Solving
Partial Differential Equation
With Fourier Neural Operator

- Burgers' Equation & Darcy Flow -

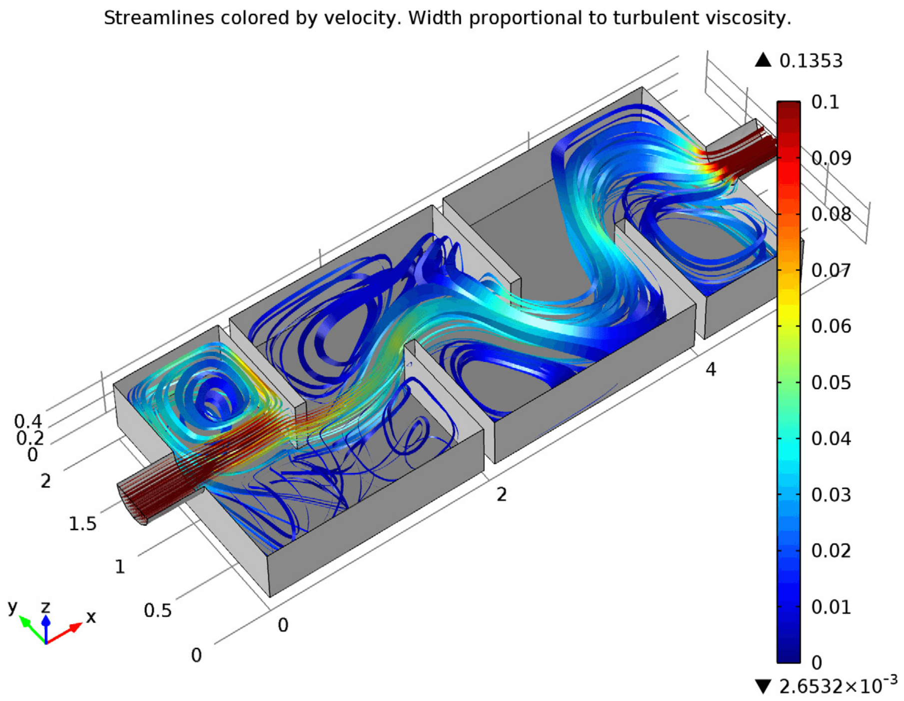
23.11.08.(Wed)

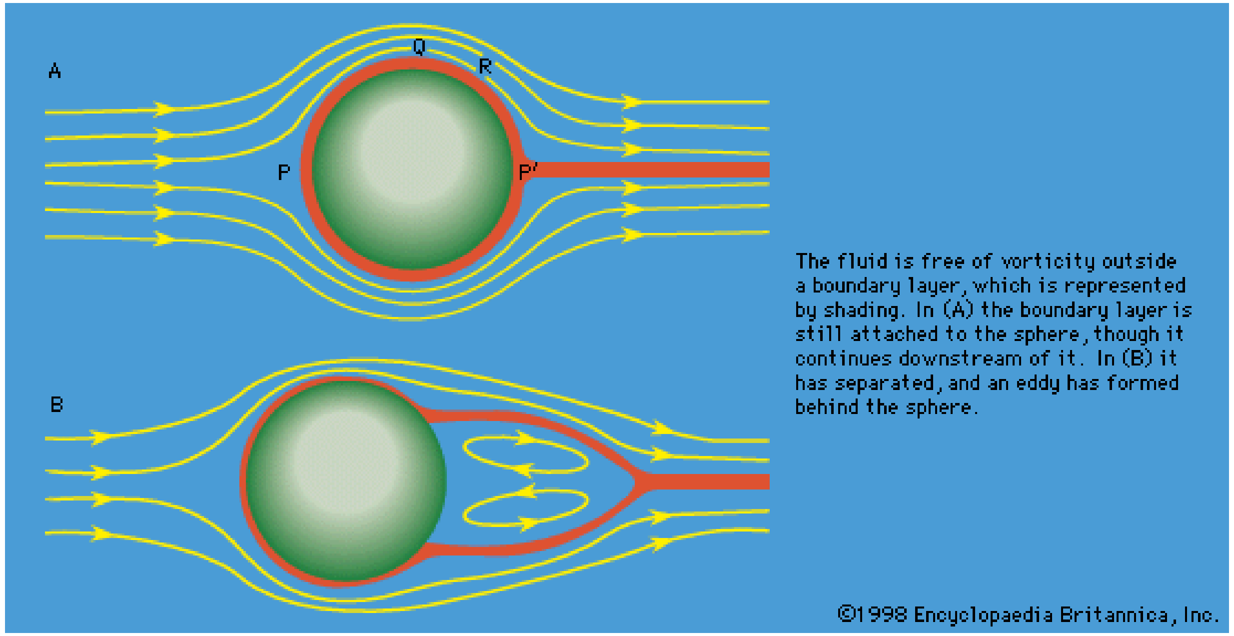
Professor: Yoo, Kee-youn

Jo Min-geon, Han Se-hee, Lee Jae-woo

Contents

Why solve PDE?





