Learning to Iteratively Solve Routing Problems with Dual-Aspect Collaborative Transformer

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recently, Transformer has become a prevailing deep architecture for solving vehicle routing problems (VRPs). However, the original Transformer is less effective in learning *improvement* models because its positional encoding (PE) method is not suitable in representing VRP solutions. This paper presents a novel *Dual-Aspect Collaborative Transformer* (DACT) to learn embeddings for the node and positional features separately, instead of fusing them together as done in the original PE, so as to avoid potential noises and incompatible attention scores. Moreover, the positional features are embedded through a novel *cyclic positional encoding* (CPE) method to capture the circularity and symmetry of VRP solutions. We train DACT using Proximal Policy Optimization, and design a curriculum learning strategy for better sample efficiency. We apply DACT to solve the traveling salesman problem (TSP) and capacitated vehicle routing problem (CVRP). Results show that DACT outperforms existing Transformer based improvement models, and exhibits better capability of generalizing across different problem sizes.

1 Introduction

Vehicle Routing problems (VRPs), such as the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP) which consider finding the optimal route for a single or fleet of vehicles to serve a set of customers, have ubiquitous real-world applications [1–3]. Despite being intensively studied in the Operations Research community, VRPs still remain challenging due to their NP-hard [4] nature. Recent studies on learning neural heuristics are gathering attention as promising extensions to traditional hand-crafted heuristics for solving VRPs (e.g., [5–13]), where reinforcement learning (RL) [14] is usually exploited to train a deep neural network as an efficient solver without hand-crafted rules. A salient motivation is that deep neural networks may learn better heuristics by identifying useful patterns directly from data in an end-to-end fashion.

Solutions to VRPs, i.e., routes, are sequences of nodes (customer and depot locations). Naturally, deep models for Natural Language Processing (NLP), which deal with sequence data as well, are ideal choices for solving VRPs. Given its remarkable performances in various NLP tasks, Transformer [15] is standing at the forefront in the learning based methods for VRPs (e.g., [6, 8–13]). The original Transformer encodes a sentence, i.e., a sequence of words, into a unified set of embeddings by injecting word positional information into its word embeddings through positional encoding (PE). When it comes to VRPs, while is not required in *construction* models, positional information is critical for deep models that learn *improvement* heuristics since the input are solutions to be improved.

Although some success has been achieved [10–12], learning *improvement* heuristics for VRPs based on the original PE methods of Transformer is yet lacking from our perspective. Firstly, directly applying addition operation on PE vectors and the embeddings in absolute PE method could limit the representation of the model [16], as it can bring mixed correlations and induce noisy biases in the

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

self-attention module (see Appendix A for more details). Secondly, existing PE methods tend to fuse the node and positional information into one unified representation. NLP tasks such as translation may benefit from this owing to the deterministic and instructive nature of the positional information [15]. However, such design may not be optimal for routing tasks because the positional information therein can be non-deterministic and sometimes even random. This may yield incompatible or conflicting attention scores in the encoder[1]. Finally, current PE methods are designed for encoding linear sequences, which could be less effective to model circular VRP solutions[2]. As will be shown in our experiments, this limitation could severely damage the generalization performance on different problem sizes, since the cyclic feature of VRP solutions is not correctly reflected in the encoding.

In this paper, we address the above issues and contribute to the line of using RL to learn improvement heuristics for VRPs. We introduce a novel architecture for routing tasks, called *Dual-Aspect Collaborative Transformer (DACT)*, where we revisit the solution representations of existing methods and propose to learn separated groups of embeddings for the node and positional features of a routing solution. Our DACT follows the encoder-decoder structure [10]. In the encoder, we design a *cross-aspect referential attention mechanism* for better perceiving and capturing the consistence and differentiation between the two heterogeneous information. The decoder then collects action distribution proposals from these two aspects and synthesize them to output the final action distribution. Meanwhile, we design a novel *cyclic positional encoding* (CPE) method to capture the circularity and symmetry of VRP solutions, which is demonstrated to be more effective and suitable for encoding cyclic sequences than existing ones. As the last contribution, we design a simple yet effective curriculum learning strategy to improve the sample efficiency. This further leads to faster and more stable convergence of RL training. Extensive experiments show that our method outperforms existing Transformer based improvement models, and also generalizes well across different problem sizes.

2 Related work

Positional encoding in Transformer. The original Transformer model in [15] introduced *positional encodings* (PE) to describe the absolute position of elements in the sequence to make use of the sequence order information. In [17], the relative PE method was further proposed to better capture the relative order of tokens in the sequence. To avoid the mixed and noisy correlations between word semantics and positional information in the absolute PE, the Transformer with United Positional Encoding was proposed for NLP in [16] to separate the two heterogeneous information when calculating attention scores. However, as mentioned previously, existing PE methods are less effective in solving VRP since they only learn one unified set of embeddings which fuse the node and positional information, and are designed for encoding only linear sequences.

Deep models for VRP. Various deep architectures such as Recurrent Neural Network (RNN), Graph Neural Network (GNN), and Transformer have been employed in solving VRPs.

RNN based models. As the pioneering work of solving VRP using deep neural networks, Pointer Network [18] adopted RNN and attention mechanism [19] based on supervised learning to solve TSP (extended to RL in [20] and further extended to CVRP in [21]). While the models in [18, 20–22] learn construction heuristics, NeuRewriter [5] learns improvement heuristic for CVRP by using bi-directional LSTM to encode the positional information in a solution. In a recent work [12], the conditional variational autoencoder was adopted to learn a continuous and latent search space for VRP, where high-quality solutions were taken as input and encoded by the RNN structure. However, recurrence structures in RNN are less efficient in both representation and computation [6].

