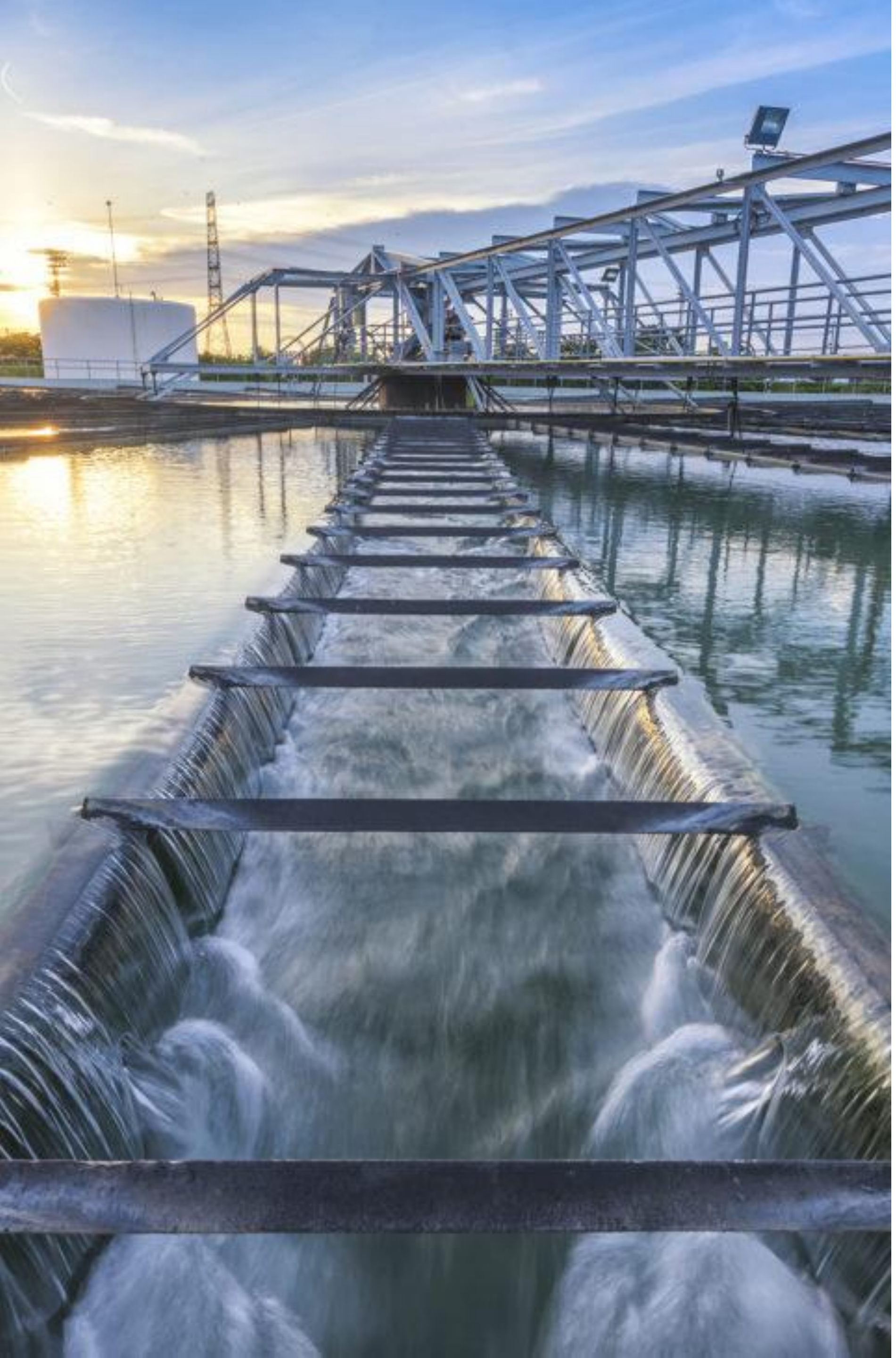


Water Supply Prediction

(Review: K-water #2 AI Competition)

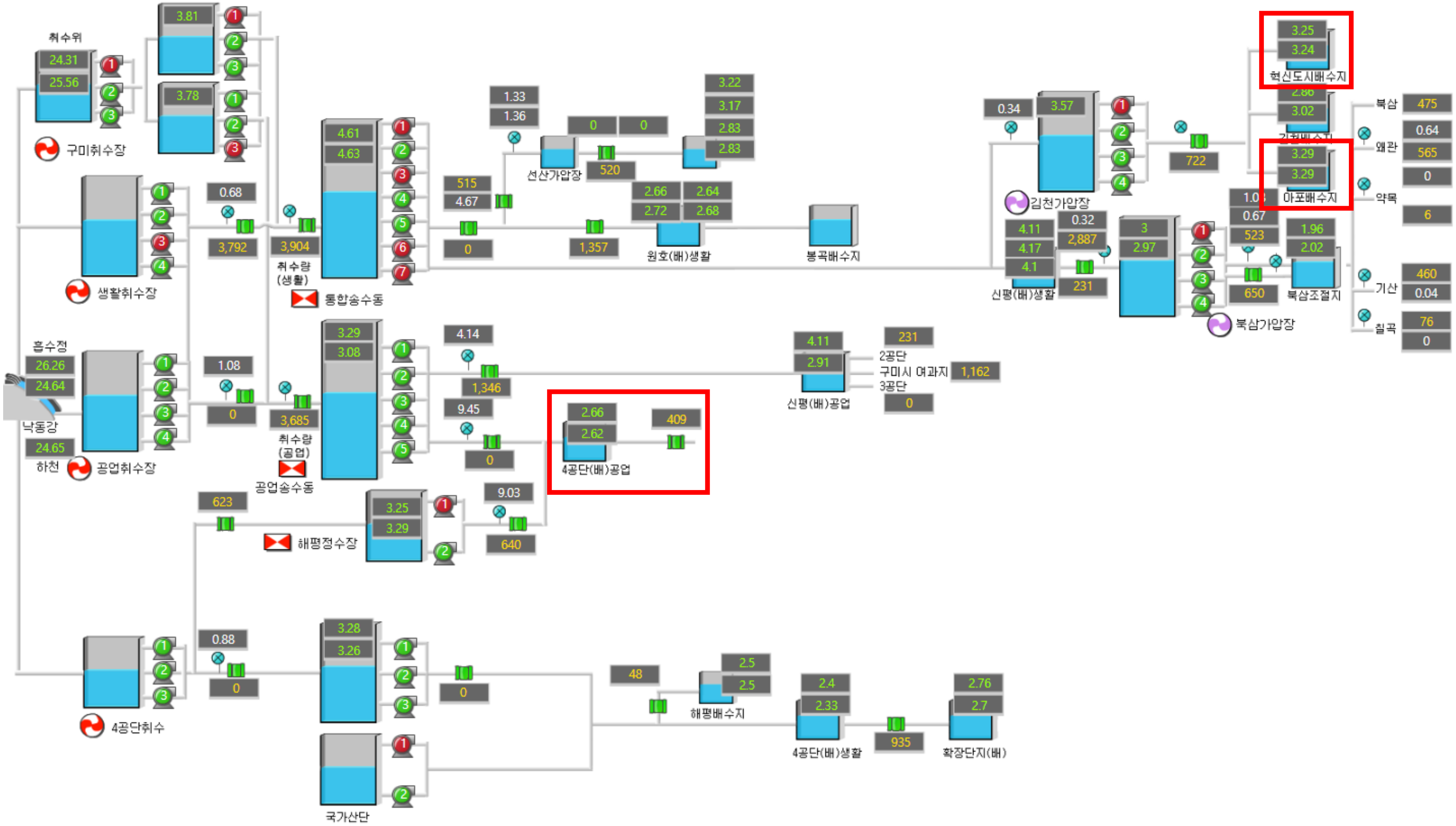
주경원

K-water 연구관리처 AI연구센터

2023년 5월 23일 (화)

0. K-water AI 경진대회

- 구미권 광역상수도 용수계통도



0. K-water AI 경진대회

- 구미권 배수지의 용수공급량 예측 알고리즘
  - Task #1: 행정 및 주거지역 (경북 김천시 율곡동)
  - Task #2: 농업지역 (경북 김천시 아포읍)
  - Task #3: 산업단지 (경북 구미시 산동읍, 장천면, 양포동)
- 미래 유출유량 적산차 예측 (2주, 336시간)
- 기대효과
  - 용수 공급량 예측을 통한 가압장 운영 최적화 및 에너지비용 절감
  - 물산업 AI기술 선도 기관으로서 K-water의 대국민 위상 제고

0. K-water AI 경진대회

- 자료기간 및 훈련/테스트 기간의 설정

Train				Test (Public, Private)	
2017	2018	2019	2020	2021	2022

목표값(y) 공개 (2017–2020)

목표값(y) 비공개 (2021–2022)

- BUT, 시계열 모델은 입력자료로 목표값이 활용되어야 함

시간	1시	2시	3시	4시	5시	6시	7시	8시	9시	10시	11시	12시	13시	…
참값(y)	3	4	5	4	3	4	5	4	3	4	5	4	3	
6시에 예측	입력자료 (참값 필요)						예측구간							
7시에 예측	입력자료 (참값 필요)							예측구간						
…														
12시에 예측	입력자료 (참값 필요)												예측구간	

0. K-water AI 경진대회

- 리더보드 구간(2021~2022년) 최종 설계방향
  - 2021년(Public): 리더보드 구간이지만 정답도 함께 공개
  - 2022년(Private): 대회 중 제출이 없으며, 추후 Inference 코드로 검증

2021년 (Public, 공개)	2022년 (Private, 비공개)

  - (장점) 실제 현장의 운영상황을 고려한 모델 설계가 가능
  - (단점) 참가자들의 Data Leakage 실수가 잦을 것으로 예상됨
    - 실제로 많은 참가자들의 기권 및 탈락사유가 됨
  - (단점) 순위권 참가자들의 Private 점수 채점이 자동으로 불가능
  - (단점) 정답이 공개되어 있어 Public 리더보드 모니터링 강화 필요

0. K-water AI 경진대회

- 시계열 예측 연구추이
  - (ARMA) ARIMA, SARIMA, …
  - (RNN) RNN, LSTM, GRU, …
  - Transformers
    - Informer, FEDformer, Autoformer, YFormer, PatchTST, …
  - Simple Linear
    - Nlinear, Dlinear

- 경진대회 우수 알고리즘
  - Nlinear
  - SCInet
  - (ensembled) Ridge, Lasso
  - Autoformer
  - Hybrid (CNN + Transformer)

1. Algorithms

K water AI Lab

Search | Browse State-of-the-Art | Datasets | Methods | More | Sign In

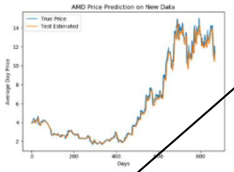
Time Series

Time Series Forecasting

Edit

254 papers with code • 14 benchmarks • 17 datasets

Time Series Forecasting is the task of fitting a model to historical, time-stamped data in order to predict future values. Traditional approaches include moving average, exponential smoothing, and ARIMA, though models as various as RNNs, Transformers, or XGBoost can also be applied. The most popular benchmark is the ETTh1 dataset. Models are typically evaluated using the Mean Square Error (MSE) or Root Mean Square Error (RMSE).



