Advanced Chiral Metasurface Engineering in the Mid-Infrared Region with Different Neural Network Approaches

Supervisor

Dr. Luo Tony | Dr. Han Daoru

Presenter

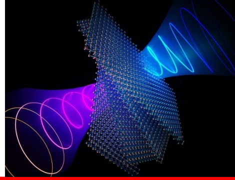
Jiang Chen | Nithin Shyam Soundararajan | Xiangkai Zeng

- 1. The Introduction of Chiral Metamaterial
- 2. DL models for Chiral Metamaterial Design
- 3. Models Performance Comparison

The Introduction of Chiral Metamaterial

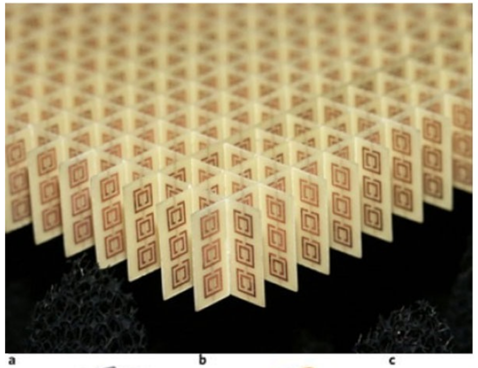
Introduction

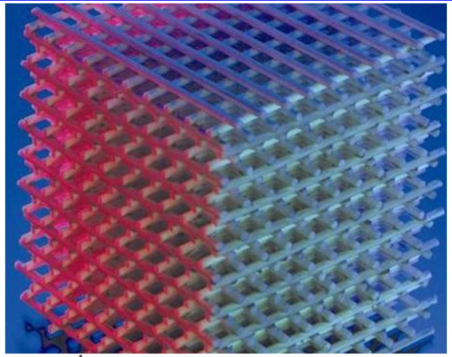
DL-GPU Team 3

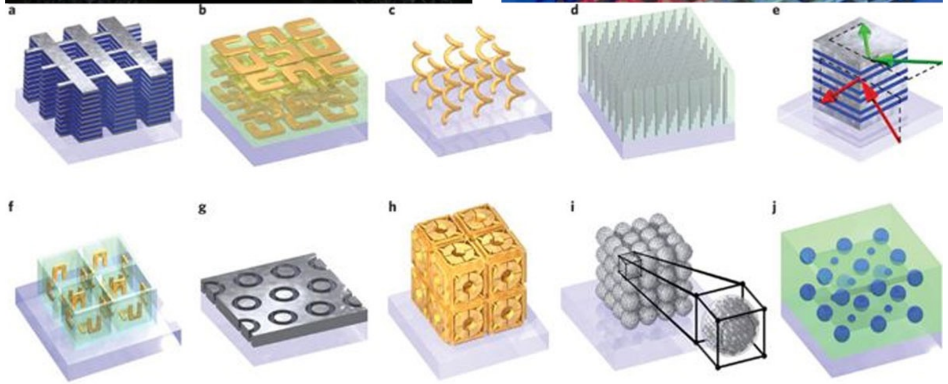


- What is metamaterial?

A metamaterial is any material engineered to have a property that is rarely observed in naturally occurring materials.



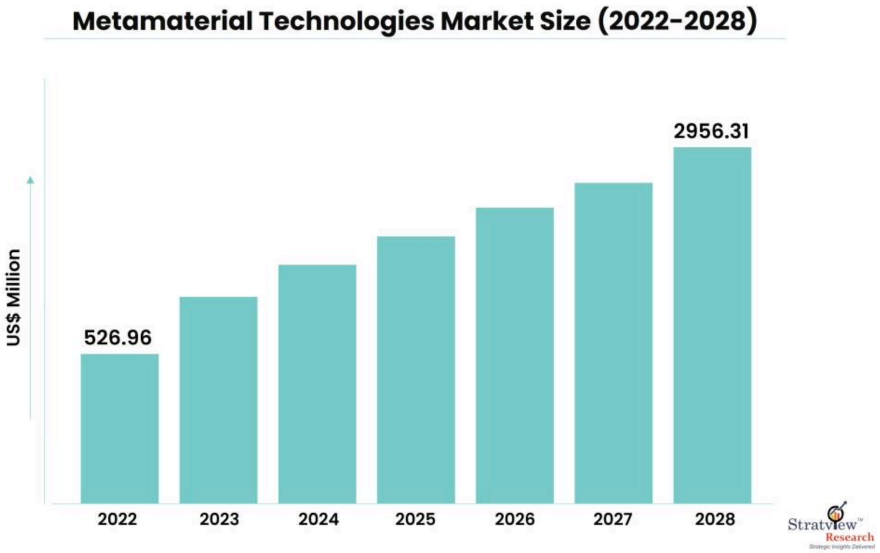


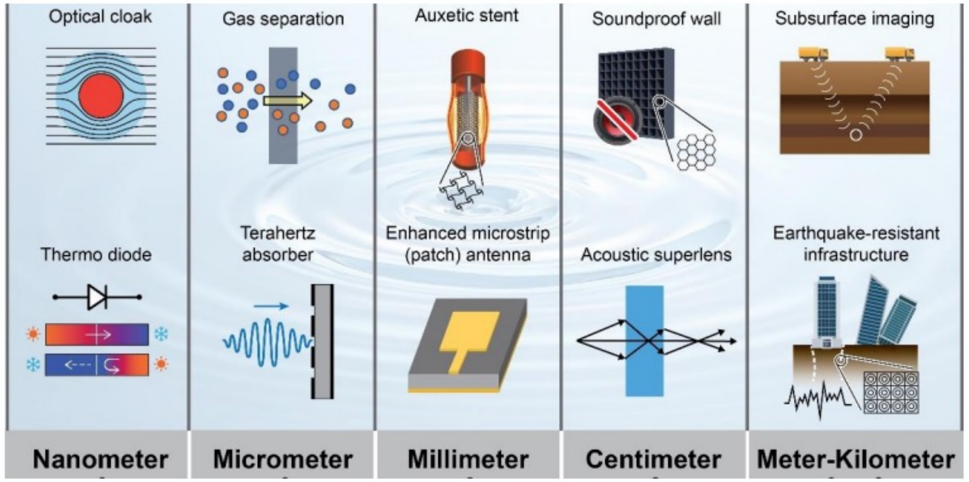


Picture @

Philip, E., Zeki Güngördü, M., Pal, S. *et al.* Review on Polarization Selective Terahertz Metamaterials: from Chiral Metamaterials to Stereometamaterials. *J Infrared Milli Terahz Waves* **38**, 1047–1066 (2017). https://doi.org/10.1007/s10762-017-0405-y

https://www.grandviewresearch.com/industry-analysis/metamaterials-market

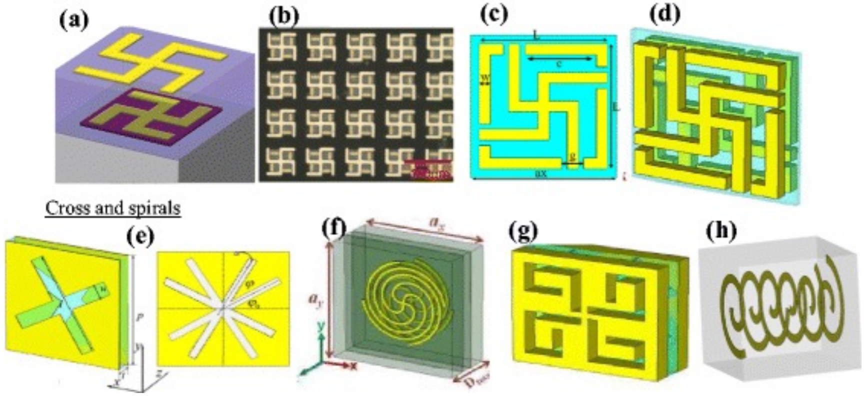




Introduction

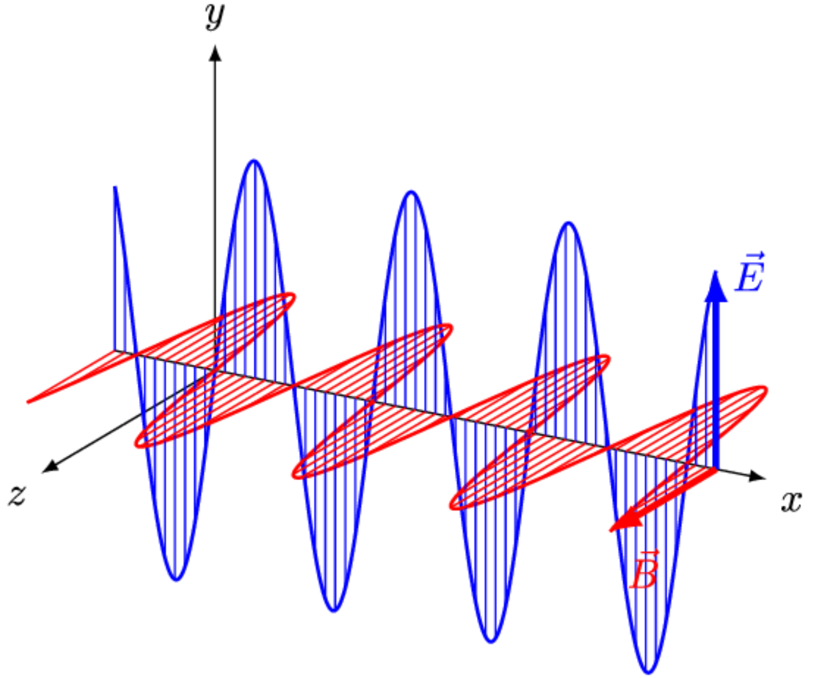
- ## Chiral Metamaterial

  Chiral metamaterials (CMMs), which are artificial materials that lack any planes of mirror symmetry, possess strong ability to rotate the plane of polarization of electromagnetic waves.

