# Reconstruction of Flow Field

干雨杨

2024 年 5 月 20 日

## 目录

# 1 Training

### Structure

## 1.1 Configs

## 1.2 Dataset

**Data**: 40 sequences(36 Training, 4 Validating),

$T = 10s, \Delta t = 1/32s$,

**Get_item**: 三张 frame, 形成 3*channels,

$data.shape = batch \cdot 3 * resolution^2$

**Normalization**: 正态正则化

## 1.3 Model

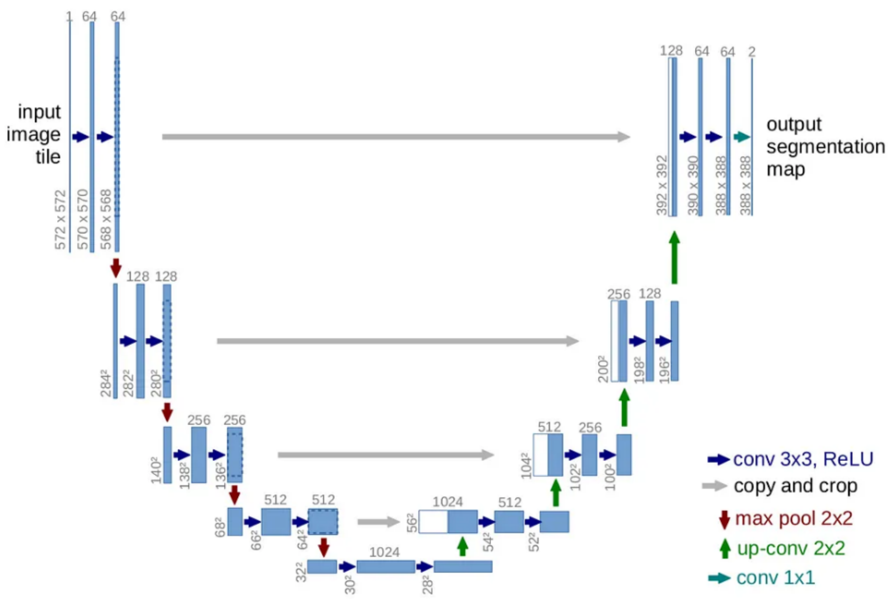


图 1: U-Net

### 1.3.1 DownSampling

降采样包括结构为 [Resnet, [Attention]] + [DownSample]

```
for i_level in range(self.num_resolutions):
    block = nn.ModuleList()
    attn = nn.ModuleList()
    block_in = ch*in_ch_mult[i_level]
    block_out = ch*ch_mult[i_level]
    for i_block in range(self.num_res_blocks):
        block.append(ResnetBlock(in_channels=block_in,
                                 out_channels=block_out,
                                 temb_channels=self.temb_ch,
                                 dropout=dropout))
        block_in = block_out
        if curr_res in attn_resolutions:
            attn.append(AttnBlock(block_in))
    down = nn.Module()
    down.block = block
    down.attn = attn
    if i_level != self.num_resolutions-1:
        down.downsample = Downsample(block_in, resamp_with_conv)
        curr_res = curr_res // 2
    self.down.append(down)
```

图 2: DownSampling

1. **Resnet Block**: Conv(x) + f(g(x) + h(t))
2. ch_mult: Downsample 时的通道变化
3. timestep_sample:

   [randint[0, timestep]] + [timestep - previous]

### 1.3.2 Mid

中间层包括 [Resnet, Attn, Resnet]

### 1.3.3 UpSampling

和 DownSampling 相同的结构, 加上了 Skip-Connection (直接 concat), 再通过 Conv 保持和降采样的 channel 一致