GNN based models. In [23], GNN was combined with deep Q-learning to learn greedy heuristic for TSP. Based on supervised learning, Joshi et al. [7] used GNN to predict the probability of each edge appearing in an optimal TSP tour (i.e., heatmap). This idea was extended in [24] with additional components such as graph sampling and heatmap merging to enable generalization to larger TSP instances. However, these GNN models are limited to TSP, and often require post-processing to build feasible solutions from heatmaps (e.g., beam search in [7] and Monte-Carlo tree search in [24]).

[1] For example, nearby customers share very similar node embeddings, but they might be far apart in a solution. This may cause disharmony and disturbance when calculating embedding correlations in self-attention and thus affect the accuracy of the policy.

[2] The relative PE method seems to help, however, it is found to be even worse than the absolute PE method [10], partly due to the incompatibility issue caused by learning the unified representation introduced above.

Transformer based models. Recently, Transformer style networks have received much attention in VRP. The Attention Model (AM) [6] was the first success of such models for learning construction heuristics, which was then exploited by many recent works. Xin et al. [8] proposed a Multi-Decoder Attention Model (MDAM) to train multiple diverse policies to improve the solution quality. Based on AM, Kwon et al. [9] proposed POMO (Policy Optimization with Multiple Optima) to force diverse rollouts to enhance performance. Besides the above construction models, several methods exploit Transformer architecture to learn improvement heuristics. Hottung and Tierney [25] integrated large neighborhood search with learning based heuristics for CVRP and split delivery VRP. Lu et al. [11] proposed the L2I model that learns to select local search operators from a pool of traditional ones. Both methods use a Transformer style attention module to encode a solution, but the positional information is captured in the node features (information of previous and next nodes) instead of using PE methods. Though L2I was shown to outperform LKH3 [26], it is limited to CVRP and the run time is considerably longer than other neural methods. Wu et al. [10] proposed a novel Transformer based model which learns to pick node pair in each improvement step to perform a certain pairwise local operator (e.g., swap or 2-opt). However, it suffers from the inaccurate representation of positional information, since the original PE method is used to encode the position of each node in a solution.

3 Problem formulation

We define a VRP instance as a group of N nodes to visit, where the *node feature* x_i of node i contains 2-dim coordinates and other problem-specific features (e.g., customer demand). A solution δ consists of a sequence of nodes visited in order where we denote p_i to be the position (indices) of node i in the solution which is deemed as the *positional feature* of node i. The objective is to minimize the total travel distance $D(\delta)$ in the Euclidean space, where the solution needs to satisfy problem-specific constraints. Below we formulate the RL task of learning improvement heuristic.

Starting with an initial yet complete solution, the RL policy tries to improve the solution iteratively. At each time step, the policy automatically selects a pair of nodes and locally adjusts the solution using a preset pairwise operator (details are given in Appendix B). We model such improvement process as the form of Markov Decision Process (MDP).

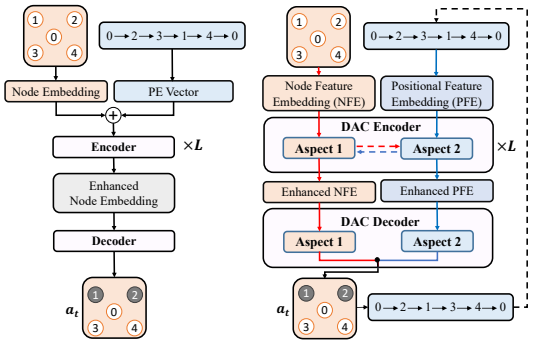


Figure 1: Transformer frameworks for solving VRPs. (a) the original one; (b) ours.

State. For an instance with N nodes, a state describes current solution δ_t using its node and positional features of each node, i.e., $s_t = \Psi(\delta_t) = \{x_1^t, ..., x_N^t, p_1^t, ..., p_N^t\}$.
Action. The action $a_t = (i, j)$ specifies a node pair (i,j) for the pairwise operator.
Reward. The reward function is defined as, $r_t = D(\delta_t^*) - min\left[D(\delta_{t+1}), D(\delta_t^*)\right]$ where δ_t^* is the best solution found until time t [10]. The reward $r_t > 0$ if and only if a better solution is found.
Policy. The policy π_θ is parameterized by the proposed DACT model with parameters θ. At each time step, the action (i, j) is obtained by sampling the stochastic policy for both training and inference.
Transition. The next state s_{t+1} is originated from s_t by performing the preset pairwise operator on the given node pair (action). Our state transient is deterministic, in the sense that it always accepts the next solution as the next state (infeasible solutions will be masked), regardless of its objective value. With such simple rule, the RL agent is expected to automatically learn how to combine multiple steps of simple local movements to achieve better solutions, even if some of them may worsen the current solution. The trajectory can be factorized as $\pi_\theta(s_T|s_0) = \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t)$. Note that the time limit T of our trajectory can be any reasonable user-specified value according to the allowed time budget. Hence, our MDP can have infinite horizon and we consider the reward discount factor $\gamma < 1$.

