Decision Diffuser

"Decision Diffuser"的工作基于两个前置工作

1. **Diffuser**：旨在通过轨迹规划解决离线强化学习问题，使用一个带有时间注意力机制的一维UNet来规划轨迹。Diffuser能够根据预定的最终状态来规划轨迹。
2. **Compositional Visual Generation with Composable Diffusion Models**：提出了一种结合Classifier-Free条件的方法，以实现条件的叠加。、

Du Yilin及其导师参与了这两个前置工作，将这两项研究结合起来是一个自然的发展。

这篇论文主要有两个贡献：

1. **结合Diffuser与Classifier-Free Guidance**：通过使用Diffusion Model规划路径，Decision Diffuser引入Classifier-Free Guidance，从而实现符合特定要求的路径规划。
2. **改进Diffuser的动作规划**：Diffuser通过Diffusion Model预测未来的状态和动作，而Decision Diffuser在此基础上做出了创新——仅用Diffusion Model规划未来状态，动作规划则由另一个Action Inverse Model完成。

然而，这个项目也面临着几个挑战：

- 核心代码未公开：Decision Diffuser扩展了Diffuser的代码，但关键部分，特别是技能组合的实现细节并没有公开。公开的代码也不支持技能组合。
- 数据集未公开：作者并未公开他们使用的Offline Dataset，需要进行手动收集。
- 训练难度和时间：Diffusion Model难以训练，原始代码在GeForce GTX 1080显卡上需要数天的训练时间。
- 依赖性问题：随着时间的推移，某些依赖组件失去支持，需要手动调整依赖库代码。

针对上述挑战，我作出了以下贡献：

1. **实现组合技能功能**：通过研究Compositional Visual Generation论文和数学推理，我实现了Decision Diffuser的组合技能核心功能，并进行了演示。我的代码支持使用技能组合和最终状态作为条件。
2. **训练时长问题**原代码使用同一个优化器来训练Action Inverse Model和Diffusion Model。Action Inverse Model在轨迹上进行模仿学习，而Diffusion做生成式降噪。而这两个模型的学习速度和目标存在差异。我用不同的优化器对两个模型训练，显著提高了训练效率，缩短到了几个小时，使调参成为可能。

3. **在两个环境下实现训练收敛**：我成功地在倒立摆和CondCircle环境下实现了训练收敛。
4. **解决数据集问题**：我设计了名为CondCircle的强化学习环境，该环境要求多重限制的组合。我训练了PPO算法作为专家，并收集了约6000条数据。

CondCircle环境中的数据准备

在CondCircle环境中，智能体根据其与中心点在特定半径内的接近程度获得奖励。本研究涉及训练两个具有不同目标的智能体：一个旨在达到1.5半径内，另一个旨在超出1.0半径。这两个策略是分开训练的，没有进行联合训练。