## 1.4 Time_embedding

参考了 Transformer 的位置编码

```
def get_timestep_embedding(timesteps, embedding_dim):
    """
    This matches the implementation in Denoising Diffusion Probabilistic Models:
    From Fairseq.
    Build sinusoidal embeddings.
    This matches the implementation in tensor2tensor, but differs slightly
    from the description in Section 3.5 of "Attention Is All You Need".
    """
    assert len(timesteps.shape) == 1

    half_dim = embedding_dim // 2
    emb = math.log(10000) / (half_dim - 1)
    emb = torch.exp(torch.arange(half_dim, dtype=torch.float32) * -emb)
    emb = emb.to(device=timesteps.device)
    emb = timesteps.float()[:, None] * emb[None, :]
    emb = torch.cat([torch.sin(emb), torch.cos(emb)], dim=1)
    if embedding_dim % 2 == 1:  # zero pad
        emb = torch.nn.functional.pad(emb, (0, 1, 0, 0))
    return emb
```

图 3: 时间编码

$$\vec{p_t} = \begin{cases} sin(\omega_k * t) \quad if \quad i = 2k \\ cos(\omega_k * t) \quad if \quad i = 2k+1 \end{cases} \tag{1}$$

间隔插入 / 直接拼接并无影响

shape: [time_step(batch)] $\rightarrow$ [time_step, channels]

## 1.5 Loss

### 1.5.1 Noise_Estimate_Loss

1. **Input:**

   $X = Weighted(X_0, e)$, $\hat{e} = Model(X, t)$

2. **L_2 loss** $Loss = \Sigma(e - \hat{e})^2$

### 1.5.2 Physical Guidance

1. Governing Equation(Imcompressible):

$$\frac{\partial \omega(\mathbf{x},t)}{\partial t} + \mathbf{u}(\mathbf{x},t)\cdot\nabla\omega(\mathbf{x},t) = \frac{1}{\mathrm{Re}}\nabla^2\omega(\mathbf{x},t) + f(\mathbf{x}), \quad \mathbf{x}\in(0,2\pi)^2, t\in(0,T] \tag{2}$$

2. Fourier Transform:

   对于中间的 channel

```
w_h = torch.fft.fft2(w[:, 1:-1], dim=[2, 3])
```

   定义波数

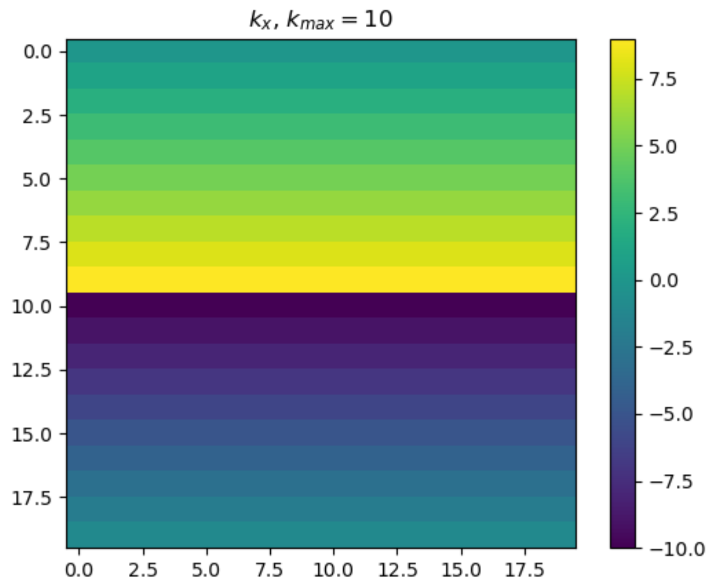


图 4: 波数示例

```
# wt = (w[:, 2:, :, :] - w[:, :-2, :, :]) / (2 * dt)
wt = (w[:, 2:, :, :] - w[:, :0, :, :]) / (2 * dt)
```

图 5: sth wrong

```
residual = wt + (advection - (1.0 / re) * wlap + 0.1*w[:, 1:
                                   -1]) - f
```

```
residual_loss = (residual**2).mean()
dw = torch.autograd.grad(residual_loss, w)[0]
```