4 Dual-aspect collaborative Transformer model

As illustrated in Figure 1(a), when solving a VRP instance, the original Transformer encodes a solution by fusing the node and positional information into a unified set of embeddings, which

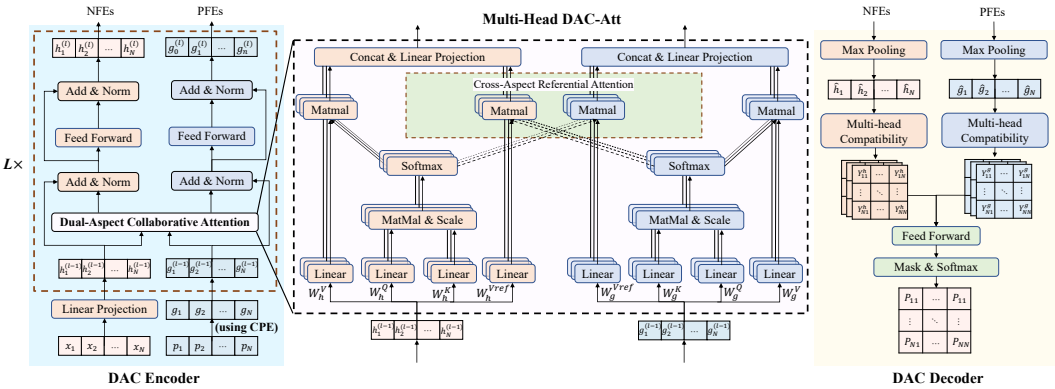


Figure 2: Architecture of our policy network: dual-aspect collaborative Transformer (DACT).

may impair the model performance. As illustrated in Figure 1(b), our Dual-Aspect Collaborative Transformer (DACT) model leverages separate *aspects* of embeddings to encode a routing solution. In the DAC *encoder*, the self-attention scores are computed individually for each aspect to avoid correlation noises, and a *cross-aspect referential attention* mechanism is proposed to enable an aspect to effectively exploit the cue from the other as an optional reference. The DAC *decoder* then collects action distribution proposals from both aspects and synthesize them to output the final action distribution. The concrete architecture of our DACT model is presented in Figure 2, where we take the TSP with N nodes as an illustration example.

4.1 Dual-aspect solution representation

Specifically, we learn two sets of embeddings, i.e., the *node feature embeddings* (NFEs) for node representation and the *positional feature embeddings* (PFEs) for positional representation.

NFEs. Following [10], the NFE h_i of the node i is initialized as the linear projection of its node feature x_i with output dimension $dim = 64$.

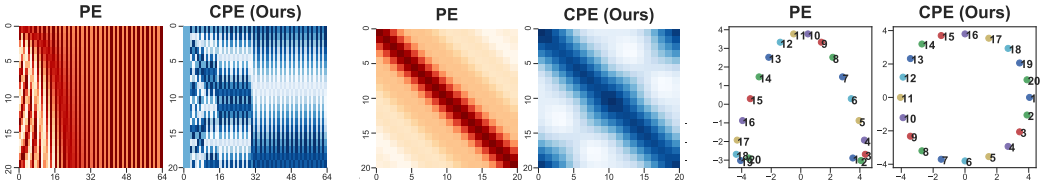
PFEs. The PFE g_i of the positional feature p_i is initialized as a real-valued vector ($dim = 64$) by applying our cyclic positional encoding (CPE), which is designed based on cyclic Gray codes [27].

Figure 3: An example of cyclic Gray code where 4 digits are used to encode $N = 16$ nodes. The top left shows the base symmetry pattern '1001' in Gray code, and the top right plots its representation in our method.

As illustrated in Figure 3, the original cyclic Gray codes present a cyclic property ('1110' in the last column is adjacent to '1111' in the first column) and an adjacency similarity property (the codes in any two adjacent columns only differ in one digit), both of which are desirable for encoding cyclic sequences like the VRP solutions. To preserve these properties in designing our CPE, we follow two observed key patterns: 1) each numerical digit contains a periodic cycle with reflectional symmetry, e.g., the '10|01' in the lowest digit; and 2) the higher the numerical digit, the longer the period. Accordingly, we create similar patterns by exploiting the sinusoidal functions in Eq. (1), where a periodic function with period $\frac{4\pi}{\omega_d}$ (induced by modulus) is used to generate the cycle with reflectional symmetry (e.g., the top right in Figure 3),

$$\overrightarrow{g_i}^{(d)} := \begin{cases} sin(\omega_d \cdot \left|(z(i) \bmod \frac{4\pi}{\omega_d}) - \frac{2\pi}{\omega_d}\right|), \text{ if } d \text{ is even} \\ cos(\omega_d \cdot \left|(z(i) \bmod \frac{4\pi}{\omega_d}) - \frac{2\pi}{\omega_d}\right|), \text{ if } d \text{ is odd} \end{cases} \tag{1}$$

where $z(i) = \frac{i-1}{N}\frac{2\pi}{\omega_d}\left\lceil \frac{N+1}{2\pi/\omega_d} \right\rceil$ is to make N nodes linearly spaced in the generated pattern; the angular frequency ω_d is decreasing along the vector dimension to make the wavelength longer (the wavelength is chosen from $[N^{\frac{1}{\lceil dim/2 \rceil}}, N]$, and see Appendix C for detailed formulation). In Figure 4, we visualize the comparison between the absolute PE method and our CPE method based on a TSP



(a) (b) (c)

Figure 4: Comparison of our CPE method with absolute PE method on a TSP instance with 20 nodes. (a) the embedding vectors, (b) the correlations (dot products) between every two embeddings, and (c) the top two principal components after PCA (principal component analysis) projection.

instance with 20 nodes. Figure 4(a) demonstrates that our real-valued base symmetry pattern has a longer cyclic period as the digit grows. Figure 4(b) indicates that our method (blue) is able to correctly reflect the the adjacency between the head and tail of the sequence in a cyclic solution, whereas the PE method (red) fails to do so. Figure 4(c) verifies that our CPE vectors are well distributed in space with the desired cyclic and adjacency similarity properties.