→ Partial Differential Equation solves various engineering problems in the real field.

Picture 1: https://www.comsol.com/multiphysics/navier-stokes-equations
Picture 2: https://www.britannica.com/science/Navier-Stokes-equation

Problem on Solving PDE

- **The traditional numerical method of solving PDEs**

  - Finite differential method (FDM), finite element method (FEM), finite volume method (FVM), etc

  - Too much calculation and taking a long time

→ Study of methods using neural networks

→ **Fourier Neural Operator** emerged

What is FNO?

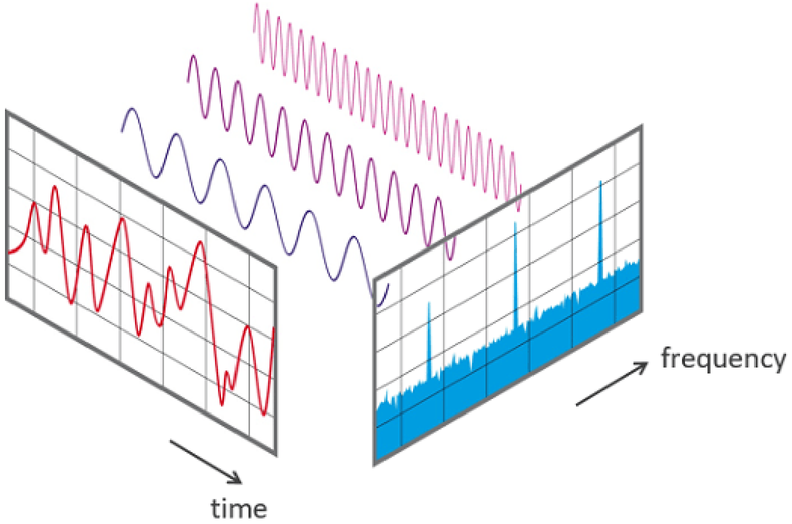
- FNO; **F**ourier **N**eural **O**perator

  = **Deep Learning + Fourier Transform**

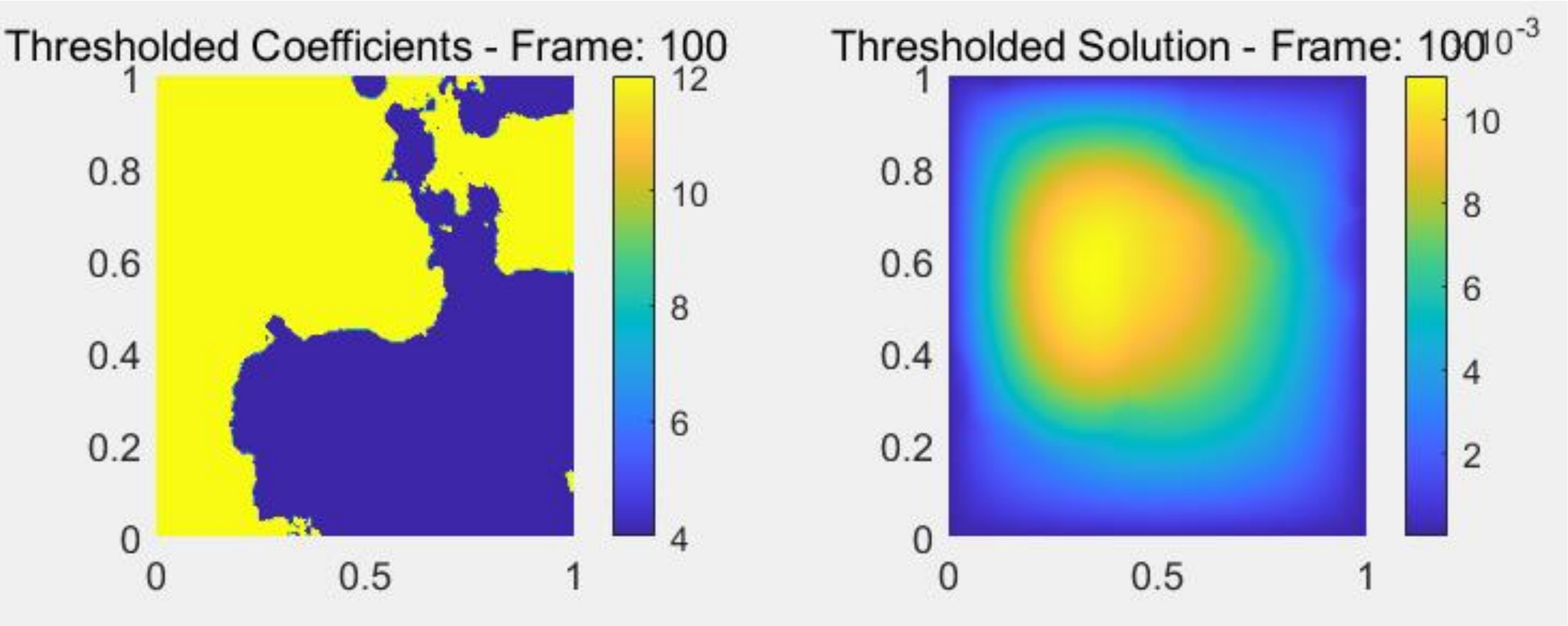
- Approximate the differential equation of the general space domain to a deep learning model