(Image credit: ThaiBinh Nguyen)

Benchmarks

Add a Result

These leaderboards are used to track progress in Time Series Forecasting

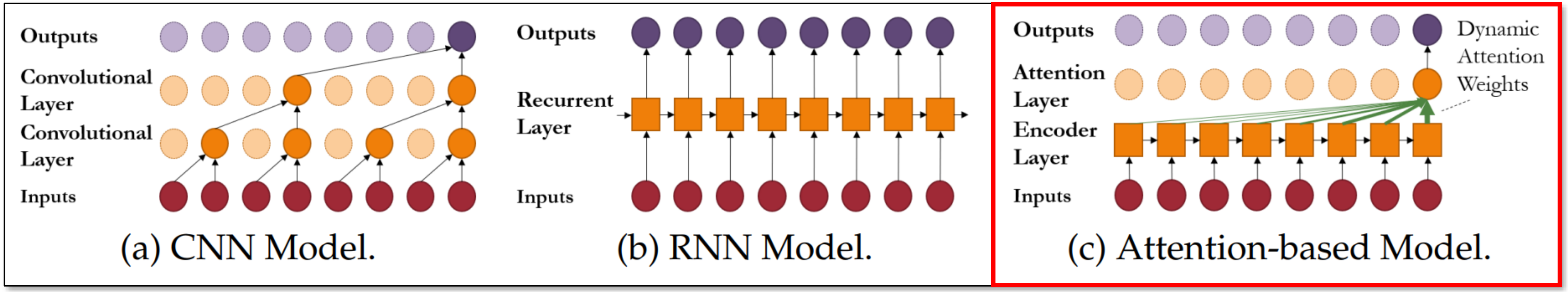
Trend	Dataset	Best Model	Paper	Code	Compare
	ETTh1 (720)	NLinear(univariate)			See all
	ETTh1 (336)	NLinear			See all
	ETTh2 (720)	NLinear			See all
	ETTh2 (336)	SCINet (Univariate)			See all
	ETTh1 (24)	DLinear			See all
	ETTh1 (48)	NLinear			See all
	ETTh1 (168)	NLinear			See all
	ETTh2 (24)	SCINet (Univariate)			See all
	ETTh2 (48)	SCINet (Univariate)			See all
	ETTh2 (168)	SCINet (Univariate)			See all

Content

Rank	Model	MAE	MSE	Paper	Code	Result	Year	Tags
1	NLinear (univariate)	0.226	0.080	Are Transformers Effective for Time Series Forecasting?			2022	
2	FiLM (univariate)	0.240	0.09	FiLM: Frequency improved Legendre Memory Model for Long-term Time Series Forecasting			2022	
3	DLinear	0.274		Are Transformers Effective for Time Series Forecasting?			2022	
4	SCINet (Univariate)	0.316	0.156	SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction			2021	
5	QuerySelector	0.3730	0.2136	Long-term series forecasting with Query Selector -- efficient model of sparse attention			2021	
6	Yformer	0.394	0.226	Yformer: U-Net Inspired Transformer Architecture for Far Horizon Time Series Forecasting			2021	
7	Transformer	0.4213	0.2501	Long-term series forecasting with Query Selector -- efficient model of sparse attention			2021	
8	Informer	0.435	0.269	Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting			2020	
9	NLinear (Multivariate)	0.453	0.440	Are Transformers Effective for Time Series Forecasting?			2022	
10	FiLM (Multivariate)	0.472	0.465	FiLM: Frequency improved Legendre Memory Model for Long-term Time Series Forecasting			2022	
11	SCINet (Multivariate)	0.582	0.612	SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction			2021	
12	ARIMA	0.766	0.659	Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting			2020	

1. Algorithms

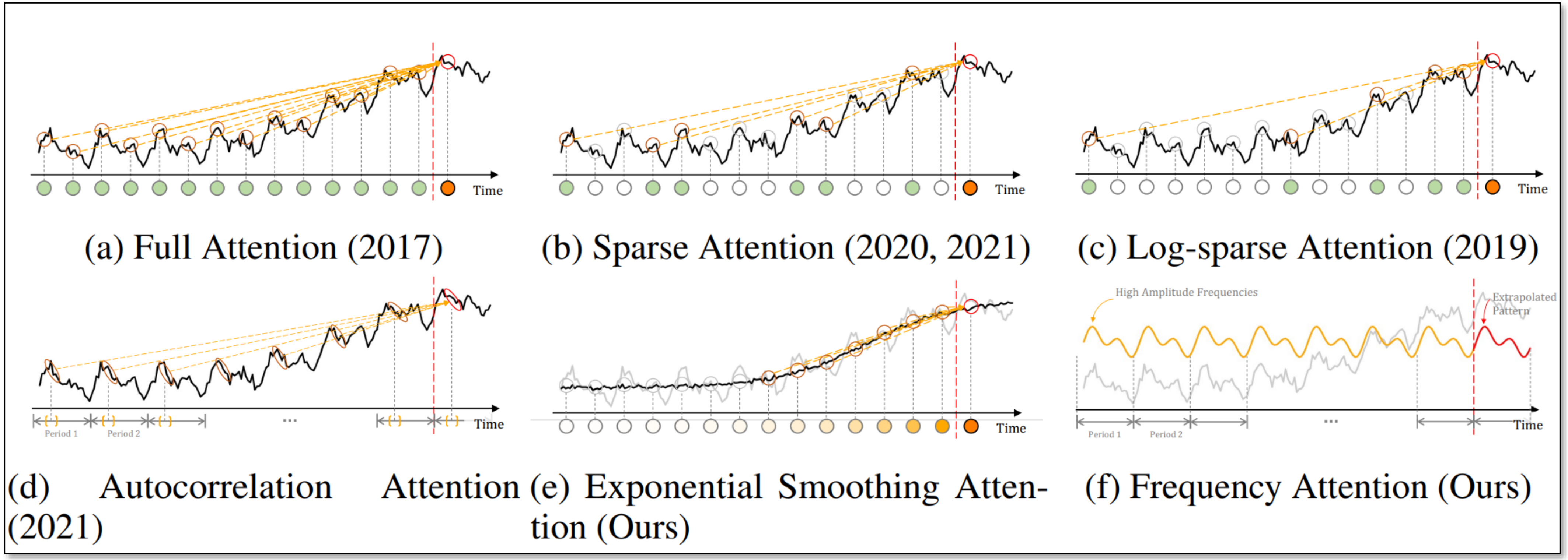
- CNN : 먼 과거의 정보를 추출하기 위해 ConvNet 활용
- RNN : 고전적인 딥러닝 접근 방법 (gradient vanishing)
- Attention : 최근 SOTA 모델의 기본 구조
  - Autoformer, Informer, Yformer, FEDformer 등등



Bryan and Zohren (2021), "Time Series Forecasting with Deep Learning", Philosophical Transaction of the Royal Society

1. Algorithms

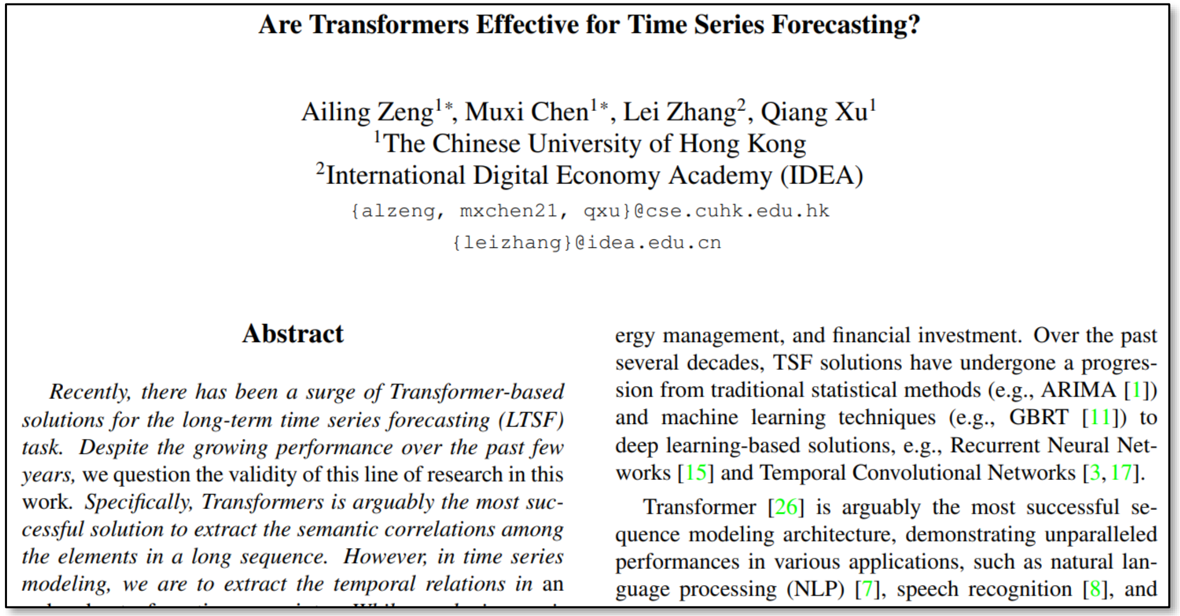
- Attention-Based Transformers



Woo et al. (2023) ETSformer: Exponential Smoothing Transformers for Time-series Forecasting (ICLR 2023)

1.1 Linear Model

- Linear : Simple one-layer linear model
- Nlinear : 입력 sequence 마지막 값을 빼서 간단한 정규화
- DLinear : Trend 고려하여 decomposition 수행



Are Transformers Effective for Time Series Forecasting?

Ailing Zeng[1*], Muxi Chen[1*], Lei Zhang[2], Qiang Xu[1]
[1]The Chinese University of Hong Kong
[2]International Digital Economy Academy (IDEA)
{alzeng, mxchen21, qxu}@cse.cuhk.edu.hk
{leizhang}@idea.edu.cn

Abstract

Recently, there has been a surge of Transformer-based solutions for the long-term time series forecasting (LTSF) task. Despite the growing performance over the past few years, we question the validity of this line of research in this work. *Specifically, Transformers is arguably the most successful solution to extract the semantic correlations among the elements in a long sequence. However, in time series modeling, we are to extract the temporal relations in an*

ergy management, and financial investment. Over the past several decades, TSF solutions have undergone a progression from traditional statistical methods (e.g., ARIMA [1]) and machine learning techniques (e.g., GBRT [11]) to deep learning-based solutions, e.g., Recurrent Neural Networks [15] and Temporal Convolutional Networks [3, 17].

