aCat: Automatically Choosing Anchor Tokens in Prompt for Natural Language Understanding

Anonymous ACL submission

Abstract

P-tuning has demonstrated that anchor tokens are beneficial for improving the performance of downstream tasks. However, selecting anchor tokens manually may result in subjective or suboptimal results. In this paper, we present aCat to choose anchor tokens automatically. Following the framework of the soft-hard prompt paradigm, aCat achieves the automatic construction of prompt templates. Experiments conducted on natural language understanding benchmarks demonstrate the effectiveness of our proposed method. On the seven datasets of SuperGlue, the proposed method has higher accuracy than the P-tuning, and the average accuracy is higher than P-tuning V2.

1 Introduction

Recent research shows that the prompt-based paradigm can reduce the gap between downstream tasks and pre-training tasks (Liu et al., 2023a). The proper prompt template has a significant effect on the downstream task (Liu et al., 2023a). Designing more appropriate prompts for further utilizing the ability of the pre-trained language models is challenging (Zhou et al., 2022; Schick and Schütze, 2021a,b).

Hard prompts (a.k.a discrete prompts) (Shin et al., 2020; Liu et al., 2023a) and soft prompts (a.k.a continuous prompts) (Deng et al., 2022; Liu et al., 2022b) are two types of prompts. Hard prompts consist of human-interpretable natural language, while soft prompts are continuous prompts that perform prompting directly in the embedding space of the model.

Rather than relying solely on hard prompt or soft prompt, the soft-hard prompt paradigm incorporates trainable token embeddings into the hard prompt. Compared to soft or hard prompts, the soft-hard prompt reduces the number of parameters (Liu et al., 2023a; Lang et al., 2022) and provides a certain degree of interpretability (Liu et al., 2023b). P-tuning(Liu et al., 2023b) is one of the typical soft-hard prompt paradigm. The hard part, called anchor tokens, is a few words designed manually. The soft part, called pseudo tokens, is used to generate pseudo tokens embedding by a prompt encoder. P-tuning found that selecting the task-related anchor tokens could further improve performance on the downstream task (Liu et al., 2023b). However, the anchor token in P-tuning still needs to be selected manually, which may result in subjective or suboptimal results.

To address the problem of manually selecting anchor tokens, we propose aCat (Automatically Choosing Anchor Tokens) for automatically selecting anchor tokens. In aCat, we introduce a corrector to determine which prompt tokens can serve as anchor tokens. To train the corrector, we formulate the problem of anchor token automatic selection under a reinforcement learning framework: the decision is made based on the features of training examples and prompt tokens, while the policy is learned towards maximizing the performance of the downstream task.

To verify the effectiveness of our approach in natural language understanding, we conducted extensive experiments on 7 NLP datasets (BoolQ, CB, WIC, RTE, MultiRC, WSC, and COPA), including sentiment analysis, paraphrase similarity matching, and natural language inference. Taking P-tuning as baseline, our experimental results show that aCat enhances the performance on all datasets.

In addition, we also compared the performance of aCat with P-tuning V2, the next generation version of P-tuning. P-tuning V2 applies continuous prompts for every layer of the pre-trained model (Liu et al., 2022a). Clearly, P-tuning V2 requires introducing more trainable parameters to obtain prompts suitable for downstream tasks. In contrast, aCat does not increase the number of trainable parameters with the depth of the pre-trained model