4.2 The encoder

Similar to [6, 10], we stack $L(L=3)$ DAC encoders. In each DAC encoder, we retain relatively independent coding paths for NFEs and PFEs as in Eq. (2) and Eq. (3), respectively, each of which consists of a shared Dual-Aspect Collaborative Attention (**DAC-Att**) sub-layer and an independent feed-forward network (**FFN**) sub-layer. The DAC-Att takes both sets of embeddings as input and then outputs their respective enhanced embeddings, i.e., NFEs $\{\tilde{\boldsymbol{h}}\}_{i=1}^{N}$ and PFEs $\{\tilde{\boldsymbol{g}}\}_{i=1}^{N}$. All sub-layers are followed by skip connection [28] and layer normalization [29].

$$h_i^{(l)} = \mathbf{LN}\left(h_i' + \mathbf{FFN}_h^{(l)}(h_i')\right), h_i' = \mathbf{LN}\left(h_i^{(l-1)} + \tilde{\boldsymbol{h}}_i^{(l)}\right), \tag{2}$$

$$g_i^{(l)} = \mathbf{LN}\left(g_i' + \mathbf{FFN}_g^{(l)}(g_i')\right), g_i' = \mathbf{LN}\left(g_i^{(l-1)} + \tilde{\boldsymbol{g}}_i^{(l)}\right). \tag{3}$$

DAC-Att. The DAC-Att sub-layer enhances each set of embedding mainly from its own aspect, while leveraging the attention scores from the other aspect to achieve the synergy. Given the two sets of embeddings[3], $\{h_i\}_{i=1}^{N}$ and $\{g_i\}_{i=1}^{N}$, we first compute the self-attention scores from their own aspects,

$$\alpha_{i,j}^{h} = \frac{1}{\sqrt{d_k}}\left(h_i W_h^Q\right)\left(h_j W_h^K\right)^T, \quad \alpha_{i,j}^{g} = \frac{1}{\sqrt{d_k}}\left(g_i W_g^Q\right)\left(g_j W_g^K\right)^T, \tag{4}$$

where we use independent projection matrices W_h^Q, W_h^K, W_g^Q and $W_g^K \in \mathbb{R}^{dim \times d_k}$ to calculate *queries* and *keys* in each aspect. The obtained attention scores are further normalized to $\tilde{\alpha}_{i,j}^{h}$ and $\tilde{\alpha}_{i,j}^{g}$ via the Softmax function. Note that the attention scores here are computed from their own aspect of NFE and PFE, which eliminates possible noise and conduces to correctly describe the incompatible node pair relationships in different aspects of VRP solutions.

To achieve synergy of the two aspects, we then exploit a *cross-aspect referential attention* mechanism. It allows the computed attention scores shared between the two aspects, as an additional reference for both contradistinction and collaboration, which is formulated as follows,

$$\text{out}_i^{h} = \text{Concat}\left[\sum_{j=1}^{N}\tilde{\alpha}_{i,j}^{h}\left(h_j W_h^V\right), \sum_{j=1}^{N}\tilde{\alpha}_{i,j}^{g}\left(h_j W_h^{Vref}\right)\right], \tag{5}$$

$$\text{out}_i^{g} = \text{Concat}\left[\sum_{j=1}^{N}\tilde{\alpha}_{i,j}^{g}\left(g_j W_g^V\right), \sum_{j=1}^{N}\tilde{\alpha}_{i,j}^{h}\left(g_j W_g^{Vref}\right)\right], \tag{6}$$

where we calculate the weighted sum of attention *values* based on both the weights derived from its own aspect and the other; $W_h^V, W_g^V \in \mathbb{R}^{dim \times d_v}$ are trainable parameter matrices for formulating

[3]We omit the encoder index l for better readability.

values in each aspect; and $W_h^{Vref}, W_g^{Vref} \in \mathbb{R}^{dim \times d_v}$ are parameter matrices for each aspect to generate referential values which represent adjacency from the other aspect. Finally, we adopt the *multi-head* in Transformer [15] to further enhance the attention with the output NFE $\tilde{h}_i$ and PFE $\tilde{g}_i$ as follows,

$$\begin{gathered}\tilde{h}_i, \tilde{g}_i = \textbf{DAC-Att}\left(W^Q,\ W^K, W^V, W^{V_{ref}}, W^O\right), \\ \tilde{h}_i = \text{Concat}\left[\text{head}_{i,1}^h, ..., \text{head}_{i,m}^h\right] W_h^O,\ \tilde{g}_i = \text{Concat}\left[\text{head}_{i,1}^g, ..., \text{head}_{i,m}^g\right] W_g^O,\end{gathered} \tag{7}$$

where $\text{head}_{i,k}^h = out_{i,k}^h$, $\text{head}_{i,k}^g = out_{i,k}^g$, and $W_h^O, W_g^O \in \mathbb{R}^{2md_v \times dim}$ are trainable parameter matrices for the final output projection. In our model, we adopt $m = 4$ and $d_k = d_v = 16$.

FFN. Our FFN sub-layer has only one hidden layer with 64 hidden unites and adopts the ReLU activation function. The parameters of $\textbf{FFN}_h$ and $\textbf{FFN}_g$ are different for each group of embeddings.

4.3 The decoder

In the DAC decoder, the two sets of embeddings $\{h_i^{(L)}\}_{i=1}^N$ and $\{g_i^{(L)}\}_{i=1}^N$ are first passed through a **Max-pooling** sub-layer and a **Multi-head compatibility** sub-layer to independently generate diversified node-pair selection proposals from their own aspect. The generated proposals are then aggregated through a feed-forward aggregation (**FFA**) sub-layer to output the final action distribution.