Transformer [26] is arguably the most successful sequence modeling architecture, demonstrating unparalleled performances in various applications, such as natural language processing (NLP) [7], speech recognition [8], and

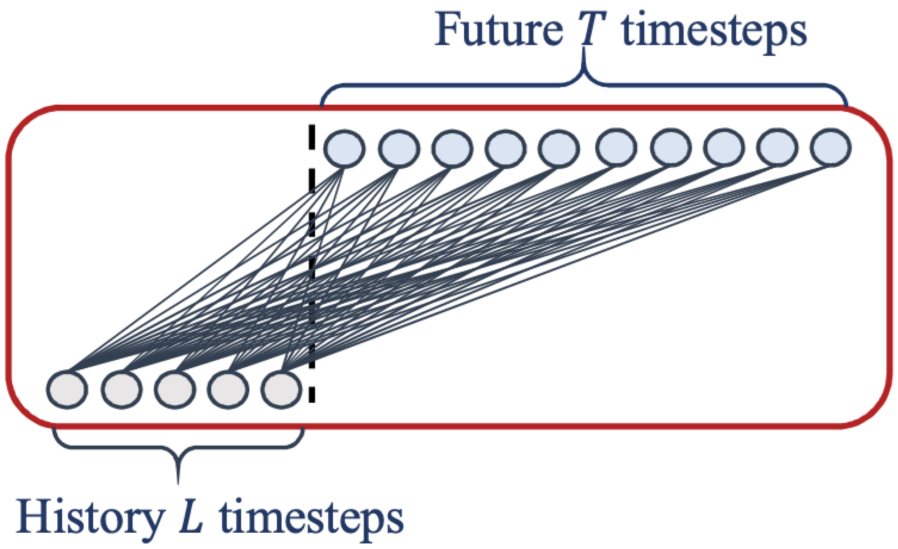


Figure 2: Illustration of the basic linear model.

Zeng et al., Are Transformers Effective for Time Series Forecasting? (AAAI 2023)

1.2 Autoformer

- Auto-correlation을 활용한 Transformer
- Trend와 Seasonal 분리하여 적용

Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting

Haixu Wu, Jiehui Xu, Jianmin Wang, Mingsheng Long (✉)
School of Software, BNRist, Tsinghua University, China
{whx20,xjh20}@mails.tsinghua.edu.cn, {jimwang,mingsheng}@tsinghua.edu.cn

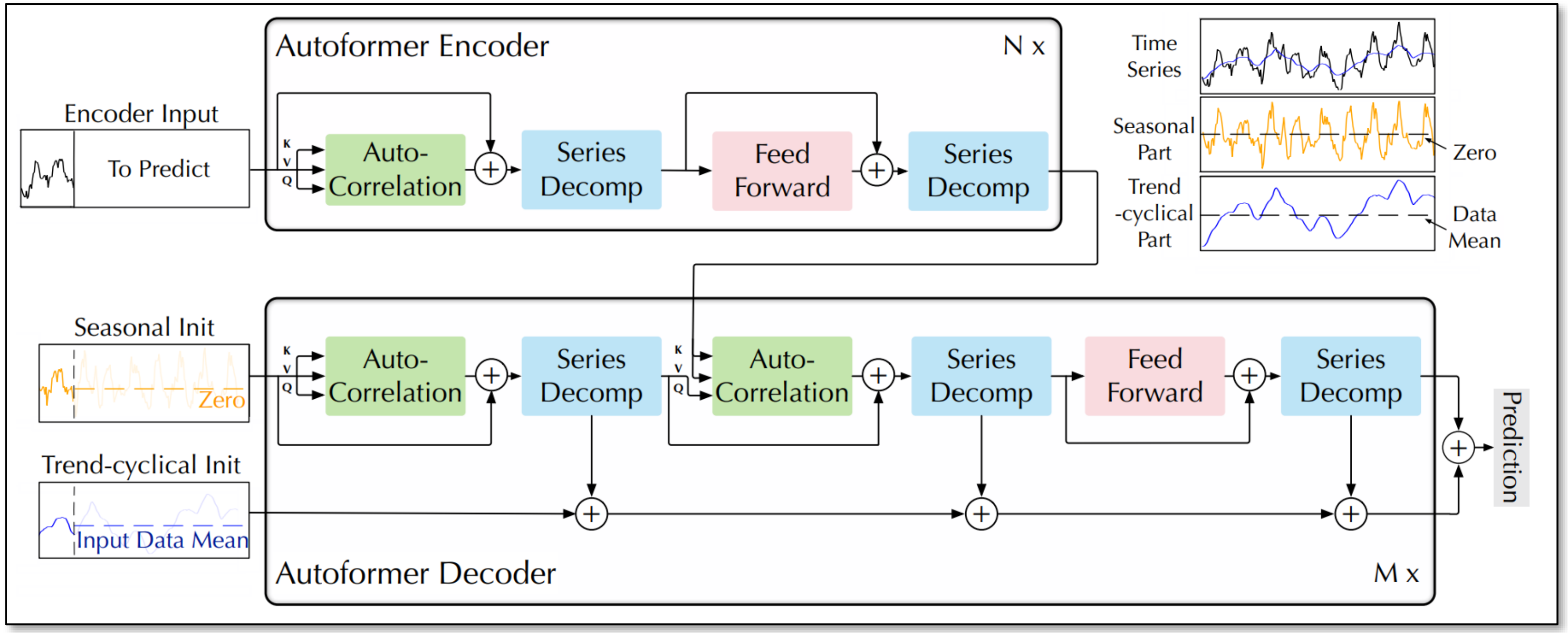
Abstract

Extending the forecasting time is a critical demand for real applications, such as extreme weather early warning and long-term energy consumption planning. This paper studies the *long-term forecasting* problem of time series. Prior Transformer-based models adopt various self-attention mechanisms to discover the long-range dependencies. However, intricate temporal patterns of the long-term future prohibit the model from finding reliable dependencies. Also, Transformers have to adopt the sparse versions of point-wise self-attentions for long series efficiency, resulting in the information utilization bottleneck. Going beyond Transformers, we design *Autoformer* as a novel decomposition architecture with an *Auto-Correlation* mechanism. We break with the pre-processing convention of series decomposition and renovate it as a basic inner block of deep models. This design empowers Autoformer with

Wu et al. Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting (NeurIPS 2021)

1.2 Autoformer

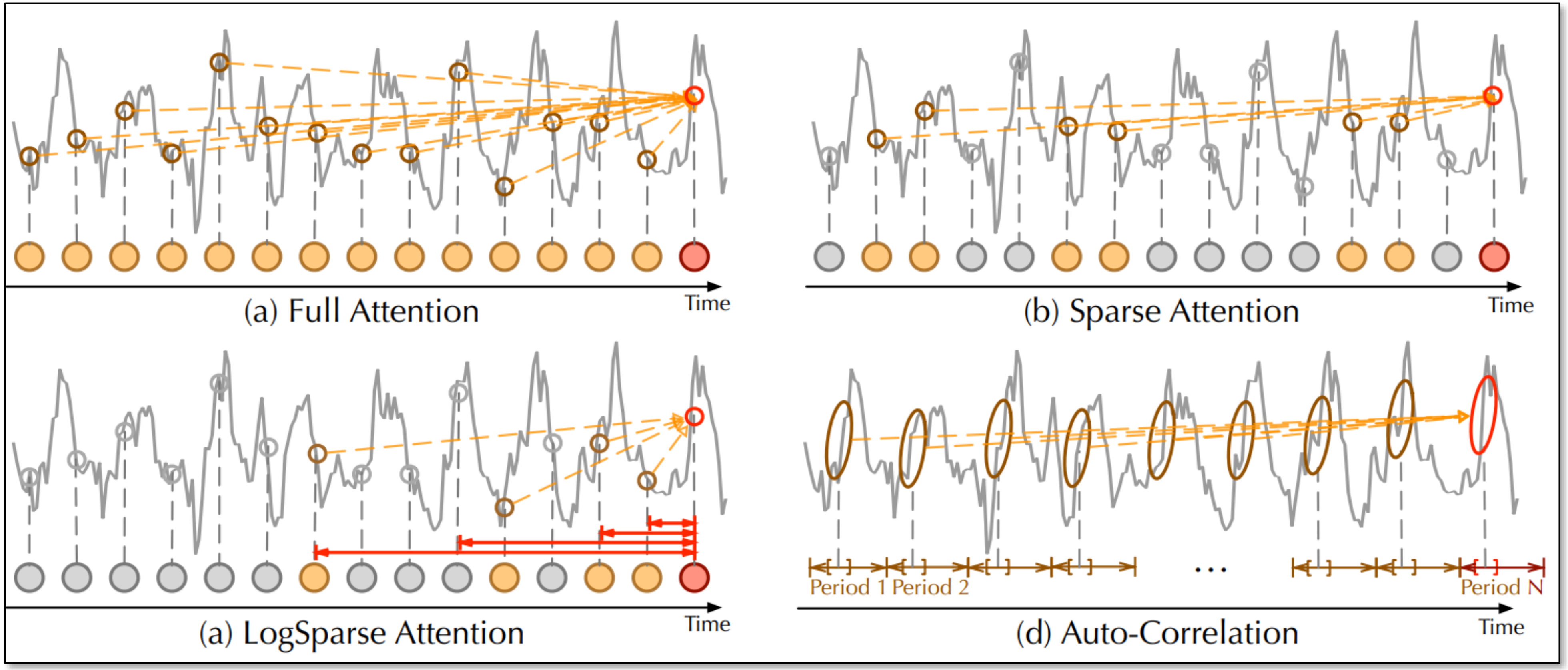
- Decoder의 입력에서 Trend와 Seasonal을 분리
- Positional Encoding 자리에 Auto-correlation 레이어
- Encoder, Decoder에 Series Decomp 블록을 사용



Wu et al. Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting (NeurIPS 2021)

1.2 Autoformer

- Series 레벨에서의 dependency를 찾아 attention을 수행



Wu et al. Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting (NeurIPS 2021)

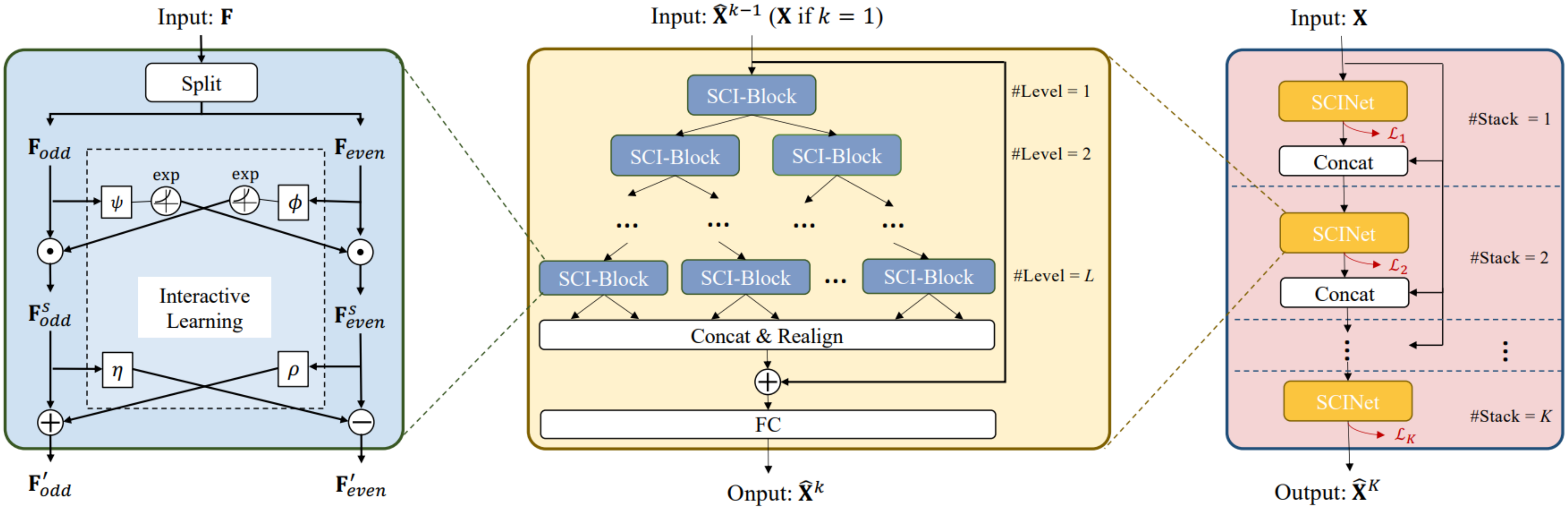
1.3 SCINet

- Temporal Convolution Network의 개선형
  - Sample convolution and interaction network (SCINet)
  - 다양한 시계열 해상도로부터 반복적으로 추출 및 교환하는 계층적 구조
  - 계층적 구조를 위해 기본 단위인 SCI블럭을 사용
    - 입력 데이터를 다운샘플링하고, 특징들을 추출하는 역할을 수행
  - ETT 데이터셋에서 기존 SOTA인 Informer, Yformer를 갱신함

