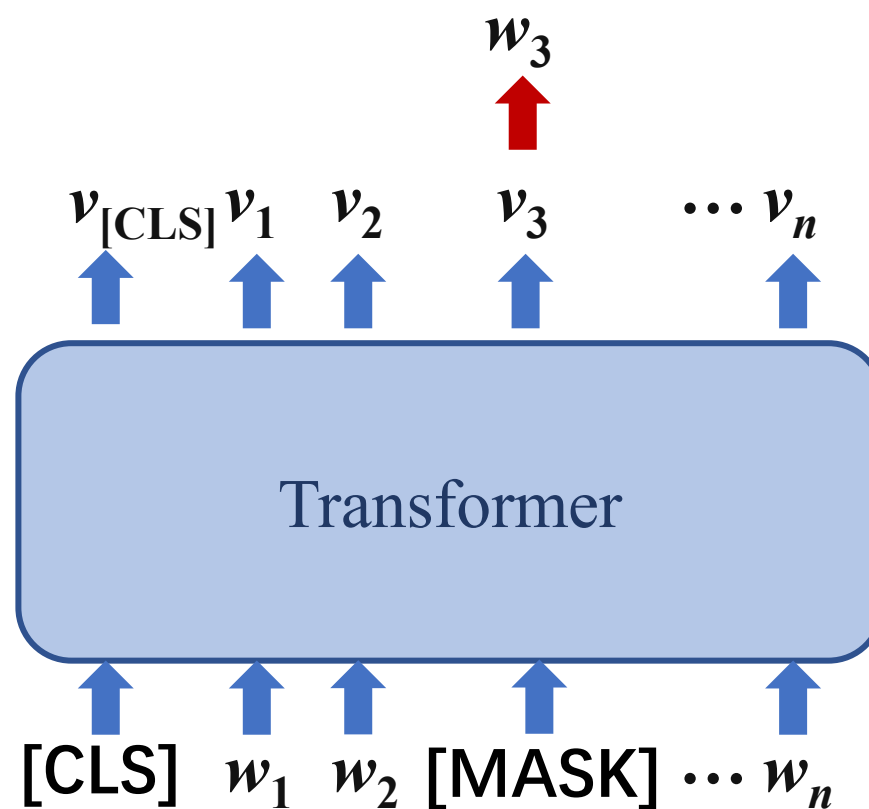
# Pretrain 预训练



➤ 在无标记语料上通过自监督任务进行预训练



Left-to-Right LM



Masked LM



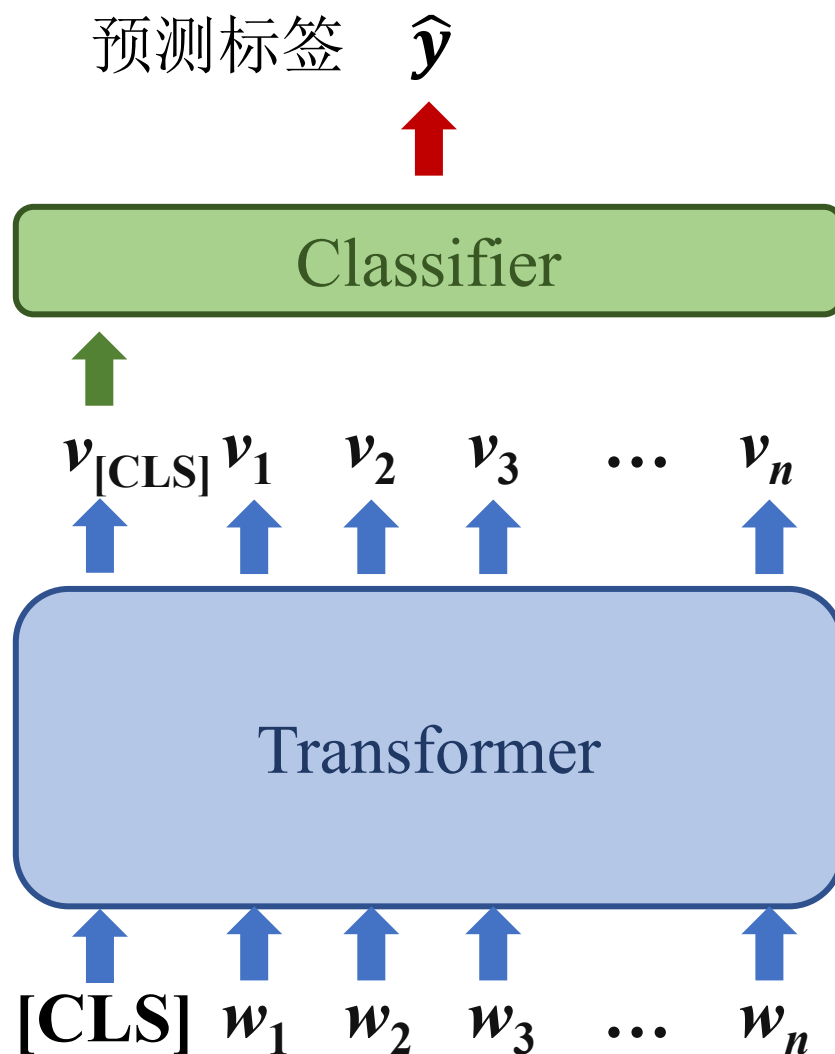
# Fine-tune 微调



➤在有标记语料上进行微调