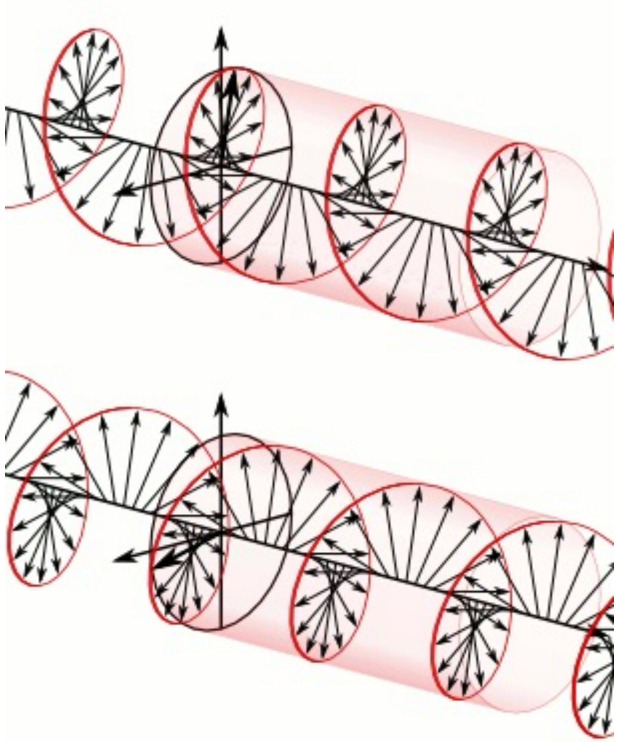


Picture @

Philip, E., Zeki Güngördü, M., Pal, S. *et al.* Review on Polarization Selective Terahertz Metamaterials: from Chiral Metamaterials to Stereometamaterials. *J Infrared Milli Terahz Waves* **38**, 1047–1066 (2017). https://doi.org/10.1007/s10762-017-0405-y



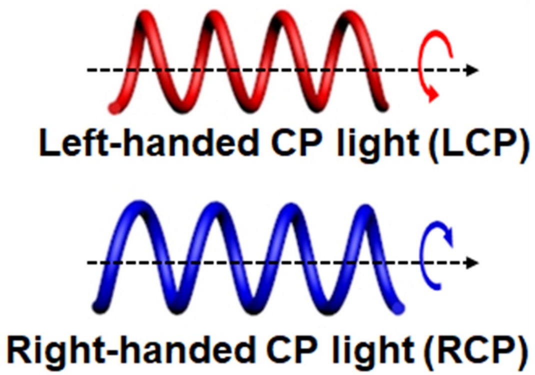
EM Wave

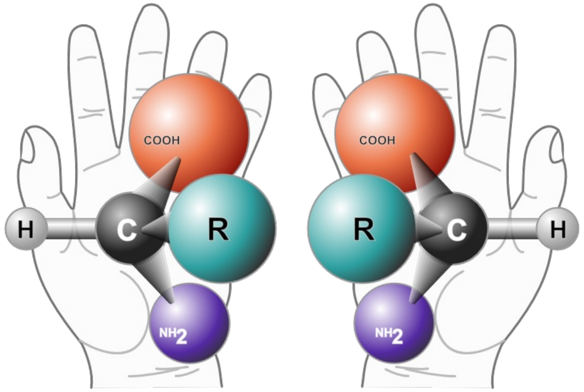


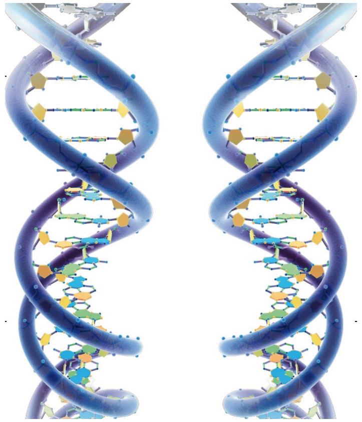
LCP

RCP

Introduction

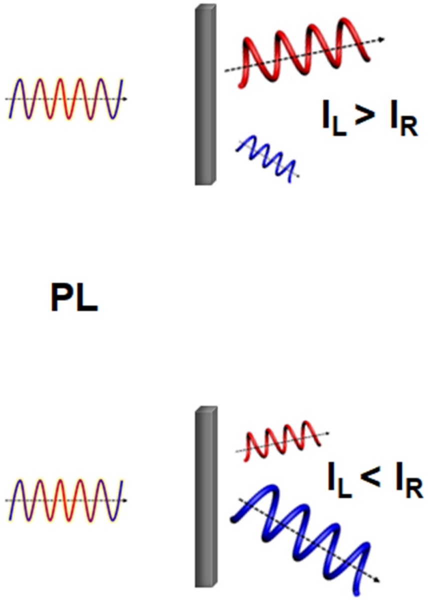


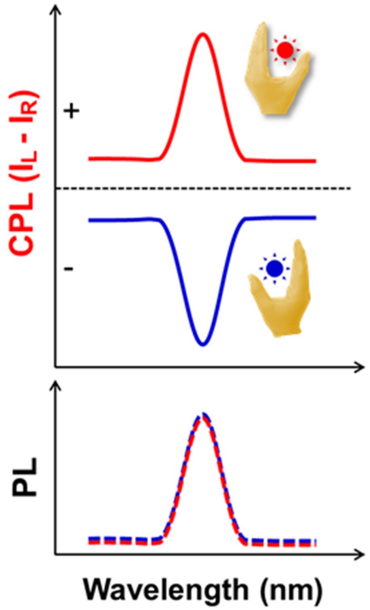




Chiral Molecules

DNA



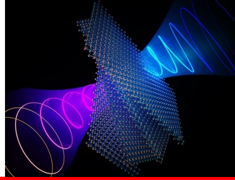


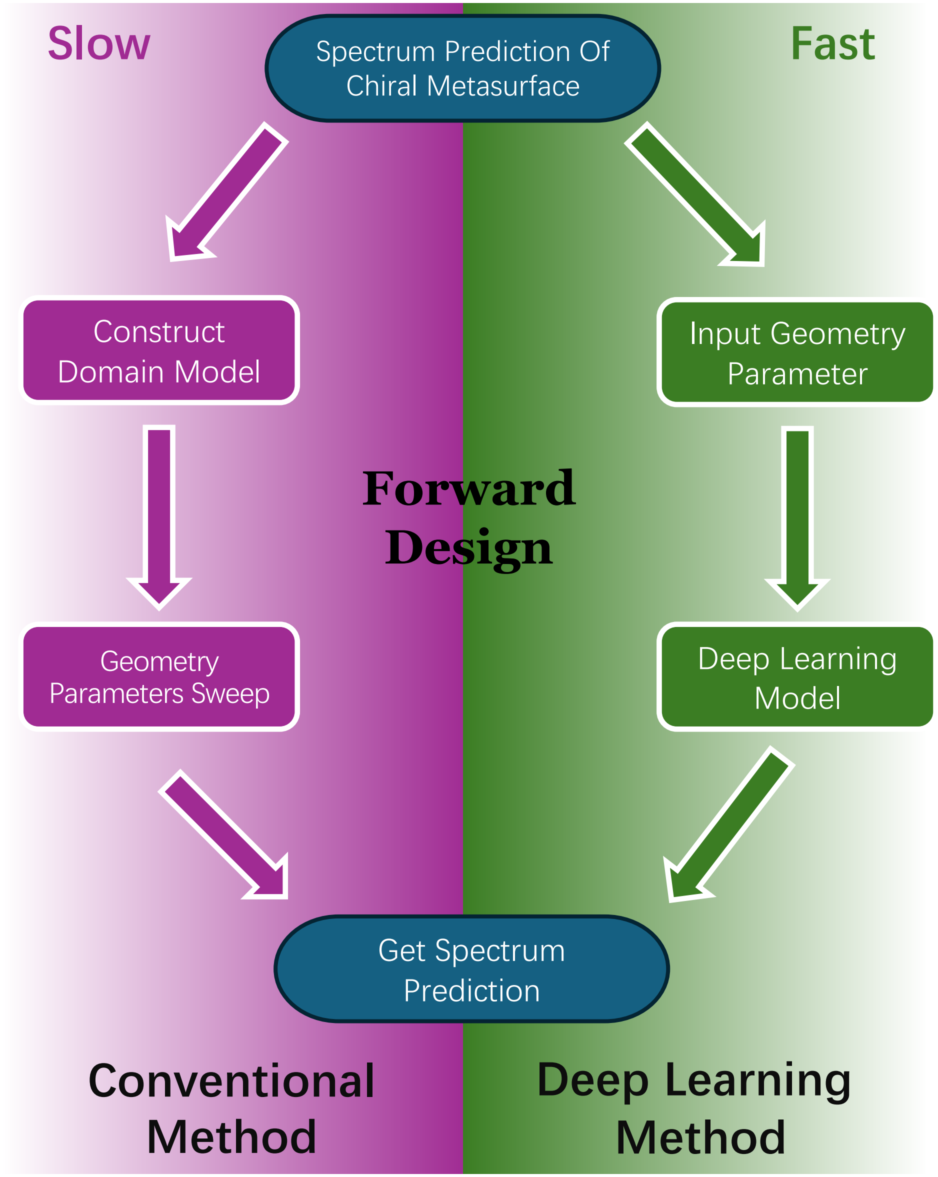
For Nature chiral molecular like Protein, DNA, etc. The inherent CPL is weak to be detected.

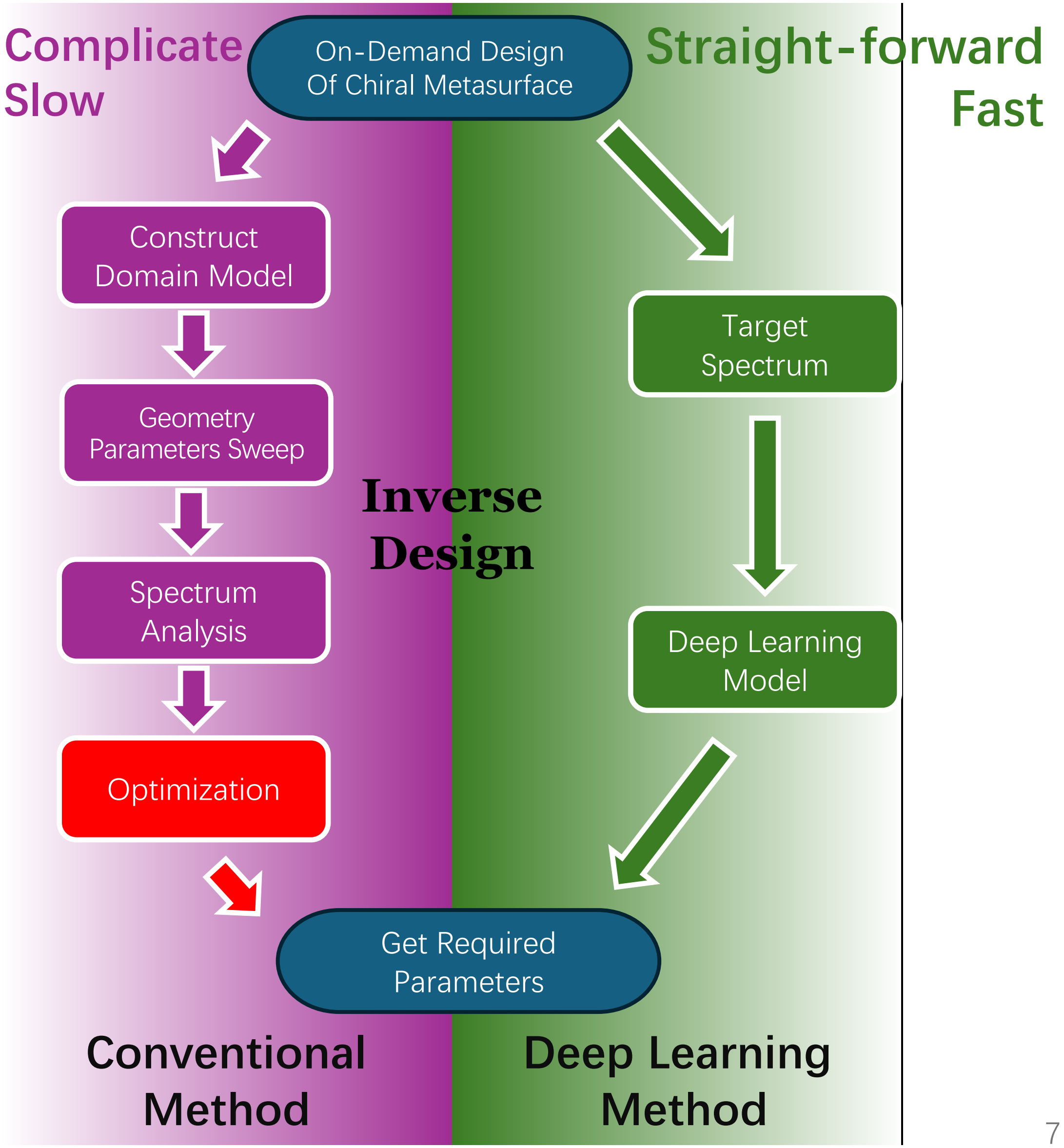
Chiral Metamaterial can increase the signal via *surface enhanced plasma* principle to increase the detection limit.