Liu et al., SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction. (NeurIPS 2022)

1.3 SCINet

- SCINet Architecture

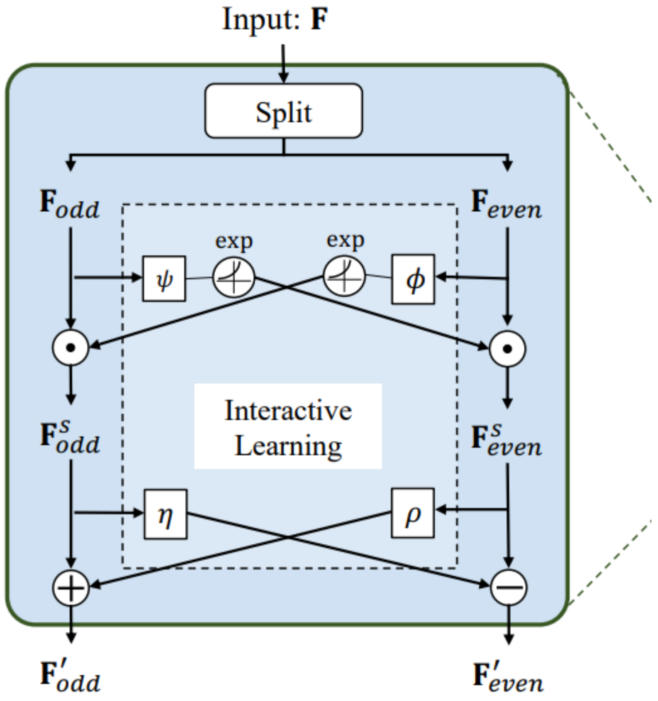


(a) SCI-Block (b) SCINet (c) Stacked SCINet

Liu et al., SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction. (NeurIPS 2022)

1.3 SCINet

- SCI-Block
  - Split 및 Interaction Module
    - Split : 짝수/홀수 번째 sequence로 구분
    - Interaction : 각 convolution의 정보 교환
  - $F_{odd}^{s} = F_{odd} \odot \exp(\phi(F_{even}))$
  - $F^{`}_{odd} = F_{odd}^{s} + \rho(F_{even}^{s})$

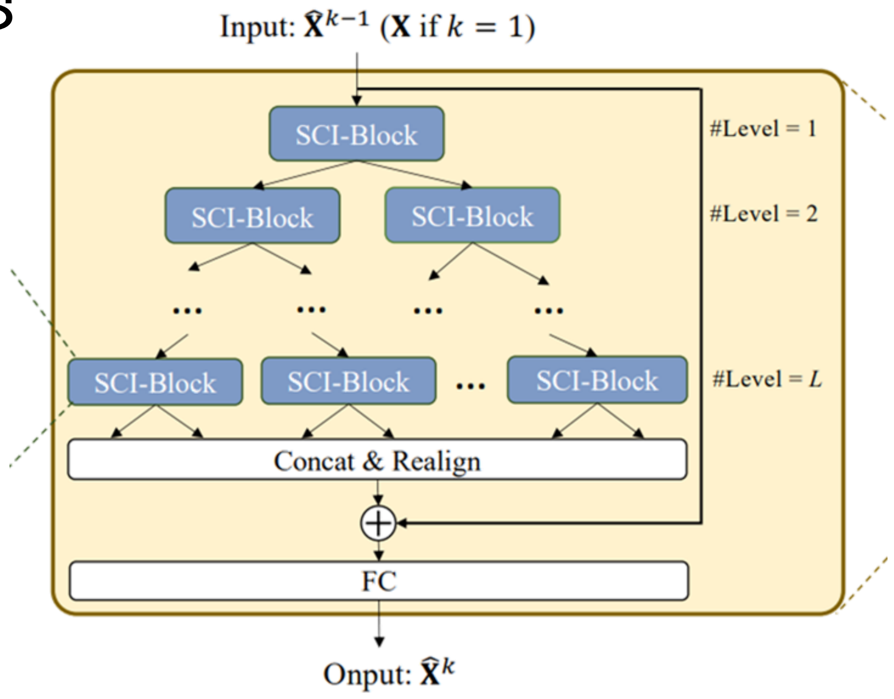


(a) SCI-Block

Liu et al., SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction. (NeurIPS 2022)

1.3 SCINet

- SCINet
  - Level이 올라갈수록 long-term 정보가 반영
    - Level 1 (1, 3, 5, 7, ⋯)
    - Level 2 (1, 5, 9, 13, ⋯)
    - Level 3 (1, 9, 17, 25, ⋯)
  - L Level까지 진행 후 residual connection
  - 출력(예측) 시계열 길이로 변환 후 예측

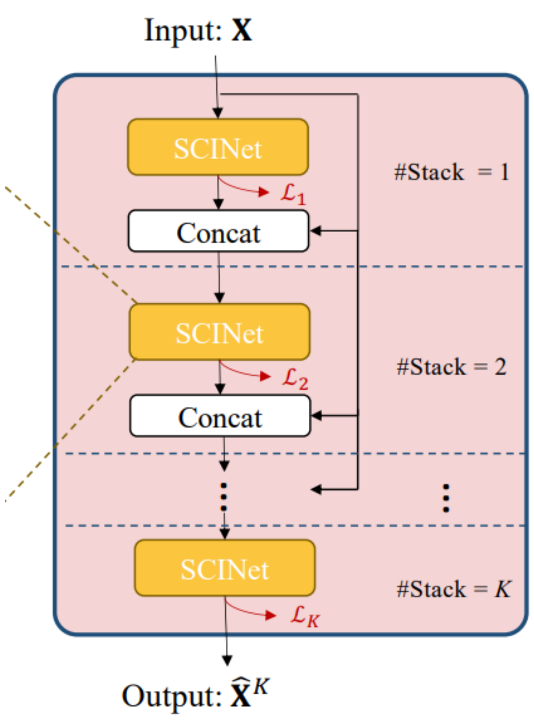


(b) SCINet

Liu et al., SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction. (NeurIPS 2022)

1.3 SCINet

- Stacked SCINet
  - SCINet을 K개 만큼 stack
  - $\mathcal{L}_k = \frac{1}{\tau}\sum_{i=0}^{\tau}\left\|\hat{x}_i^{\,k} - x_i\right\|$
  - $\mathcal{L} = \sum_{k=1}^{K} \mathcal{L}_k$



(c) Stacked SCINet

Liu et al., SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction. (NeurIPS 2022)

2. Benchmark

Time Series Forecasting on ETTh1 (336)

Rank	Model	MAE	MSE	Paper	Code	Result	Year	Tags
1	**NLinear**	0.226	0.081	Are Transformers Effective for Time Series Forecasting?			2022	
2	**FiLM** (Univariate)	0.229	0.083	FiLM: Frequency improved Legendre Memory Model for Long-term Time Series Forecasting			2022	
3	**SCINet** (Univariate)	0.231	0.087	SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction			2021	
4	**DLinear**	0.244	0.098	Are Transformers Effective for Time Series Forecasting?			2022	
5	**QuerySelector**	0.2844	0.1267	Long-term series forecasting with Query Selector -- efficient model of sparse attention			2021	
6	**Transformer**	0.3201	0.1541	Long-term series forecasting with Query Selector -- efficient model of sparse attention			2021	
7	**Yformer**	0.365	0.195	Yformer: U-Net Inspired Transformer Architecture for Far Horizon Time Series Forecasting			2021	
8	**Informer**	0.387	0.222	Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting			2020	
9	**FiLM** (Multivariate)	0.445	0.442	FiLM: Frequency improved Legendre Memory Model for Long-term Time Series Forecasting			2022	
10	**SCINet** (Multivariate)	0.494	0.491	SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction			2021	
11	**ARIMA**	0.593	0.468	Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting			2020	

3. Code Review

```
print(f"Train Data Shape: {train_df.shape}")
print(f"Test Data Shape: {test_df.shape}")
```
[11] ✓ 0.0s

```
Train Data Shape: (35063, 2)
Test Data Shape: (8424, 2)
```

```
train_df
```
[12] ✓ 0.0s

	datetime	구미 혁신도시배수지 유출유량 적산차
0	2017-01-01 01:00:00	138.0
1	2017-01-01 02:00:00	237.0
2	2017-01-01 03:00:00	128.0
3	2017-01-01 04:00:00	14.0
4	2017-01-01 05:00:00	11.0
...	...	...
35058	2020-12-31 19:00:00	328.0
35059	2020-12-31 20:00:00	347.0
35060	2020-12-31 21:00:00	335.0
35061	2020-12-31 22:00:00	141.0
35062	2020-12-31 23:00:00	112.0

