# RDT (Robotics Diffusion Transformer) 说明文档(结合部分源码)

## 目录



## 1. 模型概述

RDT (Robotics Diffusion Transformer) 是一个专门为机器人控制任务设计的生成式模型，结合了**扩散模型**和**Transformer架构**的优势。

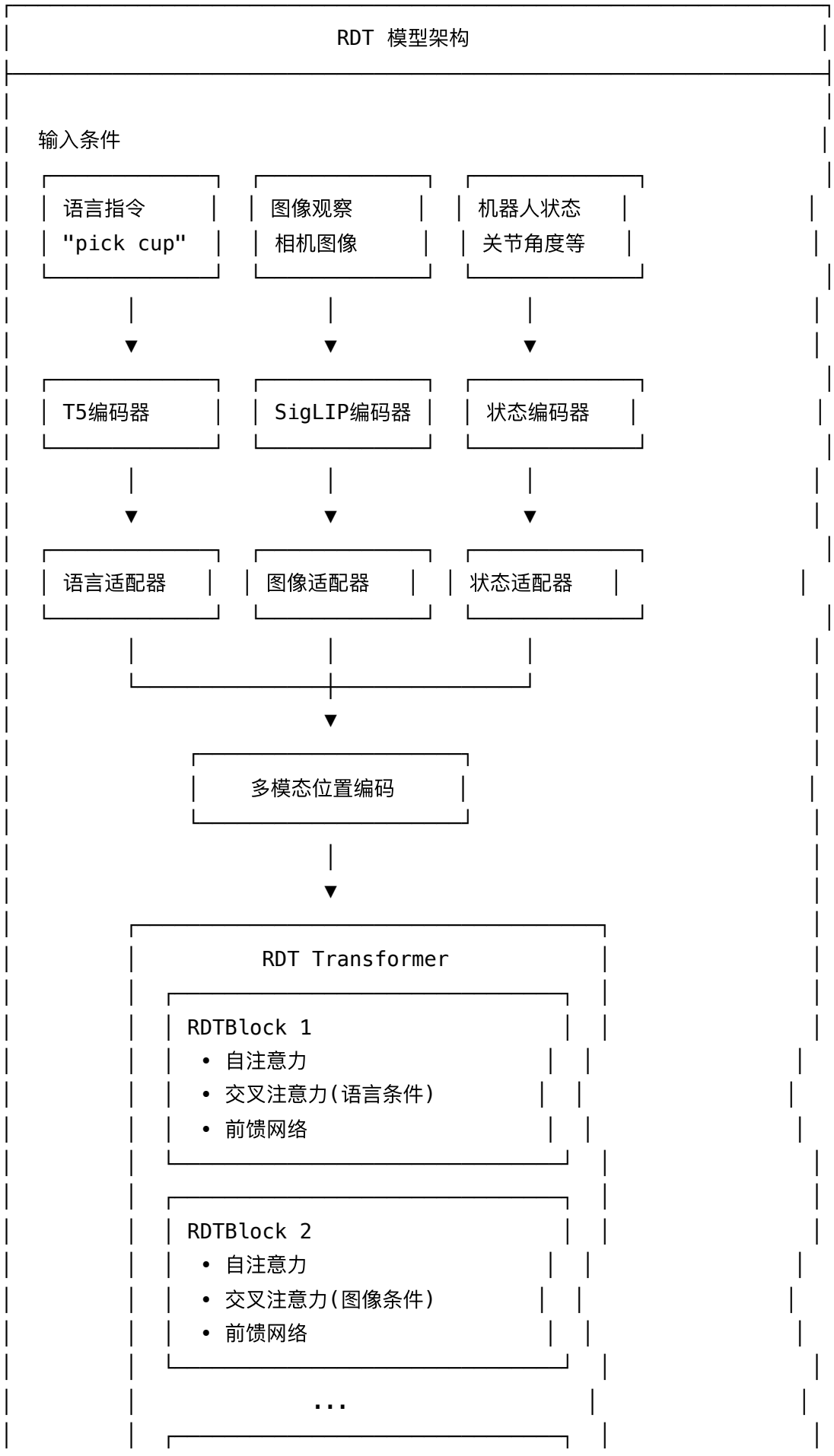
### 1.1 核心思想

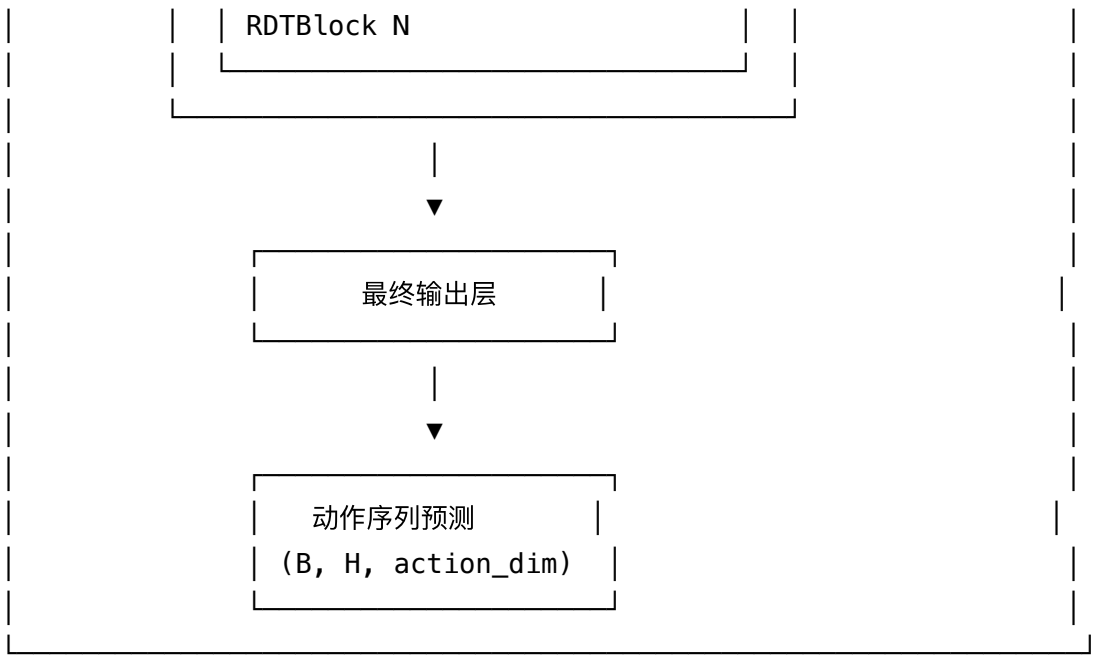
- **问题建模**：将机器人动作序列生成视为一个条件扩散过程
- **输入条件**：语言指令 + 视觉观察 + 当前状态
- **输出目标**：未来H步的动作序列
- **生成方式**：通过迭代去噪过程从随机噪声中生成连贯的动作序列

### 1.2 主要优势

1. **多模态融合**：统一处理语言、视觉和状态信息
2. **序列建模**：生成时间一致的动作序列
3. **条件生成**：根据具体任务需求生成相应动作
4. **扩展性强**：支持不同模态的灵活配置

# 2. 整体架构


RDT 模型架构
输入条件
语言指令
"pick cup"
图像观察
相机图像
机器人状态
关节角度等
T5编码器
SigLIP编码器
状态编码器
语言适配器
图像适配器
状态适配器
多模态位置编码
RDT Transformer
RDTBlock 1
• 自注意力
• 交叉注意力(语言条件)
• 前馈网络
RDTBlock 2
• 自注意力
• 交叉注意力(图像条件)
• 前馈网络
...




# 3. 核心组件详解

## 1. RDTRunner（模型运行器）

**职责**：整合所有组件，处理训练和推理逻辑

**关键子组件**：

- **条件适配器**：将不同模态特征映射到统一空间(hidden_size)
  - `lang_adaptor`：语言特征适配器
  - `img_adaptor`：图像特征适配器
  - `state_adaptor`：状态特征适配器
- **噪声调度器**：管理扩散过程
  - `noise_scheduler`：训练时的DDPM调度器
  - `noise_scheduler_sample`：推理时的DPM求解器