  -> Use the model to predict solutions

  💥 In this process, Fourier transform is used to take advantage of the characteristics of the frequency domain

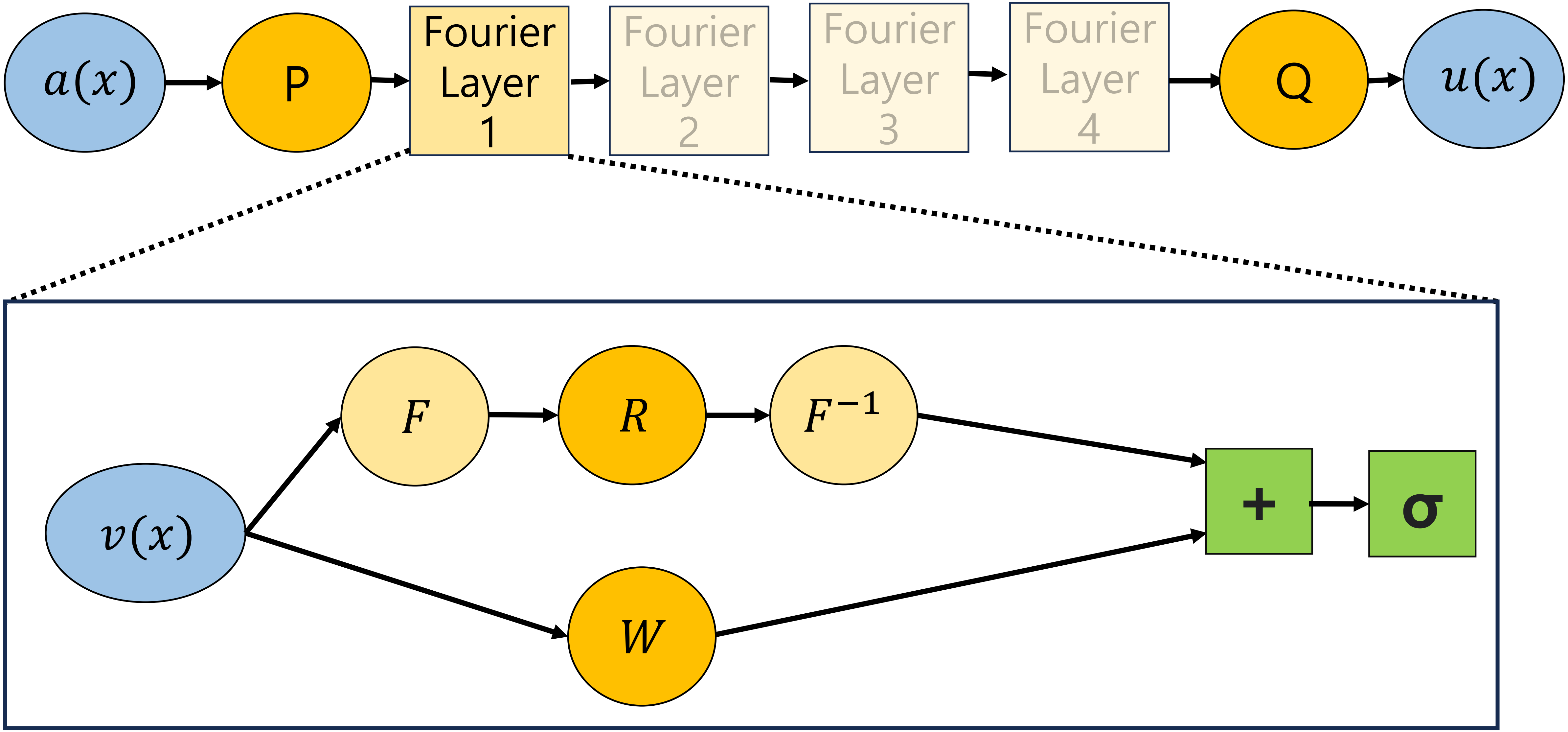


Darcy Flow - initial condition: coefficient



Full structure of FNO

- Initial condition: u(x,0) or coefficient



Full structure of FNO - code

#class SpectralConv2d(nn.Module)