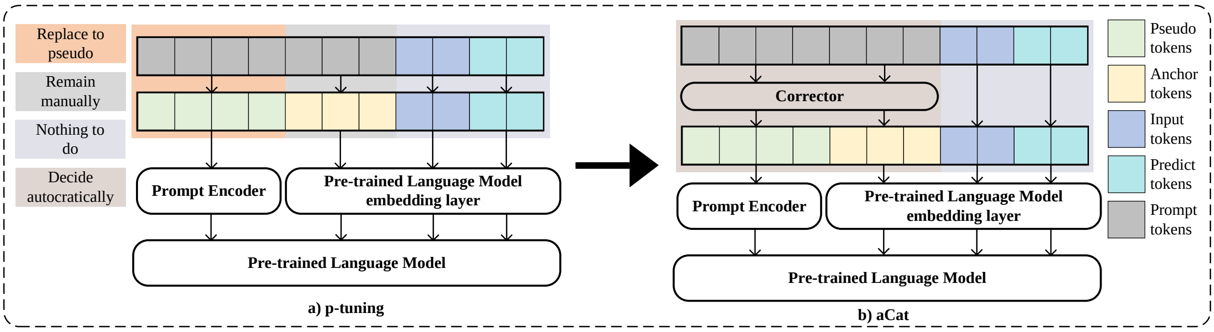

Figure 1: The difference between P-tuning and aCat

layers. The experimental results on six datasets demonstrate that the performance of aCat is comparable to P-tuning V2, and the average accuracy surpasses P-tuning V2 when using bert-base as a pre-training language model.

Our contributions are twofold. First, we systematically study how to select anchor tokens from prompt tokens automatically. Second, we propose a two-stage method for fine-tuning the prompt-based pre-trained language model, prompt encoder, and corrector agent.

2 Proposed method

Given the k-th training sample (x_k, y_k) in dataset D, prompt-based paradigm firstly constructs the prompt template $T = \{[\mathrm{W}_{0:i}], \mathrm{x}_k, [\mathrm{W}_{i+1:m}], \hat{y}_k\}$, where $[\mathrm{W}_{0:i}]$ and $[\mathrm{W}_{i+1:m}]$ represent a series of predefined prompt tokens, $\hat{y}_k$ denotes the tokens used for outputting the predicted results, and m represents the total number of prompt tokens in template T. In the soft-hard prompt paradigm, some prompt tokens will be chosen as anchor tokens, and others will be converted into pseudo tokens. Then, T can be converted to $T' = \{[\mathrm{P}_{0:j}][\mathrm{A}_{j+1:i}], \mathrm{x}_k, [\mathrm{P}_{i+1:r}][\mathrm{A}_{r+1:m}], \hat{y}_k\}$, where $[\mathrm{P}_{0:j}]$ and $[\mathrm{P}_{i+1:r}]$ represent pseudo tokens, $[\mathrm{A}_{j+1:i}]$ and $[\mathrm{A}_{r+1:m}]$ represent anchor tokens. Commonly, a pre-trained embedding layer e is employed to map x_k, $\hat{y}_k$ and anchor tokens to embeddings, and a prompt encoder is used to encode pseudo tokens to trainable continuous embeddings. Then, the converted T' will be fed into the pre-trained model to perform downstream tasks.

2.1 Overall architecture of aCat

aCat consists of three parts: (1) the corrector, to effectively determine which prompt tokens should be kept as anchor tokens; (2) the prompt encoder, to encode some prompt tokens to the pseudo tokens; and (3) the pre-trained language model, to perform the downstream tasks together with the soft-hard prompt. Figure 1 shows the architecture of aCat and denotes the difference between P-tuning and aCat. Note that P-tuning chooses anchor tokens manually based on the relationship between predefined prompt tokens W_i and x_k. But for aCat, anchor tokens are chosen by a corrector.

2.2 Constructing prompt template using corrector

A corrector of aCat consists of a frozen pre-trained language model M_f and an agent. Given the original prompt template T. M_f is used to encode prompt tokens in T as

$$h(T) = \{h([W_{0:i}]), h(\mathrm{x}_k), h([W_{i+1:m}]), h(\hat{y}_k)\} \tag{1}$$

where $h(\cdot)$ represents the output of M_f, and $h(\hat{y}_k)$ is obtained by using one or multiple [mask] tokens. Agent is used to determine which prompt tokens in T should be retained as anchor tokens.

Then, we obtain the final template T', using T', we use the loss $\mathcal{L}$ to finetune the pre-trained language model to perform the downstream task:

$$\mathcal{L} = -\sum_{\mathrm{x}_k, y_k \in \mathcal{D}} logP(\hat{y}_k = y_k | T^{'}; M, \theta) \tag{2}$$

where M represents the trainable pre-trained language model, θ represents the parameters of the prompt encoder.