## 2. RDT（核心变换器）

**职责**：序列到序列的变换，核心推理引擎

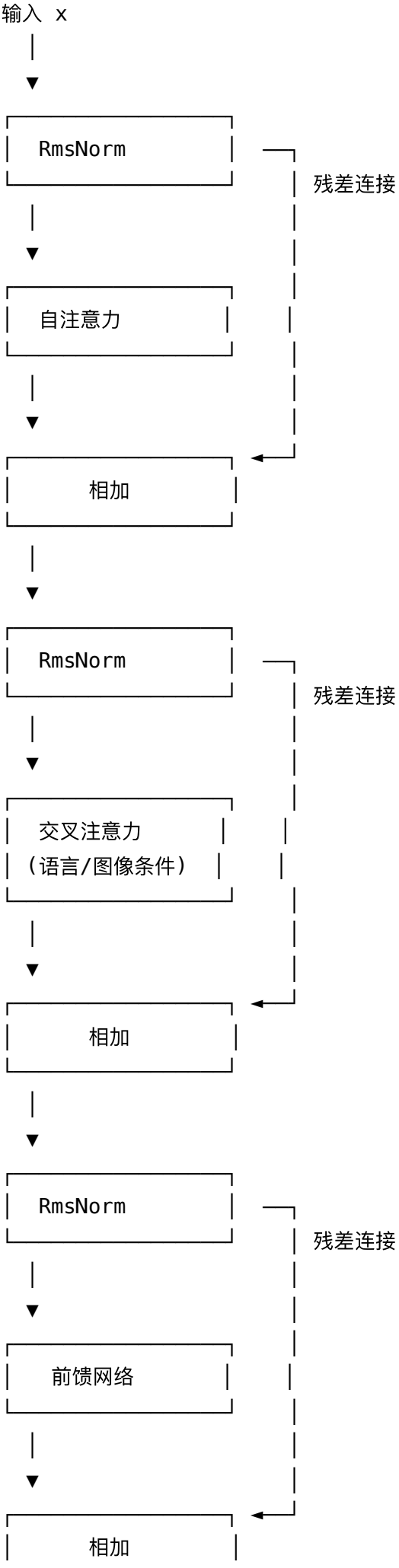
**输入序列结构**：

```
[时间步嵌入] [控制频率嵌入] [当前状态] [动作序列1] [动作序列2] ... [动作序列H]
     1D          1D           1D         1D         1D              1D
```

**变换器层数**：28层（可配置）
**隐藏维度**：1152（可配置）
**注意力头数**：16（可配置）

## 3. RDTBlock（变换器基本块）

每个RDTBlock包含三个子层：


输入 x
RmsNorm
残差连接
自注意力
相加
RmsNorm
残差连接
交叉注意力
（语言/图像条件）
相加
RmsNorm
残差连接
前馈网络
相加

```
└──────────────────┘
   |
   ▼
输出 x
```

**层交替策略**：

- 偶数层（0,2,4...）：交叉注意力使用语言条件
- 奇数层（1,3,5...）：交叉注意力使用图像条件

# 4. 嵌入和编码

## 4.1 序列结构：

```
Token类型:  [时间步] [控制频率] [状态] [动作1] [动作2] ... [动作H]
位置索引:      0         1        2      3      4          H+2
```

1. 嵌入生成阶段：系统将时间步t和控制频率freq通过 正余弦位置编码 + 专门的嵌入层或MLP 转换为高维向量表示，即从一个常量转为一个可以训练的嵌入向量，嵌入向量维度为 hidden_dim， 同时处理其他输入模态（如状态信息、动作序列），维度转为 hidden_size， 准备下一步的拼接。
2. 序列构建阶段：将所有模态的嵌入向量按照预定义的顺序进行拼接，形成一个统一的多模态token序列。这个序列的结构为：[timestep_token, ctrl_freq_token, state_token, action_token_1, action_token_2, ..., action_token_horizon]。 得到初步的输入 x 。
3. 位置编码注入阶段：使用多模态条件位置编码函数为整个序列生成位置编码矩阵，该矩阵同时包含模态身份信息和序列内位置信息。然后通过元素级加法将位置编码与token表示相加，确保每个token都携带完整的位置和模态信息 。 x = x + x_pos_embed

```
# 获取多模态条件位置编码
x_pos_embed = get_multimodal_cond_pos_embed(
    embed_dim=hidden_size,
    mm_cond_lens=OrderedDict([
        ('timestep', 1),        # 时间步：1个token
        ('ctrl_freq', 1),       # 控制频率：1个token
        ('state', 1),           # 当前状态：1个token
        ('action', horizon),    # 动作序列：H个token
    ])
)
```