- def __init__(self, in_channels, out_channels, modes1, modes2)
- def compl_mul2d(self, input, weights)
- def forward(self, x)

#class MLP(nn.Module)

- def __init__(self, in_channels, out_channels, mid_channels)
- def forward(self, x)

#class FNO2d(nn.Module)

- def __init__(self, modes1, modes2, width)
- def forward(self, x)
- def get_grid(self, shape, device)

#class SpectralConv2d(nn.Module)

```python
def forward(self, x): #모델에서 실행되어야하는 계산을 정의. (input을 넣어서 어떤 계산을 진행하여 output이 나올지를 정의)
    batchsize = x.shape[0]

    #Compute Fourier coeffcients u[illegible]tor of e^(- something constant). 입력 데이터를 받아 Fourier 계수로 변환
    x_ft = torch.fft.rfft2(x)
```

F

```python
    # Multiply relevant Fourier modes. 논문에서의 R(Linear Transtform)을 의미
    out_ft = torch.zeros(batchsize, self.out_channels,  x.size(-2), x.size(-1)//2 + 1, dtype=torch.cfloat, device=x.device)
    # x.size(-2): (batchsize, x=s, y=s, c=3)중 뒤에서 두번째인 y,
    # x.size(-1)//2 + 1: 3//2+1=2
    out_ft[:, :, :self.modes1, :self.modes2] = ₩
        self.compl_mul2d(x_ft[:, :, :self.modes1, :self.modes2], self.weights1)
    out_ft[:, :, -self.modes1:, :self.modes2] = ₩
        self.compl_mul2d(x_ft[:, :, -self.modes1:, :self.modes2], self.weights2)
    # x_ft[:, :, -self.modes1:, :self.modes2]가 input,  self.weights2가 weights
```

R

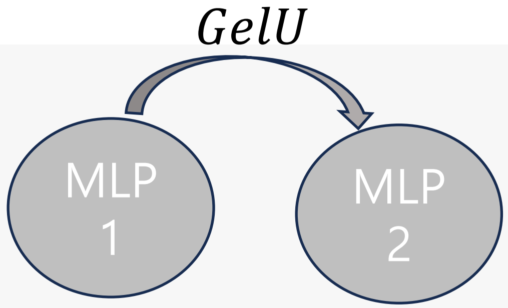
```python
    #Return to physical space
    x = torch.fft.irfft2(out_ft, s=(x.size(-2), x.size(-1)))
    return x
```

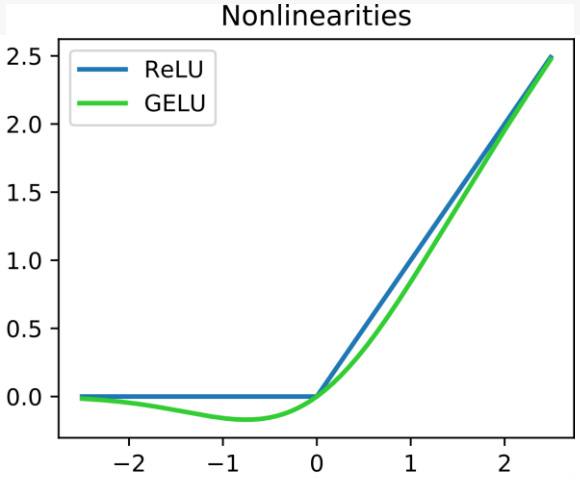
F^{-1}

- def forward(self, x): conduct fourier transform inside the Fourier layer

#class MLP(nn.Module)

```
def forward(self, x):
    x = self.mlp1(x) #첫번째 합성곱 레이어 통과
    x = F.gelu(x) #GelU 활성화함수 적용
    x = self.mlp2(x) #두번째 합성곱 레이어 통과
    return x
```





- def forward(self, x)
- Use GelU as the activation function.
- : GelU: Smoothing version of RelU
- : After $F \rightarrow W \rightarrow F^{-1}$ process, will go through this MLP before adding with W
- : This MLP class is also used by Neural Network Q