数据集用于Decision Diffuser。目标是创建走向在1.5半径圆内和1.0半径圆外的轨迹。

结果

环境1倒立摆

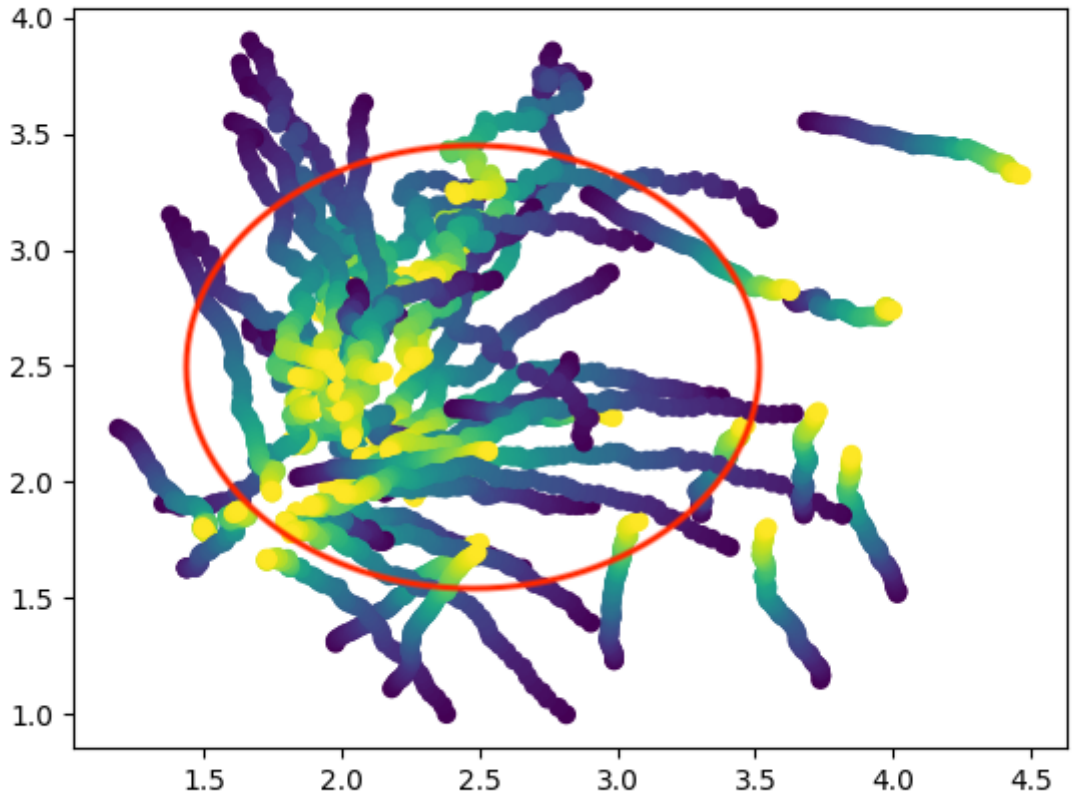
- 在倒立摆环境中实现了收敛，达到最优控制。
- 数据集从GitHub获取。

环境2 CondCircle

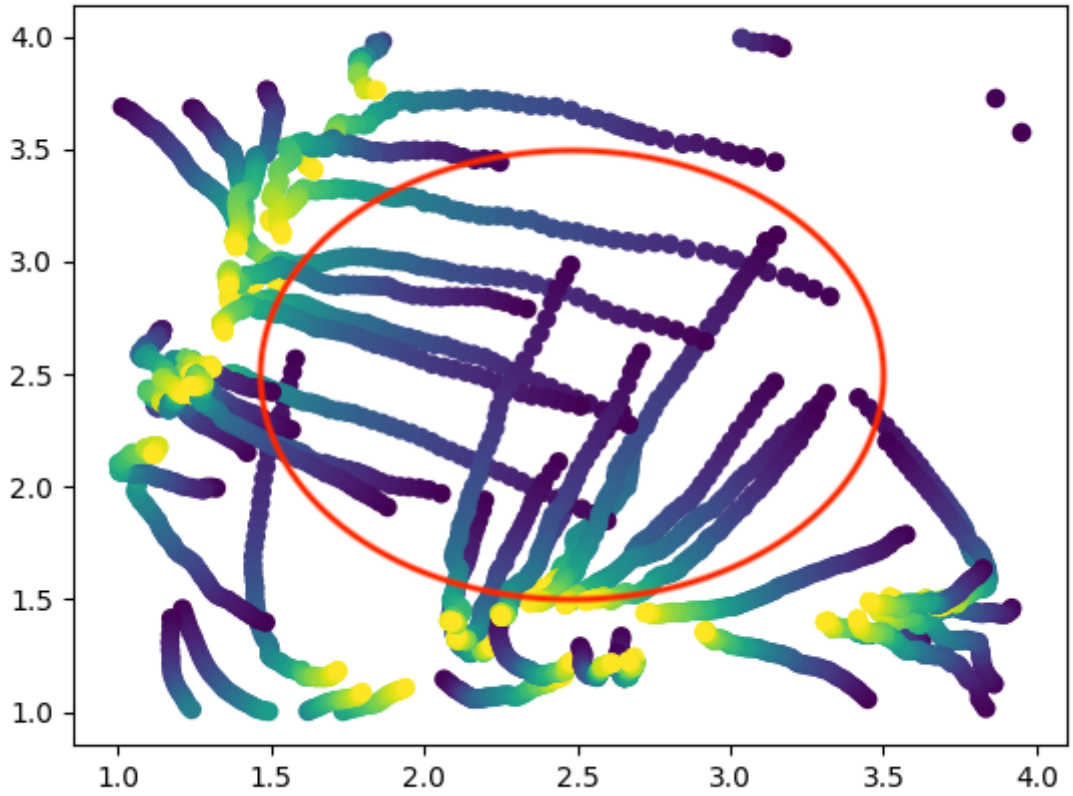
- 成功模仿满足特定条件的专家轨迹。
- 用 Classifier Free Guidance 的组合生成新的OOD轨迹。