## 4.2. 多模态位置编码设计理念

同时考虑模态身份编码(时间步t/控制频率freq/状态信息state/动作序列actions)和模态内位置编码(pos)

```
位置编码 = 模态身份编码 + 模态内位置编码
```

原始的位置编码维度和嵌入维度相同，为 hidden_dim ， 现在分两种情况考虑：

1. 不采用模态编码，则位置编码维度(pos_dim)为 hidden_dim
2. 采用模态内位置编码，此时取 hiddden_dim 的前半部分，作为模态编码，区分模态；后半部分作为位置编码。四种模态，每一种的所有 tokens 采用相同的模态编码。

```
num_modalities = len(mm_cond_lens)

# 初始化模态位置嵌入
modality_pos_embed = np.zeros((num_modalities, embed_dim))

if embed_modality:
    # 采用模态编码，获取各种模态的嵌入（放在前半部分）
    modality_sincos_embed = get_1d_sincos_pos_embed_from_grid(
        embed_dim // 2,
        torch.arange(num_modalities)
    )
    modality_pos_embed[:, :embed_dim // 2] = modality_sincos_embed
    # 后半部分用于位置嵌入
    pos_embed_dim = embed_dim // 2
else:
    # 不采用模态编码，整个嵌入都用于位置嵌入
    pos_embed_dim = embed_dim
```

## 4.3 位置编码

**正弦-余弦编码公式**：

```
PE(pos, 2i)   = sin(pos / 10000^(2i/d))     # 偶数维度用正弦
PE(pos, 2i+1) = cos(pos / 10000^(2i/d))     # 奇数维度用余弦
```

**优势**：

- **唯一性**：每个位置都有唯一编码
- **相对性**：相邻位置的编码相似
- **外推性**：可以处理训练时未见过的位置

- **多尺度**：包含不同频率的位置信息

# 5. 其他多模态处理

## 5.1 条件适配器系统

**设计目标**：将不同模态的特征映射到统一的隐藏空间

```
# 语言适配器
lang_adaptor: MLP(lang_token_dim → hidden_size)

# 图像适配器
img_adaptor: MLP(img_token_dim → hidden_size)

# 状态适配器
state_adaptor: MLP(state_token_dim * 2 → hidden_size)  # *2 因为包含状态+掩码
```

## 5.2 交叉注意力机制

**核心创新**：让动作序列"查询"条件信息

```
# 在每个RDTBlock中
def forward(x, condition, mask):
    # x: 主序列(动作序列)
    # condition: 条件序列(语言或图像)
    # mask: 注意力掩码

    Q = Linear_q(x)               # Query来自主序列
    K = Linear_k(condition)       # Key来自条件序列
    V = Linear_v(condition)       # Value来自条件序列

    attention = softmax(Q @ K.T / sqrt(d))
    output = attention @ V
```

**交替注意力策略**：

- Layer 0, 2, 4, ... → 关注语言条件
- Layer 1, 3, 5, ... → 关注图像条件
- 渐进式多模态信息整合

## 5.3 掩码处理机制

**语言掩码**：处理变长序列

```
lang_mask[i] = True   # 有效token
lang_mask[i] = False  # 填充token
```

**动作掩码**：指示有效动作维度

```
action_mask[i] = 1.0  # 有效动作维度
action_mask[i] = 0.0  # 无效动作维度
```

# 6. 训练流程

## 6.1 扩散前向过程

**目标**：学习从噪声恢复动作序列

```
def compute_loss(self, ...):
    # 步骤1：采样随机噪声
    noise = torch.randn(action_gt.shape)

    # 步骤2：随机采样时间步
    timesteps = torch.randint(0, num_train_timesteps, (batch_size,))

    # 步骤3：向真实动作添加噪声
    noisy_action = noise_scheduler.add_noise(action_gt, noise, timesteps)

    # 步骤4：预测噪声或原始动作
    pred = model(noisy_action, conditions...)

    # 步骤5：计算损失
    if prediction_type == 'epsilon':
        loss = MSE(pred, noise)      # 预测噪声
    elif prediction_type == 'sample':
        loss = MSE(pred, action_gt)  # 预测原始动作
```