#class FNO2d(nn.Module)

```python
def __init__(self, modes1, modes2,  width):
    super(FNO2d, self).__init__()

    """
    The overall network. It contains 4 layers of the Fourier layer.
    1. Lift the input to the desire channel dimension by self.fc0 .
    2. 4 layers of the integral operators u' = (W + K)(u).
        W defined by self.w; K defined by self.conv .
    3. Project from the channel space to the output space by self.fc1 and self.fc2 .

    input: the solution of the coefficient function and locations (a(x, y), x, y)
    #coeff: a(x, y) 위치: x, y
    input shape: (batchsize, x=s, y=s, c=3)
    output: the solution #u(x,y)
    output shape: (batchsize, x=s, y=s, c=1) ?
    """

    self.modes1 = modes1
    self.modes2 = modes2
    self.width = width #width는 뭘 가리킬까?
    self.padding = 9 # pad the domain if input is non-periodic

    self.p = nn.Linear(3, self.width) # input channel is 3: (a(x, y), x, y)
    #논문 그림에서 a(x)가 v(x)로 되도록 만드는 P
    #nn.Linear(): 선형회귀모델. 입력차원:3 출력차원:width
    self.conv0 = SpectralConv2d(self.width, self.width, self.modes1, self.modes2)
    #in_channels: self.width, out_channels: self.width, modes1: self.modes1, modes2: self.modes2
    self.conv1 = SpectralConv2d(self.width, self.width, self.modes1, self.modes2)
    self.conv2 = SpectralConv2d(self.width, self.width, self.modes1, self.modes2)
    self.conv3 = SpectralConv2d(self.width, self.width, self.modes1, self.modes2)
    self.mlp0 = MLP(self.width, self.width, self.width)
    self.mlp1 = MLP(self.width, self.width, self.width)
    self.mlp2 = MLP(self.width, self.width, self.width)
    self.mlp3 = MLP(self.width, self.width, self.width)
    self.w0 = nn.Conv2d(self.width, self.width, 1)
    self.w1 = nn.Conv2d(self.width, self.width, 1)
    self.w2 = nn.Conv2d(self.width, self.width, 1)
    self.w3 = nn.Conv2d(self.width, self.width, 1)
    self.q = MLP(self.width, 1, self.width * 4) # output channel is 1: u(x, y)
    #입력채널: width, 출력채널: 1, 중간채널: 입력채널*4
```

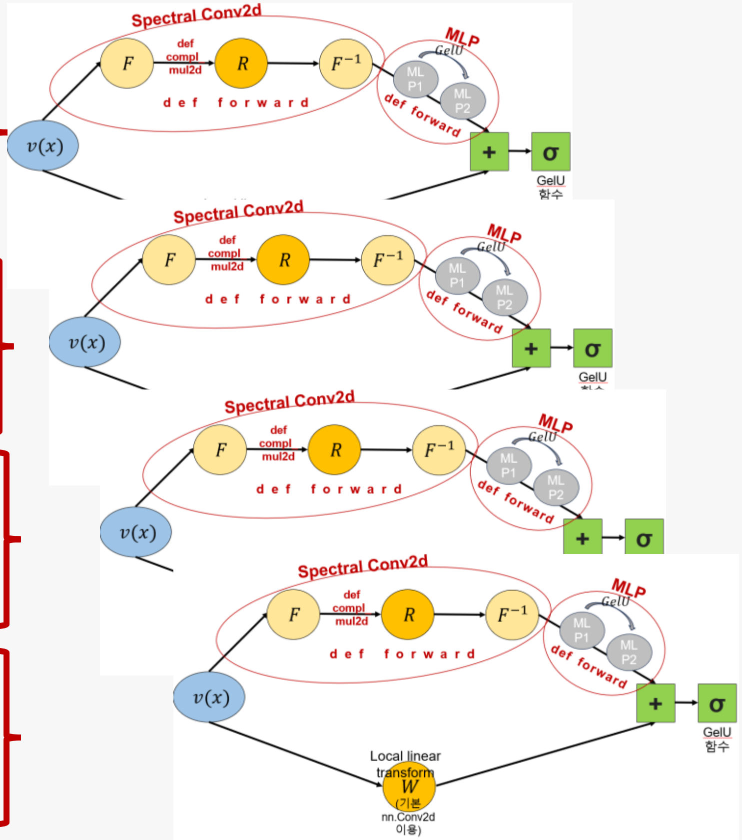
- def __init__(self, modes1, modes2, width)