CondCircle 可视化结果

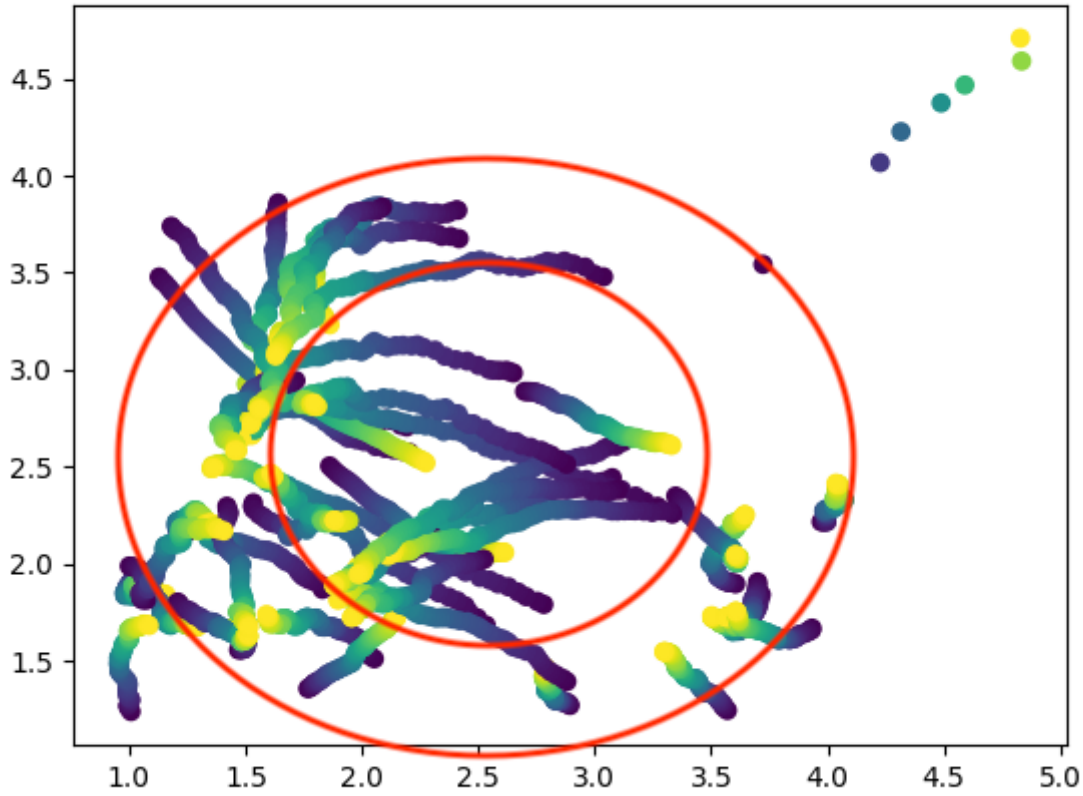
- 图像1：朝向半径1.5圆内部（中心在[2.5, 2.5]）的轨迹计划。



- 图像2：朝向半径1.0圆外部（中心在[2.5, 2.5]）的轨迹计划。