## 6.2 训练数据流

输入数据：
├── 语言指令："pick up the red cup"
├── 图像序列：[img_t-1, img_t] × 3个相机
├── 当前状态：机器人关节角度、末端位置等
└── 目标动作：未来H步的动作序列

预处理：
├── 语言编码：T5编码器 → 语言tokens
├── 图像编码：SigLIP编码器 → 图像tokens
├── 状态处理：归一化 + 掩码
└── 动作处理：归一化 + 掩码

模型训练：
├── 条件适配：不同模态映射到统一空间
├── 位置编码：多模态位置信息注入
├── 扩散过程：加噪 → 预测 → 计算损失
└── 反向传播：更新模型参数

## 6.3 损失函数设计

**主要损失**：均方误差损失

```
loss = MSE(predicted_noise, actual_noise)
```

**优化策略**：

- 使用AdamW优化器
- 学习率预热 + 余弦衰减
- 梯度裁剪防止梯度爆炸
- EMA模型稳定训练

# 7. 推理流程

## 7.1 条件采样过程

**目标**：从随机噪声生成连贯动作序列

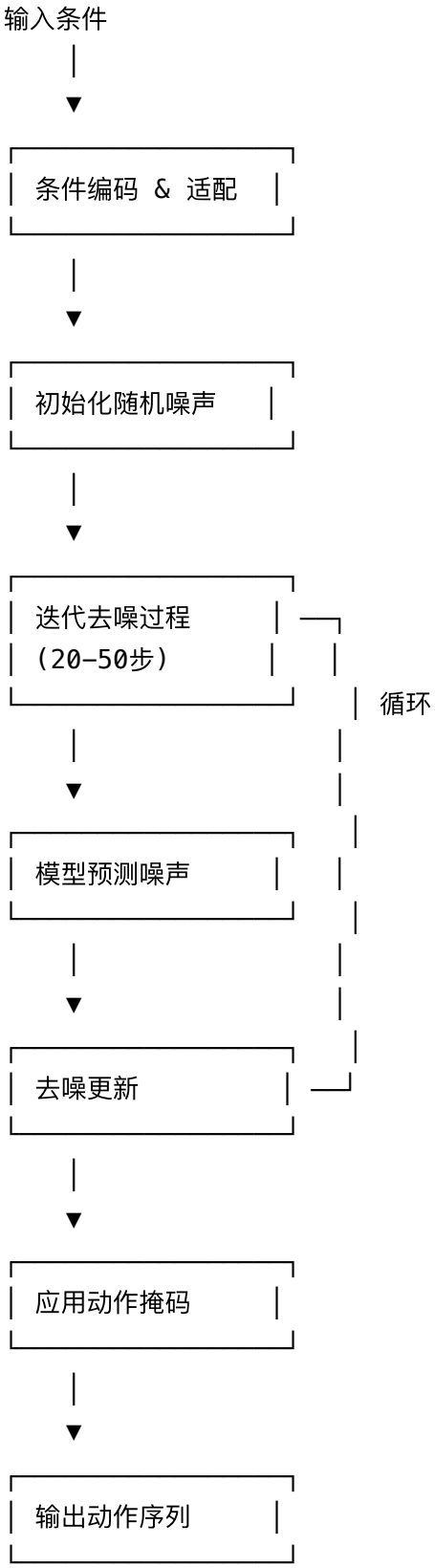
```
def conditional_sample(self, conditions...):
    # 步骤1：初始化随机噪声
    noisy_action = torch.randn(batch_size, horizon, action_dim)

    # 步骤2：设置推理时间步
    scheduler.set_timesteps(num_inference_timesteps)

    # 步骤3：迭代去噪
    for t in scheduler.timesteps:
        # 预测噪声
        model_output = model(noisy_action, t, conditions...)

        # 去噪步骤
        noisy_action = scheduler.step(model_output, t, noisy_action).prev_sample

    # 步骤4：应用动作掩码
    final_action = noisy_action * action_mask

    return final_action
```

## 7.2 推理时间步调度

**训练vs推理**：

- 训练：1000个时间步（DDPM）
- 推理：20-50个时间步（DPM-Solver）
- 推理加速：通过更高效的求解器实现

# 7.3 推理流程图


输入条件
条件编码 & 适配
初始化随机噪声
迭代去噪过程
(20–50步)
循环
模型预测噪声
去噪更新
应用动作掩码
输出动作序列