

FlowMixer: A Constrained Neural Architecture for Interpretable Spatiotemporal Forecasting

This repository contains the implementation code for the paper "FlowMixer: A Constrained Neural Architecture for Interpretable Spatiotemporal Forecasting".

Overview

FlowMixer is a neural architecture that leverages constrained matrix operations within a reversible mapping framework to achieve interpretable spatiotemporal forecasting. The architecture introduces:

- **Constrained Non-negative Matrix Mixing:** Time mixing with positive definite matrices and feature mixing with stochastic attention mechanisms
- **Kronecker-Koopman Eigenmode Framework:** Provides interpretable spatiotemporal patterns through eigendecomposition
- **Semi-Orthogonal Basic Reservoir (SOBR):** Enables effective modeling of chaotic dynamical systems
- **Semi-group Properties:** Allow algebraic manipulation of prediction horizons without retraining

Code Structure

Core Implementations (Jupyter Notebooks)

1. **1_FlowMixer_Interactive_Pytorch.ipynb** - Interactive PyTorch Implementation
 - Full PyTorch implementation with interactive configuration
 - Supports multiple mixer types (standard, exponential, periodic, exponential-periodic)
 - Includes EMA, Mixup augmentation, and Kronecker-Koopman eigenmode analysis
 - Interactive experiment runner with comprehensive visualizations
2. **2_FlowMixer_Time_Series_Forecasting.ipynb** - TensorFlow Time Series Implementation
 - TensorFlow implementation focused on long-horizon forecasting
 - Comprehensive experiments on ETT, Weather, Electricity, and Traffic datasets
 - Includes RevIN and TD-RevIN normalization variants
 - Automated hyperparameter tuning and result collection
3. **3_Chaotic_Attractors_Prediction.ipynb** - Chaotic Systems Prediction
 - Implementation with SOBR (Semi-Orthogonal Basic Reservoir)
 - Experiments on Lorenz, Rössler, and Aizawa chaotic attractors
 - Comparison with Reservoir Computing and N-BEATS
 - Visualization of attractor predictions and phase space plots
4. **4_CylinderFlow_Prediction.ipynb** - 2D Turbulent Flow Prediction
 - CuPy-accelerated fluid dynamics simulation
 - 2D flow around cylinder using Navier-Stokes equations
 - FlowMixer implementation for vorticity field prediction
 - High-resolution flow visualization and error analysis

Requirements

Python Dependencies

```
# Core ML Libraries
tensorflow>=2.8.0
torch>=1.12.0
numpy>=1.21.0
pandas>=1.3.0
scikit-learn>=1.0.0

# Visualization
matplotlib>=3.5.0
seaborn>=0.11.0

# GPU Acceleration (for fluid dynamics)
cupy>=10.0.0 # Optional, for 4_CylinderFlow_Prediction.ipynb

# Additional utilities
networkx>=2.6.0 # For reservoir computing graphs
tqdm>=4.62.0 # Progress bars

# Jupyter environment
jupyter>=1.0.0
ipywidgets>=7.6.0 # For interactive widgets
```

Hardware Requirements

- **GPU:** NVIDIA GPU with CUDA support recommended
- **Memory:** 16GB+ RAM for large-scale experiments
- **Storage:** 2GB+ for datasets and results
- **Environment:** Jupyter Notebook or JupyterLab

Dataset Setup

Time Series Datasets

Create a `./data/` directory and download the following CSV files:

- `ETTh1.csv`, `ETTh2.csv` - Electricity Transformer Temperature (hourly)
- `ETTM1.csv`, `ETTM2.csv` - Electricity Transformer Temperature (minutely)
- `Weather.csv` - Weather forecasting dataset
- `ECL.csv` - Electricity Consuming Load dataset
- `TrafficL.csv` - Traffic dataset

These datasets are commonly available from the TSF benchmarks and should be placed with 'date' as the first column.

Chaotic Systems

No external datasets required - chaotic attractors are generated using the built-in ODE solvers.

Fluid Dynamics

No external datasets required - flow simulations are generated using the CuPy-based Navier-Stokes solver.

Running Experiments (Interactive Notebooks)

1. Interactive PyTorch Experiments

Open and run 1_FlowMixer_Interactive_Pytorch.ipynb:

- **Configuration Section:** Modify dataset, mixer type, and training parameters using interactive widgets
- **Experiment Execution:** Run cells sequentially for complete experiment
- **Outputs:** Training curves, prediction samples, and automatic Kronecker-Koopman eigenmode analysis
- **Visualizations:** Comprehensive eigenmode decomposition plots and interpretability analysis

2. Time Series Forecasting Benchmarks

Open and run 2_FlowMixer_Time_Series_Forecasting.ipynb:

- **Dataset Selection:** Configure experiments for ETT, Weather, Electricity, Traffic
- **Hyperparameter Sections:** Each dataset has optimized configurations
- **Execution:** Run all cells to reproduce Table 1 results from the paper
- **Outputs:** Results automatically saved to ./flowmixer_results/ with formatted comparison tables

3. Chaotic Systems Prediction

Open and run 3_Chaotic_Attractors_Prediction.ipynb:

- **System Generation:** Automatic generation of Lorenz, Rössler, and Aizawa attractors
- **Model Comparison:** FlowMixer+SOBR vs Reservoir Computing vs N-BEATS
- **Execution:** Run all cells for complete chaotic systems analysis
- **Outputs:** Attractor trajectory plots and phase space visualizations (Figure 3 reproduction)

4. 2D Turbulent Flow Prediction

Open and run 4_CylinderFlow_Prediction.ipynb:

- **Requirements:** CuPy installation required for GPU-accelerated fluid simulation

- **Simulation:** Real-time 2D Navier-Stokes simulation around cylinder ($Re=150$)
- **Training:** FlowMixer training on vorticity field sequences
- **Outputs:** Flow field predictions, vorticity visualizations, and error analysis (Figure 4 reproduction)

Key Results Reproduction

Time Series Forecasting (Table 1)

1. Open 2_FlowMixer_Time_Series_Forecasting.ipynb
2. Run all dataset sections (ETTh1, ETTh2, ETTm1, ETTm2, Weather, Electricity, Traffic)
3. Results automatically formatted and displayed as tables
4. CSV files saved in ./flowmixer_results/ for further analysis

Chaotic Attractors (Figure 3)

1. Open 3_Chaotic_Attractors_Prediction.ipynb
2. Execute all cells to generate predictions for all three attractors
3. Comparison plots automatically generated and displayed
4. High-quality figures saved as PDF files

Kronecker-Koopman Analysis (Figure 2)

1. Open 1_FlowMixer_Interactive_Pytorch.ipynb
2. Set show_eigenmode_analysis = True in configuration
3. Run experiment - eigenmode analysis automatically triggered
4. Comprehensive eigenmode decomposition plots generated

Turbulent Flow Prediction (Figure 4)

1. Open 4_CylinderFlow_Prediction.ipynb
2. Ensure CuPy is installed for GPU acceleration
3. Run all cells - simulation, training, and prediction automatically executed
4. Vorticity field predictions with error maps generated

Interactive Configuration

Key Parameters (Configurable via Widgets)

- **seq_len:** Input sequence length (96, 192, 336, 720, 1024+)
- **pred_len:** Prediction horizon (96, 192, 336, 720)
- **mixer_type:** Architecture variant
 - 'standard': Basic constrained mixing
 - 'exp': Matrix exponential time mixing
 - 'periodic': Kronecker structure for seasonalities

- 'exp-periodic': Combined exponential and periodic
- **revin**: Normalization type
 - 1: RevIN (feature-wise)
 - 2: TD-RevIN (time-dependent)
- **dropout_rate**: Regularization strength (0.0-0.5)
- **learning_rate**: Optimizer learning rate (1e-4 to 1e-1)

Advanced Options

- **use_ema**: Exponential Moving Average for stable training
- **mixup_alpha**: Data augmentation strength
- **grad_clip**: Gradient clipping for stability
- **optimizer**: SGD vs AdamW selection

Expected Runtime

Single GPU (NVIDIA A100)

- **ETT datasets**: ~2 seconds/epoch, ~50-100 epochs per experiment
- **Weather/Traffic**: ~10-30 seconds/epoch, ~50-100 epochs
- **Electricity**: ~120 seconds/epoch, ~50-100 epochs
- **Chaotic systems**: ~5-10 minutes total per attractor
- **Fluid dynamics**: ~30 minutes simulation + 10-15 minutes training

Interactive Execution

- Each notebook cell executes independently
- Progress bars and real-time plotting for monitoring
- Early stopping implemented to prevent overtraining
- Automatic checkpoint saving for long experiments

Architecture Implementation Details

Core Transformation

FlowMixer implements:

Core transformation

$F(X) = \text{phi_inverse}(W_t @ \text{phi}(X) @ W_f.T)$

Where:

phi: Reversible mapping (RevIN/TD-RevIN)

W_t : Time mixing matrix = $\alpha * I + W * W$ (positive definite)

W_f : Feature mixing matrix (stochastic attention)

Kronecker-Koopman Framework

```
# Eigenmode decomposition
W_f ⊗ W_t = P @ D @ P-1

# Spatiotemporal modes
Φij = qi ⊗ pj # Kronecker product of eigenvectors

# Horizon manipulation
X(t) = Σ aij * Φij * exp(t * log(λi * μj))
```

SOBR Enhancement

```
# Semi-Orthogonal Basic Reservoir
S(X) = σ(Ut @ X @ Uf.T) # Dimensional lifting
# Where Uf @ Uf.T = I, Ut @ Ut.T = I
```

Troubleshooting

Common Issues

1. **CuPy Installation:** For fluid dynamics, ensure CUDA-compatible CuPy version
2. **Memory Issues:** Reduce batch_size or seq_len for large datasets
3. **Dataset Loading:** Ensure CSV files have 'date' column and proper formatting
4. **GPU Memory:** Clear GPU cache between experiments in Jupyter

Performance Tips

- Use GPU acceleration when available, further optimize code if need (jit)
- Enable mixed precision training for larger models
- Adjust batch size based on available memory-not recommended
- Use early stopping to prevent overtraining