- 图像3：融合条件1和条件2生成的轨迹，这些轨迹融合了两种的特点结合了两种条件的轨迹，轨迹要么落在环上，要么在前往圆环的路上。



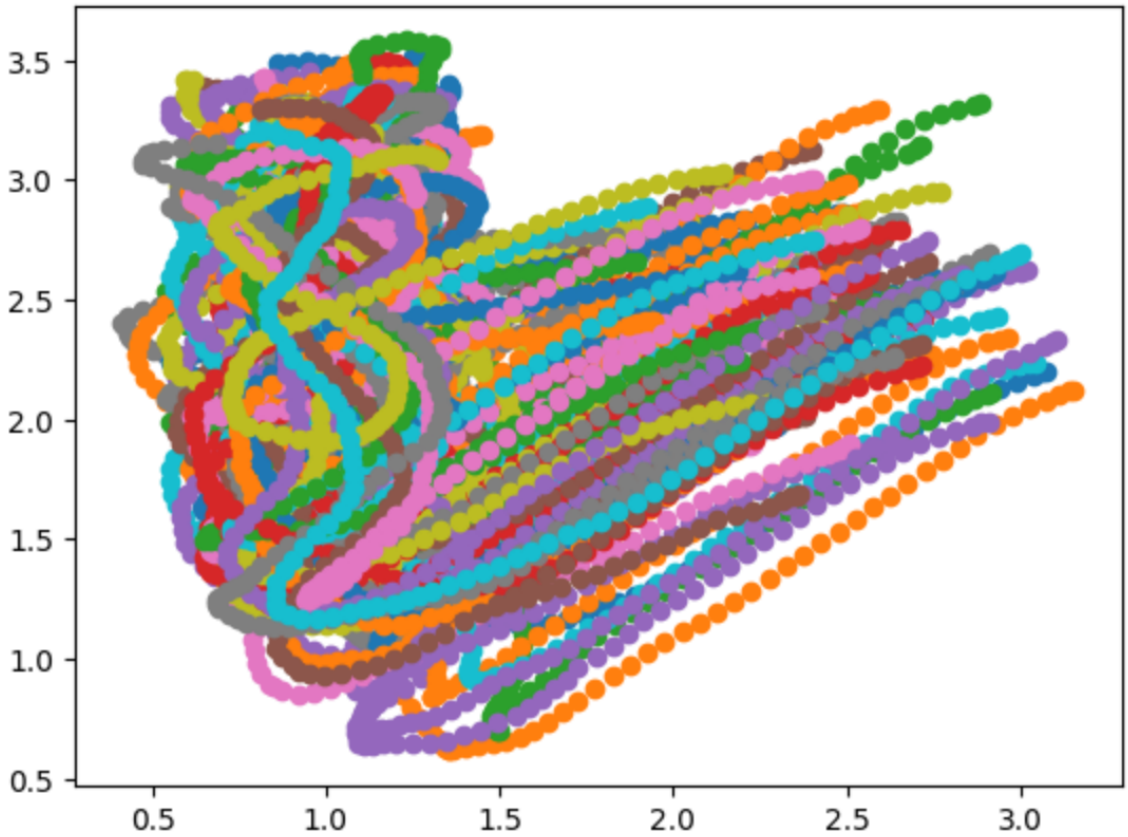
但是由于专家轨迹较为单一，生成的轨迹不是最优的。如果能有更多时间收集更加多样化的专家轨迹，调整guidance 强度，效果可以更好。

