TrailBlazer : Trajectory Control for Diffusion-Based Video Generation Supplementary Material

1 IMPLEMENTATION

In this section we describe details of our implementation, including the core library, hyperparameters, and other pertinent information. Our method is developed using PyTorch 2.01 [5] and the Diffusers library version 0.21.4 from Huggingface [2]. We override the Diffusers pipeline `TextToVideoSDPipeline` to produce our implementation.

Parameters are selected as follows: We use classifier-free guidance with a strength of 9, conduct 40 denoising steps, and use a video resolution of 512x512 for the conventional stable diffusion backward denoising process. For all the comparisons with Peekaboo [3] we use their official repository[1] at the commit `6564274` (12 Feb 2024). In our comparisions we utilize a resolution of 576x320 as employed in the Peekaboo code to ensure fair assessment.

In addition, We use MotionCtrl's [8] official repository[2] at the commit `3d0ec04` (20 Jul 2024), and VideoComposer [7] repository[3] at the commit `490ed21` (11 Nov 2023) for baseline comparisons in the paper. Please refer to Sec 4.1 of the main text for detailed information on the experiment setup. MotionCtrl and VideoComposer employ different backbone diffusion models, each with unique inference parameters such as video length and aspect ratios. However, we adjust our bounding box configuration to accommodate their specific requirements.

Regarding the parameters specific to our proposed method, the majority of our results are generated using the default values outlined as follows: We execute 5 editing steps for both prompt and temporal attention, denoted as $N_S \equiv N_M \equiv 5$. The editing coefficients $c_w \equiv 0.001$ and $c_s \equiv 0.1$ are used in both prompt and temporal attention in most cases. The number of trailing attention maps $|\mathcal{T}|$ is the only parameter that needs to be tuned. Generally, $10 \leq |\mathcal{T}| \leq 20$ yields satisfactory results in practice and we set $|\mathcal{T}| \equiv 15$ for our paper results.

As highlighted in Sec 1. in the main text, we adapt the pre-trained ZeroScope[4] [1] T2V model. This model is fine-tuned from the initial weights of ModelScope[5] [4] utilizing nearly ten thousand clips, each comprising 24 frames as training data. Consequently, we adhere to the recommended practice of setting the length of the synthesized sequence to 24 frames, drawing insights from user experiences shared in relevant blogs.[6]

Prompt cross-attention editing is performed at several resolutions with a module with the following architecture:

```
transformer_in.transformer_blocks.0.attn2
down_blocks.0.attentions.0.transformer_blocks.0.attn2
down_blocks.0.attentions.1.transformer_blocks.0.attn2
down_blocks.1.attentions.0.transformer_blocks.0.attn2
down_blocks.1.attentions.1.transformer_blocks.0.attn2
down_blocks.2.attentions.0.transformer_blocks.0.attn2
down_blocks.2.attentions.1.transformer_blocks.0.attn2
up_blocks.1.attentions.0.transformer_blocks.0.attn2
up_blocks.1.attentions.1.transformer_blocks.0.attn2
up_blocks.1.attentions.2.transformer_blocks.0.attn2
up_blocks.2.attentions.0.transformer_blocks.0.attn2
up_blocks.2.attentions.1.transformer_blocks.0.attn2
up_blocks.2.attentions.2.transformer_blocks.0.attn2
up_blocks.3.attentions.0.transformer_blocks.0.attn2
up_blocks.3.attentions.1.transformer_blocks.0.attn2
up_blocks.3.attentions.2.transformer_blocks.0.attn2
```

For temporal attention editing, we found that a multiple-resolution approach was not necessary and produced unpredictable results. Instead, temporal attention editing uses a single layer:

```
mid_block.attentions.0.transformer_blocks.0.attn2
```

2 FURTHER ABLATIONS

Given the limited space in the primary text, here we describe additional ablation tests to substantiate our proposed approach. Broadly, we illustrate the impact of the spatial and temporal placement of guidance bounding boxes *(bboxes)* on the overall result quality, exploring the effect of various bbox speed and size choices directed by user keyframing. To see details, please zoom in to the experiment images, and **especially refer to our supplementary video.** Note that the animated green bboxes in our supplementary video are manually annotated to visualize the type of intended motion and do not exactly reflect the motion derived from true bbox keyframes. This was done because we do not have a tool to automatically add such "bbox animations" to videos. The identical green bbox animation is added for both our method and the baseline. The quantitative measures (e.g. mIoU) reported in the paper use the true interpolated bboxes rather than these video annotations.

Fig. 1 illustrates video synthesis using the pre-trained ZeroScope model **without** applying our approach. Broadly, all the synthesized results exhibit fine details with plausible temporal coherence as would be seen in a real video featuring relatively slow motion. However, several undesirable effects may be introduced alongside this realism. For example, the synthesized subject is often positioned in the same general area near the center of the images regardless of portrayed motion, and subjects like a galloping horse do not conveying the notion of speed. Additionally, artifacts such as extra or missing limbs (e.g., the cat in the second row) or other implausible results occasionally occur.