2.3 RL based corrector training

We exploit the policy gradient method (Sutton et al., 1999) to learn the corrector. Now, we describe the state, the action, the reward, and the optimization in our reinforcement learning approach.

State State refers to the agent's state s_i on the i-th position. The agent decides the replacement result of the i-th token in the prompt tokens of T based on a series of states $s_1, s_2, ...s_i$.

For each prompt token $h([W_i])$ at position i in T, we concatenate $h([W_i])$ with $h(\hat{y}_k)$ to get the agent's state s_i for the i-th position as

$$s_i = F([h([W_i]); h(\hat{y}_k)]) \tag{3}$$

where $s_i \in \mathbb{R}^{2*d}$, with d being the dimension of the hidden vector and $F(\cdot)$ representing the concatenation function.

Action All prompt tokens of T are associated with one agent. An agent chooses between two possible actions, selecting the prompt tokens as anchor tokens or not. We use a policy function $\pi_{\theta_c}(s_i, \gamma_i)$ to decide actions.

$$\pi_{\theta_c}(s_i, \gamma_i) = \gamma_i \sigma(s_i, \theta_c) + (1 - \gamma_i)(1 - \sigma(s_i, \theta_c)) \tag{4}$$

where $\sigma(\cdot)$ is the RELU activation function, and $\theta_c \in \mathbb{R}^{2*d}$ is the trainable parameters of agents.

In practice, a multi-layer perceptron (MLP) and RELU activation function is adopted as the agent. We use the agent's state s_i to decide the action γ_i to decide whether the prompt token will be retained as an anchor token at each position i as

$$\gamma_i = \arg\max_{\gamma_i \in \gamma}(\pi_{\theta_c}(\gamma_i \mid s_i)) \tag{5}$$

Here, $\pi_{\theta_c}(\cdot)$ represents policy function, and $\gamma \in \{0, 1\}$ refers to the set of actions. If γ_i is 0, we replace the prompt token at position i with a pseudo token. If γ_i is 1, keep it as an anchor token.

Reward The reward function is associated with the performance of downstream tasks. It indicates the quality of the decisions made by the current agents. We use the negative of the loss $\mathcal{L}$ as the reward for the k-th instance.

$$reward_k = -\mathcal{L} \tag{6}$$

Optimization Following (Sutton et al., 1999), we store episode = {action, observation, reward} in an experience pool H to train the corrector. Then, we update the parameters θ_c of agent using H according to the standard policy gradient, that is,

$$\theta_c \leftarrow \theta_c + \beta \sum_i reward_k \nabla_{\theta_c} \pi_{\theta_c}(s_i, \gamma_i) \tag{7}$$

where

$$\mathcal{L}_{\text{rl}} = -\sum_i reward_k \log \pi_{\theta_c}(s_i, \gamma_i) \tag{8}$$

and β is the learning rate.

2.4 Two-stage method to train aCat

We propose a two-stage method to train the pre-trained language model for downstream tasks and the corrector, as shown in Algorithm 1 in Appendix.

In stage one, we fix the parameters of the corrector and train the parameters of the prompt encoder and the pre-trained language model by minimizing the loss in Eq.(2).

During stage two, we fix the prompt encoder and pre-trained language model and train the parameters of the corrector according to the loss function in Eq.(7)(8). We iteratively conduct stage one and stage two in turn.

3 Experiments

3.1 Main Results

We compare the proposed aCat with the following baselines in Table 1.

fine-tuning refers to using vanilla fine-tuning of bert-base-cased to complete downstream tasks.

No-anchor: refers to no prompt that will be retained as an anchor token.

P-tuning described in (Liu et al., 2023b), and we show the performance reported in (Liu et al., 2023b).

Random selection randomly decides which prompt tokens need to be retained as anchor tokens.

To explore the impact of the number of correctors on performance, we tried two strategies to construct correctors, as shown below.

aCat-multi-agents each prompt token uses an independent corrector. It means that for multiple correctors, we set them as independent parameters and train them independently.

aCat-single-agent all prompt tokens use the same corrector. It means that we share parameters with all correctors.

Following P-tuning, we adopt Accuracy as the main evaluation measure and compare F1 on dataset CB and EM on dataset MultiRC.