- Use all the classes created so far

- 4 Fourier Layers

- Set variables for Conv2d, MLP, and General Linear Transformation(W) processes in each layer

```python
def forward(self, x): #순전파
    grid = self.get_grid(x.shape, x.device) #shape: x.shape, device: x.device. 따라서 x만 설정
    #따라서 grid의 shape(차원): (batchsize, size_x, size_y, 2)
    x = torch.cat((x, grid), dim=-1)
    #cat속의 input인 x shape: (batchsize, x=s, y=s, c=3)라고 추측
    # cat한 이후 x의 shape: (batchsize, s, s, c+2) 려나?
    #Q.x의 shape는? input데이터에서 확인
    ##Q.몇인지는 데이터에서 확인가능한가?

    x = self.p(x) # 이 결과 x.shape?
    x = x.permute(0, 3, 1, 2) #(batchsize, c+2(아마 3+2=5), s(아마 높이), s(아마 너비))
    x = F.pad(x, [0,self.padding, 0,self.padding]) #(batchsize, c+2(아마 3+2=5)+9, s(아마 높이

    x1 = self.conv0(x) #SpectralConv2d
    x1 = self.mlp0(x1)
    x2 = self.w0(x)
    x = x1 + x2
    x = F.gelu(x)

    x1 = self.conv1(x)
    x1 = self.mlp1(x1)
    x2 = self.w1(x)
    x = x1 + x2
    x = F.gelu(x)

    x1 = self.conv2(x)
    x1 = self.mlp2(x1)
    x2 = self.w2(x)
    x = x1 + x2
    x = F.gelu(x)

    x1 = self.conv3(x)
    x1 = self.mlp3(x1)
    x2 = self.w3(x)
    x = x1 + x2

    x = x[..., :-self.padding, :-self.padding]
    x = self.q(x)
    x = x.permute(0, 2, 3, 1)
    return x
```



#class FNO2d(nn.Module)

- def forward(self, x)