2.1 Exploration and Ablation: Varied static bbox sizes

Fig. 2 shows the effect of the size of the bbox without considering motion. The results indicate that the bbox size significantly influences the outcome. In extreme cases, the top row illustrates that a smaller bbox may yield unexpected entities in the area (e.g., white smoke next to the horse) or information leakage to the neighboring area (e.g., the blue attribute affecting the road). In contrast, the bottom row demonstrates that a overly large bbox can lead to broken results in general (e.g., the fish disappearing into the coral reef, and the strange blue pattern in place of the expected blue car). We expect this may be in large part due to the centered-object bias [6] in the pre-trained model's training data.

[1]https://github.com/microsoft/Peekaboo
[2]https://github.com/TencentARC/MotionCtrl
[3]https://github.com/ali-vilab/videocomposer
[4][2]:cerspense/zeroscope_v2_576w
[5][2]:damo-vilab/modelscope-damo-text-to-video-synthesis
[6]https://zeroscope.replicate.dev/

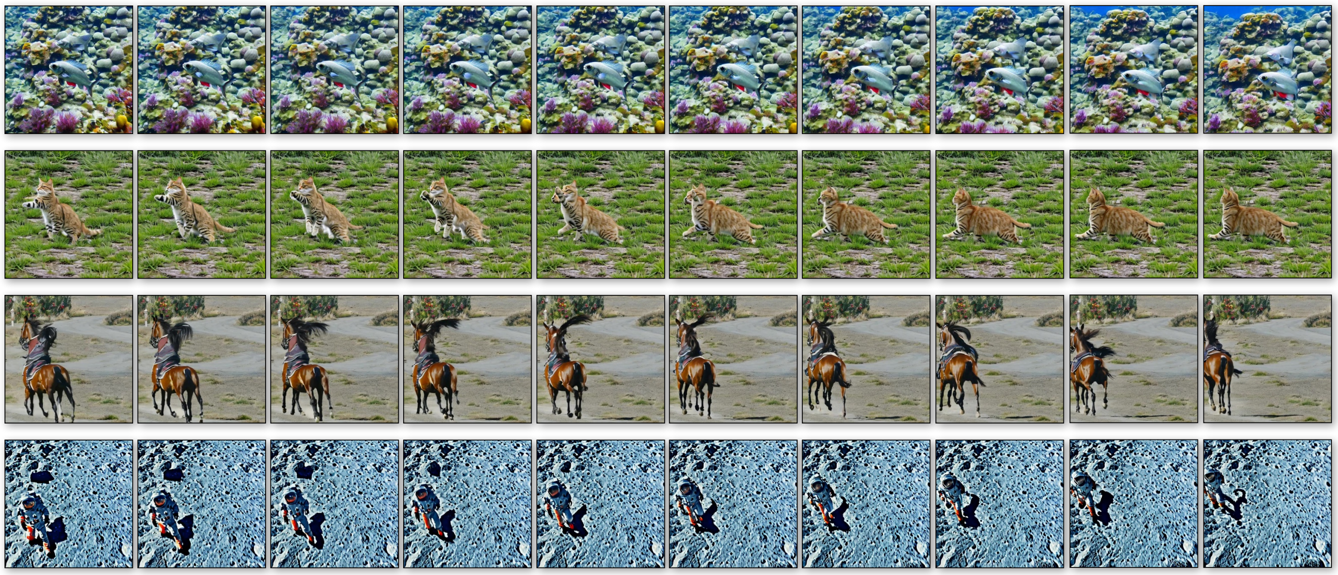
Figure 1: Baseline results. Each row shows equally-spaced frames sampled from a video generated using ZeroScope *without applying our trajectory control approach*. The prompts used starting from the first row: "A [fish] swimming in the sea", "The [cat] running on the grass field", "The [horse] galloping on the road", and "An [astronaut] walking on the moon". These prompts are reused in subsequent examples in these supplementary results.

Our recommended bbox size falls within the range of 30% to 60% for optimal reconstruction quality. Note that very small- or large-sized bboxes can still be employed in our approach, but they are best specified for a particular frame rather than the entire sequence. This is demonstrated, for example, in Fig. 3 guiding the swimming fish.

2.2 Exploration and Ablation: Varied dynamic bbox sizes

Fig. 3 demonstrates video synthesis with a dynamically changing bbox size. In the top-left example, the bbox grows larger and then shrinks, resulting in a perspective effect where the fish swims towards the camera and then away from it. The frame highlighted in red indicates the middle keyframe with a large bbox. This aligns with our main text results in Fig. 6, showcasing that the animated tiger respect the bbox size and its trajectory. The top-right example is a comparison to the top-left, portraying the fish only swimming toward the camera.

The second and the third rows show a comparison of the same bbox condition with the prompt words "fish" (second row), and "sardine" (third row), respectively. This experiment aims to assess how well our method adapts to large bbox size variations, represented by the short/wide target bbox on the left and tall/thin target bbox on the right. The result on the left indicates that the output from the "fish" prompt does not adequately conform to the short-wide aspect ratio of the bounding box, whereas the result from the "sardine" prompt can more closely adjust to the desired bbox thanks to the elongated shape of the sardine. Conversely, in the experiment on the right, both "fish" and "sardine" perform well with the tall/thin bounding box, since the tall aspect ratio can be satisfied by a fish facing directly toward or away from the camera. In general we expect that the obtained results will mimic the situations found in ZeroScope's training data, while views that are outside the typical data (such as a fish swimming vertically, or a horse at the top of the image) will be difficult to synthesize.