Based on the experimental results shown in table 1, the bold values represent the best results per dataset, and we can see that aCat exhibits an improvement in accuracy in all datasets compared with the baselines. It indicates that automatic template construction achieves better performance in soft-hard prompt paradigms. We also find that aCat-single-agent outperformed aCat-multi-agents on more data sets. One explanation is that using a

Table 1: Main results

dataset	BoolQ (Acc.)	CB (Acc.)	CB (F1)	WiC (Acc.)	RTE (Acc.)	MultiRC (EM)	MultiRC (F1a)	WSC (Acc.)	COPA (Acc.)
fine-tuning	72.9	85.1	73.9	71.1	68.4	16.2	66.3	63.5	67
no-anchor	73.8	89.2	92.1	67.3	71.2	14.7	64.7	61.2	64.3
p-tuning	73.9	89.2	92.1	68.8	71.1	14.8	63.3	63.5	72
random selection	73.5	91.2	**93.4**	67.9	70.5	15.6	67.6	62.8	69.3
aCat-multi-agents	73.9	89.9	92.6	**68.9**	71.2	15.6	**67.6**	**64.7**	72
aCat-single-agents	**74**	**91.7**	93.1	67.2	**71.7**	**16.9**	67.4	63.8	**72.3**

Table 2: Compare with P-tuning V2

dataset	BoolQ (Acc.)	RTE (Acc.)	CB (Acc.)	WiC (Acc.)	WSC (Acc.)	COPA (Acc.)	avg. (Acc.)
P-tuning V2	71.19	70.03	82.14	**69.5**	**65.4**	67.4	70.94
aCatmulti-agents	73.9	71.2	89.9	68.9	64.7	72	73.43
aCatsingle-agent	**74**	**71.7**	**91.7**	67.2	63.8	**72.3**	**73.45**

single agent can make the content more coherent. In an aCat-multi-agents setting, each agent is independent of the others, which will destroy the coherence of the context. The use of aCat-single-agent can take into account the coherence of the context. On WIC and WSC, the performance of aCat-multi-agents exceeds aCat-single-agent. We deem that the semantic information of anchor tokens itself in WIC is not strong. Therefore, we do not require strong semantic correlation.
Further experiments, such as s_i using different token information, can be found in table b3 in Appendix.

3.2 Comparasion with P-tuning V2

We also compare our method with P-tuning V2 on six datasets, and we show the experimental results in Table 2. As for the MultiRC dataset, there are no reported performance records using Bert-base in p-tuning V2, so we do not list the results on MultiRC. The performances of P-tuning V2 are reported according to the results in (Yang et al., 2022; Zhang et al., 2023). Our method outperforms P-tuning V2 on four datasets and achieves higher average accuracy with fewer parameters. We deem that it is crucial to retain the task-specific prompt tokens, and simply initializing the embedding parameters of prompt tokens and training them with the pre-trained language model are difficult to yield the optimal performance.

4 Conclusion

Soft-hard prompt paradigm has been proven effective for natural language understanding. Selecting some tokens in prompt as anchor tokens can further improve the performance of the downstream tasks in prompt-based learning. To achieve automatic anchor token acquisition, we propose aCat, an automatically choosing anchor tokens method. By introducing a corrector agent, our method achieves automatic selection of anchor tokens from pre-defined prompts. Experiments conducted on seven benchmark natural language understanding datasets demonstrate that aCat achieves better performances compared to previous baselines under the soft-hard paradigm.

Limitations

The approach proposed in this paper has some limitations, specifically that we only use policy gradient method to train our corrector. Additionally, it would be beneficial to extend the idea to other efficient learning methods, such as avoiding the two-stage process for training the corrector. Exploring a unified solution for existing methods might be valuable in future research.

References

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. 2022. RLPrompt: Optimizing discrete text prompts with reinforcement learning.

In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Hunter Lang, Monica N Agrawal, Yoon Kim, and David Sontag. 2022. Co-training improves prompt-based learning for large language models. In *International Conference on Machine Learning*, pages 11985–12003. PMLR.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023a. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022a. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland. Association for Computational Linguistics.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023b. Gpt understands, too. *AI Open*.