35063 rows × 2 columns

```
test_df
```
[13] ✓ 0.0s

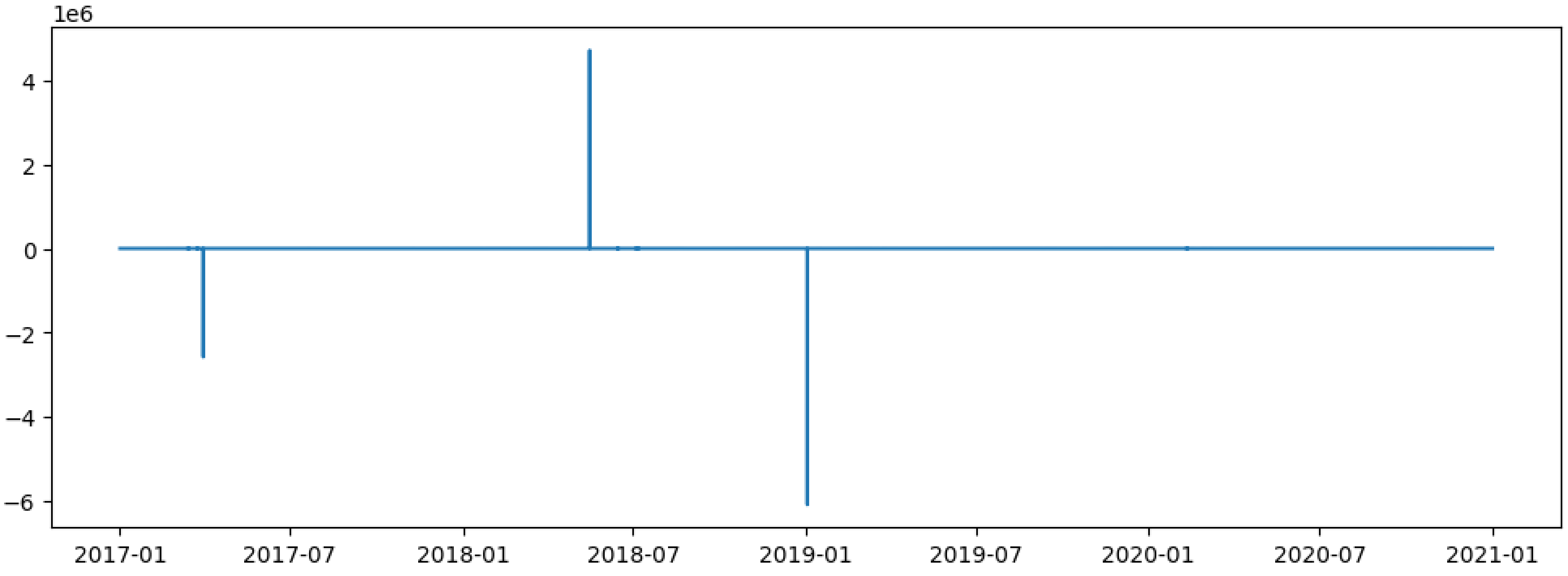
	datetime	구미 혁신도시배수지 유출유량 적산차
0	2021-01-01 00:00:00	106.0
1	2021-01-01 01:00:00	184.0
2	2021-01-01 02:00:00	277.0
3	2021-01-01 03:00:00	197.0
4	2021-01-01 04:00:00	72.0
...	...	...
8419	2021-12-17 19:00:00	327.0
8420	2021-12-17 20:00:00	513.0
8421	2021-12-17 21:00:00	396.0
8422	2021-12-17 22:00:00	350.0
8423	2021-12-17 23:00:00	197.0

8424 rows × 2 columns

3. Code Review

```
plt.figure(figsize=(12,4))
plt.plot(train_df['value']);
```

[33] ✓ 0.1s



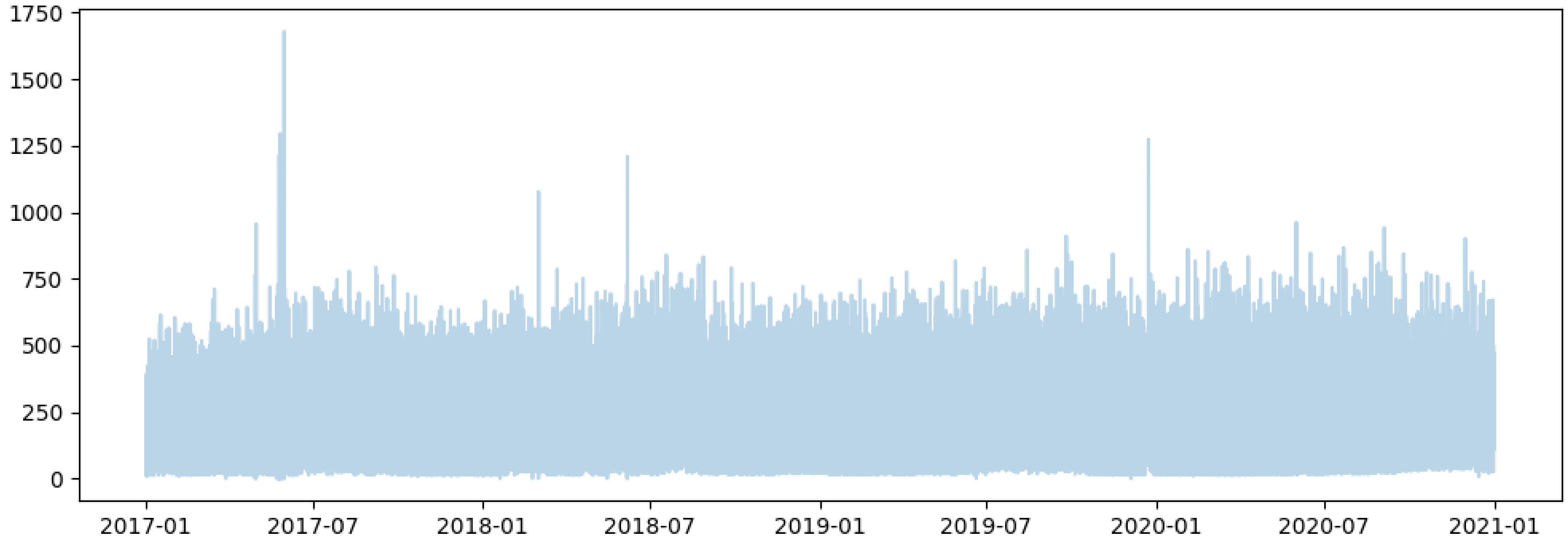
3. Code Review

```
train_df[train_df['value'] > 2000] = np.NaN
train_df[train_df['value'] < 0] = np.NaN
train_df['value'] = train_df['value'].interpolate()
```

[34] ✓ 0.0s

```
plt.figure(figsize=(12,4))
plt.plot(train_df['value'], alpha=0.3);
```

[35] ✓ 0.2s

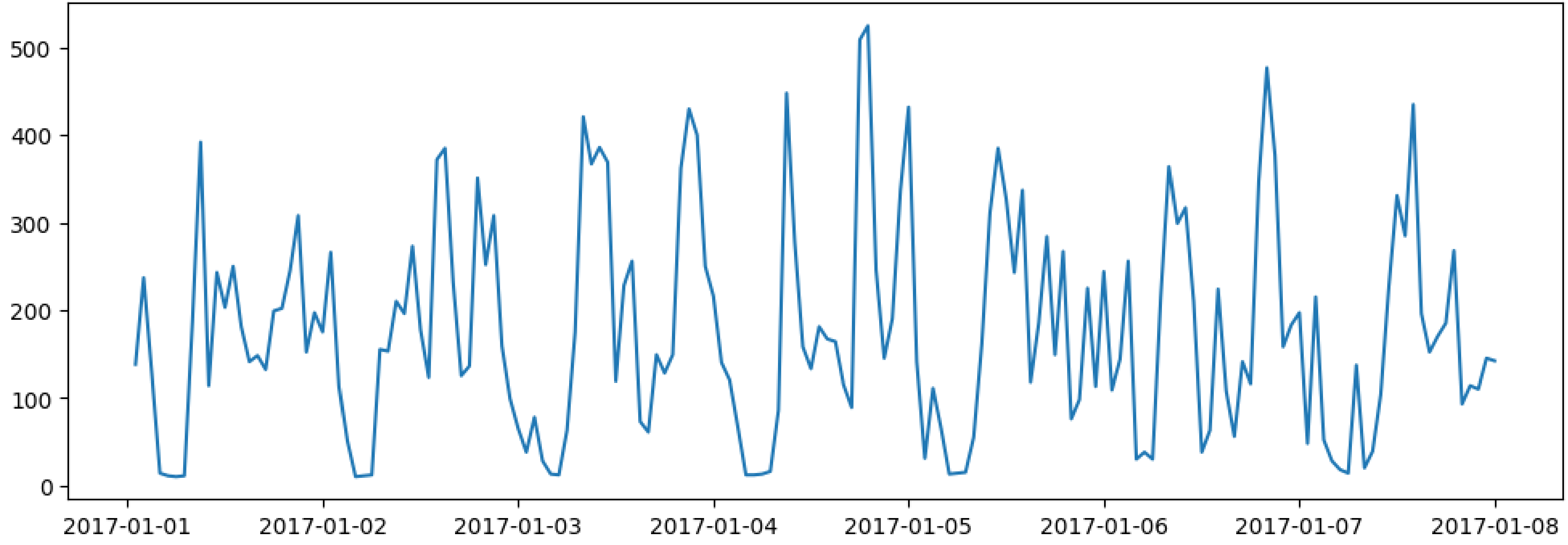


3. Code Review

```
plt.figure(figsize=(12,4))
plt.plot(train_df['value'][:168]);
```

[40] ✓ 0.1s

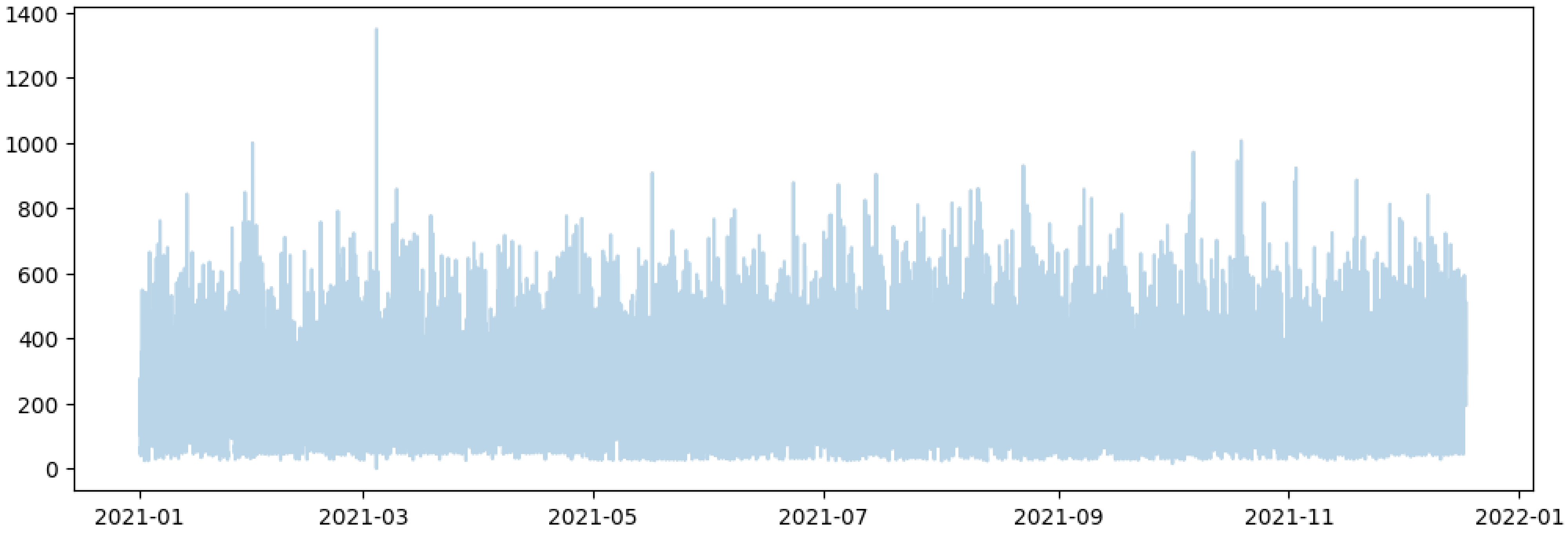
...