As with all our results, we see that the guided subject *approximately* follows the specified bounding box, but does not exactly lie within the bbox. While this is a disadvantage for some purposes, we argue that it is also an advantage for casual users – if the subject exactly fit the bounding box it would require the user to imagine the correct aspect ratio of the subject under perspective (a difficult task for a non-artists) as well as do per-frame animation of the bbox to produce the oscillating motion of the swimming fish seen here.

2.3 Exploration and Ablation: Speed control with multiple keys

Fig. 4 demonstrates controlling the subject's speed through varying the number of keyframes in the video synthesis. Given the recommended sequence length $N_f = 24$ for ZeroScope, we show the result of adding different keyframes in between the start and end keyframes at the left/right image boundary, simulating the cat running back and forth on the grass field. It is clear that the cat moves relatively naturally according to the motion flow indicated by the yellow arrows. For instance, the cat looks back first before turning around, rather than showing an unnatural motion where the position of the head and tail is instantaneously swapped. As the cat moves faster, motion blur also introduced in the result. We found that this motion blur is hard to eliminate using negative prompts.

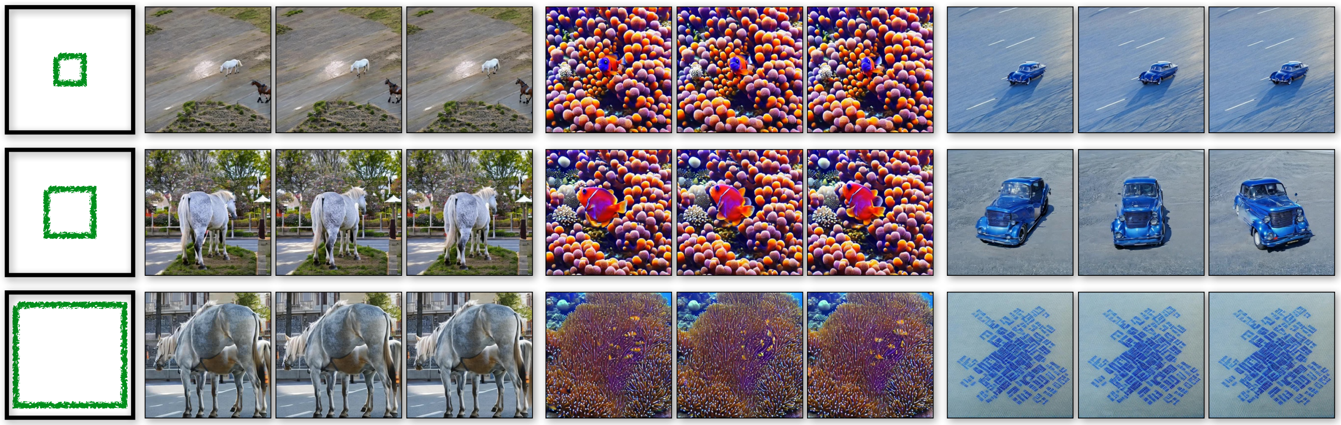

Figure 2: Static bbox sizes. Each row shows the result of a static square bbox positioned at the center, where the width and height are 25%, 50%, and 90% of the original image size (represented by the the green square on the left). The prompts used in the three sets of the experiments are: "The [white horse] standing on the street", "The [fish] swimming in the sea", and "The [blue car] running on the road".