Xiaochen Liu, Yang Gao, Yu Bai, Jiawei Li, Yinan Hu, He-Yan Huang, and Boxing Chen. 2022b. Psp: Pre-trained soft prompts for few-shot abstractive summarization. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 6355–6368.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Timo Schick and Hinrich Schütze. 2021a. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269.

Timo Schick and Hinrich Schütze. 2021b. It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Haoran Yang, Piji Li, and Wai Lam. 2022. Parameter-efficient tuning by manipulating hidden states of pretrained language models for classification tasks. *arXiv e-prints*, pages arXiv–2204.

Zhen-Ru Zhang, Chuanqi Tan, Haiyang Xu, Chengyu Wang, Jun Huang, and Songfang Huang. 2023. Towards adaptive prefix tuning for parameter-efficient language model fine-tuning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1239–1248.

Jie Zhou, Le Tian, Houjin Yu, Zhou Xiao, and Hui Su. 2022. Dual context-guided continuous prompt tuning for few-shot learning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 79–84.

A Appendix

A.1 Experiment setting

To compare with P-tuning, we perform experiments on the SuperGlue benchmark (Wang et al., 2019). A detailed description of SuperGlue can be found in P-tuning. Following P-tuning, we conducted experiments on BoolQ, MultiRC, CB, RTE, WiC and COPA. We use the training sets and the development sets of each different task as D_{train} and D_{dev}.

We reformulate NLU tasks into MLM tasks. We designed the prompt pattern followed by P-tuning. The examples of prompt patterns can be found in Table b1. For pre-trained language model M and M_f, we use bert-base-cased (Devlin et al., 2019). We divide the dataset D into τ batches for $D_{train_{st2}}$ and train the corrector at each α step δ times. The hyperparameters of M and M_f, such as learning rate α, batch size, and $epoch$ on different dataset are shown in Table b2.

We set different hyperparameters by balancing the performance of different datasets and the size of the dataset, which though α,τ,δ to adjust, in which τ means we divide the data of τ batches as $D_{train_{st2}}$, α means that we train stage1 α times and then switch to stage 2. δ means that we repeat training δ times when training the corrector in stage2. $epoch$ means that stage1 and stage2 together are executed for a total of $epoch$ times.

We use the AdamW optimizer (Loshchilov and Hutter, 2019) with a linearly decayed learning rate and use early stopping to avoid overfitting the training data. All experiments are conducted on NVIDIA A800 with 80GB memory.

Finally, since selecting different positions in a sentence will yield different information, resulting in different s_i. We additionally explored the impact of using information from different positions in the sentence on performance. Results are shown in Table b3.

In all our experiments, according to some of the prompts given in Table b1, we will repeat each pattern three times to obtain the average performance of each pattern and record the pattern with the highest performance in each data set.

B Further results

B.1 Comparison of experimental performance of different splicing strategies

We replace $h(\hat{y}_k)$ in Eq.(3) to explore the performance when s_i uses different token information. The experimental results are shown in Table b3, where (mask+anchor) is the original setting described in Eq.(3), (only anchor) means $s_i = \mathrm{F}([h([W_i])])$, and (cls+anchor) is $s_i = \mathrm{F}([h([W_i]); h([cls])])$. We found that the combination of (mask+anchor) will achieve more ideal performance. This indicates that combining anchor prompts and mask tokens can provide more information and construct better prompts. Taking the dataset COPA as an example, COPA needs to predict downstream tasks through autoregression (get results from multiple mask tokens), so it tends to provide more information to the corrector, enabling better automatic selection of anchor tokens. Thus, it achieves better performance.

Algorithm 1 overall algorithm

Pre-process: We split the dataset D into D_{train} and D_{dev}, and split D_{train} into $D_{train_{st1}}$ for the first stage and $D_{train_{st2}}$ for the second stage to make the training data belongs to the same field.

