

Table 1: Hyper-parameters settings.

Method	Symbol	Value	Description
C3	M	150	# of prompt tokens
	r	1	# of demonstrations
	η	4.0	threshold
	-	12	batch size
	-	0.3	learning rate for PROMPT-TUNING
	-	4	beam size
	-	1×10^{-4}	learning rate for FINE-TUNING
	-	50	evaluation is done every # epochs for PROMPT-TUNING of T5-BASE or T5-SMALL
	-	25	evaluation is done every # epochs for PROMPT-TUNING of T5-LARGE
	-	10	evaluation is done every # epochs for FINE-TUNING of T5-BASE or T5-SMALL
	-	5	evaluation is done every # epochs for FINE-TUNING of T5-LARGE
	-	15000	# of maximum epochs for PROMPT-TUNING
	-	300	# of maximum epochs for FINE-TUNING
	-	512	maximum # of input tokens
	-	10	stop training when the specified metric worsens for # evaluations
Other Baselines	-	32	batch size
	-	2×10^{-4}	learning rate
	-	15	memory size of replayed examples in EMR, EMAR, TR and EWC
	-	5	beam size
	-	55	# of maximum epochs
	-	1	# of maximum epochs for the second iterations in EMAR
	-	2	N-way of MAML
	-	1	K-shot of MAML
	-	25	# of meta-task in MAML
	-	300	maximum # of input tokens
	-	1	# of demonstrations
	-	1.0	regulation weight in EWC

A Hyperparameter Details

The detailed settings of the hyper-parameters used in our experiments are presented in Table 1.

B Aligning GPT’s tokenizer and T5’s tokenizer

When calling OpenAI’s API to utilize text-davinci-003 (hereinafter referred to as GPT), we can get a predicted SQL query $\mathbf{p}_{0:L}$ made up of L GPT tokens and a sequence of output distributions $\mathbf{s}_{1:L}$ where each $\mathbf{s}_i \in \mathbf{s}$ is the probabilities of the most probable tokens at each decoding step, corresponding to \mathbf{p}_i . Our objective is to convert the GPT output distributions to the the format of T5 tokenizer and get the target distributions \mathbf{t} where each $\mathbf{t}_i \in \mathbf{t}$ contains the probabilities of T5 tokens, making it trainable for the student models. To achieve this goal, we apply the algorithm shown in Algorithm 1.

First, we evaluate whether GPT produces the correct SQL by comparing \mathbf{p} with the gold SQL \mathbf{g} . If not, we expand \mathbf{g} as a sequence of one-hot distributions for student’s training. Otherwise, we tokenize the GPT predicted SQL query using the T5 tokenizer and get a sequence of one-hot distributions $\mathbf{q}_{1:K}$ made up of K T5 tokens with probability 1. And our target is to align \mathbf{s} and \mathbf{q} . Here we apply an one-to-one strategy. If there exists a one-to-one mapping of a T5 token \mathbf{q}_i and a GPT token \mathbf{p}_j , we use the GPT distribution \mathbf{s}_j as the T5 distribution. In other cases where a mismatch exists, we simply take the one-hot distribution \mathbf{q}_i as the label.

Algorithm 1 Aligning GPT’s tokenizer and T5’s tokenizer

Require: GPT predicted SQL query $\mathbf{p}_{1:L}$, GPT output distributions $\mathbf{s}_{1:L}$, gold SQL query \mathbf{g} , T5 Tokenizer \mathbf{T} .

```
1: initialize  $\mathbf{t} = []$ 
2: function FINDONETOONEMAPPING( $\mathbf{q}_i, \mathbf{p}, \mathbf{s}$ )
3:   if Exists an one-to-one mapping of  $\mathbf{q}_i$  and  $\mathbf{p}_j$  then
4:     return  $s_j$ 
5:   else
6:     return None
7:   end if
8: end function
9: if  $\mathbf{p}$  is equivalent to  $\mathbf{g}$  then
10:   $\mathbf{q}_{1:K} = \mathbf{T}.\text{TOKENIZE}(\mathbf{p})$ 
11:  for  $i := 1$  to  $K$  do
12:    if FINDONETOONEMAPPING( $\mathbf{q}_i, \mathbf{p}, \mathbf{s}$ ) is not None then
13:       $\mathbf{t}.\text{APPEND}(\text{FINDONETOONEMAPPING}(\mathbf{q}_i, \mathbf{p}, \mathbf{s}))$ 
14:    else
15:       $\mathbf{t}.\text{APPEND}(\mathbf{q}_i)$ 
16:    end if
17:  end for
18: else
19:   $\mathbf{t} = \mathbf{g}$ 
20: end if
21: return  $\mathbf{t}$ 
```