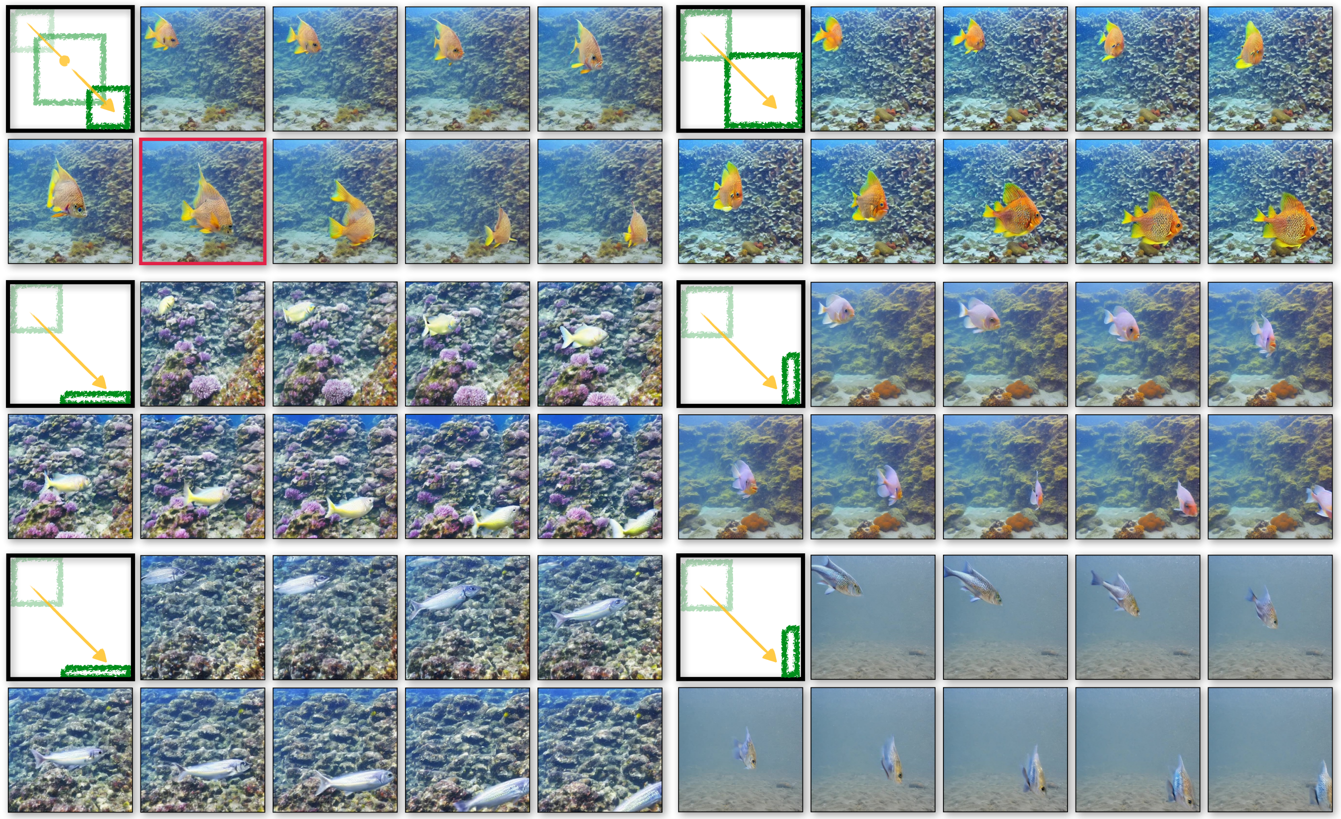

Figure 3: Dynamic bbox sizes. The result showcases six synthesized video sequences with the subject directed by the yellow arrow starting at the position indicated by green bbox. The number of the bboxes (corresponding to the number of keyframes used in the experiment) is, clockwise from top-left, $|\mathcal{K}|$ = 3, 2, 2, 2, 2, and 2, respectively. The prompt used in each result: "The [X] swimming in the sea", where "[X]" denotes the "fish" for the first and second rows, and "sardine" for the third row.

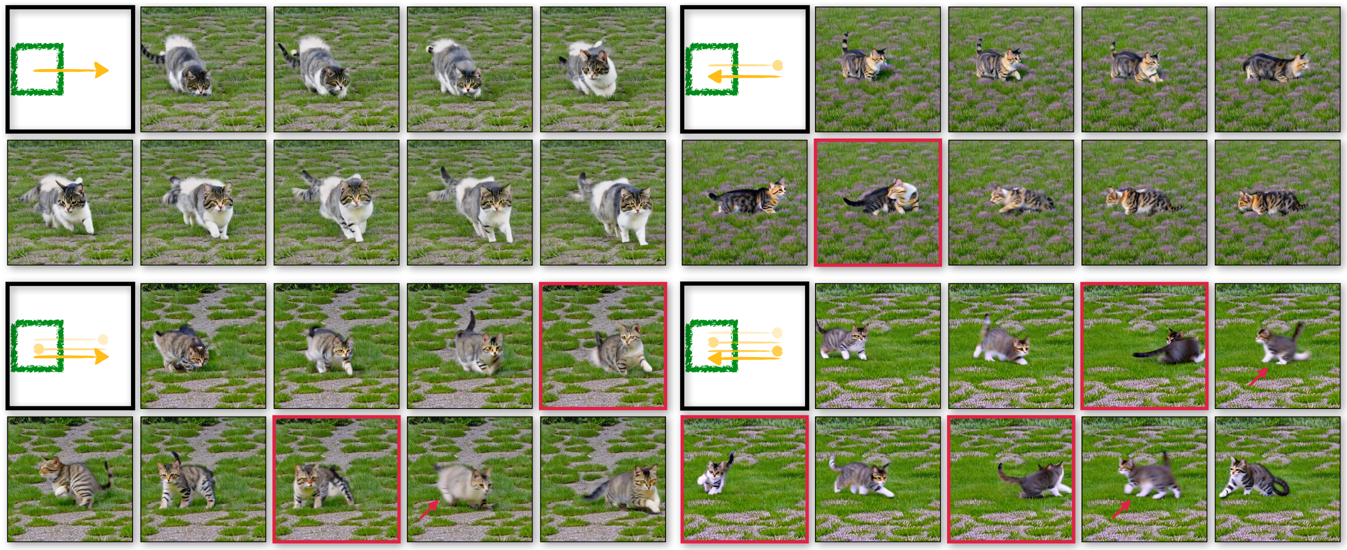

Figure 4: Speed Test: number of keyframes. This result shows four synthesized video sequences with the cat's motion directed according to the yellow arrows starting from the position indicated by green bbox. The number of the arrows denotes the number of keyframes (excluding the start/end keyframes) used in each experiment. Specifically, starting from the top-left and proceeding in left/right top/down (English reading) order, there are $|\mathcal{K}| = 2, 3, 4,$ and 5, keyframes, respectively. The frames highlighted with red correspond to the user-specified keyframes, excluding the start and end keyframes. The prompt used for all experiments is "A [cat] running on the grass field". The red arrows in the bottom-right example highlight the motion blur introduced by the fast movement.