Max-pooling. For each set of embeddings, we adopt the max-pooling sub-layer in [10] to aggregate the global representation of all N embeddings into each respective one[4].

Multi-head compatibility. The compatibility sub-layer [10] computes the attention scores for each embedding pair following Eq. (4), where the obtained scores are deemed as a proposal distribution for selecting a node pair. Our enhanced attention scores are computed using multiple heads in order to select each node pair with different thresholds for diversity. Meanwhile, we calculate two separated attention score matrices $Y_k^h, Y_k^g \in \mathbb{R}^{N \times N}$ (of head k) from the two aspects independently. Accordingly, the two action distribution proposals would be different due to their aspect-specific focus and cognitions of the current solution, which will provide the subsequent FFA layer with a rich pool of proposals and allow our model to be more flexible and robust.

FFA. Once all proposals from the two aspects are collected, a feed-forward network with four layers (dimensions are $2m$, 32, 32 and 1, respectively) and a ReLU activation is used to aggregate them,

$$\tilde{Y}_{i,j} = \textbf{FFA}\left(Y_{i,j,1}^g, ..., Y_{i,j,m}^g, Y_{i,j,1}^h, ...Y_{i,j,m}^h\right), \tag{8}$$

where $m = 4$ is the number of heads; the network is shared for each node pair to preserve the ability of generalization on solutions of different lengths; the output $\tilde{Y}_{i,j}$ is a scalar indicating the likelihood of selecting the node pair (i, j) as an action. Afterwards, $\tilde{Y}_{i,j}$ is further gauged in the range of $[-C, C](C = 6)$ to control the entropy of the final action distribution, i.e., $\hat{Y}_{ij} = C \cdot \text{Tanh}(\tilde{Y}_{i,j})$, where we mask the infeasible [5] node pairs (i', j') as $\hat{Y}_{i'j'} = -\infty$ following [10]. Lastly, the likelihoods are normalized using Softmax function to obtain the final action distribution $P_{i,j}$.

4.4 Reinforcement learning algorithm

To train our DACT policy π_θ, we adopt proximal policy optimization (PPO) [30] with n-step return estimation, which is detailed in Appendix D. Meanwhile, we also design a *curriculum learning strategy* to stabilize and improve the training performance.

Curriculum learning strategy. The strategy in [10] sets a maximum of T_{train} steps for training and estimates future returns after T_{train} steps by *bootstrapping* [31]. However, due to the concern of training cost, T_{train} is usually much smaller than the actual T for inference (e.g., 200 v.s. 10k), which may leave the agent a poor chance of observing high-quality solutions (states) during training. Consequently, it may cause high variance for estimation using bootstrapping because the value function is mostly fitted on low-quality solutions, and may render it less knowledgeable in estimating long-term future returns accurately. In this paper, we tackle this issue by designing a simple yet

[4] E.g., for $\hat{h}$, $\hat{h}_i = h_i^{(L)} W_h^{Local} + \max\left[\{h_i^{(L)}\}_{i=1}^N\right] W_h^{Global}$, where $W_h^{Local}, W_h^{Global} \in \mathbb{R}^{64 \times 64}$ are parameters.

[5] Besides, we also mask all the diagonal elements since they are not meaningful to the pair-wise operators, and the node pair selected at the last step to forbid possible dead loops [10].

efficient strategy which gradually prescribes higher-quality solutions as the initial states for training. In doing so, 1) it increases the probability for the agent to observe higher-quality solutions and thus reduce the estimation variance of the value function; 2) it increases the difficulty of the learning task (higher-quality solutions are harder to improve) in a gradual manner and allows the agent to achieve better sample efficiency [32]. In practice, those higher-quality solutions can be easily achieved by running the current policy for a few (T_{init}) steps on the randomly generated solutions, where T_{init} could be slightly increased as the epoch grows. In our algorithm (see line 5 of Algorithm 1 in Appendix D), we increase T_{init} according to an S-shaped logistic function which has a gentler slope at the beginning and the end to ensure more stable training.

5 Experiments

We evaluate our DACT model on two representative routing problems, i.e., TSP and CVRP [6, 9, 10]. For each problem, we follow [6] to randomly generate instances on the fly for three sizes, i.e., $N =$ 20, 50 and 100. Initial experiments with three operators including *2-opt*, *swap* and *insert* show that 2-opt performs best for both TSP and CVRP, hence we report results of our method based on it. Following [10], we use randomly generated initial solutions for training and the solutions generated by nearest insertion for inference. Since each problem has its own constraints and node features, we adjust the input, feasibility masks, and problem-dependent hyperparameters for each problem. Our DACT is trained and tested on a GPU server equipped with TITAN RTX GPU cards and Intel i9-10940X CPU at 3.30 GHz. More details and discussions are provided in Appendix E and F.