以文本分类为例：

通常采用  $v_{[CLS]}$  作为文档嵌入



*positive*

I love this movie.

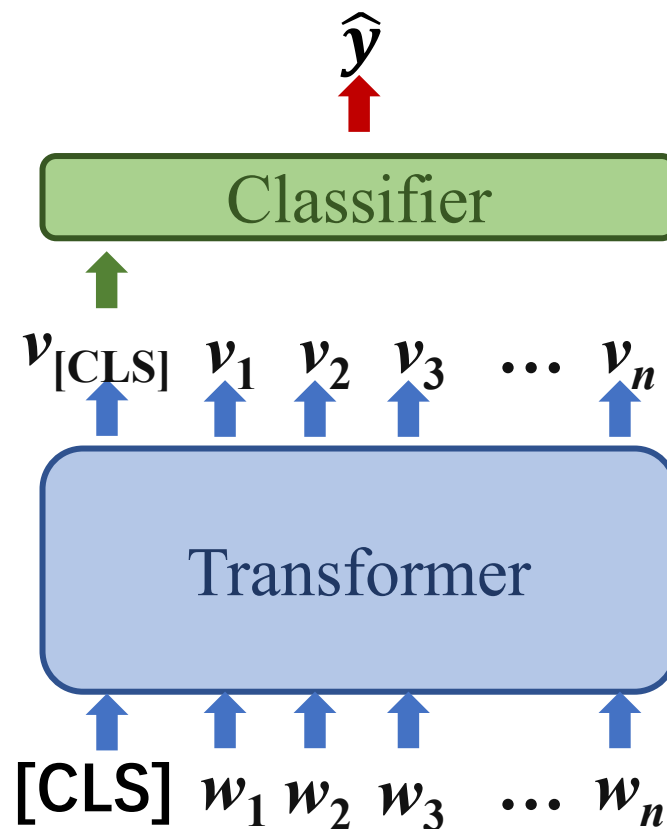
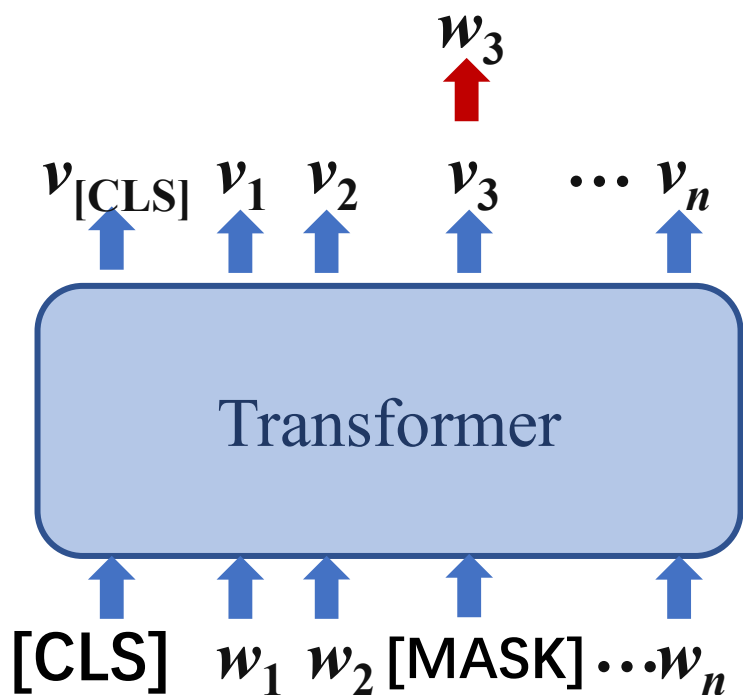
# Prompt Learning