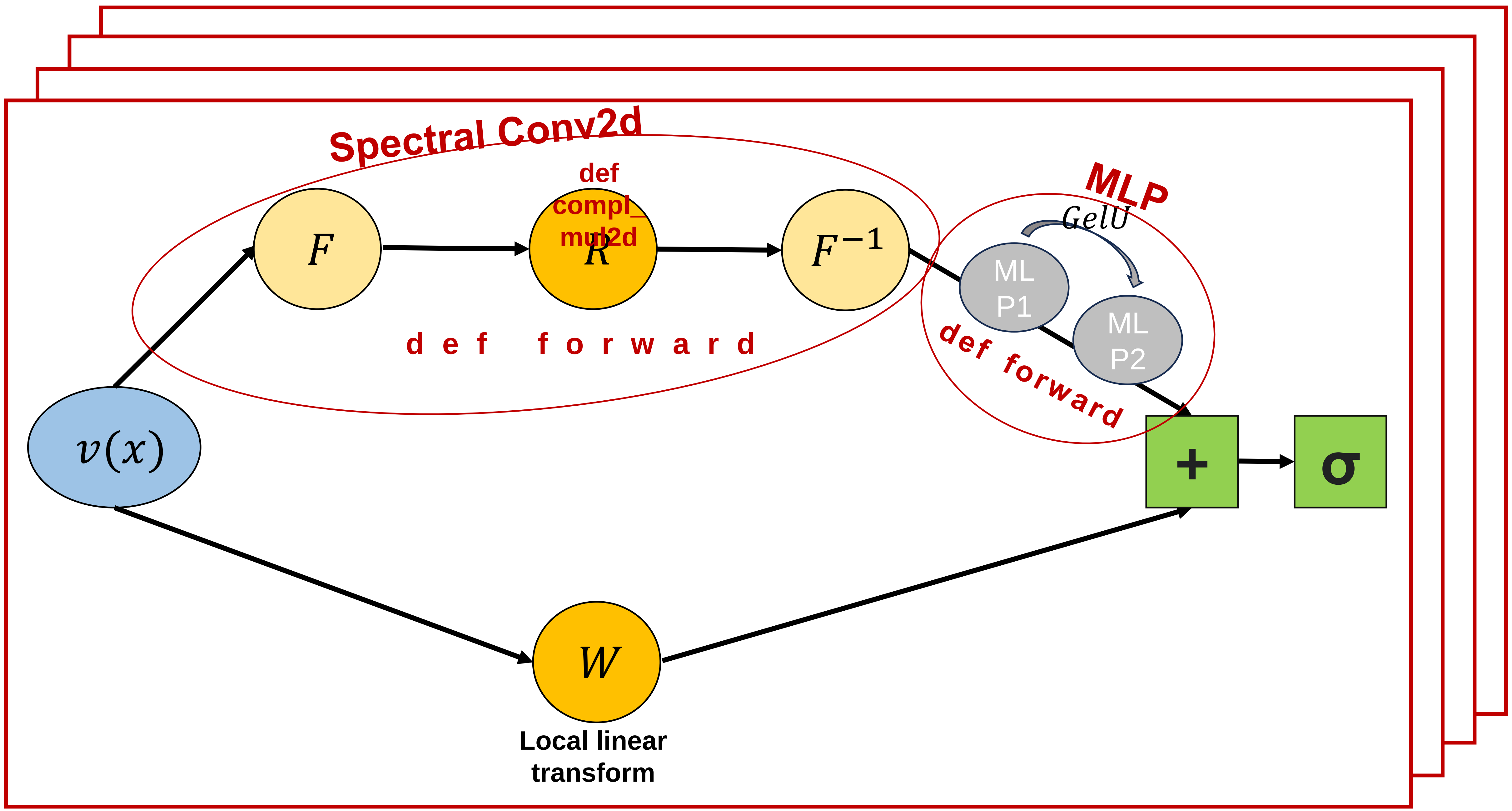
- design **overall architecture**

<Process for each layer>

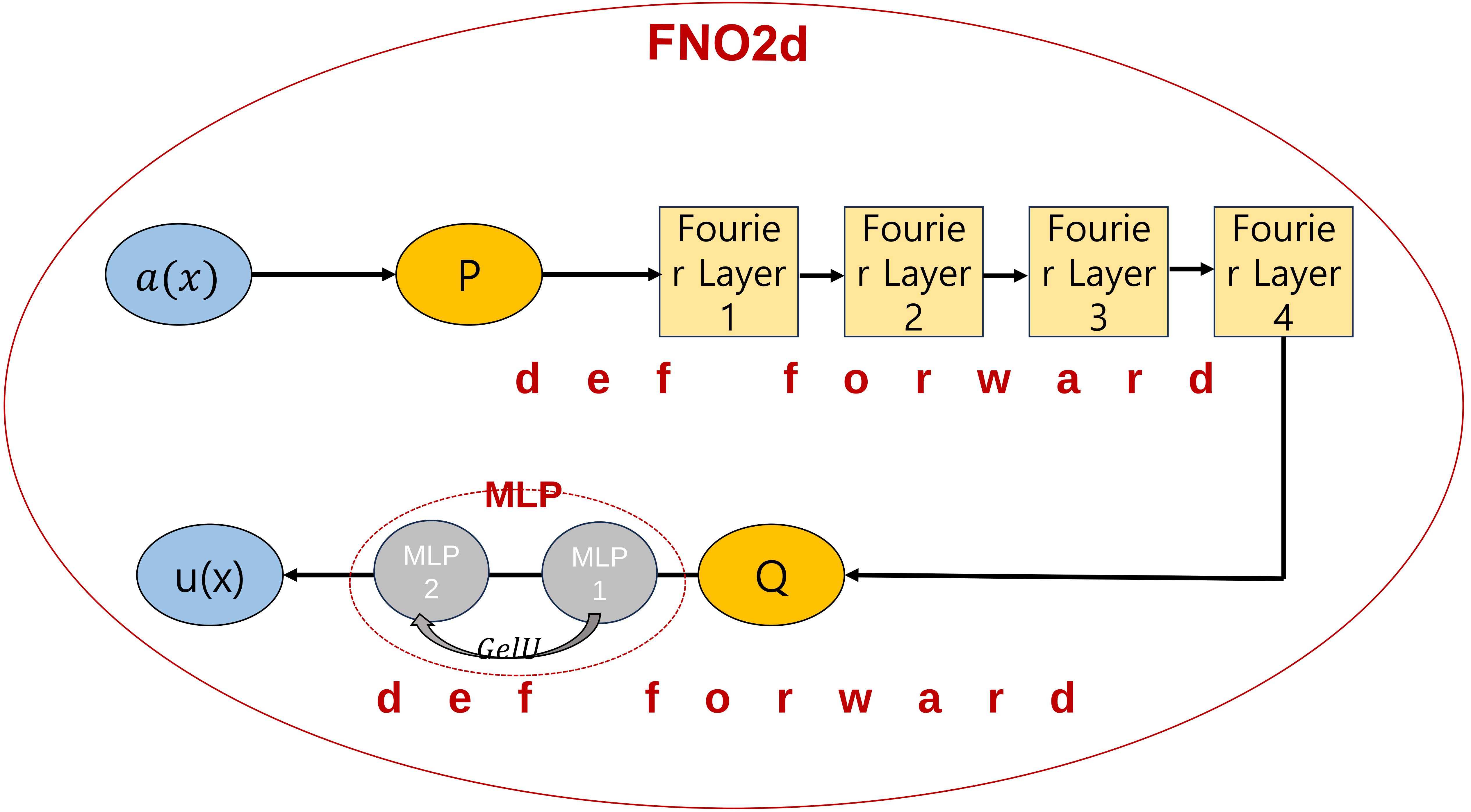
- Sum results of Conv2d, MLP process and results of general linear transformation (W), and goes through activation functions