Training. We train our model with $E = 200$ epochs and $B = 20$ batches per epoch with batch size 600 for TSP and CVRP [6]. Regarding the n-step PPO algorithm, we set $n = 4$, $T_{train} = 200$ for TSP, and $n = 10$, $T_{train} = 500$ for CVRP. PPO performs $K = 3$ mini-batch updates per batch with its objective function clipped by a threshold $\epsilon = 0.1$. We adopt the Adam optimizer [33] with a learning rate $\eta_\theta = 10^{-4}$ for π_θ and $\eta_\phi = 3 \times 10^{-5}$ for v_ϕ, both of which are decayed with $\beta = 0.985$ per epoch for convergence. We use pretrained models for TSP50 and CVRP50 to train TSP100 and CVRP100 for faster convergence, while for other cases the model parameters are initialized randomly. We enable all state transitions of the MDP and the masking for feasibility of a batch to be performed in parallel on GPU for higher efficiency. Training time varies with different problem sizes. An epoch takes 6m (minutes), 9m, and 13m for TSP20, TSP50 and TSP100; 21m, 35m, and 53m for CVRP20, CVRP50 and CVRP100, which are shorter than those reported in [10]. Our PyTorch code of DACT models for TSP and CVRP will be made publicly available soon.

5.1 Comparison studies

In Table 1, we compare our DACT with, (1) learning based *improvement* models, including Wu et al. [10], Neural-2-Opt [22] (TSP only), NewRewriter [5] (CVRP only), NLNS [25] (CVRP only), (2) learning based *construction* models, including AM-sampling [6], GCN-BS [7] (TSP only), MDAM-BS [8], POMO [9], (3) learning based optimization approach equipped with the heuristic optimizer, such as CVAE-Opt-DE [12], and (4) strong conventional solvers including Concorde [34], LKH [26, 35], and OR-Tools [36]. All results are averaged over 10,000 randomly generated instances unless specified otherwise (e.g., the ones marked with # only infer 2,000 instances), and we report the metrics of objective values, (optimality) gaps and run time. Regarding baselines, we follow the results reported in their original papers, which may not include all the three metrics. For TSP, Concorde is adopted to get the optimal solutions, based on which the optimality gaps of other methods are calculated. CVRP is harder to be solved optimally, and the gaps are calculated based on solutions of LKH. Note that even for the baselines which infer 10,000 random instances, their objective values might be slightly different from ours (e.g., the ones marked with ‡), therefore we focus more on gaps for fair comparison. The run time is also hard to compare due to various factors (e.g., GPU/CPU models, batch sizes, Python v.s. C++). For DACT, we report the time for inferring all 10,000 instances with multiple GPU cards in (.), and a small batch (512 instances) with one single GPU card in {.}.

Pertaining to TSP, our DACT with inference step limit of 5,000 (T=5k) outperforms the traditional solver OR-Tools and all improvement models in terms of optimality gap, including Wu et al. [10] which learns a unified set of embeddings for the node and positional features using Transformer. It also

[6] Except CVRP100 for which we use 512 due to the limited GPU memory.

Table 1: Comparison with various baselines on TSP and CVRP.

	Method	N=20			N=50			N=100		
		Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
TSP	Concorde	3.83	-	(3m)	5.70	-	(10m)	7.76	-	(1h)
	LKH	3.83	0.00%	(38s)	5.70	0.00%	(5m)	7.76	0.00%	(20m)
	OR-Tools	3.86	0.94%	(42s)	5.85	2.87%	(5m)	8.06	3.86%	(23m)
	Neural-2-Opt [22]	3.84‡	**0.00%**	(15m)	5.70	0.12%	(29m)	7.83	0.87%	(41m)
	Wu et al. [10] (T=5k)	3.83	**0.00%**	(1h)	5.70‡	0.20%	(1.5h)	7.87	1.42%	(2h)
	DACT (T=1k)	3.83	0.04%	{7s}(24s)	5.70	0.14%	{16s}(1m)	7.89	1.62%	{48s}(4m)
	DACT (T=5k)	3.83	**0.00%**	{32s}(2m)	5.70	0.02%	{2m}(6m)	7.81	0.61%	{4m}(18m)
	DACT (T=10k)	3.83	**0.00%**	{1m}(5m)	5.70	0.01%	{3m}(13m)	7.79	0.37%	{8m}(40m)
	DACT×4 augment	3.83	**0.00%**	{3m}(10m)	5.70	**0.00%**	{10m}(1h)	7.77	**0.09%**	{29m}(2.5h)
	GCN-BS [7]	3.84‡	0.01%	(12m)	5.70	0.01%	(18m)	7.87	1.39%	(40m)
	AM-sampling [6]	3.84‡	0.08%	(5m)	5.73	0.52%	(24m)	7.94	2.26%	(1h)
	MDAM-BS [8]	3.84‡	**0.00%**	(3m)	5.70	0.03%	(14m)	7.79	0.38%	(44m)
	POMO [9]	3.83	0.04%	(1s)	5.70‡	0.21%	(2s)	7.80	0.46%	(11s)
	POMO×8 augment [9]	3.83	**0.00%**	(3s)	5.69‡	0.03%	(16s)	7.78	0.15%	(1m)
	CVAE-Opt-RS [12]	-	**0.00%**#	6m#	-	0.20%#	17m#	-	1.35%#	51m#
	CVAE-Opt-DE [12]	-	**0.00%**#	11m#	-	0.02%#	22m#	-	0.34%#	55m#
CVRP	LKH	6.14	0.00%	1h	10.38	0.00%	4h	15.68	0.00%	8h
	OR-Tools	6.46	5.68%	2m	11.27	8.61%	13m	17.12	9.54%	46m
	NeuRewriter [5]	6.15#	-	6m#	10.51#	-	11m#	16.10#	-	17m#
	NLNS [25]	6.19#	-	6m#	10.54#	-	11m#	15.99#	-	16m#
	Wu et al. [10] (T=5k)	6.12‡	0.39%	(2h)	10.45	0.70%	(4h)	16.03‡	2.47%	(5h)
	DACT (T=1k)	6.15	0.28%	{16s}(33s)	10.61	2.13%	{43s}(2m)	16.17	3.18%	{2m}(5m)
	DACT (T=5k)	6.13	-0.00%	{1m}(3m)	10.48	1.01%	{3m}(8m)	15.92	1.55%	{8m}(23m)
	DACT (T=10k)	6.13	-0.04%	{2m}(6m)	10.46	0.79%	{6m}(16m)	15.85	1.12%	{16m}(45m)
	DACT×6 augment	6.13	**-0.08%**	{11m}(35m)	10.39	**0.14%**	{32m}(1.5h)	15.71	**0.19%**	{1.5h}(4.5h)
	AM-sampling [6]	6.25	1.87%	(6m)	10.62	2.40%	(28m)	16.23‡	3.72%	(2h)
	MDAM-BS [8]	6.14	0.18%	(5m)	10.48	0.98%	(15m)	15.99‡	2.23%	(1h)
	POMO [9]	6.17‡	0.82%	(1s)	10.49	1.14%	(4s)	15.83	0.98%	(19s)
	POMO×8 augment [9]	6.14‡	0.21%	(5s)	10.42	0.45%	(26s)	15.73	0.32%	(2m)
	CVAE-Opt-RS[12]	6.15#	-	16m#	10.54#	-	38m#	16.11#	-	1.5h#
	CVAE-Opt-DE [12]	6.14#	-	21m#	10.40#	-	41m#	15.75#	-	1.5h#