## 提示学习



➤ 预训练-微调范式的存在问题：

预训练阶段的自监督任务与微调阶段的有监督任务存在较大的差距



# Prompt Learning

## 提示学习

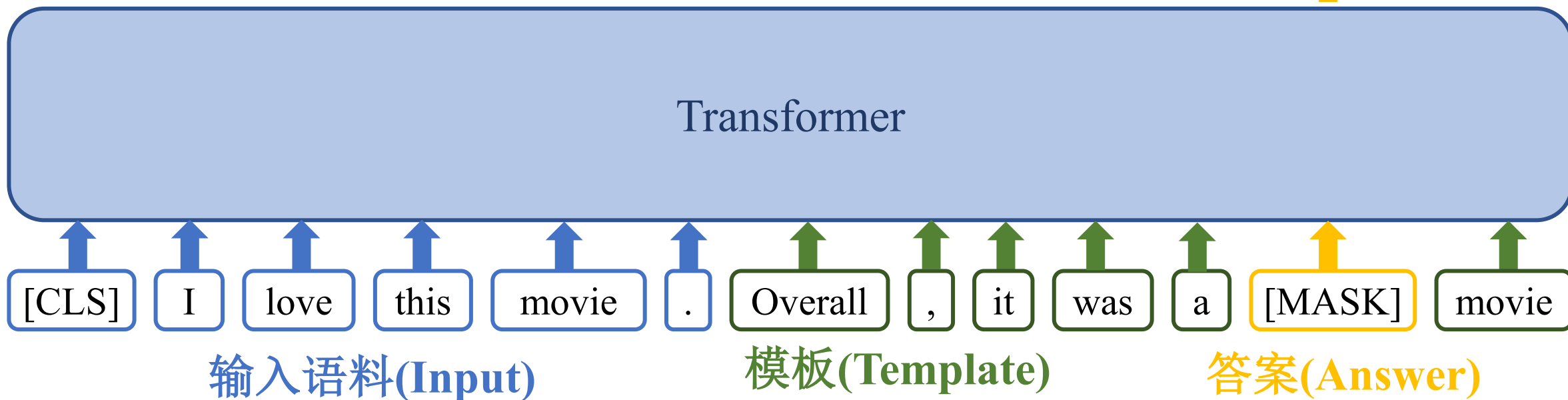


➤ 目标：拉近预训练与目标任务的距离

对应标签：

*positive*

完型填空： good / fantastic / interesting

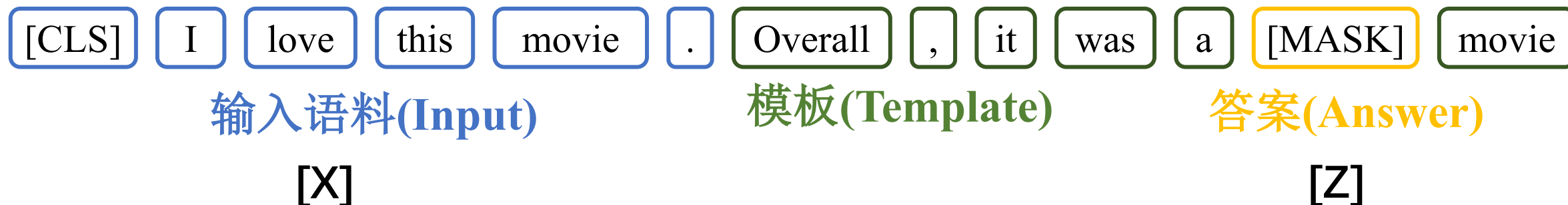


# Prompt Learning

## 提示学习



➤形式化:



提示(Prompt): [X] [Template] [Z]