2.4 Exploration and Ablation: Controlling speed with different placement of a single keyframe

Fig. 5 shows the results of moving the subject with increasing speeds. The first row shows the astronaut moving with constant speed obtained by the linearly interpolating bboxes at the left and right of the image. Starting from the second row, the astronaut holds the position of the first bbox on the left side of the image for some period of time, then moves more rapidly to the right side of the image, as illustrated in the second column of the figure. This is obtained by changing the timing of a single "middle" keyframe $\mathcal{K}_{f_1}$, where the first keyframe and the middle keyframe have the same bbox location (e.g., $\mathcal{B}_{f_0} \equiv \mathcal{B}_{f_1}$). Similar to the results in Fig. 4, the synthesis may generate motion blur and artifacts when the speed is high (e.g., last row).

2.5 Exploration and Ablation: Irregular trajectory

We illustrate irregular trajectories determined by varied keyframes in Fig. 6. The four experiments involve a zigzag trajectory (top-left), a triangle trajectory (top-right), a *discontinuous* trajectory (bottom-left), and a down-pointing triangle trajectory (bottom-right). In every result the horse shows high-speed running with motion blur. However, the results with turning points show limitations in depicting the horse quickly turning around and may show artifacts. For example, in the third frame of the down-pointing triangle case, the horse appears to swap its head and tail. Difficulty portraying this turn is somewhat expected, as horses cannot naturally execute tight high-speed turns, unlike cats or dogs. On the other hand, the down-pointing triangle video naturally introduces a perspective-like size change as the horse moves higher in the image, similar to the previous results in Fig. 3, and also the tiger example Fig. 6 in our main text. In summary, maintaining consistency between the prompt and the timing and location of the keyframed bounding boxes is crucial for producing realistic results.

3 CODE

The core of TrailBlazer is the attention processor, whose implementation is shown in Sec. 3.2 below.[7] In addition, subject morphing is achieved by modifying the `forward` function to allow interpolated prompt embeddings as conditioning. We omit irrelevant code in order to simplify this section.[8]

[7]`diffusers.models.attention_processor.AttnProcessor`
[8]`diffusers.models.unet_3d_condition.UNet3DConditionModel`

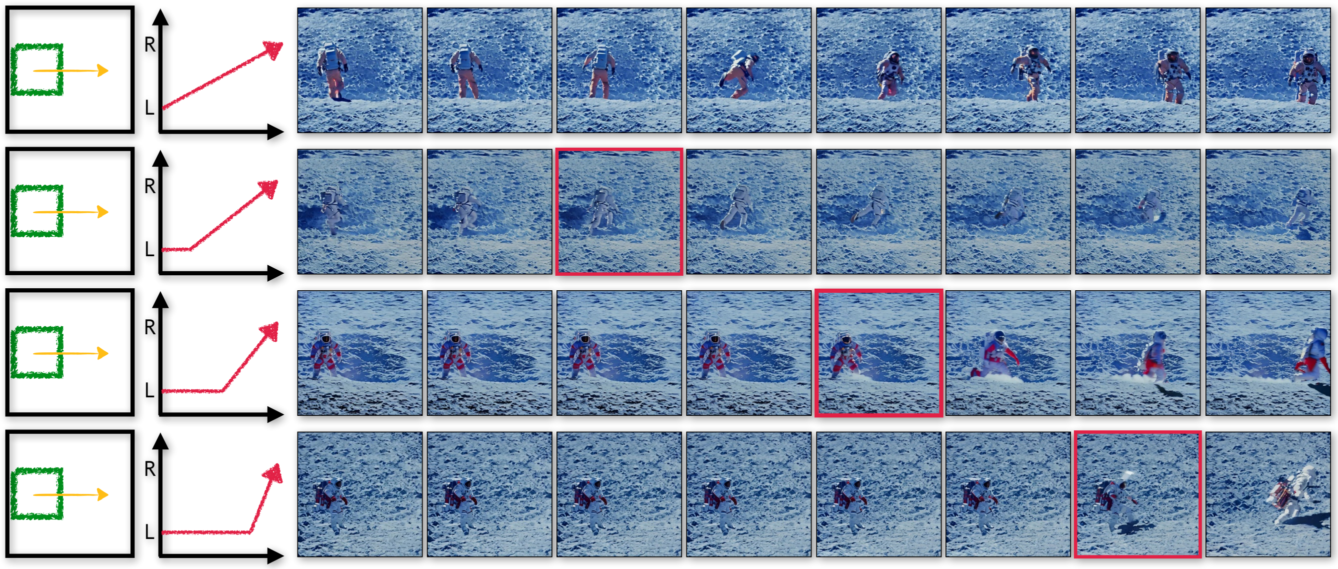

Figure 5: Speed Test: the timing of a keyframe. The result shows four synthesized video sequences with the subject directed according to the yellow arrow starting at the position indicated by green bbox, as illustrated in the first column. All experiments except the first use three keyframes ($|\mathcal{K}| = 3$), where the timing of the internal keyframe (e.g., $\mathcal{K}_{f_1}$) controls the duration of a stationary phase and the speed of the subsequent motion, as illustrated in the second column. The vertical and horizontal axis in the second column represent the left/right position and timing, respectively. The frame outlined in red indicates the frame controlled by $\mathcal{K}_{f_1}$, corresponding to the time when the astronaut starts to move. The prompt used for all experiments: "The [astronaut] walking on the moon".