Picture @
Liu, X.; Jin R.H. Recent advances in circularly polarized luminescence generated by inorganic materials. *Chem. Synth.* **2022**, *2*, 7. http://dx.doi.org/10.20517/cs.2022.01

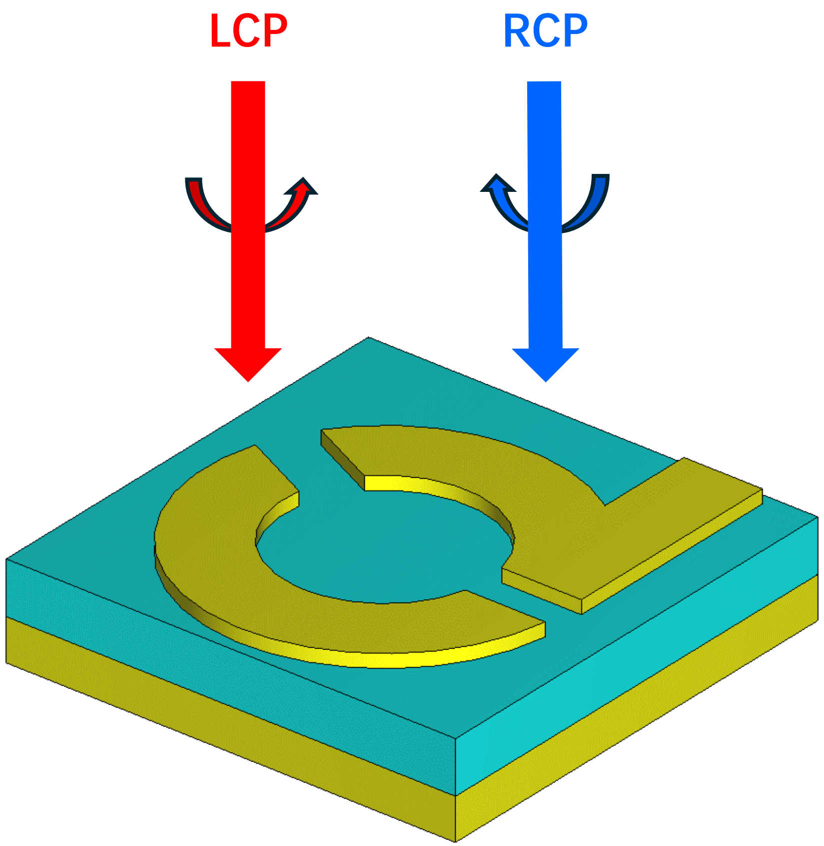
Introduction







Conventional Method

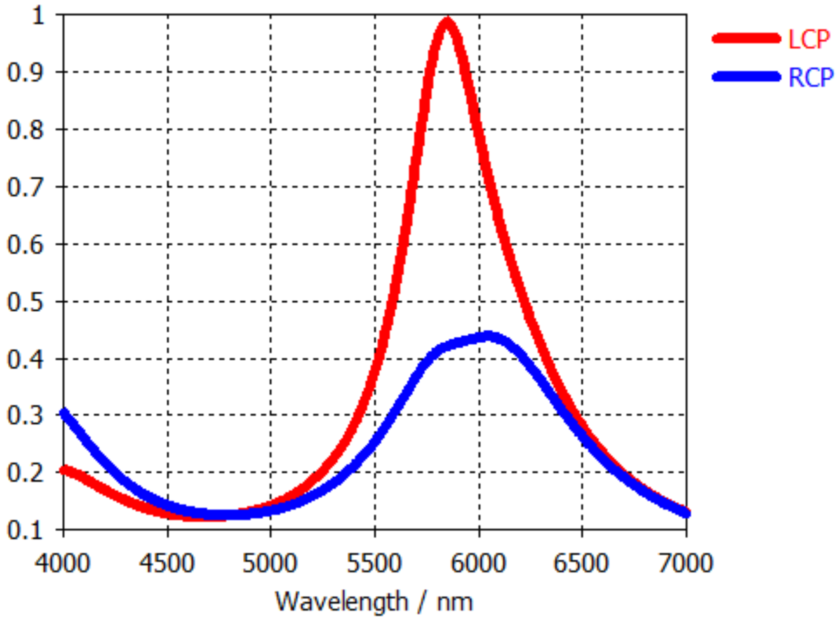


Domain model with LCP and RCP incident

Forward Design

CST Studio Suite 2022

Inverse Design



Absorption Spectrum for LCP and RCP

Conventional FIT Principle

CST Studio Suite 2022

The conventional method Finite Integration Technique (FIT) is based on the solution of discretized set of *Maxwell's Equations*.

This numerical method provides a universal spatial discretization scheme applicable to various electromagnetic problems ranging from static field calculations to high frequency applications in time or frequency domain.

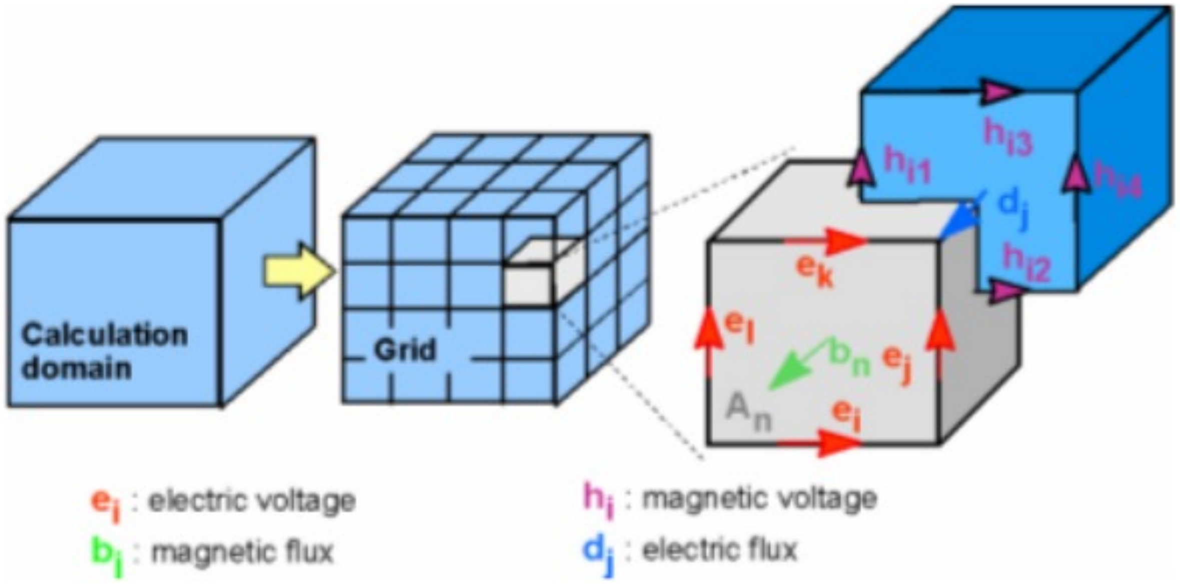
$$\oint_{\partial A} \vec{E} \cdot d\vec{s} = -\int_A \frac{\partial \vec{B}}{\partial t} \cdot d\vec{A}$$

$$\oint_{\partial \mathrm{A}} \vec{\mathrm{H}} \cdot \mathrm{d}\vec{\mathrm{s}} = \int_{\mathrm{A}} \left(\frac{\partial \vec{\mathrm{D}}}{\partial \mathrm{t}} + \vec{\mathrm{J}} \right) \cdot \mathrm{d}\vec{\mathrm{A}}$$

$$\oint_{\partial \mathrm{V}} \vec{\mathrm{D}} \cdot \mathrm{d}\vec{\mathrm{A}} = \int_{\mathrm{V}} \rho \ \mathrm{dV}$$

$$\oint_{\partial \mathrm{V}} \vec{\mathrm{B}} \cdot \mathrm{d}\vec{\mathrm{A}} = 0$$

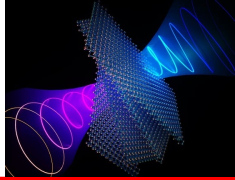
Integral form of Maxwell's Equation



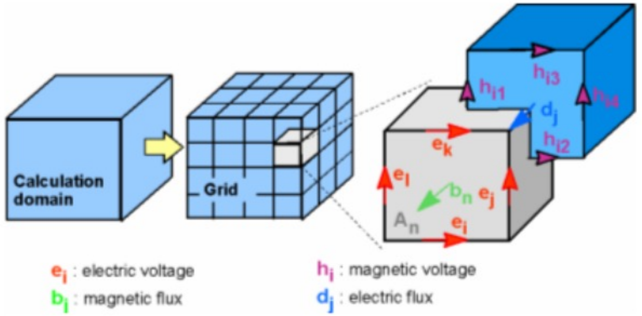
Discretization with Small Mesh to fit the geometry