the obj. values, gaps or time are obtained based on 2,000 instances in their original papers, and not directly comparable to ours.
‡ the obj. values obtained by Concorde or LKH in their original papers are slightly different from ours since the 10,000 instances are randomly generated. E.g., for TSP20, the optimal values according to our running of Concorde is 3.83, while 3.84 in [6–8, 22]; for TSP50, ours is 5.70, while 5.69 in [9, 10]. Therefore, we focus more on gaps for fair comparison.

outstrips construction methods including AM-sampling on TSP50, TSP100, and GCN-BS on TSP100. With larger step limit (T=10k), our DACT further boosts the solution qualities and outperforms other construction methods including MDAM with beam search (MDAM-BS), and POMO (the state-of-the-art). To further reduce the gaps, we also leverage the data augmentation technique in POMO (which considers flipping node coordinates without changing the optimal solution) to solve same instances multiple times in different ways. Although the inference time increases (we run data augmentation in serial on the same GPUs), our DACT with 4 augments not only outstrips POMO with 8 augments but also achieves the lowest objective values and gaps among all learning based models. In particular, our method almost optimally solved TSP20 and TSP50 with gap lower than 0.005%, and 0.09% on TSP100, which is superior to most of the recent learning based models. Pertaining to CVRP, our DACT with T=5k produces lower gaps than that of improvement models including NeuRewriter and NLNS. It also performs much better than Wu et al. [10] except on CVRP50. With T=10k and 6 augments[7], our DACT exhibits even better performance than the highly specialized heuristic solver LKH on CVRP20 and delivers the smallest gap of 0.19% on CVRP100 against other learning based methods including POMO with 8 augments. Compared with CVAE-Opt-DE, despite that it is averaged over less instances and integrated with differential evolution algorithms, our objective values are still lower. (We also test our DACT on benchmark dataset [37] in Appendix F.)

In terms of the run time, our DACT is competitive against all learning based methods except POMO, which learns a construction model by sampling multiple trajectories. It is possible to adopt a similar strategy in our DACT to find better solutions earlier. For example, we may force diverse rollouts for an improvement model by fixing different node pairs as the first actions, as POMO fixes different nodes as the first starting nodes in a construction model. Since our focus is to ameliorate the Transformer in the improvement models, we will investigate these possible extensions in the future.

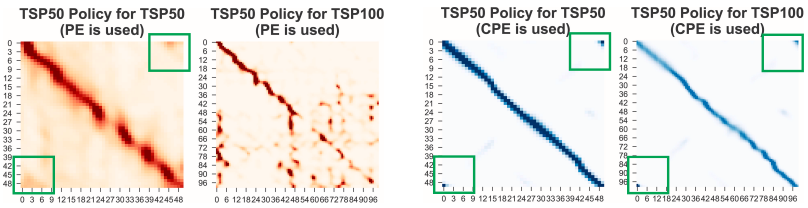
[7]The gaps will further decrease to 0.09% if we adopt 8 augments, with run time of about 7h.

Table 2: Dual v.s. single aspect representation.

Step Limit	Method	N=50		N=100	
		Obj.	Gap	Obj.	Gap
T = 1000	SA-T	5.72	0.35%	8.03	3.49%
	DACT	**5.70**	**0.14%**	**7.89**	**1.62%**
T = 5000	SA-T	5.70	0.05%	7.88	1.55%
	DACT	**5.70**	**0.02%**	**7.81**	**0.61%**

Table 3: PE v.s. CPE on generalization.

Method	N=20		N=100	
	Obj.	Gap	Obj.	Gap
DACT-PE (T=5k)	3.84	0.21%	8.38	7.93%
DACT-CPE (T=5k)	**3.83**	**0.10%**	**7.99**	**2.98%**
Wu et al. [10] (T=5k)	3.91	2.14%	9.03	16.37%
OR-Tools	3.83	0.00%	8.06	3.87%



(a) (b)

Figure 5: Visualization of the attention scores for the encoder when a trained model is used to solve instances with a larger size. (a) using PE method; (b) using CPE method (ours).

