Dear Reviewer qUxm,

Thank you for your great recognition of our work. We likewise hope that our work can make some modest contributions to the progress of this field. We tried our best to address all the concerns and questions, and update the main paper and appendix(marked blue) in the new version. Please let us know if you have any further concerns or questions to discuss.

Best,

Paper 1571 Authors

W1. Please provide simpler explanations and more visual examples for the interpolation combined with diffusion.

A1. As shown in Figure 2(main paper), our model is based on the U-Net architecture.

The first step is to construct the input matrix $\hat{x}*N$ by integrating multiple components: task class labels obtained from our classifier, visual observation features (V_s and V_g), and the action sequence $a*{1:T}$. This matrix is then refined through our masked projection during initialization to constrain the action space.

Subsequently, the intermediate latent features F_j generated by the latent space temporal logical interpolation module are incorporated into the diffusion model to progressively refine the input matrix.

The U-Net model, represented by the central gray box in Figure 2(main paper), comprises three downsampling modules, two upsampling modules, and one middle module. Each of these modules contains two residual temporal blocks, which serve as the integration points for features.

As depicted in Figure 3(b)(main paper), the interpolated features derived from the interpolation module undergo a linear transformation to form keys and values (LF_j), ensuring dimensional compatibility with the input matrix. Within the residual temporal block module, the input matrix $\hat{x}_n$ is processed through Convolutional Blocks (CB) and combined with timesteps from TimeMLP (TM), resulting in $\left[CB(\hat{x}_n) + TM(t)\right]$ (where t is randomly sampled), which serves as the query. The query and key-value pairs are then processed via Multi-head Attention mechanisms, facilitating the integration of interpolated features and the input matrix. The resulting matrix is further refined through LayerNorm, Linear layers, and Mish activation, before passing through a Convolutional Block to progress to the subsequent residual temporal block.

We've added more detailed visualizations in Appendix D to illustrate the feature maps of interpolated features.

Q1. Balance of current loss function and other possible kinds of loss function.

A2. While applying weights solely to endpoints initially enhances performance by emphasizing these crucial positions, this approach leads to excessive focus on endpoints at the expense of middle actions. To address this limitation, we introduce a task-adaptive masked proximity loss that implements a more balanced weighting strategy - applying stronger supervision at endpoints while gradually relaxing constraints towards the middle through our weighted gradient loss mechanism. This approach helps achieve better equilibrium between endpoint accuracy and intermediate state supervision. As demonstrated in the comparison table below, our method significantly improves action prediction accuracy across all positions, with particularly notable gains in middle actions (showing a 60.6% improvement for a_3 compared to 18.2% for a_5). These results validate that our approach successfully balances the model's attention between endpoints and intermediate actions.

Table 1: Comparison of action accuracy at each timestep between endpoint weighting and gradient weighting.

Action	Weight on both sides	Gradient Weight	Improvement
a_1	62.32	74.26	+19.2%
a_2	26.55	42.64	+60.6%
a_3	21.45	34.44	+60.6%

Action	Weight on both sides	Gradient Weight	Improvement
a_4	30.76	43.03	+39.9%
a_5	61.10	72.19	+18.2%

We explored several alternative loss functions, including Cross-Entropy (CE) and combinations with ranking loss. Our experiments revealed that MSE loss is most effective for this task. The key advantage of MSE loss lies in its ability to model comprehensive distance relationships between actions throughout the entire action matrix. In contrast, CE loss only computes individual position-wise losses, limiting its capacity to capture complex action dependencies. While we attempted to enhance performance by combining MSE with ranking loss, which focuses on adjacent action relationships, this combination did not yield improvements. We attribute this to potential conflicts between the optimization objectives - MSE's holistic distance modeling versus ranking loss's local relationship focus - which may have counteracted each other's benefits.

Table 2: More experiments for exploring different types of loss functions when T=5.

Loss	SR↑
Cross Entropy	7.77
MSE	11.89
MSE+Ranking	8.50
MSE+GW+M	**15.26**

More information about the ranking loss function:

For each sequence in a batch, let $a \in \mathbb{R}^{T \times A}$ be the probability distribution of predicted actions by the model (where T is the number of time steps and A is the dimension of the action space). Let $\bar{a} \in \mathbb{N}^T$ be the true action index sequence. The steps to calculate the ranking loss are as follows:

1. Sort the predicted probabilities of actions for each time step t in descending order to obtain the sorted indices:
   $$R_t = \text{argsort}(a_t) \quad \text{for } t \in [1,T].$$

2. Find the position (rank) of the true action in the sorted list:
   $$\text{rank}_t = \text{position}(\bar{a}_t \text{ in } R_t) \quad \text{for } t \in [1,T].$$

3. Calculate the rank difference between adjacent time steps:
   $$\text{diff}*t* = *text{rank}_t* - *text{rank}*{t+1} \quad \text{for } t \in [1,T-1].$$

4. Define an order mask that only calculates the loss when the true action indices satisfy a specific order relationship:
   $$\text{mask}*t* = *mathbb{1}[\bar{a}_t* > *bar{a}*{t+1}] \quad \text{for } t \in [1,T-1].$$

5. The final ranking loss is:
   $$L_R = \frac{1}{T-1}\sum_{t=1}^{T-1} \text{ReLU}(\text{diff}_t \cdot \text{mask}_t).$$

Where $\text{ReLU}(x) = \max(0, x)$, and $\mathbb{1}[\cdot]$ is the indicator function, which is 1 when the condition is true and 0 otherwise.

The purpose of this loss function design is to ensure that when in the true action sequence, the index of the previous action is larger than the index of the subsequent action ($\bar{a}*t* > *bar{a}*{t+1}$), the model should give a higher ranking (i.e., a smaller rank value) to the subsequent action. If this rule is violated, a positive loss value will be produced

---

Q2. About the classification and procedure planning results on COIN and NIV.

A3. There are two possible explanations for this phenomenon:

- **Classification results**: The dataset size significantly impacts classification performance, particularly in terms of task and action diversity. While COIN contains a larger variety of tasks and actions compared to NIV, this actually makes NIV classification more straightforward due to its less complex task structure and more focused action space.

- **Procedure planning performance**: The effectiveness of procedure planning is primarily determined by the average sequence length of actions within tasks. The model generates intermediate action sequences by leveraging visual features from the start (V_s) and goal (V_g) states. However, the reliability of visual information gradually diminishes towards the middle of the sequence. Given that NIV contains longer action sequences on average, this degradation of visual information in the middle portions results in decreased prediction accuracy.