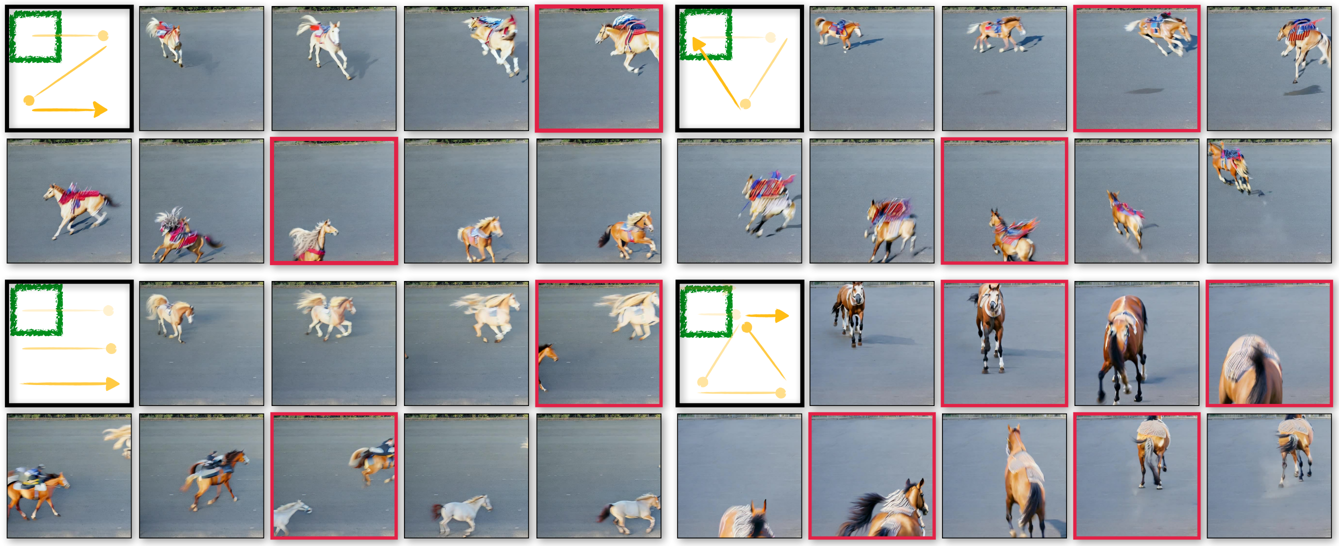

Figure 6: Irregular trajectory. The figure shows four synthesized video sequences with the horse subject directed according to the yellow arrows starting from the position indicated by green bbox. The frames highlighted in red correspond to keyframes. The start and end keyframes are not indicated. The prompt used for all examples: "A [horse] galloping on the road".