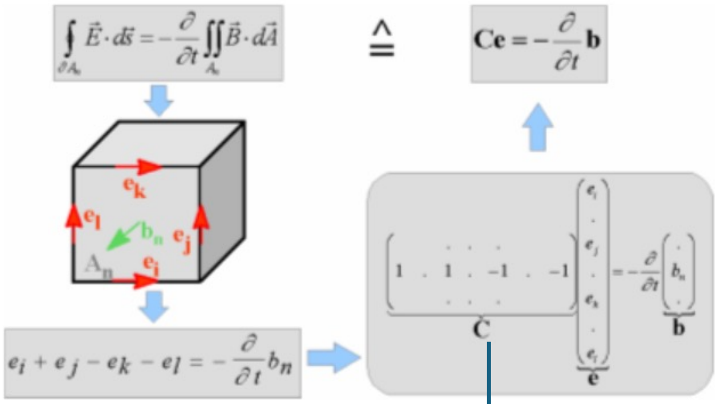
Conventional FIT Principle

DL-GPU Team 3



CST Studio Suite 2022





Faraday's Law

Discrete Operator

$$\oint_{\partial A} \vec{E}\cdot d\vec{s} = -\int_A \frac{\partial \vec{B}}{\partial t}\cdot d\vec{A}$$

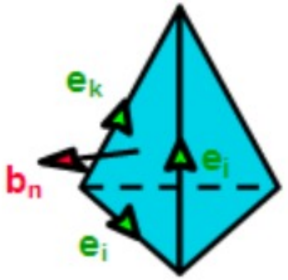
$$\oint_{\partial A} \vec{H}\cdot d\vec{s} = \int_A \left(\frac{\partial \vec{D}}{\partial t} + \vec{J}\right)\cdot d\vec{A}$$

$$\oint_{\partial V} \vec{D}\cdot d\vec{A} = \int_V \rho \; dV$$

$$\oint_{\partial V} \vec{B}\cdot d\vec{A} = 0$$

Integral form of Maxwell's Equation

Tetrahedral Mesh



$$\mathcal{C}e = -\frac{d}{dt}b$$

$$\tilde{\mathcal{C}}h = \frac{d}{dt}d + j$$

$$\tilde{\mathcal{S}}d = q$$

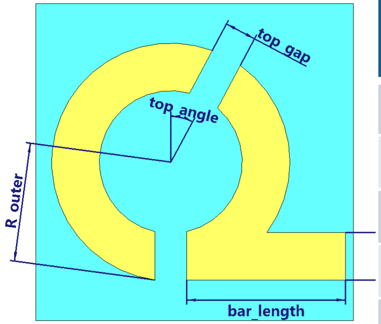
$$\mathcal{S}b = 0$$

Discretized form of Maxwell's Equation

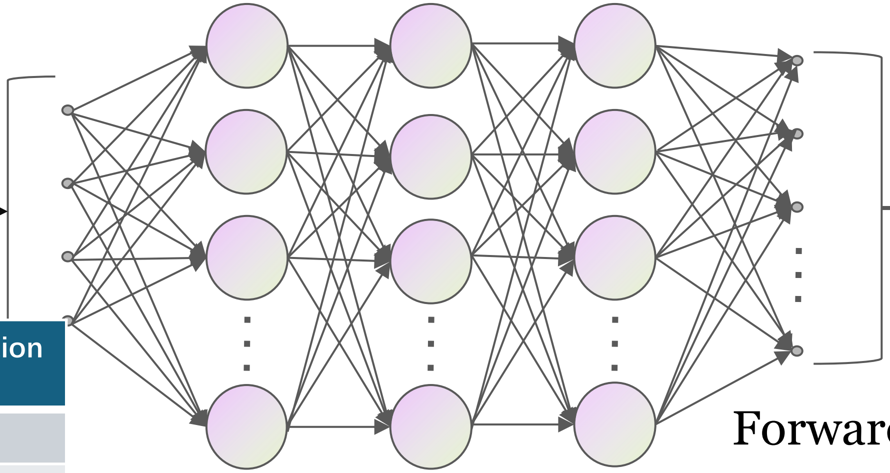
Deep Learning Model Sketch

DL-GPU Team 3

Real Geometry Parameters

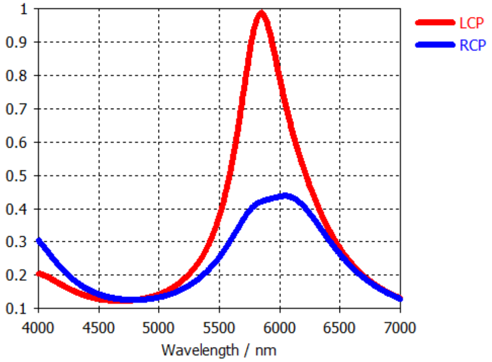


Engineering Parameter	abbreviation
top_gap	t_g
top_angle	t_a
bar_length	b_l
R_outer	r
width	w

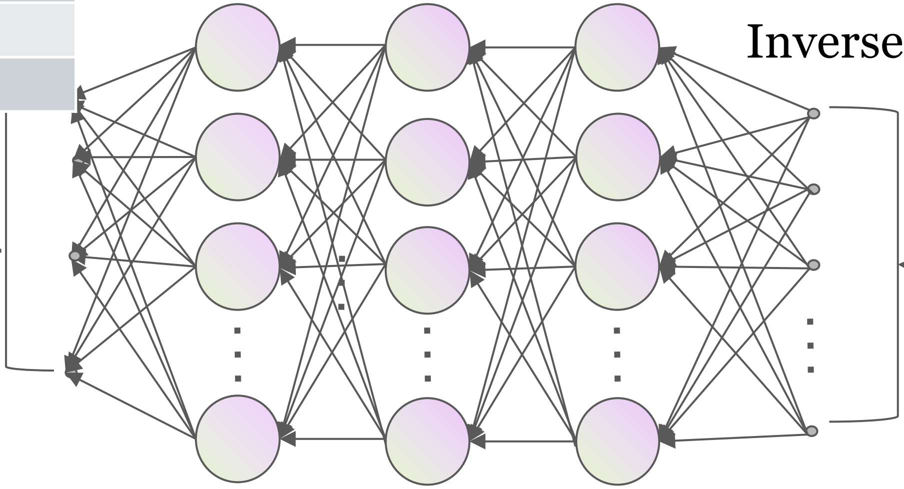


Forward Model

Predicted Spectrum
1001 points



Inverse Model



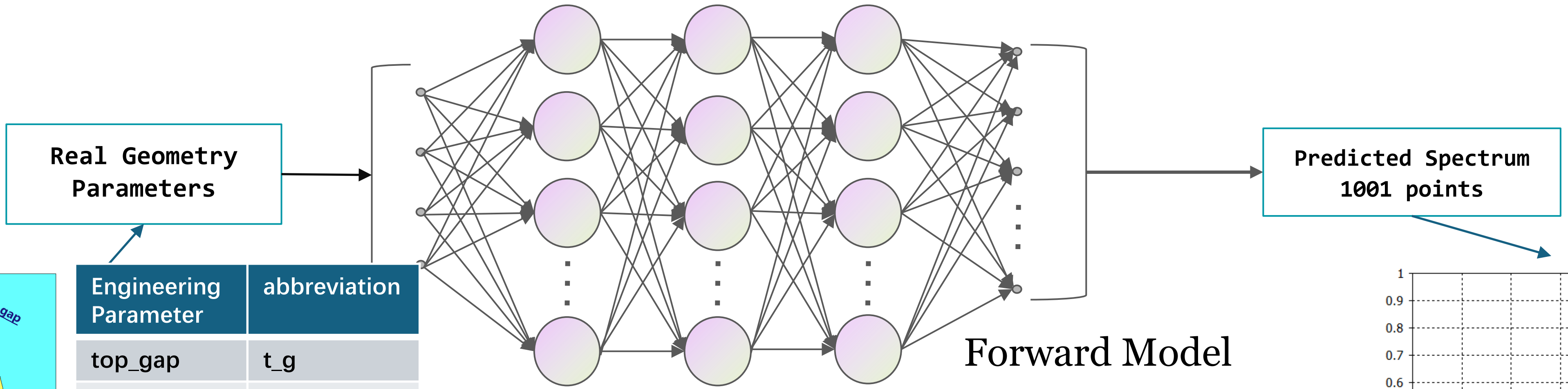
Real Spectrum
1001 points

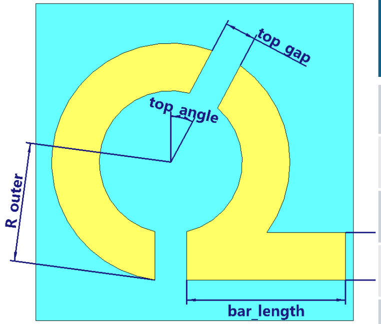
Predicted Geometry Parameters

DL models for Chiral Metamaterial Design

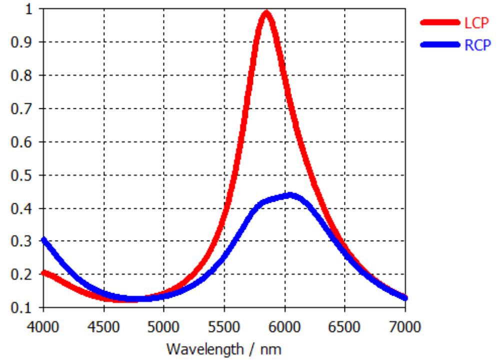
Deep Learning Model Sketch

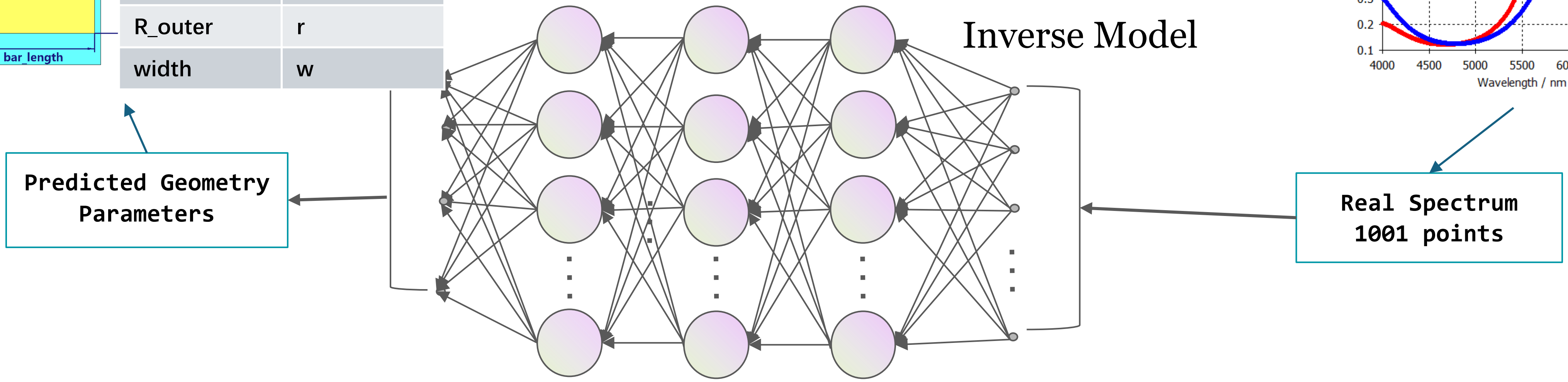
DL-GPU Team 3



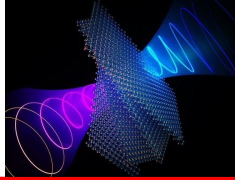


Engineering Parameter	abbreviation
top_gap	t_g
top_angle	t_a
bar_length	b_l
R_outer	r
width	w





Performance Matrix and Dataset



Dataset:

1920 data in total.

