**General Response** We thank all reviewers for their insightful comments! We are sorry that Figure 1 in the submission version of this paper is not the lastest so the descriptions in Section 4.3 are ambiguous. We will correct it in the updated version. We have revised typos and removed duplicated sentences.

**Regarding computation efficiency and memory cost.** For memory, SAC consumes smaller memory since the connections are sparse. For computation efficiency, at training time, SAC is only slightly faster than vanilla transformers, since learning which node should be linked to which node using RL is time-consuming. But at test time, SAC is significantly faster due to significantly smaller cost in self-attention computations. We will add more details in the updated version.

**To Reviewer #1** Yes, we agree that the improvements compared with state-of-the-art models are marginal. But the main goal of this paper is to **reduce the memory cost of vanilla self-attention while achieving slightly better performances**. We do not attempt to improve the results, and instead, we show that with less attention connections, the model is also strong and consistent over tasks.

**To Reviewer #2** Thanks for your careful and insightful feedback! We will correct all the typos and incorrect references in the next version. For the order, it is indeed what you think, i.e. $\{a_{11}, a_{12}, a_{21}, a_{22}, \cdots\}$, where $a_{i1}$ is the start node for the $i$-th edge and $a_{i2}$ is the end node.

`Correctness`: Yes, we cannot directly say the other four methods are "special cases" of SAC, but by imposing extra constraints when training the LSTM edge predictor, we can actually induce each of them. For example, for vanilla self-attention (take the encoder side as an example), we can feed each node $N$ times into the predictor so that it recovers vanilla self-attention with the help of distant encodings. For Transformer-XL, we can still segment the sequence and apply the above operations.

In terms of decoding in MT, all models (including baselines) use beam search. We are sorry for the confusion and will make this point clearer in the updated version.

`Weakness`: See **General Response**.

**To Reviewer #3** Thanks for you helpful and insightful comments!

`Weakness 1&2`: See **General Response**.

`Weakness 3`: We omit the parameters due to the limited space of the page. In fact, the numbers of parameters for these methods including SAC are very close, as you can see from Table 2 and Table 4 that the most parameters come from the main model, i.e. $\mathbf{\Phi}$ rather than $\mathbf{\Theta}$.

`Weakness 4`: Thank you for the sensible comment. We used a simple version of the edge predictor, in which all layers share the same structure and each node has to be connected to some other nodes for each layer. For the head adaptive strategy, we reported results for both (head adaptive or not adaptive) for different tasks. We will make these points clearer in the updated version.

**To Reviewer #4** We thank you for your insightful comments!

`Weakness 1`: See **General Response**.

`Weakness 2`: Yes, at first glance, the inference speed of SAC is slower than vanillan Transformer since it introduces the extra process of link prediction. But in fact, SAC does not need to do full self-attention, which makes a significant remedy for test speed. We will show these in the next version. As for baselines, connecting to nearest nodes is actually what CNNs do, for which many recent works have discovered for sparse self-attention. We will compare the performance and the speed of these methods.

`Correctness`: Thanks for your suggestions! The intuition of using LSTM edge predictor is to learn different attention patterns for different downstream tasks. In NMT, we find that the learned attention prefers more semantics-related words. We will give a deeper analysis and plot more figures to show how SAC construct attentions for different tasks.

`Clarity`: The LSTM predictor is initialized randomly (uniform and guassian distributions are both possible). We will clarify this in the next version.

`Reference`: We are sorry for the confusion and will make it clearer in the updatedd version.