5.2 Ablation studies

Dual aspect representation. In Table 2, we evaluate the effectiveness of our dual aspect representation against the single aspect one (SA-T) on TSP50 and TSP100, where SA-T mainly follows the Transformer based architecture in [10] but equipped with the CPE, multi-head attentions and curriculum learning strategy for fair comparison. We observe that DACT consistently outperforms SA-T, which verifies the effectiveness of the dual aspect representation.

Cyclic positional encoding. Here we show that CPE significantly improves the capability of generalizing across different problem sizes. In Table 3, we record the results of our DACT with PE and CPE, and Wu et al. [10], when the model trained on TSP50 is directly used to solve instances from TSP20 and TSP100. We see that even with PE, our DACT outperforms Wu et al. [10]. Further equipped with CPE, DACT outstrips DACT-PE and OR-Tools on TSP100. We continue to compare the two DACT variants by visualizing their attention scores. As depicted in Figure 5(a), although PE is designed for linear sequences, it did attempt to perceive the circularity of VRP solutions (as highlighted in the green boxes) but failed, which instead engenders random attention scores when generalizing to larger size. In contrast, our DACT with CPE is able to capture the circularity as depicted in Figure 5(b), which verifies the effectiveness of CPE in representing VRP solutions.

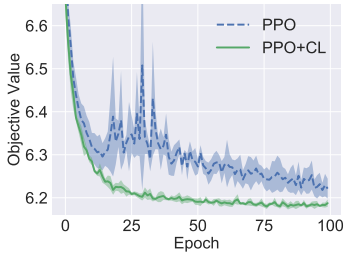


Figure 6: Training curves of PPO with and without CL on CVRP20 (random seeds 1-5).

Curriculum learning (CL) strategy. In Figure 6, we plot the training curves of PPO algorithm with and without our CL strategy, where the results are averaged over 5 independent runs with 90% bootstrap confidence intervals. It shows that our CL strategy significantly improves the sample efficiency while reducing the variance of training, which aligns with our analysis in Section 4.4.

6 Conclusions and future work

In this paper, we present a novel DACT model for solving VRPs. It learns separate groups of embeddings for the node and positional features, and is equipped with a cyclic positional encoding method to capture the circularity and symmetry of VRP solutions. Extensive experiments justified the effectiveness of DACT. A potential limitation is that DACT is more useful for learning improvement models at present, for which positional information is of great importance. In the future, we will investigate how to extend DACT to construction models, and integrate DACT with diverse rollouts strategies (e.g., POMO [9]) to further improve efficiency. We will also evaluate our method on more routing problems such as the pickup and delivery problem (PDP) [13].

References

[1] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM press, 2014.

[2] Michael Schneider, Andreas Stenger, and Dominik Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, 48(4):500–520, 2014.

[3] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[4] Jan Karel Lenstra and AHG Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[5] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 32, pages 6281–6292, 2019.

[6] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[7] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. arxiv preprint arxiv:1906.01227, 2019.

[8] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *AAAI Conference on Artificial Intelligence*, 2021.

[9] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 21188–21198, 2020.

[10] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[11] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2019.

[12] André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In *International Conference on Learning Representations*, 2021.

[13] Jingwen Li, Liang Xin, Zhiguang Cao, Andrew Lim, Wen Song, and Jie Zhang. Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[14] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 6000–6010, 2017.

[16] Guolin Ke, Di He, and Tie-Yan Liu. Rethinking the positional encoding in language pre-training. In *International Conference on Learning Representations*, 2020.

[17] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 464–468, 2018.

[18] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 2692–2700, 2015.

[19] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[20] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *International Conference on Machine Learning (Workshop)*, 2017.

[21] Mohammadreza Nazari, Afshin Oroojlooy, Martin Takáč, and Lawrence V Snyder. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9861–9871, 2018.

[22] Paulo R d O Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning*, pages 465–480, 2020.

[23] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6351–6361, 2017.

[24] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large TSP instances. In *AAAI Conference on Artificial Intelligence*, 2021.

[25] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *European Conference on Artificial Intelligence*, 2020.

[26] Keld Helsgaun. LKH-3 (version 3.0.6), 2019. URL `http://webhotel4.ruc.dk/~keld/research/LKH-3/`.

[27] Wikipedia. Gray code, 2021. URL: `http://en.wikipedia.org/w/index.php?title=Gray%20code&oldid=1023725478`. Last visited on 2020/05/19.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[29] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. Corr: abs/1607.06450, 2016.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arxiv preprint arxiv:1707.06347, 2017.

[31] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. In *International Conference on Machine Learning*, pages 4045–4054, 2018.

[32] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning*, pages 41–48, 2009.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[34] Vašek Chvátal David Applegate, Robert E. Bixby and William J. Cook. Concorde TSP Solver, 2020. URL `http://www.math.uwaterloo.ca/tsp/concorde/`.

[35] Keld Helsgaun. LKH (version 2.0.9), 2018. URL `http://webhotel4.ruc.dk/~keld/research/LKH/`.

[36] Laurent Perron and Vincent Furnon. OR-Tools (version 7.2), 2019. URL `https://developers.google.com/optimization/`.

[37] Gerhard Reinelt. TSPLIB-A traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[38] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO. In *International Conference on Learning Representations*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We provide the way to generate the data in Appendix E. Our code, pre-trained models, as well as instructions to reproduce the main results will be released soon.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5 and Appendix F.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See confidence intervals in Figure 6 and error bars in Appendix F.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We have cited them, e.g., [26, 34–37].
   (b) Did you mention the license of the assets? [Yes] We have mentioned that all used assets are open source softwares in Appendix F.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The data used in this paper are all randomly generated.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The data used in this paper are all randomly generated.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]