1536 data are used for training.
384 data are used for testing.

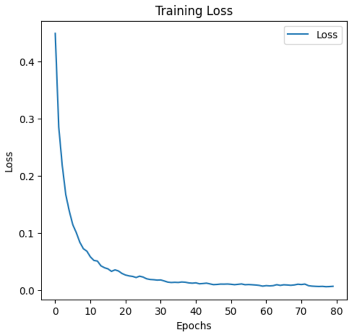
Performance Matrix:

1. R^2 Score
2. Time
3. GPU usage

FCNN Forward Result

DL-GPU Team 3

LCP Forward



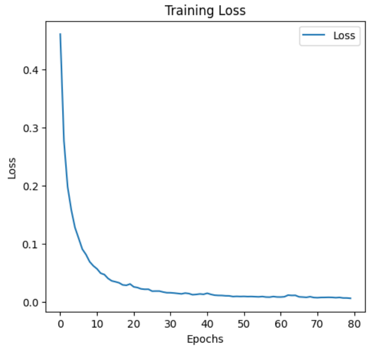
Model Parameters

```
class MLP(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(input_dim, 300)
        self.hidden2 = nn.Linear(300, 600)
        self.hidden3 = nn.Linear(600, 300)
        self.output = nn.Linear(300, output_dim)
        self.tanh = nn.Tanh()

    def forward(self, x):
        x = self.tanh(self.hidden1(x))
        x = self.tanh(self.hidden2(x))
        x = self.tanh(self.hidden3(x))
        x = self.output(x)
        return x
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

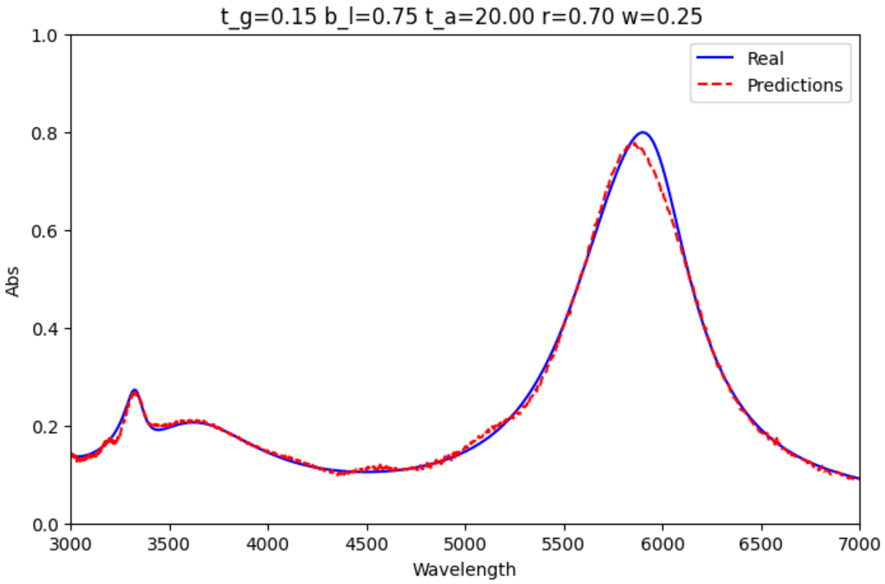
Performance Matrix:

1. R^2 Score
2. ✓ Time
3. ✓ GPU usage

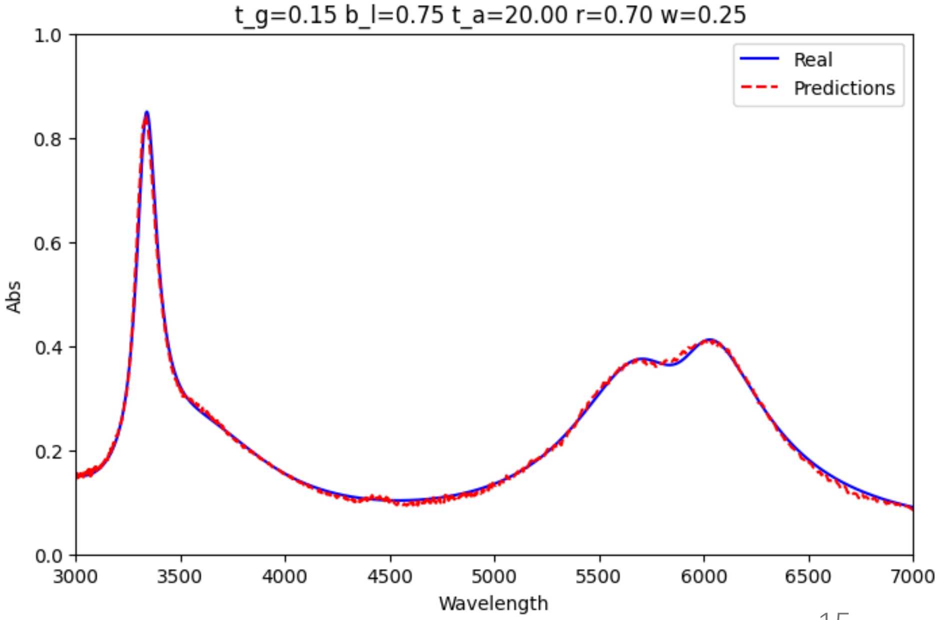
RCP Forward



Sample Test Data

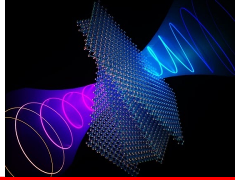


Sample Test Data

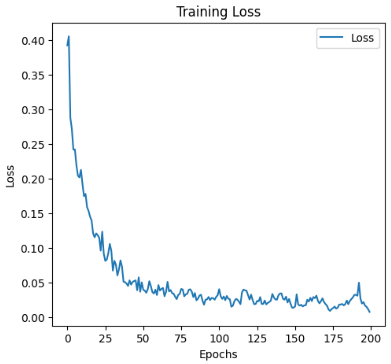


FCNN Inverse Result

DL-GPU Team 3



LCP Inverse



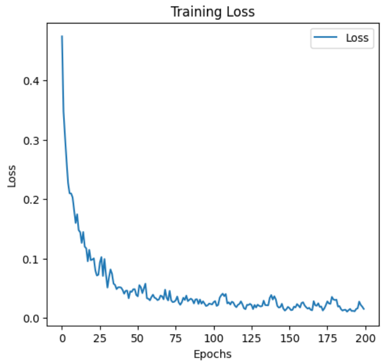
Model Parameters

```
class MLP(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(input_dim, 1000)
        self.hidden2 = nn.Linear(1000, 500)
        self.hidden3 = nn.Linear(500, 250)
        self.output = nn.Linear(250, output_dim)
        self.elu = nn.ELU()

    def forward(self, x):
        x = self.elu(self.hidden1(x))
        x = self.elu(self.hidden2(x))
        x = self.elu(self.hidden3(x))
        x = self.output(x)
        return x
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

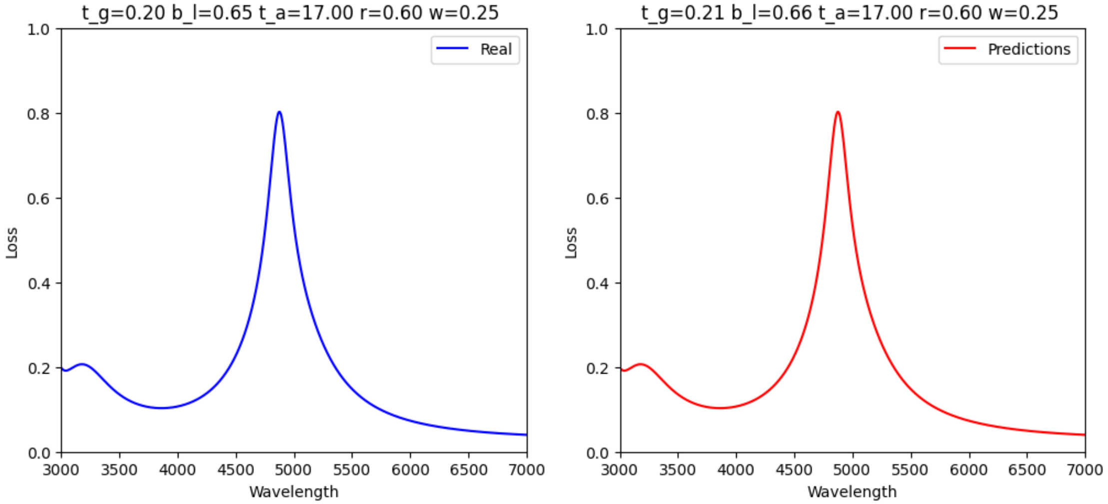
Performance Matrix:

1. R^2 Score
2. ✓ Time
3. ✓ GPU usage

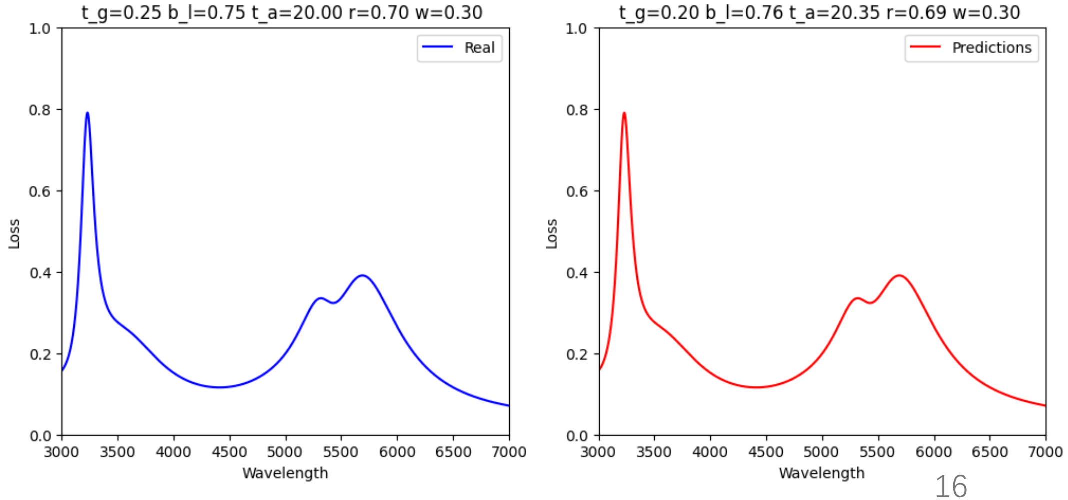
RCP Inverse



Sample Test Data



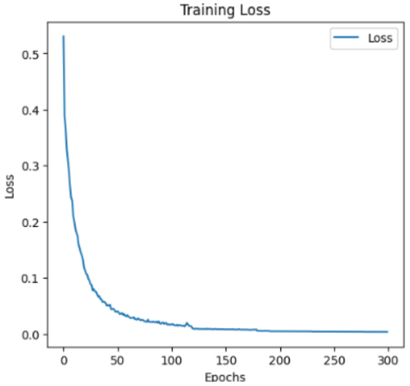
Sample Test Data



CNN Forward Result

DL-GPU Team 3

LCP Forward



Model Parameters

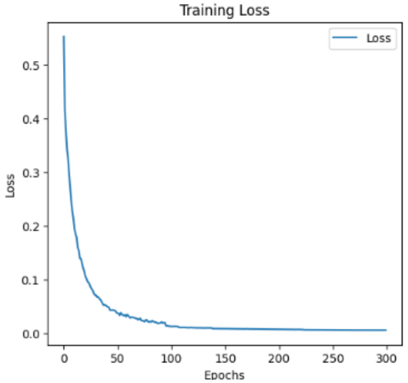
```python
class CNN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv1d(in_channels=16, out_channels=32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * input_dim, 512)
        self.fc2 = nn.Linear(512, output_dim)
        self.elu = nn.ELU()

    def forward(self, x):
        x = x.unsqueeze(1)  # Add channel dimension
        x = self.elu(self.conv1(x))
        x = self.elu(self.conv2(x))
        x = self.elu(self.conv3(x))
        x = x.view(x.size(0), -1)  # Flatten
        x = self.elu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```python
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

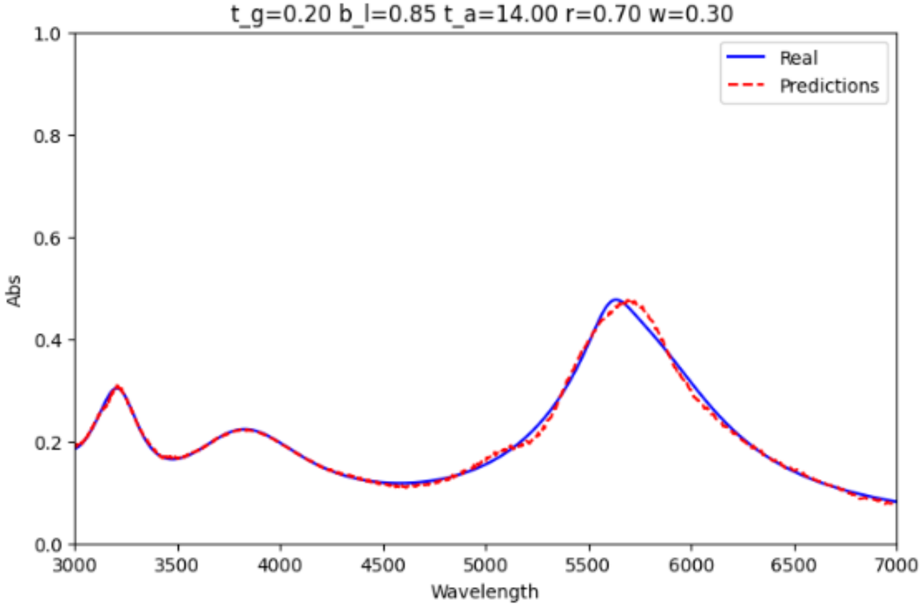
Performance Matrix:

✓ 1. R^2 Score
2. Time
3. GPU usage

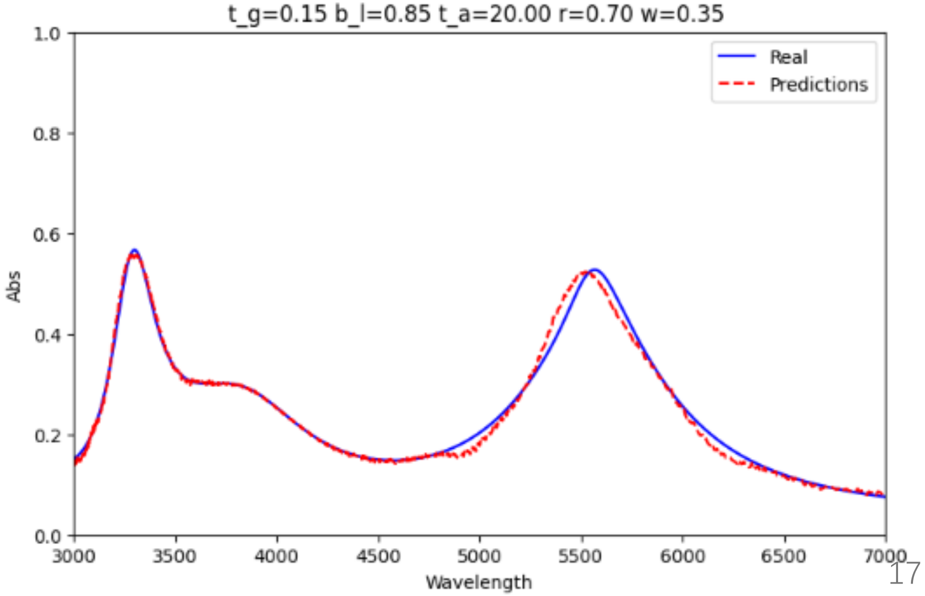
RCP Forward



Sample Test Data

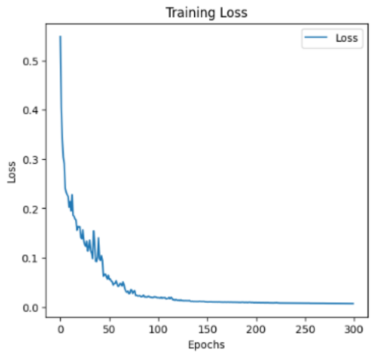


Sample Test Data



CNN Inverse Result

LCP Inverse



Model Parameters

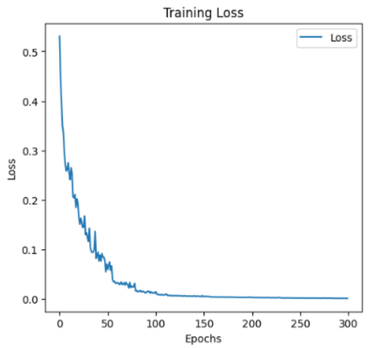
```
class CNN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv1d(in_channels=16, out_channels=32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * input_dim, 512)
        self.fc2 = nn.Linear(512, output_dim)
        self.elu = nn.ELU()

    def forward(self, x):
        x = x.unsqueeze(1)  # Add channel dimension
        x = self.elu(self.conv1(x))
        x = self.elu(self.conv2(x))
        x = self.elu(self.conv3(x))
        x = x.view(x.size(0), -1)  # Flatten
        x = self.elu(self.fc1(x))
        x = self.fc2(x)
        return x
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Performance Matrix:

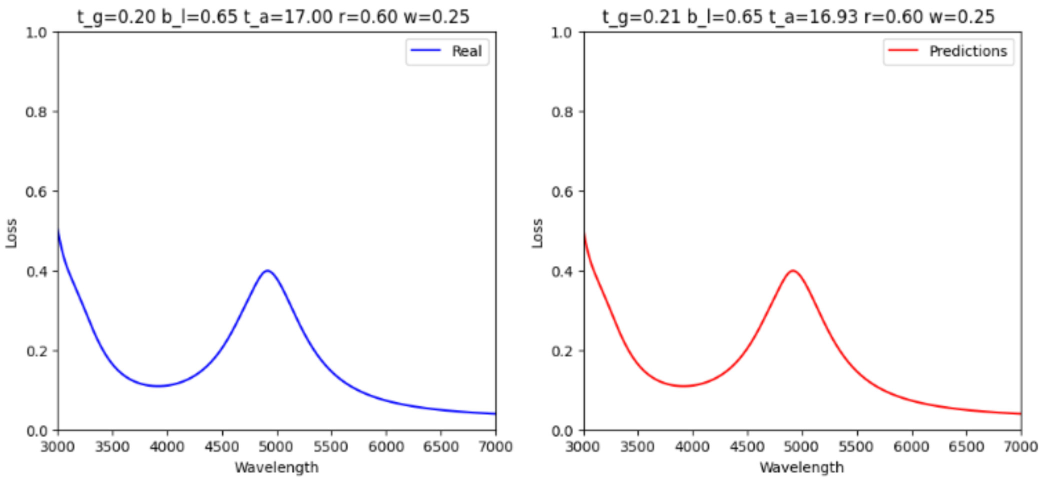
✓

1. R^2 Score
2. Time
3. GPU usage

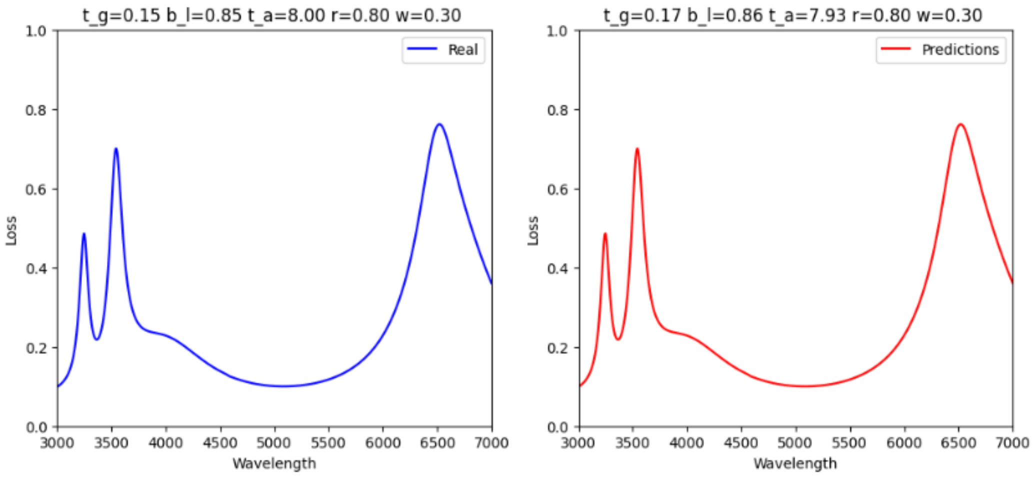
RCP Inverse



Sample Test Data

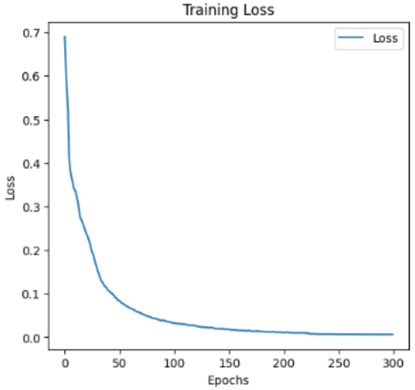


Sample Test Data



RNN Forward Result

LCP Forward



Model Parameters

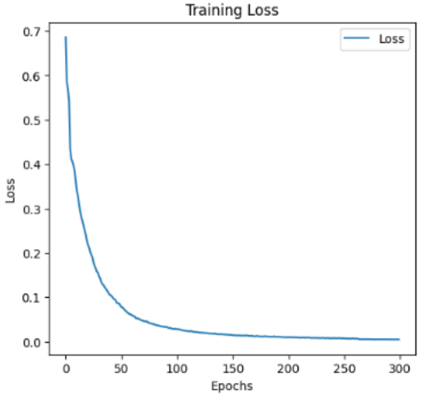
```python
class RNN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(RNN, self).__init__()
        self.rnn = nn.LSTM(input_dim, 128, num_layers=2, batch_first=True)
        self.fc1 = nn.Linear(128, 500)
        self.fc2 = nn.Linear(500, output_dim)
        self.elu = nn.ELU()

    def forward(self, x):
        x, _ = self.rnn(x.unsqueeze(1))  # Add channel dimension and pass through RNN
        x = x[:, -1, :]  # Get the last output from RNN
        x = self.elu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```python
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

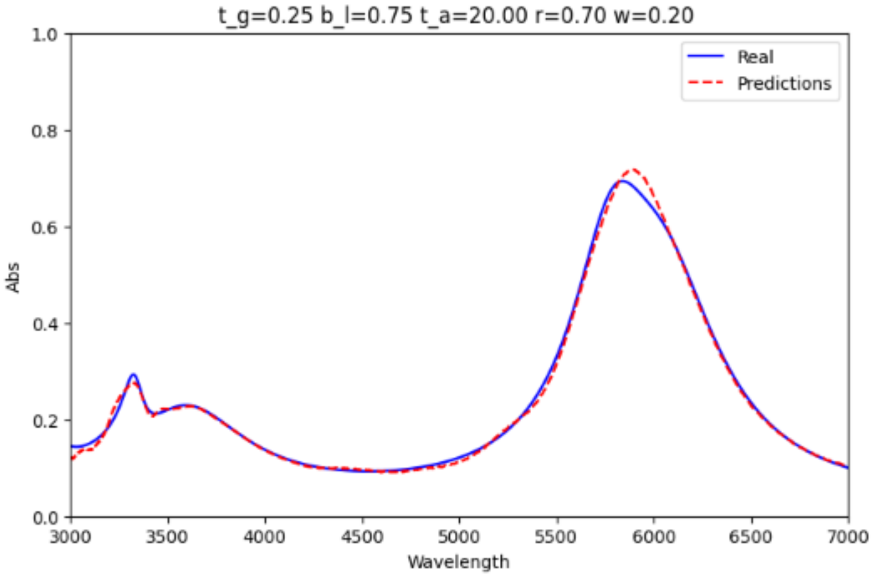
Performance Matrix:

1. R^2 Score
2. Time
3. GPU usage

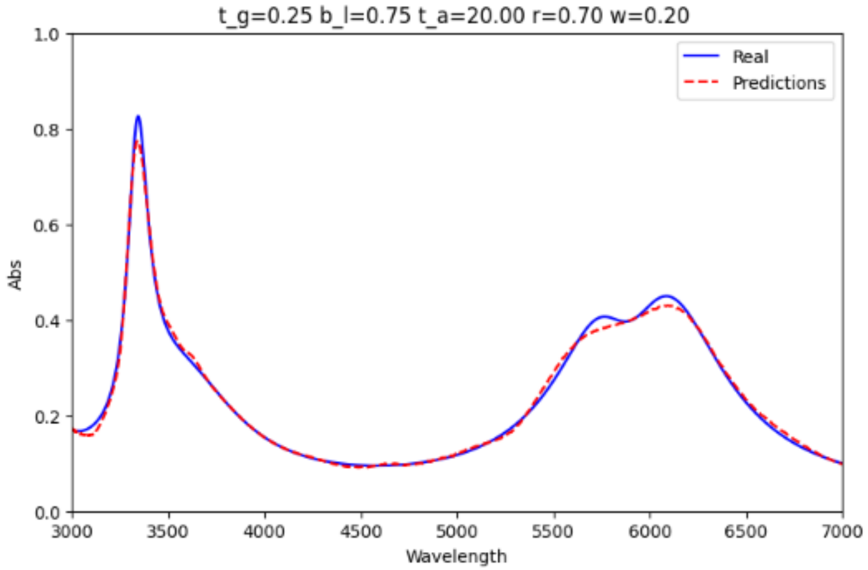
RCP Forward



Sample Test Data



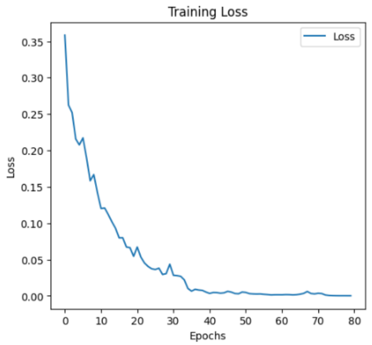
Sample Test Data



RNN Inverse Result

DL-GPU Team 3

LCP Inverse



Model Parameters

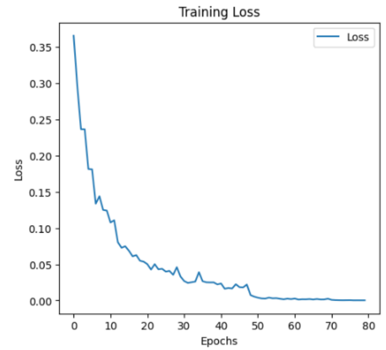
```python
class RNN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(RNN, self).__init__()
        self.rnn = nn.LSTM(input_dim, 2000, num_layers=2, batch_first=True)
        self.fc1 = nn.Linear(2000, 128)
        self.fc2 = nn.Linear(128, output_dim)
        self.elu = nn.ELU()

    def forward(self, x):
        x, _ = self.rnn(x.unsqueeze(1))  # Add channel dimension and pass through RNN
        x = x[:, -1, :]  # Get the last output from RNN
        x = self.elu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```python
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

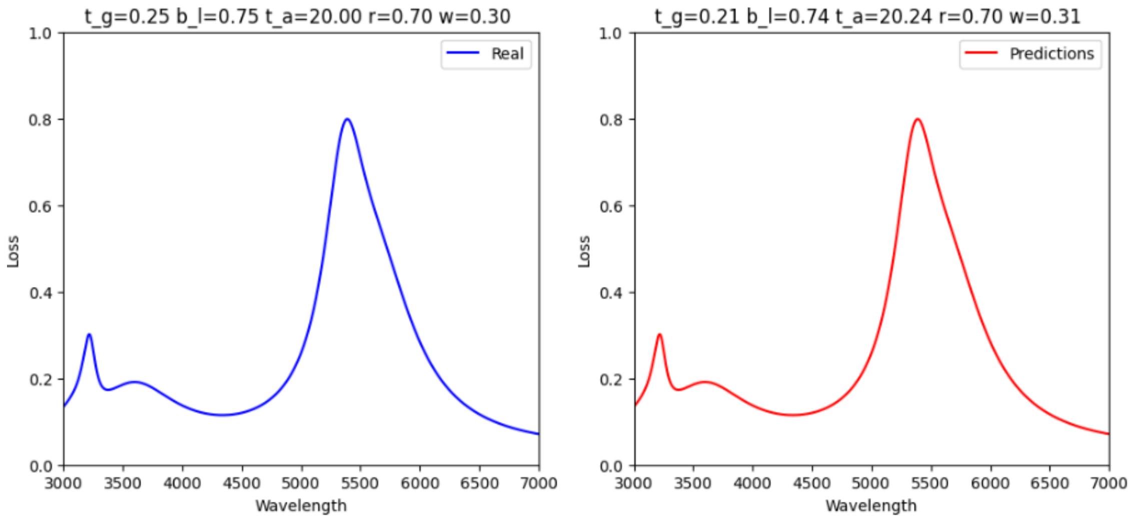
Performance Matrix:

1. R^2 Score
2. Time
3. GPU usage

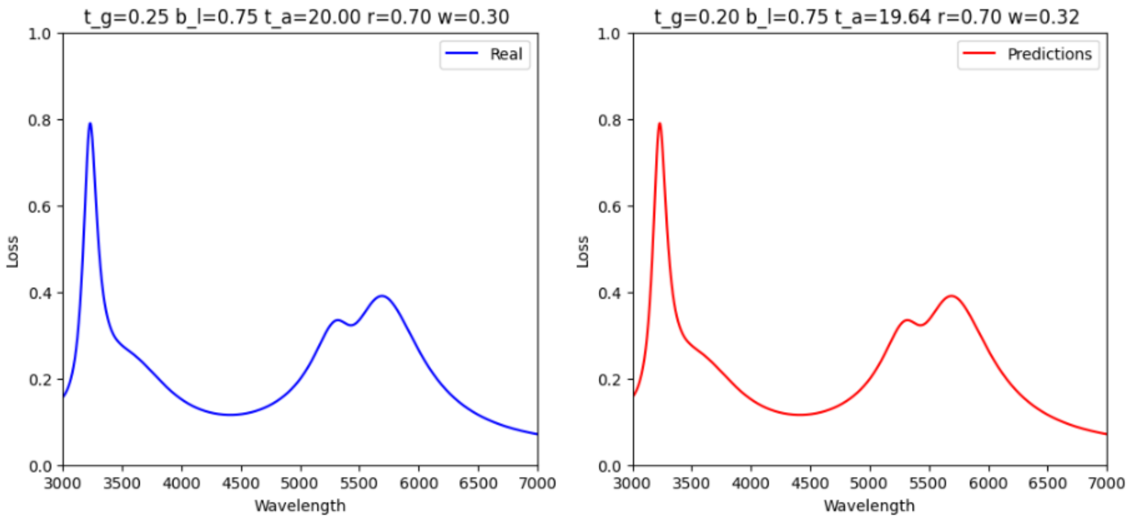
RCP Inverse



Sample Test Data



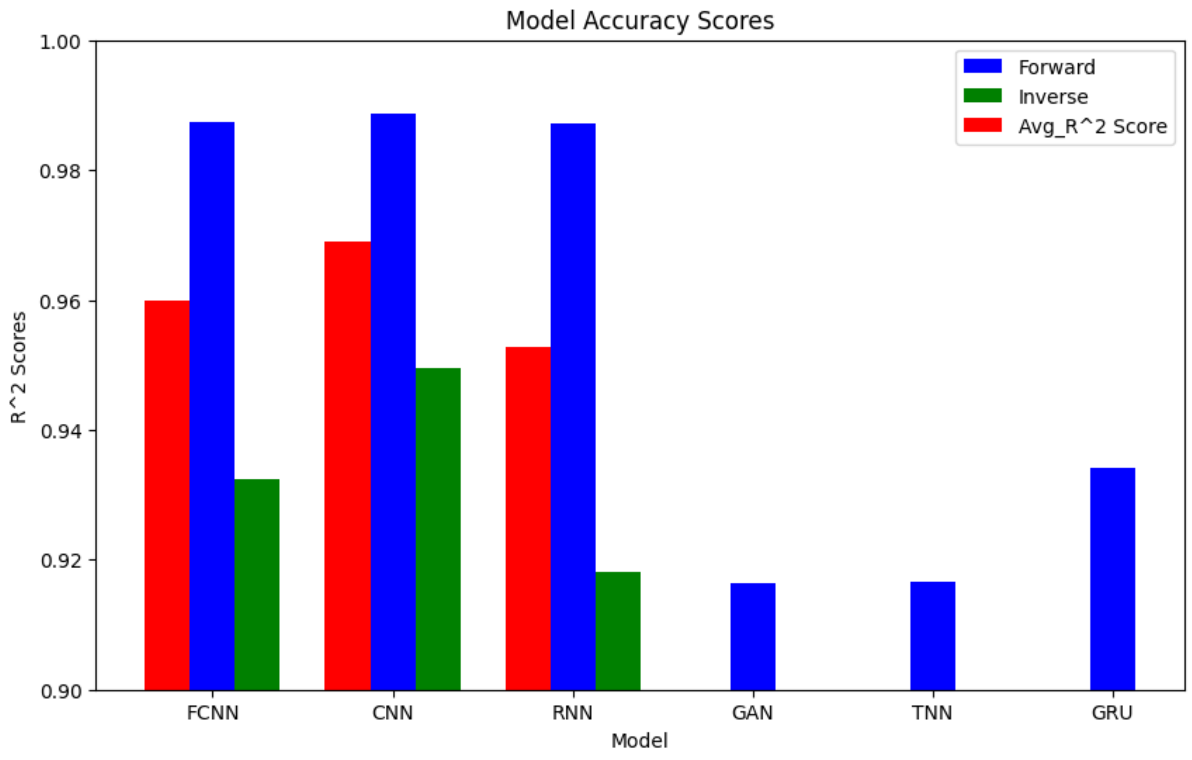
Sample Test Data



Models Performance Comparison

The accuracy performance for different models

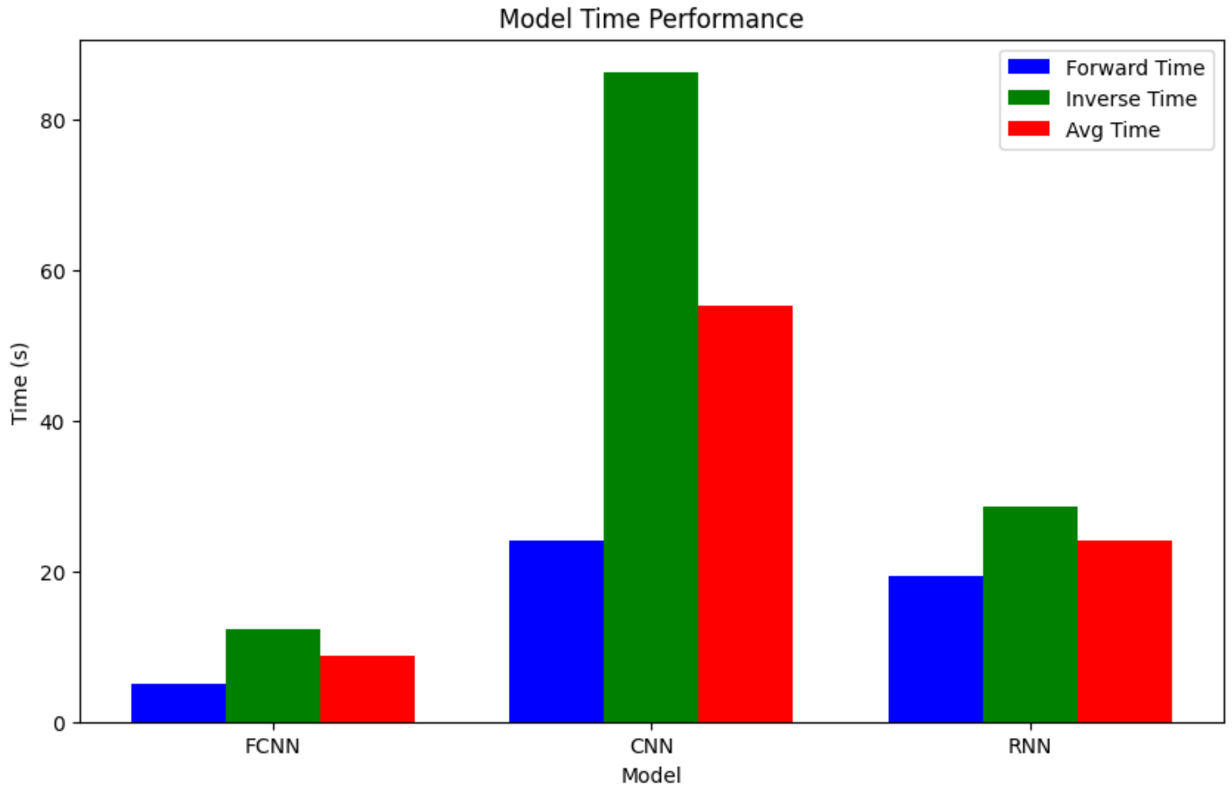
Model	Forward	Inverse	Avg_R^2 Score
FCNN	0.9873	0.9324	0.95985
CNN	0.9886	0.94945	0.969025
RNN	0.98725	0.91815	0.9527
GAN	0.91644	NAN	NAN
TNN	0.91663	NAN	NAN
GRU	0.93409	NAN	NAN



The time performance for different models

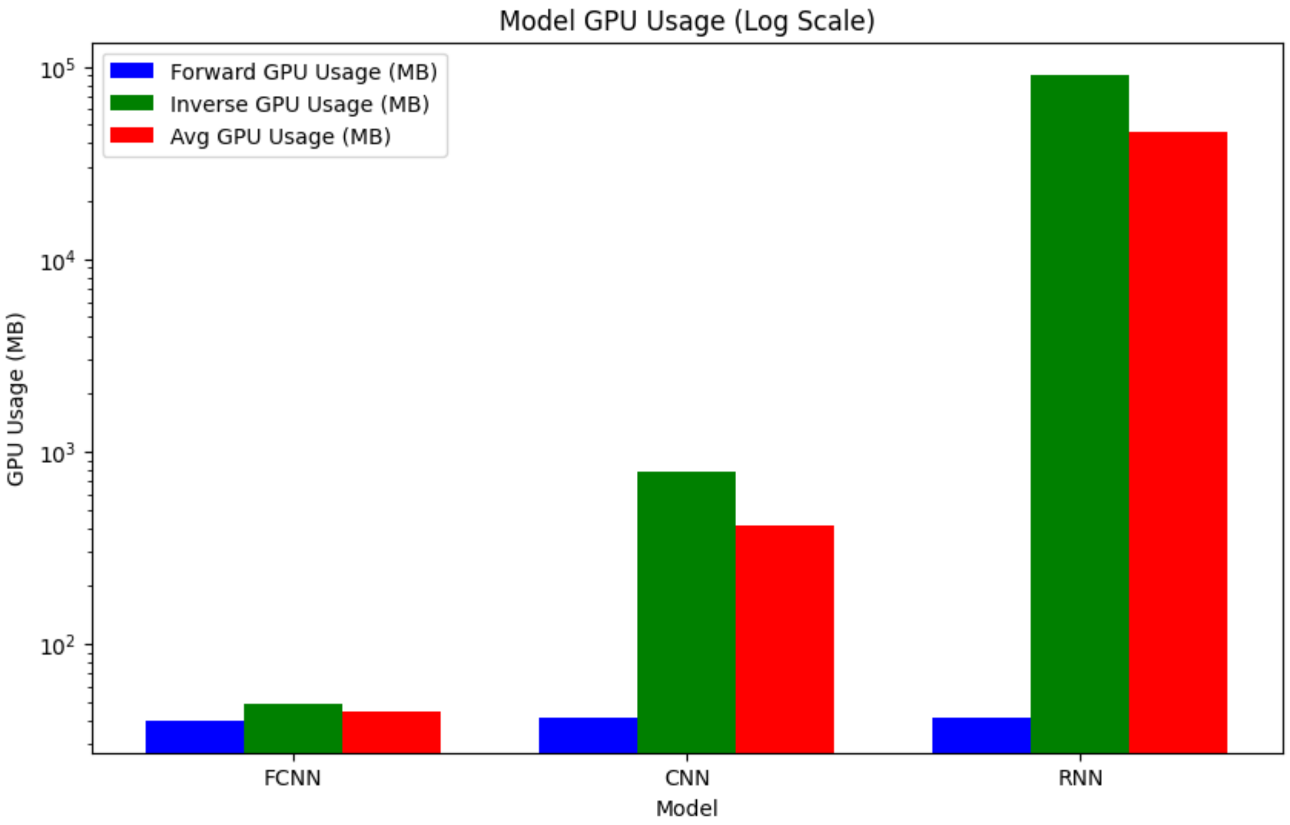
DL-GPU Team 3

Model	Forward Time (S)	Inverse Time (S)	Avg Time (S)
FCNN	5.035	12.39	8.71
CNN	24.15	86.3	55.22
RNN	19.39	28.59	23.99

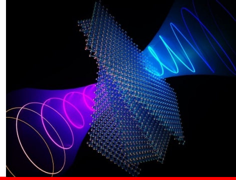


The GPU usage performance for different models

Model	Forward GPU Usage (MB)	Inverse GPU Usage (MB)	Avg GPU Usage (MB)
FCNN	39.99	48.64	44.315
CNN	41.35	780	410.695
RNN	41.56	91129	45585.3



Future Plan



- 1. Collect more data for training.

- 2. Use FCNN as the main model and combine Forward Model and Inverse Model to train the data.

- 3. Improve the accuracy by tuning the hyperparameters.

Thank you for your attention !

Question ?

Presenter

Jiang Chen | Nithin Shyam Soundararajan | Xiangkai Zeng