在实现现有代码的同时，我还克服了以下困难：

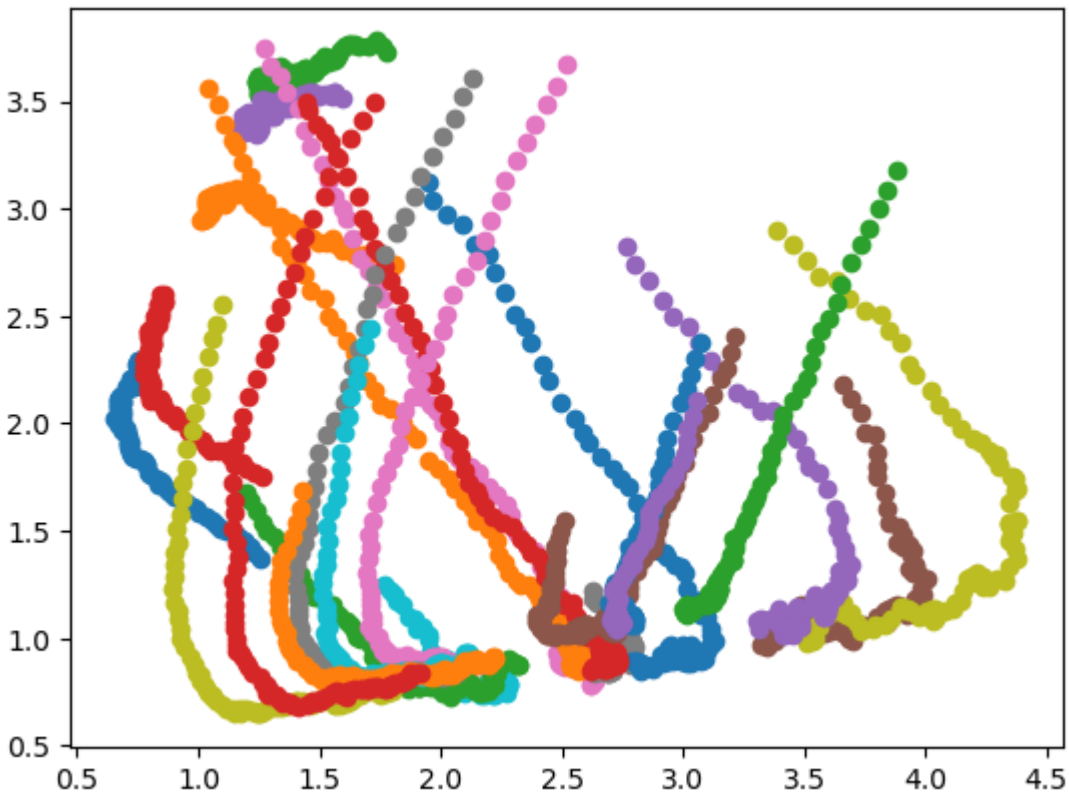
收集专家数据集有难点，一开始我使用单个专家收集轨迹数据集，因为轨迹过于单一，Diffusion Model的泛化性能很差，训练时loss收敛后测试时效果很差。以上的结果是重新收集多专家的轨迹训练的

我修改了强化学习环境，强迫智能体探索更大的空间，同时我训练了多个专家收集轨迹，因为时间原因对于每个条件我训练了三个专家，收集多样化的轨迹，如果时间更多，更多专家会得到更好的效果

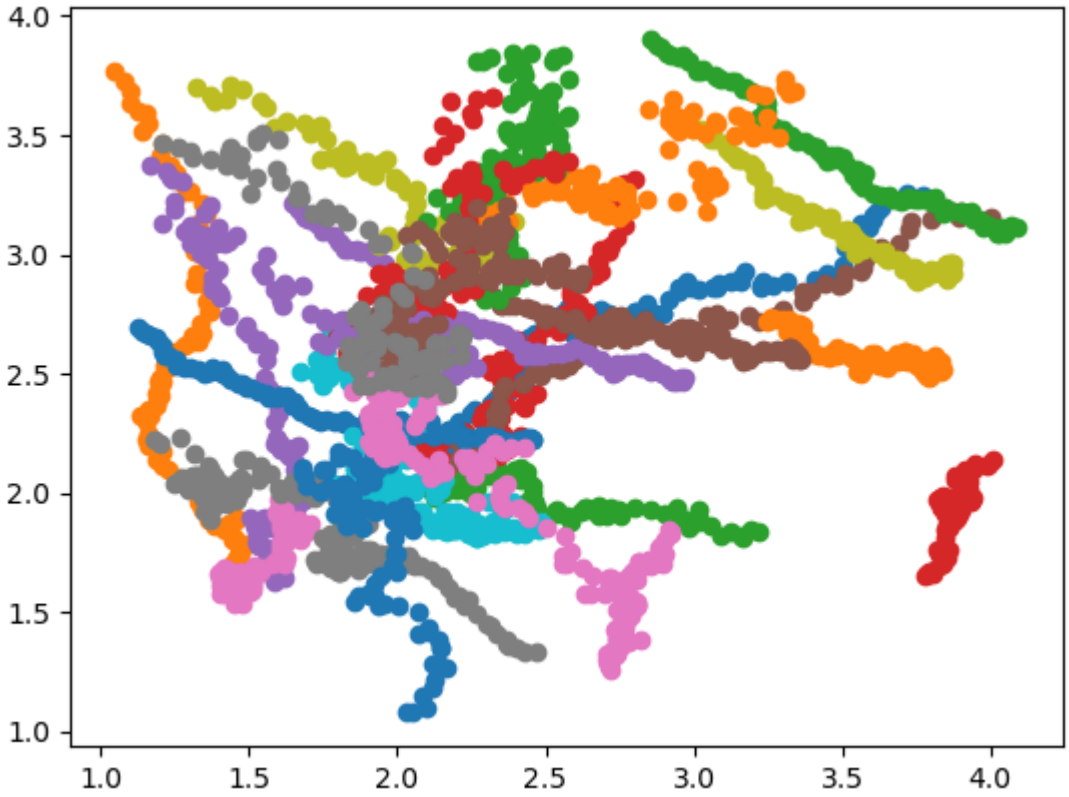
一开始的智能体使用加速度控制，但是训练时间比较长。为了加快进度，后来改为了速度控制。对于演示技能组合没有太多影响