- All four Fourier Layers are designed in the same structure

- Even in Neural Network Q, MLP is used to return the dimension



Spectral Conv2d
def
compl_
mul2d
F
R
F^{-1}
d e f f o r w a r d
MLP
$GelU$
ML
P1
ML
P2
def forward
$v(x)$
+
σ
W
Local linear
transform



FNO2d
a(x)
P
Fourier Layer 1
Fourier Layer 2
Fourier Layer 3
Fourier Layer 4
d e f f o r w a r d
MLP
u(x)
MLP 2
MLP 1
Q
GelU
d e f f o r w a r d

Burgers Equation

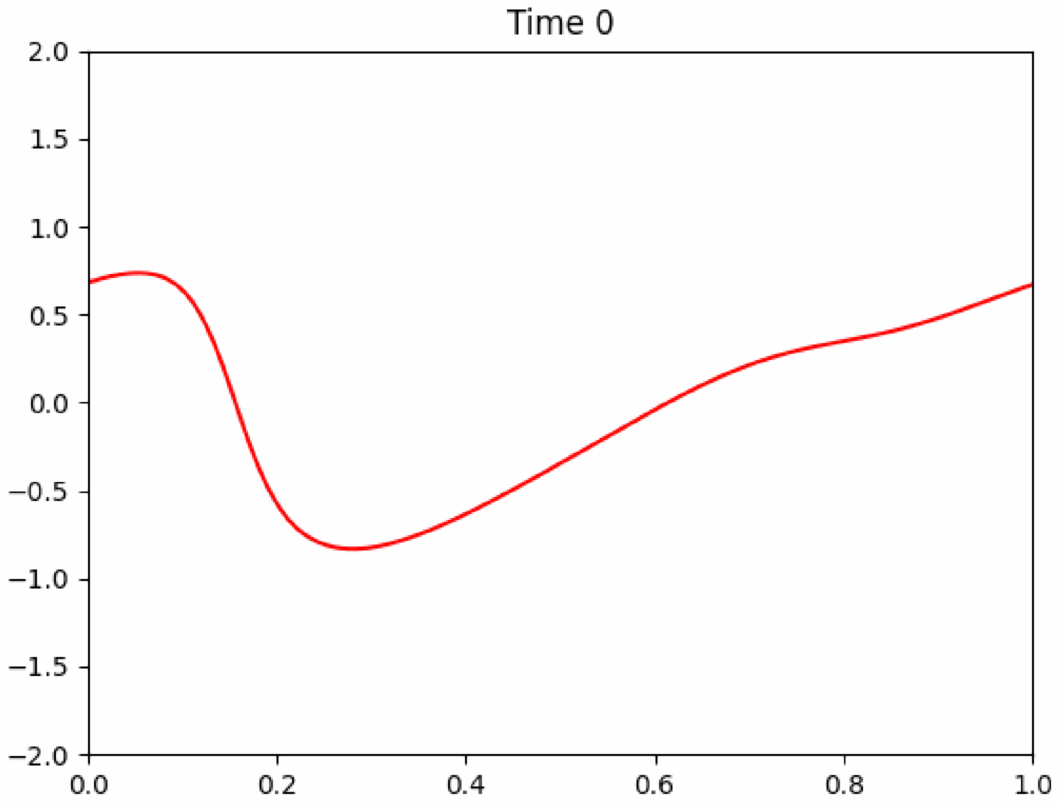
- An Equation used in modeling the one-dimensional flow of viscous fluids

$$\partial_t u(x,t) + \partial_x(u^2(x,t)/2) = \nu\partial_{xx}u(x,t), \qquad x \in (0,1), t \in (0,1]$$
$$u(x,0) = u_0(x), \qquad x \in (0,1).$$

- T=(0,1]

->How about putting the **prediction back into input**?

Animated Solution Predictions in Burgers' Equation



https://github.com/hanmind/neuraloperator/blob/main/fourier_1d_Burgers.ipynb