## 1.6 Result

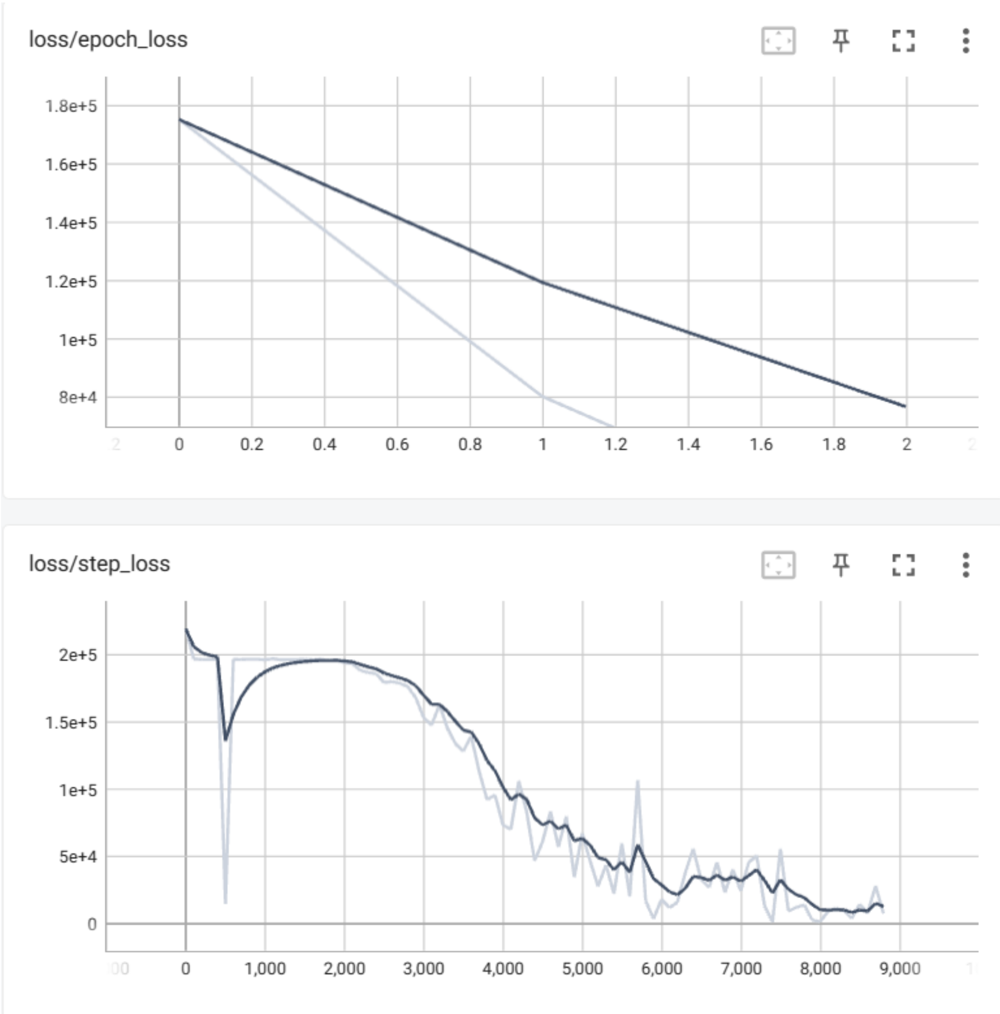


图 6: loss

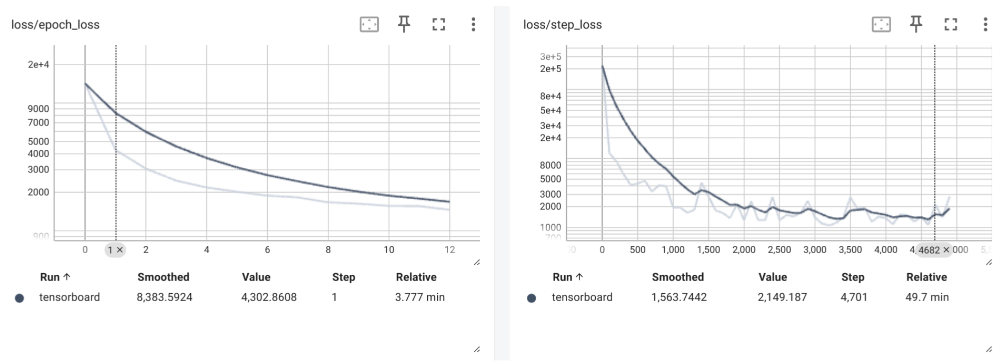


图 7: On A100

# 2 Sampling

## 2.1 Data component

1. ref_data: ground_truth
2. sampled_data: blured
3. $\alpha\_sequence$

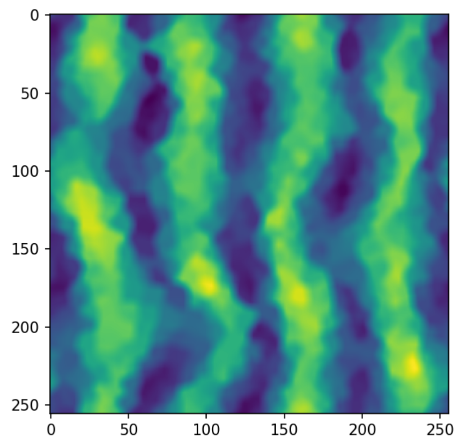

图 8: Gaussian Blur

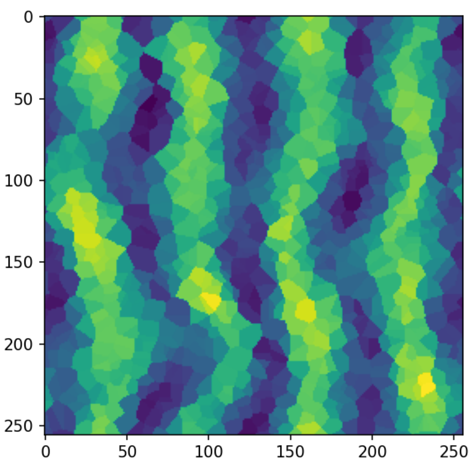

图 9: Original Sampled

图 10: Overall caption for both images.