# Prompt Learning

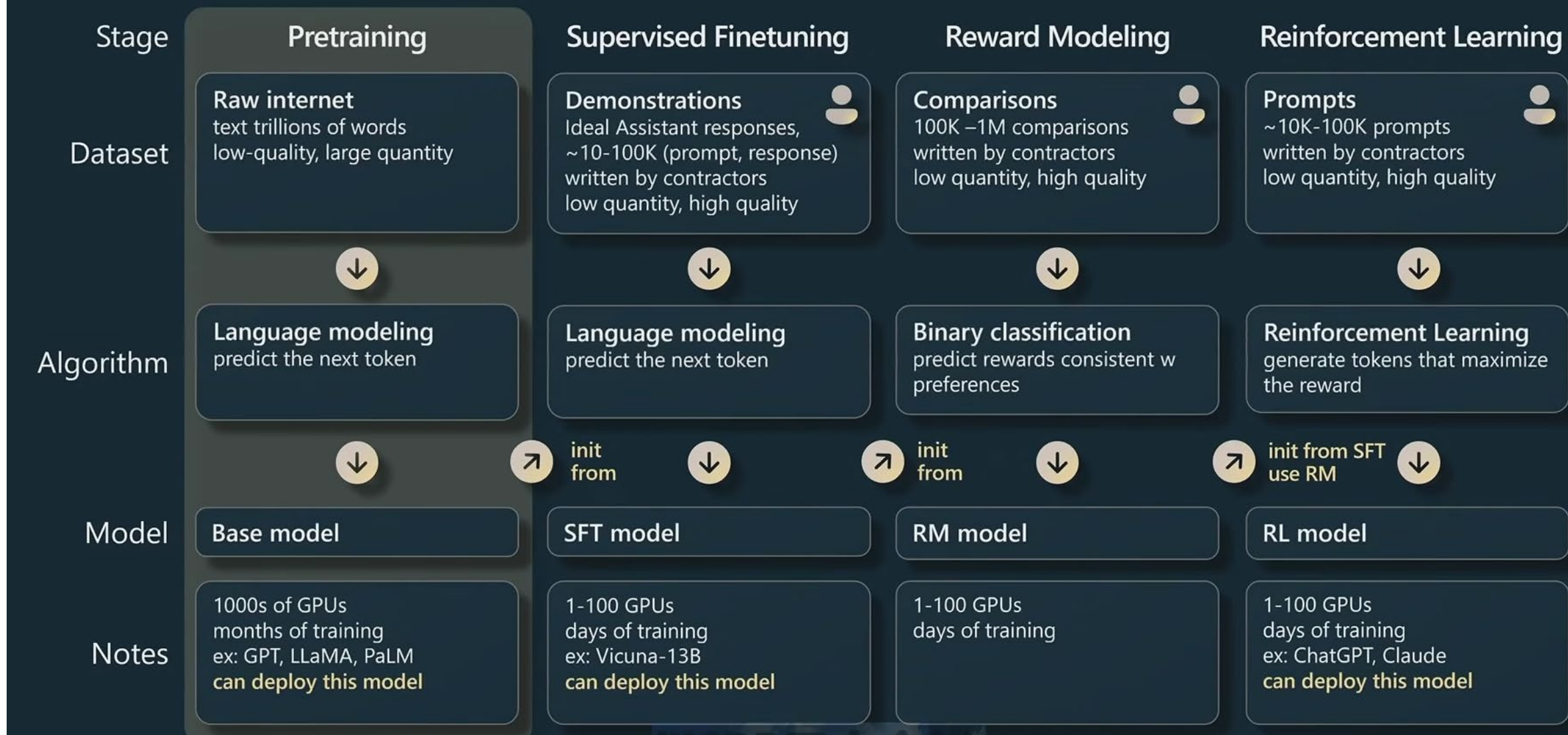
## 提示学习



➤形式化:

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

# GPT Assistant training pipeline







# Reinforcement Learning from Human Feedback (RLHF)

## 根据人类反馈进行强化学习

### ➤ 奖励建模 (Reward Modeling)

## RM Dataset

#### 3

Write a Python function that checks if a given string is a palindrome.

Challenge: Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python
def is_palindrome(word):
    # Convert the word to lowercase and remove spaces
    word = word.lower().replace(" ", "")

    # Compare the word with its reverse
    if word == word[::-1]:
        return True
    else:
        return False
```

Here's how you can use the function:

```
python
word1 = "radar"
print(is_palindrome(word1)) # Output: True

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```

The function first converts the input word to lowercase and removes any spaces. Then, it compares the word with its reverse using slicing ('[::-1]'). If the word is equal to its reverse, it is a palindrome, and the function returns "True". Otherwise, it returns "False".

#### 2

Write a Python function that checks if a given string is a palindrome.

Challenge: Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python
def is_palindrome(string):
    # Convert the string to lowercase and remove spaces
    string = string.lower().replace(" ", "")

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

This function takes a string as input, converts it to lowercase, and removes any spaces. It then checks if the reversed string is equal to the original string. If they are equal, the function returns "True", indicating that the string is a palindrome. Otherwise, it returns "False".

Here's an example of how you can use the function:

```
python
word = "radar"
if is_palindrome(word):
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

Output:

```
python
radar is a palindrome.
```

#### 1

Write a Python function that checks if a given string is a palindrome.

Challenge: Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join(char.lower() for char in s if char.isalnum())

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```

In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing ('[::-1]'). If they are equal, the string is a palindrome, and the function returns "True"; otherwise, it returns "False".

Here's an example usage of the function:

```
python
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1)) # True

string2 = "race car"
print(is_palindrome(string2)) # True

string3 = "hello world"
print(is_palindrome(string3)) # False
```

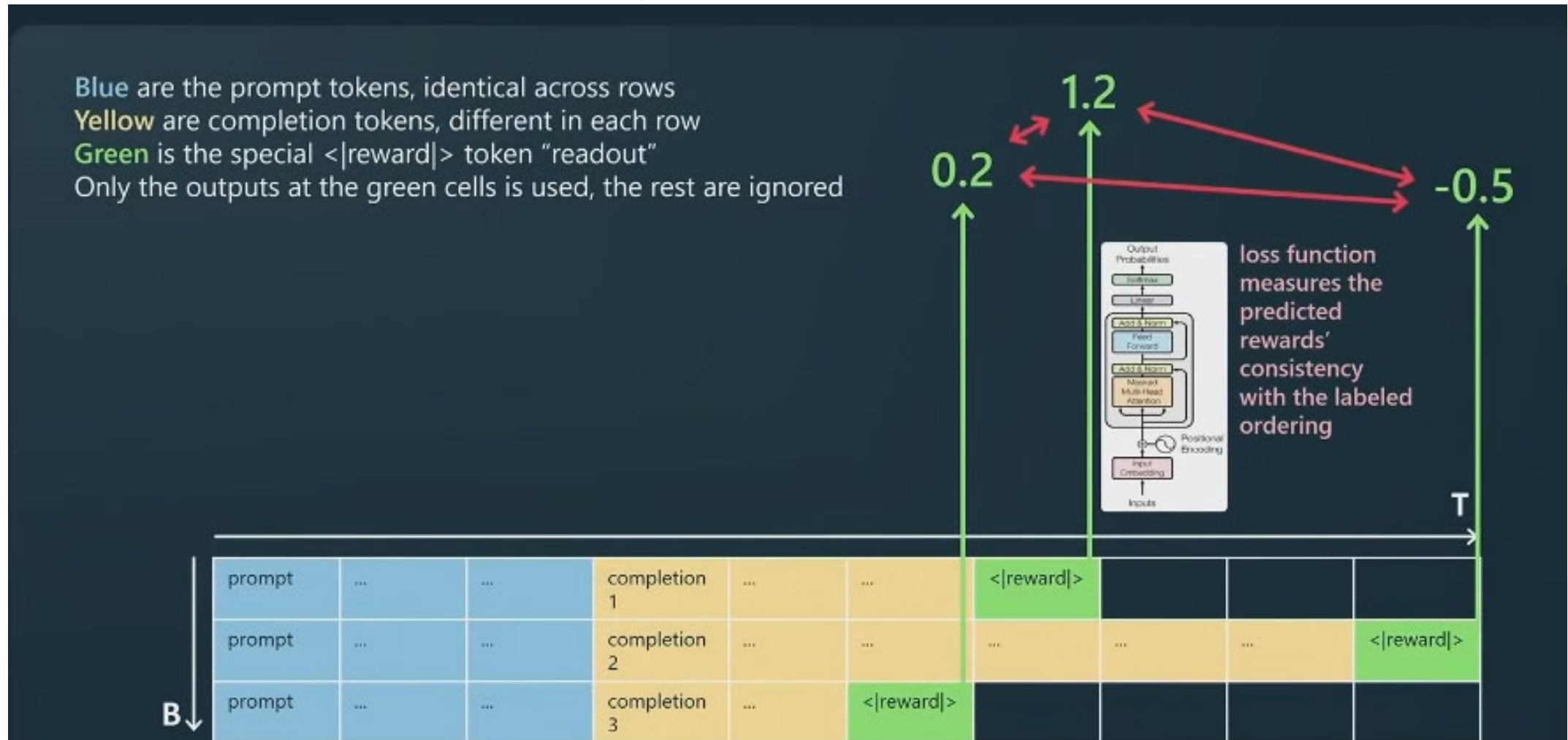
In the above example, 'is\_palindrome' is called with three different strings. The function correctly identifies the palindromes ("A man, a plan, a canal: Panama" and "race car") and returns "True". For the non-palindrome ("hello world"), it returns "False".



# Reinforcement Learning from Human Feedback (RLHF)

## 根据人类反馈进行强化学习

### ➤ 奖励建模 (Reward Modeling)







# Reinforcement Learning from Human Feedback (RLHF)

## 根据人类反馈进行强化学习

### ➤ 强化学习(Reinforcement Learning)

Blue are the prompt tokens, identical across rows

Yellow are completion tokens by the model (initialized with SFT model)

Green is the special  $\langle \text{reward} \rangle$  token "readout", RM now predicts these  
Only the yellow cells are trained on, the rest are ignored.

The sampled tokens become labels, but the training objective is weighted by the "advantage" (normalized rewards)

In this example:

- Row #1 tokens were great. These get their probabilities boosted.
- Row #2 tokens were bad. These get their probabilities decreased.
- Row #3 tokens were ~ok. These get their probabilities slightly boosted.