3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.5
1.0
1.5
2.0
2.5
3.0

单一专家生成的轨迹


3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5

多专家生成的轨迹，离开1.0



多专家生成的轨迹，进入1.5

未来工作：

1. 做更难任务的技能组合，目前在简单的 CondCircle 环境下做了实验，然而这个实验比较简单。之后的目标是找一个更困难的任务比如机器人行走上做实验
2. 更精细的调参，因为时间限制，多限制\技能组合的效果还是不太理想。可以调整guidance的权重
3. 需要收集更加多元化的数据，目前算法不是最优轨迹，也不够多样
4. 重构代码，因为时间限制，有一些代码沿用了 Decision Diffuser 的代码
5. 增加测试代码和文档，目前还没有进行测试，文档目前有不匹配的地方
6. 因为Decision Diffuser 的训练输入、输出都和其他代码不太一样，需要增加 Dataset Builder 和 适配器等

代码文档

omnisafe/algorithms/offline/decision_diffuser.py

主代码，实现了 agent 的训练功能。因为需要整个轨迹作为训练输入，所以和其他 offline rl 的训练方式和数据集不同。override 了 `_train` 和 `learn` 方法。训练使用 `SequenceDataset`。代码采用了 EMA 训练，也就是对模型训练的权重求滚动平均值，这样可以增强稳定性

omnisafe/common/offline/sequence_dataset.py

数据集的实现参考了 https://github.com/anuragajay/decision-diffuser/tree/main/code

GitHub - zhangzuyuan/decision-diffuser-simple-code

omnisafe/envs/legacy_env.py

实现了到 gym （不是 gymnasium）的适配器，可以加载我定义的 CondCircle 环境

omnisafe/models/actor/decision_diffuser_actor.py

实现了Actor类，主要是 predict 功能。predict 代码生成一个符合条件的轨迹规划，然后调用 Inverse Action Model 做下一步的动作预测

omnisafe/models/diffuser/diffusion.py

decision diffuser 算法，从 Decision Diffuser 那里修改来的

```
    def p_mean_variance(self, x, state_condition, t, cls_free_condition_list):
        if self.perform_cls_free_condition:
            # epsilon could be epsilon or x0 itself
            epsilon_uncond = self.model(
                x,
                state_condition,
                t,
                cls_free_condition_list[0],
                force_dropout=True,
            )
            epsilon = epsilon_uncond
            for cls_free_condition in cls_free_condition_list:
                epsilon_cond = self.model(
                    x,
                    state_condition,
                    t,
                    cls_free_condition,
                    use_dropout=False,
                )
                epsilon += self.cls_free_condition_guidance_w * (epsilon_cond -
epsilon_uncond)
        else:
            epsilon = self.model(x, state_condition, t)
```

这一部分是核心的修改，支持多个 Classier Free Condition的组合

参考：

https://github.com/LinXueyuanStdio/PyTorch-DDPM/blob/main/ClassifierFreeDDPM.ipynb

https://github.com/anuragajay/decision-diffuser/tree/main/code

GitHub - zhangzuyuan/decision-diffuser-simple-code