3.1 Attention manipulation functions

The code snippets in this section illustrate how TrailBlazer employs 2D Gaussian weights to generate the injection attention in the spatial and temporal attention editing steps.

```python
def get_weight_map(
    res: int, bbox_ratios: List[float], attention_probs: torch.tensor
) -> (torch.tensor, BoundingBox):
    """To get Gaussian weight map associated to specific dimension, normalized
    by the maximum of the current cross attention map

    Args:
        res(int): the resolution of this weight map
        bbox_ratios(List[float]): a list of four numbers for bbox
        attention_probs(torch.tensor): the cross attention map from the base class
    Returns:
        (torch.tensor, BoundingBox): the gaussian patch for injection and its bbox
    """
    bbox = BoundingBox(res, bbox_ratios)
    x = torch.linspace(0, bbox.height, bbox.height)
    y = torch.linspace(0, bbox.width, bbox.width)
    x, y = torch.meshgrid(x, y, indexing="ij")
    noise_patch = gaussian_2d(
        x,
        y,
        mx=int(bbox.height / 2.),
        my=int(bbox.width / 2.),
        sx=float(bbox.height),
        sy=float(bbox.width),
    )
    noise_patch.mul_(attention_probs.max())
    return noise_patch, bbox



def localized_spatial_weight_map(
    attention_probs_4d: torch.tensor, token_inds, bboxes_per_frame: List[List[float]]
):
    """Using Gaussian 2d distribution to generate weight patch as described in
    equation (1), (2) in our main text.

    Args:
        attention_probs_4d(torch.tensor):
            the cross attention in the shape of (#heads, height, width, 77)
        token_inds(List[int]):
            the set of prompt word and trailing indices from user
        bboxes_per_frame(List[float]):
            a list of bbox ratios at frame
    Returns:
        (torch.tensor): the weight map for spatial injection

    """
    dim = int(attention_probs_4d.size()[1])
    weight_map = torch.zeros_like(attention_probs_4d).half()
    attn_head_size = attention_probs_4d.shape[0] // len(bboxes_per_frame)
    for i in range(len(bboxes_per_frame)):
        weight_patch = (
            get_weight_map(
                res=dim,
                bbox_ratios=bboxes_per_frame[i],
                attention_probs=attention_probs_4d,
            )
            # to match dimension
            .unsqueeze(0)
            .unsqueeze(-1)
            .repeat(attn_head_size, 1, 1, len(token_inds))
        )
        b_idx = attn_head_size * i
        e_idx = attn_head_size * (i + 1)
        bbox.sliced_tensor_in_bbox(weight_map)[
            b_idx:e_idx, ..., token_inds
        ] = weight_patch
    return weight_map


def localized_temporal_weight_map(
    attention_probs_5d: torch.tensor, bboxes_per_frame: List[List[float]]
):
    """To generate the Gaussian weight map matching the temporal attention
    dimensions. See Sm(x,y) in equation (3) in TrailBlazer main text.

    Args:
        attention_probs_5d(torch.tensor):
            the cross attention in the shape of (
                #heads, height, width, #frames, #frames)
        bboxes_per_frame(List[float]):
            a list of four numbers determining the bbox corners
    """
    dim = int(attention_probs_5d.size()[1])
    f = attention_probs_5d.shape[-1]
    max_val = attention_probs_5d.max()
    weight_map = torch.zeros_like(attention_probs_5d).half()
    def get_patch(
        i: int, j: int, bbox_at_frame: List[float], bboxes_per_frame: List[List[float]]
    ):
        """To calculate the weight patch with distance function described in
        Sm(x,y) in the paper
        Args:
            i(int): frame index
            j(int): frame index
            bbox_at_frame(List[float]): list of four numbers for bbox
            bboxes_per_frame(List[List[float]]): all bboxes per frame
        Returns:
            (torch.tensor): the weight map for temporal injection
        """
        weight_map = (
            get_weight_map(
                res=dim,
                bbox_ratios=bboxes_per_frame,
                attention_probs=attention_probs_5d,
            )
            # NOTE: to match dimension at layer
            .unsqueeze(0).repeat(attention_probs_5d.shape[0], 1, 1)
        )
        noise_patch.mul_(attention_probs_5d.max())
        inv_noise_patch = noise_patch - noise_patch.max()
        # NOTE: distance is large when
        dist = (float(abs(j - i))) / len(bboxes_per_frame)
        final_patch = inv_noise_patch * dist + noise_patch * (1.0 - dist)
        return final_patch, bbox
    # NOTE: To form the weight patch for each pair of bboxes of frames
    for j in range(len(bboxes_per_frame)):
        for i in range(len(bboxes_per_frame)):
            patch_i, bbox_i = get_patch(bboxes_per_frame[i], i, j, bboxes_per_frame)
            patch_j, bbox_j = get_patch(bboxes_per_frame[j], i, j, bboxes_per_frame)
            bbox_i.sliced_tensor_in_bbox(weight_map)[..., i, j] = patch_i
            bbox_j.sliced_tensor_in_bbox(weight_map)[..., i, j] = patch_j

    return weight_map
```