3. Code Review

```
plt.figure(figsize=(12,4))
plt.plot(test_df['value'], alpha=0.3);
```

[41] ✓ 0.1s

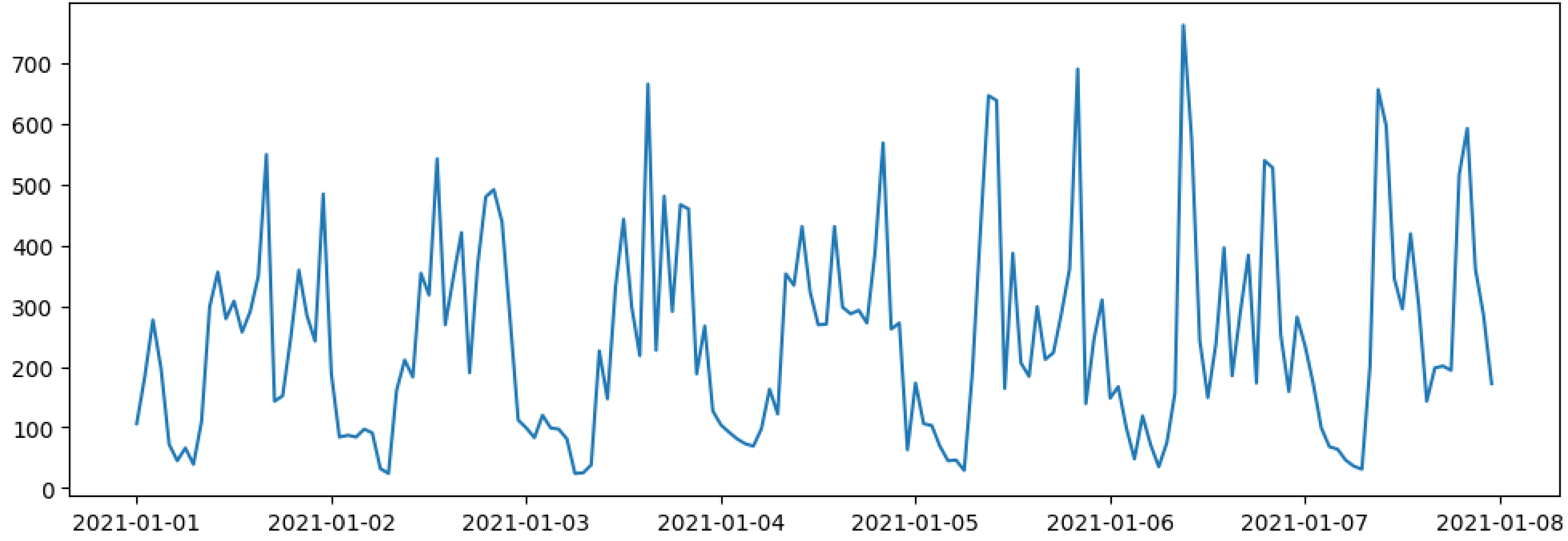


3. Code Review

```
plt.figure(figsize=(12,4))
plt.plot(test_df['value'][:168]);
```

[42] ✓ 0.1s

...



3. Code Review

- K-Fold Strategy

K-Folds Strategy

- Weights More Recent Years
- Latter is Larger

```python
train_year = [(2017, 2018), (2018, 2019), (2019, 2020), (2017, 2018, 2019), (2018, 2019, 2020)]
valid_year = [2019, 2020, 2018, 2020, 2017]
```

[17] ✓ 0.0s Python

```python
w = {2017:1, 2018:2, 2019:3, 2020:4, 2021:5}

weights = []
for years in train_year:
    weight = 0
    for year in years:
        weight += w[year]
    weights.append(weight)

weights = np.array(weights)/np.sum(weights)

print(weights)
```

[18] ✓ 0.0s Python

```
[0.1        0.16666667 0.23333333 0.2        0.3       ]
```

3. Code Review

- General Configurations
  - Loss: L1 (MAE)
  - Optimizer : AdamW
  - Scheduler : ReduceLRonPlateau
  - Framework : Pytorch (2.0)
    - CUDA Device : 3080 Ti Laptop

3.1 Linear Model

General Configuration

```
CFG = {
    'EPOCH' : 30, # Over 30 is recommended
    'WINDOW' : 24*28,
    'BATCH_SIZE' : 64,
    'LEARNING_RATE' : 1e-3,
    'device' : 'cuda' if torch.cuda.is_available() else 'cpu'
}
```

[5] ✓ 0.0s

```
# Reset Seed Function for Reproducibility
def seed_everything(SEED=42):
    print(f'RESET SEED {SEED}')
    random.seed(SEED)
    np.random.seed(SEED)
    random.seed(SEED)
    torch.manual_seed(SEED)
    torch.cuda.manual_seed(SEED)
```

[6] ✓ 0.0s

3.1 Linear Model

```
class TimeDataset(Dataset):
    def __init__(self, data, seq_len=24*14, pred_len=336, isTest=False):
        super(TimeDataset, self).__init__()
        self.data = data
        self.seq_len = seq_len
        self.pred_len = pred_len
        self.isTest = isTest

    def __len__(self):
        if self.isTest:
            return len(self.data)-self.seq_len+1
        return len(self.data) - self.seq_len - self.pred_len + 1

    def __getitem__(self, idx):
        x = self.data.iloc[idx:idx+self.seq_len, 1:2].values
        if self.isTest:
            return x
        y = self.data.iloc[idx+self.seq_len:idx+self.seq_len+self.pred_len, 1:2].values
        _y = self.data.iloc[idx+self.seq_len:idx+self.seq_len+self.pred_len, 2].values
        return x, y, _y
```

[20] ✓ 0.0s

3.1 Linear Model

```python
class Model(nn.Module):
    """
    Normalization-Linear
    """
    def __init__(self, configs):
        super(Model, self).__init__()
        self.seq_len = configs.seq_len
        self.pred_len = configs.pred_len

        # Use this line if you want to visualize the weights
        # self.Linear.weight = nn.Parameter((1/self.seq_len)*torch.ones([self.pred_len,self.seq_len]))
        self.channels = configs.enc_in
        self.individual = configs.individual
        if self.individual:
            self.Linear = nn.ModuleList()
            for i in range(self.channels):
                self.Linear.append(nn.Linear(self.seq_len,self.pred_len))
        else:
            self.Linear = nn.Linear(self.seq_len, self.pred_len)

    def forward(self, x):
        # x: [Batch, Input length, Channel]
        seq_last = x[:,-1:,:].detach()
        x = x - seq_last
        if self.individual:
            output = torch.zeros([x.size(0),self.pred_len,x.size(2)],dtype=x.dtype).to(x.device)
            for i in range(self.channels):
                output[:,:,i] = self.Linear[i](x[:,:,i])
            x = output
        else:
            x = self.Linear(x.permute(0,2,1)).permute(0,2,1)
        x = x + seq_last
        return x # [Batch, Output length, Channel]
```

```python
class Model(nn.Module):
    """
    Normalization-Linear (Simple Linear Network)
    """
    def __init__(self, seq_len, pred_len=24*14):
        super(Model, self).__init__()
        self.seq_len = seq_len
        self.pred_len = pred_len
        self.projection = nn.Sequential(
            nn.Linear(self.seq_len, self.pred_len*2),
            nn.LeakyReLU(),
            nn.Linear(self.pred_len*2, self.pred_len*3),
            nn.LeakyReLU(),
            nn.Dropout(0.5),
            nn.Linear(self.pred_len*3, self.pred_len*2),
            nn.LeakyReLU(),
            nn.Linear(self.pred_len*2, self.pred_len)
        )

    def forward(self, x):
        # x: [Batch, Input length, Channel]
        seq_last = x[:,-1:,:].detach()
        x = x - seq_last
        x = self.projection(x.permute(0,2,1)).permute(0,2,1)
        x = x + seq_last
        return x # [Batch, Output length, Channel]
```

[21] ✓ 0.0s

https://github.com/cure-lab/ltsf-linear

3.1 Linear Model

```
def train_one_epoch(model, train_loader, optimizer, loss_fn, scaler):
    model.train()
    running_loss = []
    prog_bar = tqdm(enumerate(train_loader), total=len(train_loader))
    for batch, (x_input, y_true, _) in prog_bar:
        optimizer.zero_grad()
        x_input = x_input.to(CFG['device'], torch.float)
        y_true = y_true.to(CFG['device'], torch.float)
        with torch.autocast(device_type=CFG['device'], dtype=torch.float16 if CFG['device'] == 'cuda' else torch.bfloat16):
            preds = model(x_input)
            loss = loss_fn(preds, y_true)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        running_loss.append(loss.item())
        prog_bar.set_description(f"loss: {np.mean(running_loss):.4f}")

    return running_loss
```

3.1 Linear Model

```python
def train(train_dataset, valid_dataset, fold):
    model = Model(CFG['WINDOW']).to(CFG['device'])
    loss_fn = nn.L1Loss()
    # loss_fn = WeightedMAE(alpha=1)
    optimizer = torch.optim.AdamW(model.parameters(), lr=CFG['LEARNING_RATE'])
    scaler = torch.cuda.amp.GradScaler()

    train_loader = DataLoader(train_dataset, CFG['BATCH_SIZE'], shuffle=True, drop_last=True)
    valid_loader = DataLoader(valid_dataset, CFG['BATCH_SIZE'], shuffle=False)
    # scheduler = CosineAnnealingWarmRestarts(optimizer, 4, T_mult=1, eta_min=1e-5)
    scheduler = ReduceLROnPlateau(optimizer, factor=0.25, patience=3)
    best_mae = 1e20
    for e in range(CFG['EPOCH']):
        train_loss = train_one_epoch(model, train_loader, optimizer, loss_fn, scaler)
        valid_loss, loss2 = valid_one_epoch(model, valid_loader, loss_fn)
        scheduler.step(np.mean(valid_loss))
        if best_mae>np.mean(valid_loss):
            best_mae=np.mean(valid_loss)
            torch.save(model.state_dict(), f"best_model_{fold}")
            print(f"save best model at epoch{e+1}")

    log = {
        "model" : "Nlinear",
        "window" : CFG['WINDOW'],
        "lr" : CFG['LEARNING_RATE'],
        "mae" : best_mae,
    }

    with open("model_result.txt", "a") as f:
        f.write(f"{log}\n")

    return
```

[23] ✓ 0.0s

3.1 Linear Model

```
all_preds = [np.concatenate(x) for x in all_preds]
weighted_pred=[weights[i]*p for i,p in enumerate(all_preds)]
```