$\mathcal{D}_{train_{st2}} = \{\mathcal{B}_0, \ldots, \mathcal{B}_\tau\}$, $\mathcal{D}_{train_{st1}} = \{\mathcal{B}_{\tau+1}, \ldots, \}$, where $\mathcal{B}_\tau$ refer to batch and train corrector in each step α, $epoch$, and δ times.

Initiate: Initialize model parameters M, corrector parameters θ_c and prompt encoder θ. Set up an experience pool H//Only the parameters of the agent in the corrector are trainable.

for _ $in\ range(epoch)$ **do**
    for $step\ in\ range(total\ step)$ **do**
————————————stage 1——————————-
        if $step\ (mod\ \alpha)\ != 0$ **then**
            fix corrector θ_c parameters
            set M and θ to trainable
            Optimize $\mathcal{L}$ in Eq.(2) in $\mathcal{D}_{train_{st1}}$
————————————-stage 2——————————-
        else
            set θ_c to trainable
            fix M and θ parameters
            for _ $in\ range(\delta)$ **do**
                for $Batch_\tau \in \mathcal{D}_{train_{st2}}$ **do**
                    collect $episode = \{action, state, reward\}\ though\ Eq.(3)(5)(6)$
                    Add $episode$ to the experience pool H
                end for
                update θ_c though Eq .(7)(8) and experience pool H //update corrector parameters
                clear experience pool H
            end for
        end if
    end for
end for

Table b1: Prompt pattens of different datasets.

dataset	Prompt pattern 1	Prompt pattern 2
BoolQ	***p***. the Question ***q*** ? the Answer: __.	***p*** the ***q*** ? the __.
CB	***p*** the ***q*** ? Answer: __.	***p*** question: ***q*** true, false or neither? answer: the__.
WiC	***p*** [SEP] ***q*** the ***w*** ?	***p*** [SEP] ***q*** the ***w*** ? the __.
RTE	***p*** [SEP] ***q*** ? the __.	***p*** [SEP] ***q*** ? the answer: __.
MultiRC	***p*** Question: ***q*** ? Is it ***a*** ? the__.	***p*** Question: ***q*** ? the ***a*** ? the__.
WSC	***s*** the ****p***** the__.	***s*** the pronoun ****p***** refer to __.
COPA	***c1*** or ***c2***? ***p*** so/because __.	***c1*** or ***c2***? ***p*** so/because the __.

Note: ***p*** is the passage, ***q*** is the question, ***c1*** or ***c2*** are the choices, ***s*** refers to sentence, ***p*** in WSC refers to pronoun which appears in the sentences, and '_' refers to the mask token.

Table b2: The setting of hyperparameters

	BoolQ	CB	WiC	RTE	MultiRC	WSC	COPA
α	100	1	34	15	170	1	5
τ	7	2	3	2	11	3	2
batch size	16	16	16	16	16	16	16
δ	1	1	1	50	1	10	50
epoch	10	20	10	20	10	20	20
Learning rate	2e-5	2e-5	2e-5	2e-5	2e-5	1e-5	2e-5

Table b3: Comparison of experimental performance of different splicing strategies

dataset	BoolQ (Acc.)	CB (Acc.)	CB (F1)	WiC (Acc.)	RTE (Acc.)	MultiRC (EM)	MultiRC (F1a)	WSC (Acc.)	COPA (Acc.)
SINGLE AGENT									
aCat-single-agents (mask+anchor)	**74**	**91.7**	93.1	67.2	**71.7**	**16.9**	67.4	63.8	**72.3**
aCat-single-agents (only anchor)	74	91.1	93.4	**67.6**	71.7	15.8	**67.7**	63.1	68.3
aCat-single-agents (cls+anchor)	73.8	91.1	**93.4**	67.2	70.5	16.2	67.5	**64.1**	70.7
MULTI AGENTS									
aCat-multi-agents (mask+anchor)	**74.3**	89.9	92.6	**68.9**	71.2	**15.6**	**67.6**	**63.8**	**72**
aCat-multi-agents (only anchor)	73.4	**91.1**	**93.3**	67.5	71.2	16.2	67.6	62.8	68.3
aCat-multi-agents (cls+anchor)	74.3	89.9	91.7	67.5	**71.5**	15.5	67.6	63.8	70.7