3.2 Main routine

```python
class InjecterProcessor():
    """Implementation similar to
    diffusers.models.attention_processor.AttnProcessor from the open source
    project diffusers

    """
    def __init__(
        self,
        bundle: BundleType,
        bbox_per_frame: List[BoundingBox],
        strengthen_scale: float = 0.0,
        weaken_scale: float = 1.0,
    ):
        super().__init__(bundle)
        self.strengthen_scale = strengthen_scale
        self.weaken_scale = weaken_scale
        self.bundle = bundle
        self.bbox_per_frame = bbox_per_frame

    def __call__(
        self,
        attn: diffusers.models.attention_processor.Attention,
        hidden_states: torch.tensor,
        encoder_hidden_states: torch.tensor,
    ):
        """ The caller of attention processor
        Args:
            attn(Attention): the attention with
            hidden_states(torch.tensor): the latents from previous layer
            encoder_hidden_states(torch.tensor): text condition embeddings
        Returns:
        """
        batch_size, sequence_length, _ = hidden_states.shape
        # NOTE: To Q,K,V from latent and embeddings
        query = attn.to_q(hidden_states)
        key = attn.to_k(encoder_hidden_states)
        value = attn.to_v(encoder_hidden_states)
        query = attn.head_to_batch_dim(query)
        key = attn.head_to_batch_dim(key)
        value = attn.head_to_batch_dim(value)
        attention_probs = attn.get_attention_scores(query, key)

        # NOTE: To modify the spatial cross attention
        if (
            attention_probs.shape[-1] == CrossAttnProcessorBase.MAX_LEN_CLIP_TOKENS
            and self.use_spatial_editing
        ):
            dim = int(np.sqrt(attention_probs.shape[1]))
            attention_probs_4d = attention_probs.view(
                attention_probs.shape[0], dim, dim, attention_probs.shape[-1]
            )
            #
            attention_probs_4d = self.dd_core(attention_probs_4d)
            attention_probs = attention_probs_4d.reshape(
                attention_probs_4d.shape[0], dim * dim, attention_probs_4d.shape[-1]
            )
        # NOTE: To modify temporal cross frame attention
        elif attention_probs.shape[-1] == self.num_frames and (self.use_temporal_editing):
            dim = int(np.sqrt(attention_probs.shape[0] // attn.heads))
            attention_probs_5d = attention_probs.view(
                attn.heads, dim, dim, self.num_frames, self.num_frames,
            )
            attention_probs_5d = self.dd_core(attention_probs_5d)
            attention_probs = attention_probs_5d.view(
                (attn.heads * dim * dim, self.num_frames, self.num_frames),
        )

        hidden_states = torch.bmm(attention_probs, value)
        hidden_states = attn.batch_to_head_dim(hidden_states)
        # linear projection
        hidden_states = attn.to_out[0](hidden_states)
        # dropout
        hidden_states = attn.to_out[1](hidden_states)
        return hidden_states

    def dd_core(self, attention_probs: torch.Tensor):
        """
        Args:
            attention_probs(torch.tensor): A tensor either in 4d or 5d to
                trigger spatial or temporal attention editing, respectively.
        Returns:
            (torch.tensor): modified attention map

        """
        attention_probs_modified = attention_probs.detach().clone()

        # NOTE: Spatial cross attention editing
        if len(attention_probs.size()) == 4:
            # NOTE: token_inds is for specific prompt word
            token_inds = self.bundle.get("token_inds")
            # NOTE: trailing_inds is generated by the length determined by user
            trailing_length = self.bundle.get("trailing_length")
            trailing_inds = list(
                range(self.len_prompt + 1, self.len_prompt + trailing_length + 1)
            )
            all_tokens_inds = list(set(token_inds).union(set(trailing_inds)))
            # NOTE: To generate stengthen mask Ss(x,y) in Equation (1)
            strengthen_map = localized_weight_map(
                attention_probs_modified,
                token_inds=all_tokens_inds,
                bbox_per_frame=self.bbox_per_frame,
            )
            # NOTE: To generate weakening mask Ws(x,y) in Equation (1)
            weaken_map = torch.ones_like(strengthen_map)
            zero_indices = torch.where(strengthen_map == 0)
            weaken_map[zero_indices] = self.weaken_scale
            # NOTE: weakening in Equation (2)
            attention_probs_modified[..., all_tokens_inds] *= weaken_map[
                ..., all_tokens_inds
            ]
            # NOTE: strengthen in Equation (2)
            attention_probs_modified[..., all_tokens_inds] += (
                self.strengthen_scale * strengthen_map[..., all_tokens_inds]
            )
        # NOTE: Temporal cross attention editing
        elif len(attention_probs.size()) == 5:
            # NOTE: To generate stengthen mask in Equation (3)
            strengthen_map = localized_temporal_weight_map(
                attention_probs_modified,
                bbox_per_frame=self.bbox_per_frame,
            )
            # NOTE: To generate weakening mask as in spatial attention
            weaken_map = torch.ones_like(strengthen_map)
            zero_indices = torch.where(strengthen_map == 0)
            weaken_map[zero_indices] = self.weaken_scale
            # NOTE: weakening in Equation (3)
            attention_probs_modified *= weaken_map
            # NOTE: strengthen in Equation (3)
            attention_probs_modified += self.strengthen_scale * strengthen_map

        return attention_probs_modified
```

REFERENCES

[1] cerspense. 2023. zeroscope-v2-576w. https://huggingface.co/cerspense/zeroscope-v2-576w Accessed: 2023-10-01.

[2] Huggingface. 2023. Stable Diffusion 1 Demo. https://huggingface.co/spaces/stabilityai/stable-diffusion-1 Accessed: 2023-01-01.

[3] Yash Jain, Anshul Nasery, Vibhav Vineet, and Harkirat Behl. 2023. PEEKABOO: Interactive Video Generation via Masked-Diffusion. arXiv:2312.07509 [cs.CV]

[4] Zhengxiong Luo, Dayou Chen, Yingya Zhang, Yan Huang, Liang Wang, Yujun Shen, Deli Zhao, Jingren Zhou, and Tieniu Tan. 2023. VideoFusion: Decomposed Diffusion Models for High-Quality Video Generation. arXiv:2303.08320 [cs.CV]

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG]

[6] Gergely Szabó and András Horváth. 2021. Mitigating the Bias of Centered Objects in Common Datasets. *CoRR* abs/2112.09195 (2021). arXiv:2112.09195 https://arxiv.org/abs/2112.09195

[7] Xiang Wang, Hangjie Yuan, Shiwei Zhang, Dayou Chen, Jiuniu Wang, Yingya Zhang, Yujun Shen, Deli Zhao, and Jingren Zhou. 2023. VideoComposer: Compositional Video Synthesis with Motion Controllability. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 7594–7611. https://proceedings.neurips.cc/paper_files/paper/2023/file/180f6184a3458fa19c28c5483bc61877-Paper-Conference.pdf

[8] Zhouxia Wang, Ziyang Yuan, Xintao Wang, Yaowei Li, Tianshui Chen, Menghan Xia, Ping Luo, and Ying Shan. 2024. MotionCtrl: A Unified and Flexible Motion Controller for Video Generation. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 114, 11 pages. https://doi.org/10.1145/3641519.3657518