[25] ✓ 0.0s

```
weighted_pred[0].shape
```

[36] ✓ 0.0s

```
(8425, 336)
```

3.1 Linear Model

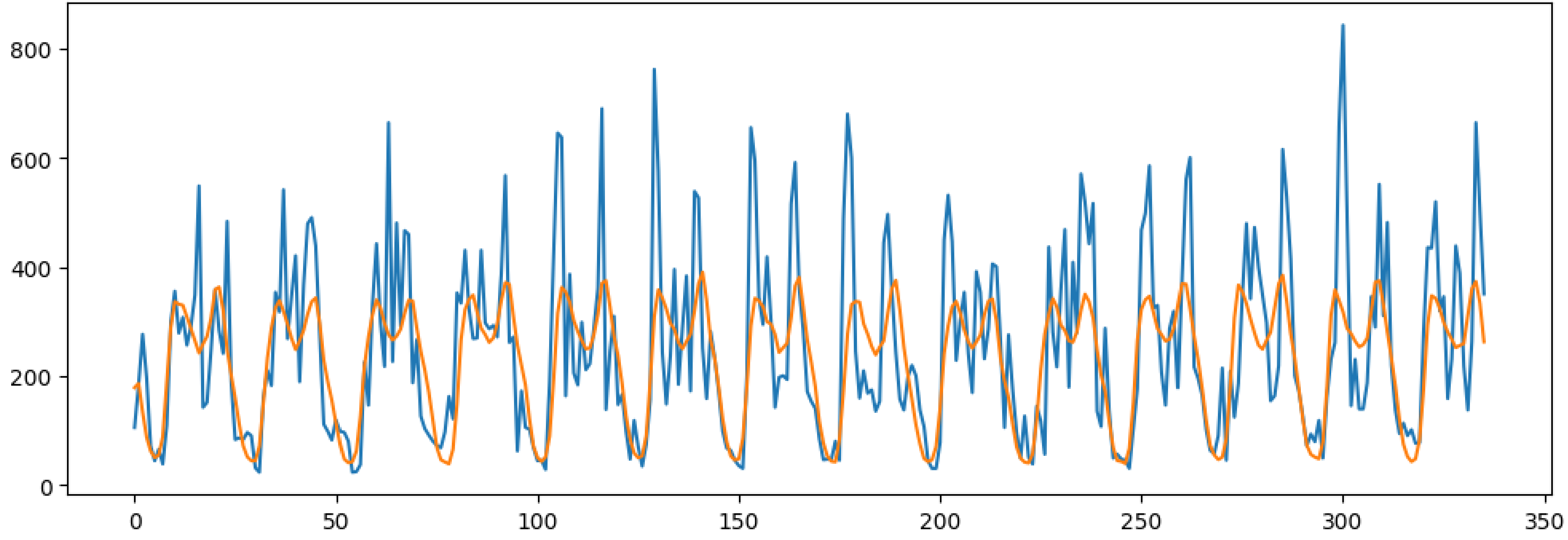
```
pred_temp = test_df[CFG['WINDOW']:CFG['WINDOW']+(24*14)].reset_index(drop=True)
pred_temp['pred'] = pd.DataFrame(np.sum(weighted_pred, axis=0)).iloc[0,:]
```

[30] ✓ 0.0s

```
plt.figure(figsize=(12,4))
plt.plot(pred_temp[['value', 'pred']]);
```

[32] ✓ 0.1s

...



3.2 SCINet

General Configuration

```
CFG = {
    'EPOCH' : 35, # Over 30 is recommended
    'WINDOW' : 24*21,
    'BATCH_SIZE' : 1024,
    'HIDDEN' : 8,
    'LEARNING_RATE' : 1e-3,
    'device' : 'cuda' if torch.cuda.is_available() else 'cpu'
}
```

[8] ✓ 0.0s

```
# Reset Seed Function for Reproducibility
def seed_everything(SEED=42):
    print(f'RESET SEED {SEED}')
    random.seed(SEED)
    np.random.seed(SEED)
    random.seed(SEED)
    torch.manual_seed(SEED)
    torch.cuda.manual_seed(SEED)
```

[3] ✓ 0.0s

3.2 SCINet

```python
class Splitting(nn.Module):
    def __init__(self):
        super(Splitting, self).__init__()

    def even(self, x):
        return x[:, ::2, :]

    def odd(self, x):
        return x[:, 1::2, :]

    def forward(self, x):
        '''Returns the odd and even part'''
        return (self.even(x), self.odd(x))
```

3.2 SCINet

```python
class Interactor(nn.Module):
    def __init__(self, in_planes, splitting=True,
                 kernel = 5, dropout=0.5, groups = 1, hidden_size = 1, INN = True):
        super(Interactor, self).__init__()
        self.modified = INN
        self.kernel_size = kernel
        self.dilation = 1
        self.dropout = dropout
        self.hidden_size = hidden_size
        self.groups = groups
        if self.kernel_size % 2 == 0:
            pad_l = self.dilation * (self.kernel_size - 2) // 2 + 1 #by default: stride==1
            pad_r = self.dilation * (self.kernel_size) // 2 + 1 #by default: stride==1

        else:
            pad_l = self.dilation * (self.kernel_size - 1) // 2 + 1 # we fix the kernel size of the second layer as 3.
            pad_r = self.dilation * (self.kernel_size - 1) // 2 + 1
        self.splitting = splitting
        self.split = Splitting()

        modules_P = []
        modules_U = []
        modules_psi = []
        modules_phi = []
        prev_size = 1

        size_hidden = self.hidden_size
        modules_P += [
            nn.ReplicationPad1d((pad_l, pad_r)),

            nn.Conv1d(in_planes * prev_size, int(in_planes * size_hidden),
                      kernel_size=self.kernel_size, dilation=self.dilation, stride=1, groups= self.groups),
            nn.LeakyReLU(negative_slope=0.01, inplace=True),

            nn.Dropout(self.dropout),
            nn.Conv1d(int(in_planes * size_hidden), in_planes,
                      kernel_size=3, stride=1, groups= self.groups),
            nn.Tanh()
        ]
        modules_U += [
            nn.ReplicationPad1d((pad_l, pad_r)),
            nn.Conv1d(in_planes * prev_size, int(in_planes * size_hidden),
                      kernel_size=self.kernel_size, dilation=self.dilation, stride=1, groups= self.groups),
            nn.LeakyReLU(negative_slope=0.01, inplace=True),
            nn.Dropout(self.dropout),
            nn.Conv1d(int(in_planes * size_hidden), in_planes,
                      kernel_size=3, stride=1, groups= self.groups),
            nn.Tanh()
        ]
```

```python
        modules_phi += [
            nn.ReplicationPad1d((pad_l, pad_r)),
            nn.Conv1d(in_planes * prev_size, int(in_planes * size_hidden),
                      kernel_size=self.kernel_size, dilation=self.dilation, stride=1, groups= self.groups),
            nn.LeakyReLU(negative_slope=0.01, inplace=True),
            nn.Dropout(self.dropout),
            nn.Conv1d(int(in_planes * size_hidden), in_planes,
                      kernel_size=3, stride=1, groups= self.groups),
            nn.Tanh()
        ]
        modules_psi += [
            nn.ReplicationPad1d((pad_l, pad_r)),
            nn.Conv1d(in_planes * prev_size, int(in_planes * size_hidden),
                      kernel_size=self.kernel_size, dilation=self.dilation, stride=1, groups= self.groups),
            nn.LeakyReLU(negative_slope=0.01, inplace=True),
            nn.Dropout(self.dropout),
            nn.Conv1d(int(in_planes * size_hidden), in_planes,
                      kernel_size=3, stride=1, groups= self.groups),
            nn.Tanh()
        ]
        self.phi = nn.Sequential(*modules_phi)
        self.psi = nn.Sequential(*modules_psi)
        self.P = nn.Sequential(*modules_P)
        self.U = nn.Sequential(*modules_U)
```

3.2 SCINet

```
class InteractorLevel(nn.Module):
    def __init__(self, in_planes, kernel, dropout, groups , hidden_size, INN):
        super(InteractorLevel, self).__init__()
        self.level = Interactor(in_planes = in_planes, splitting=True,
                    kernel = kernel, dropout=dropout, groups = groups, hidden_size = hidden_size, INN = INN)

    def forward(self, x):
        (x_even_update, x_odd_update) = self.level(x)
        return (x_even_update, x_odd_update)
```

3.2 SCINet

```
class SCINet_Tree(nn.Module):
    def __init__(self, in_planes, current_level, kernel_size, dropout, groups, hidden_size, INN):
        super().__init__()
        self.current_level = current_level


        self.workingblock = LevelSCINet(
            in_planes = in_planes,
            kernel_size = kernel_size,
            dropout = dropout,
            groups= groups,
            hidden_size = hidden_size,
            INN = INN)


        if current_level!=0:
            self.SCINet_Tree_odd=SCINet_Tree(in_planes, current_level-1, kernel_size, dropout, groups, hidden_size, INN)
            self.SCINet_Tree_even=SCINet_Tree(in_planes, current_level-1, kernel_size, dropout, groups, hidden_size, INN)

    def zip_up_the_pants(self, even, odd):
        even = even.permute(1, 0, 2)
        odd = odd.permute(1, 0, 2) #L, B, D
        even_len = even.shape[0]
        odd_len = odd.shape[0]
        mlen = min((odd_len, even_len))
        _ = []
        for i in range(mlen):
            _.append(even[i].unsqueeze(0))
            _.append(odd[i].unsqueeze(0))
        if odd_len < even_len:
            _.append(even[-1].unsqueeze(0))
        return torch.cat(_,0).permute(1,0,2) #B, L, D

    def forward(self, x):
        x_even_update, x_odd_update= self.workingblock(x)
        # We recursively reordered these sub-series. You can run the ./utils/recursive_demo.py to emulate this procedure.
        if self.current_level ==0:
            return self.zip_up_the_pants(x_even_update, x_odd_update)
        else:
            return self.zip_up_the_pants(self.SCINet_Tree_even(x_even_update), self.SCINet_Tree_odd(x_odd_update))
```

3.2 SCINet

```python
class SCINet(nn.Module):
    def __init__(self, output_len, input_len, input_dim = 9, hid_size = 1, num_stacks = 1,
                num_levels = 3, num_decoder_layer = 1, concat_len = 0, groups = 1, kernel = 5, dropout = 0.5,
                 single_step_output_One = 0, input_len_seg = 0, positionalE = False, modified = True, RIN=False):
        super(SCINet, self).__init__()

        self.input_dim = input_dim
        self.input_len = input_len
        self.output_len = output_len
        self.hidden_size = hid_size
        self.num_levels = num_levels
        self.groups = groups
        self.modified = modified
        self.kernel_size = kernel
        self.dropout = dropout
        self.single_step_output_One = single_step_output_One
        self.concat_len = concat_len
        self.pe = positionalE
        self.RIN=RIN
        self.num_decoder_layer = num_decoder_layer

        self.blocks1 = EncoderTree(
            in_planes=self.input_dim,
            num_levels = self.num_levels,
            kernel_size = self.kernel_size,
            dropout = self.dropout,
            groups = self.groups,
            hidden_size = self.hidden_size,
            INN =  modified)

        if num_stacks == 2: # we only implement two stacks at most.
            self.blocks2 = EncoderTree(
                in_planes=self.input_dim,
            num_levels = self.num_levels,
            kernel_size = self.kernel_size,
            dropout = self.dropout,
            groups = self.groups,
            hidden_size = self.hidden_size,
            INN =  modified)

        self.stacks = num_stacks

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
```