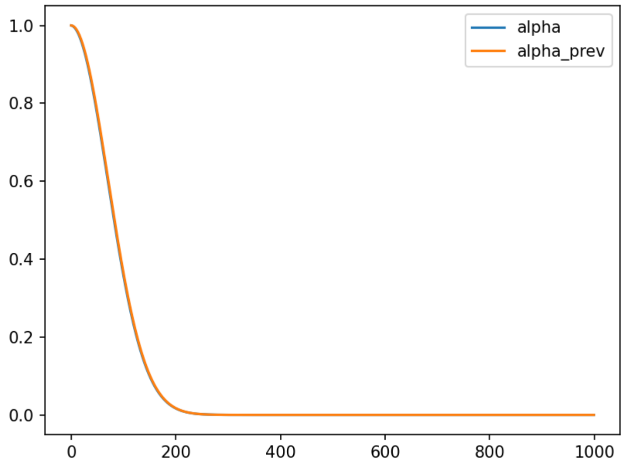


图 11: $\alpha\_sequence$

## 2.2 DDIM Process

1. Estimate noise:

```
et = (w+1)*model(xt, t, dx) - w*model(xt, t)
```

2. Denoise:

```
x0_t = (xt - et * (1 - at).sqrt()) / at.sqrt()
xt_next = at_next.sqrt() * x0_t + c2 * et - dx
```

## 2.3 Result

For 8800 steps(2 Epochs), conditional

For 32100 steps(10 Epochs), conditional

For with Attn at bottleneck:

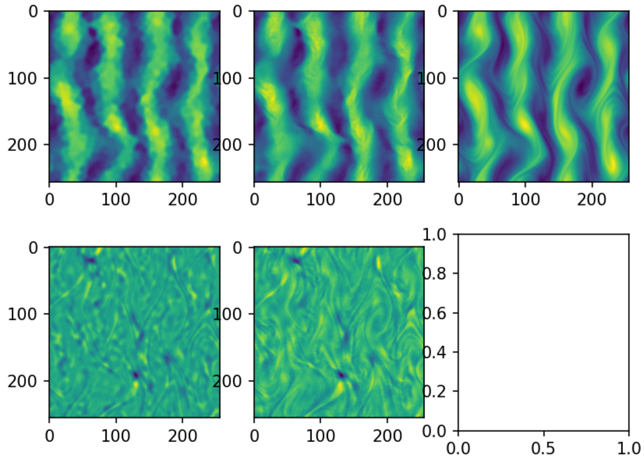


图 12: 2 epochs

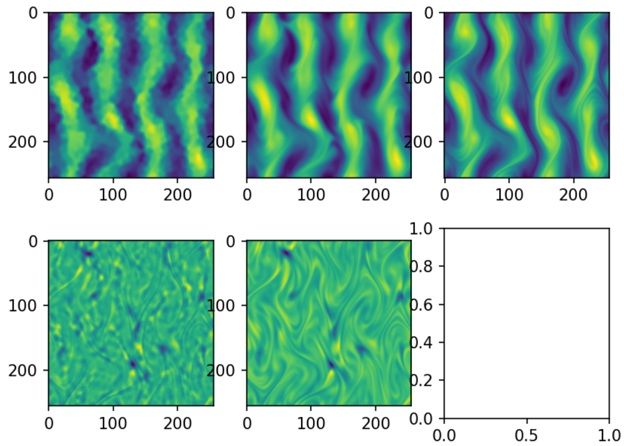


图 13: 10 epochs

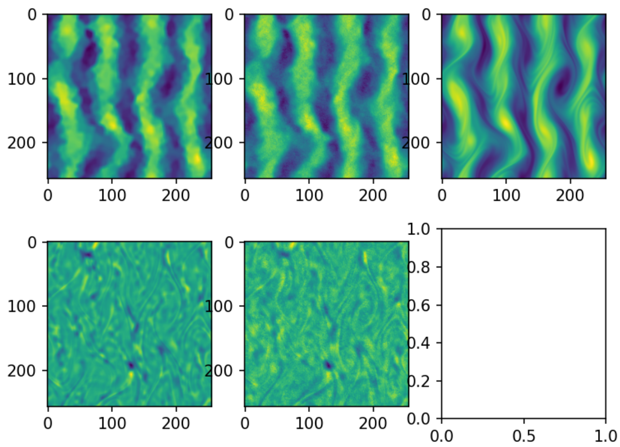

图 14: With attn, 12epoch

# 3 PINN

1. CFD funcs
2. Time Marching
3. IDE (Auxiliary PINN)
4. Dynamic Blocks

## 3.1 A-PINN

Original:

$$u^{(n)}(x) = f(x) + \lambda \int_0^x K(t)u(t)dt, \quad u(0) = a \tag{3}$$

Transform:

$$\begin{cases} u^{(n)}(x) = f(x) + \lambda \cdot v(x) \\ v(x) = \int_0^x K(t)u(t)dt \\ u(0) = a \end{cases} \tag{4}$$

Auxiliary:

$$\begin{cases} \frac{dv(x)}{dx} = K(x)u(x) \\ v(0) = 0 \end{cases} \tag{5}$$

Final:

$$\mu\frac{dI_{\mathrm{d}}\left(\tau,\mu\right)}{d\tau}=-I_{\mathrm{d}}\left(\tau,\mu\right)+\frac{\omega}{2}\int_{-1}^{1}P(\mu,\mu')I_{\mathrm{d}}\left(\tau,\mu'\right)d\mu'+\frac{\omega}{4\pi}P(\mu,\mu_0)F_0e^{-\tau} \tag{6}$$