3.2 SCINet

```python
def train_one_epoch(model, train_loader, optimizer, loss_fn, scaler):
    model.train()
    running_loss = []
    prog_bar = tqdm(enumerate(train_loader), total=len(train_loader))
    for batch, (x_input, y_true, _) in prog_bar:
        optimizer.zero_grad()
        x_input = x_input.to(CFG['device'], torch.float)
        y_true = y_true.to(CFG['device'], torch.float)
        with torch.autocast(device_type=CFG['device'], dtype=torch.float16 if CFG['device'] == 'cuda' else torch.bfloat16):
            preds = model(x_input)
            loss = loss_fn(preds, y_true)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        running_loss.append(loss.item())
        prog_bar.set_description(f"loss: {np.mean(running_loss):.4f}")

    return running_loss
```

3.2 SCINet

```python
def train(train_dataset, valid_dataset, fold):
    model = SCINet(output_len=336, input_len=CFG['WINDOW'], input_dim=1, hid_size=CFG['HIDDEN']).to(CFG['device'])
    # model = Model(WINDOW).to(device)

    loss_fn = nn.L1Loss()
    # loss_fn = WeightedMAE(alpha=1)

    optimizer = torch.optim.AdamW(model.parameters(), lr=CFG['LEARNING_RATE'])
    scaler = torch.cuda.amp.GradScaler()

    train_loader = DataLoader(train_dataset, CFG['BATCH_SIZE'], shuffle=True, drop_last=True)
    valid_loader = DataLoader(valid_dataset, CFG['BATCH_SIZE'], shuffle=False)

    # scheduler = CosineAnnealingWarmRestarts(optimizer, 4, T_mult=1, eta_min=1e-5)
    scheduler = ReduceLROnPlateau(optimizer, factor=0.25, patience=3)

    best_mae = 1e20
    for e in range(CFG['EPOCH']):
        train_loss = train_one_epoch(model, train_loader, optimizer, loss_fn, scaler)
        valid_loss, loss2 = valid_one_epoch(model, valid_loader, loss_fn)
        scheduler.step(np.mean(valid_loss))
        if best_mae>np.mean(valid_loss):
            best_mae=np.mean(valid_loss)
            torch.save(model.state_dict(), f"SCInet_best_model_{fold}")
            print(f"save best model at epoch{e}")

    log = {
        "model" : "SCInet",
        "window" : CFG['WINDOW'],
        "lr" : CFG['LEARNING_RATE'],
        "mae" : best_mae,
    }

    with open("model_result.txt", "a") as f:
        f.write(f"{log}\n")

    return
```

[42] ✓ 0.0s

3.2 SCINet

```python
all_preds = []
all_scalers = []
for fold, (ty, vy) in enumerate(zip(train_year, valid_year)):
    seed_everything()

    years = []
    for year in ty:
        years.append(total_df[(total_df.year==year)])
    train_df = pd.concat(years).reset_index(drop=True)
    valid_df = total_df[total_df.year==vy].reset_index(drop=True)

    STscaler = StandardScaler()
    STscaler.fit(train_df.iloc[:, 1:2])

    train_df.iloc[:, 1] = STscaler.transform(train_df.iloc[:, 1:2])
    valid_df.iloc[:, 1] = STscaler.transform(valid_df.iloc[:, 1:2])

    test_tmp = test_df.copy()
    test_tmp.iloc[:, 1:2] = STscaler.transform(test_df.iloc[:, 1:2])

    train_dataset = TimeDataset(train_df, seq_len=CFG['WINDOW'])
    valid_dataset = TimeDataset(valid_df, seq_len=CFG['WINDOW'])
    test_dataset = TimeDataset(test_tmp, seq_len=CFG['WINDOW'], isTest=True)

    train(train_dataset, valid_dataset, fold)

    model = SCINet(output_len=336, input_len=CFG['WINDOW'], input_dim=1, hid_size=CFG['HIDDEN']).to(CFG['device'])
    model.load_state_dict(torch.load(f"SCInet_best_model_{fold}"))
    model.eval()
    test_loader = DataLoader(test_dataset, CFG['BATCH_SIZE'], shuffle=False)
    preds = predict(model, test_loader)
    all_preds.append(preds)
    all_scalers.append(STscaler)
```

[43] ✓ 21m 31.5s

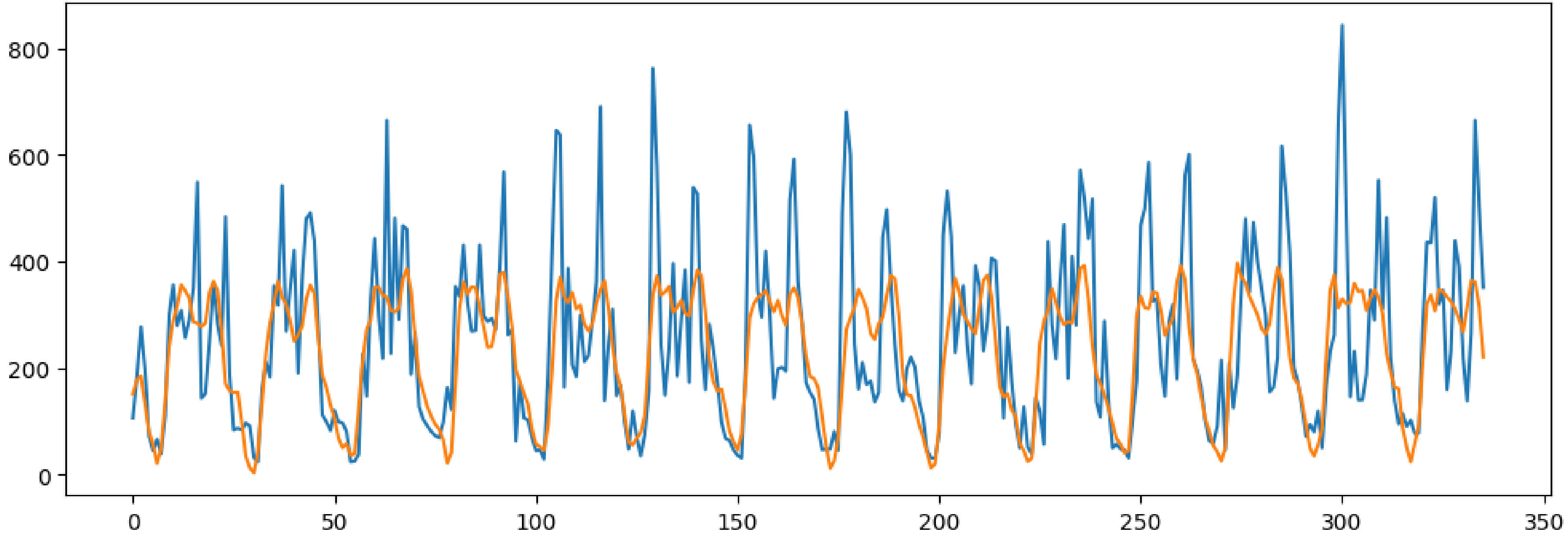
3.2 SCINet

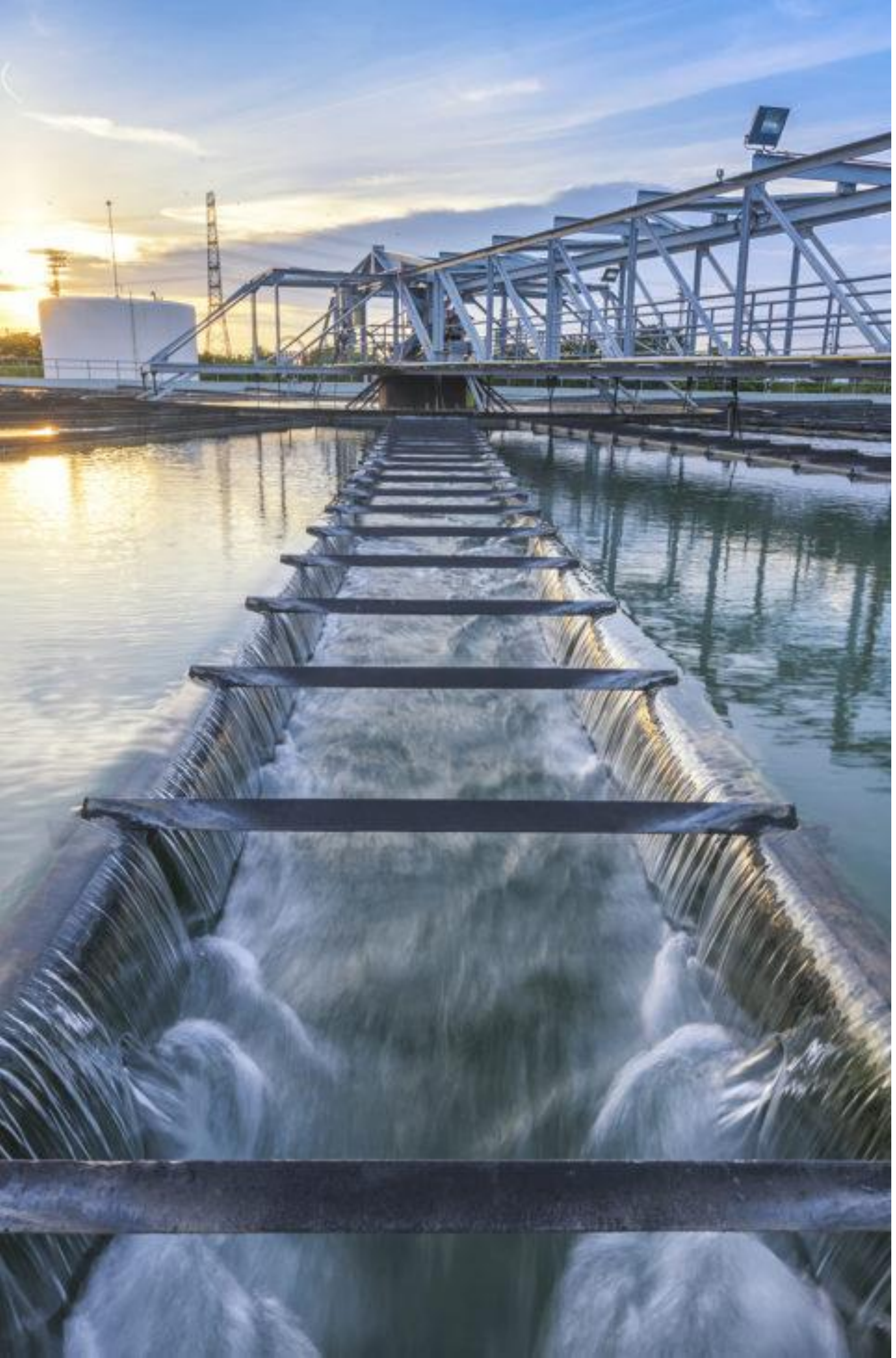
```
pred_temp = test_df[CFG['WINDOW']:CFG['WINDOW']+(24*14)].reset_index(drop=True)
pred_temp['pred'] = pd.DataFrame(np.sum(weighted_pred, axis=0)).iloc[:CFG['WINDOW']]
```

[64] ✓ 0.0s

```
plt.figure(figsize=(12,4))
plt.plot(pred_temp[['value', 'pred']]);
```

[67] ✓ 0.1s





감사합니다