Some Preliminary Transfer Learning Results for Dynamical Systems

Joshua Hammond, Brian Korgel, Michael Baldea
McKetta Department of Chemical Engineering
The University of Texas at Austin

Acknowledgments: Tyler Soderstrom, ExxonMobil

TWCCC Fall Meeting 2025

Chemical Processes are “Copies” of Dynamical Systems

- Several plants are built off the same blueprint
- Similar but not identical
- May use a module (or more) in multiple designs

How can *data driven* modeling knowledge developed for one system be leveraged in the operation and control of another “copy”

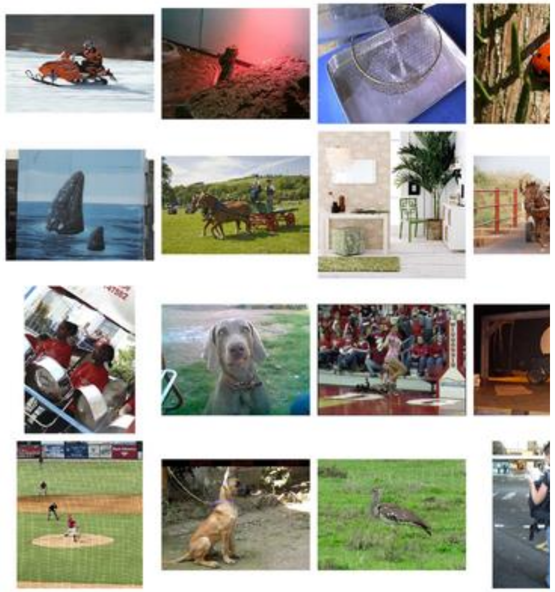
>> Transfer learning

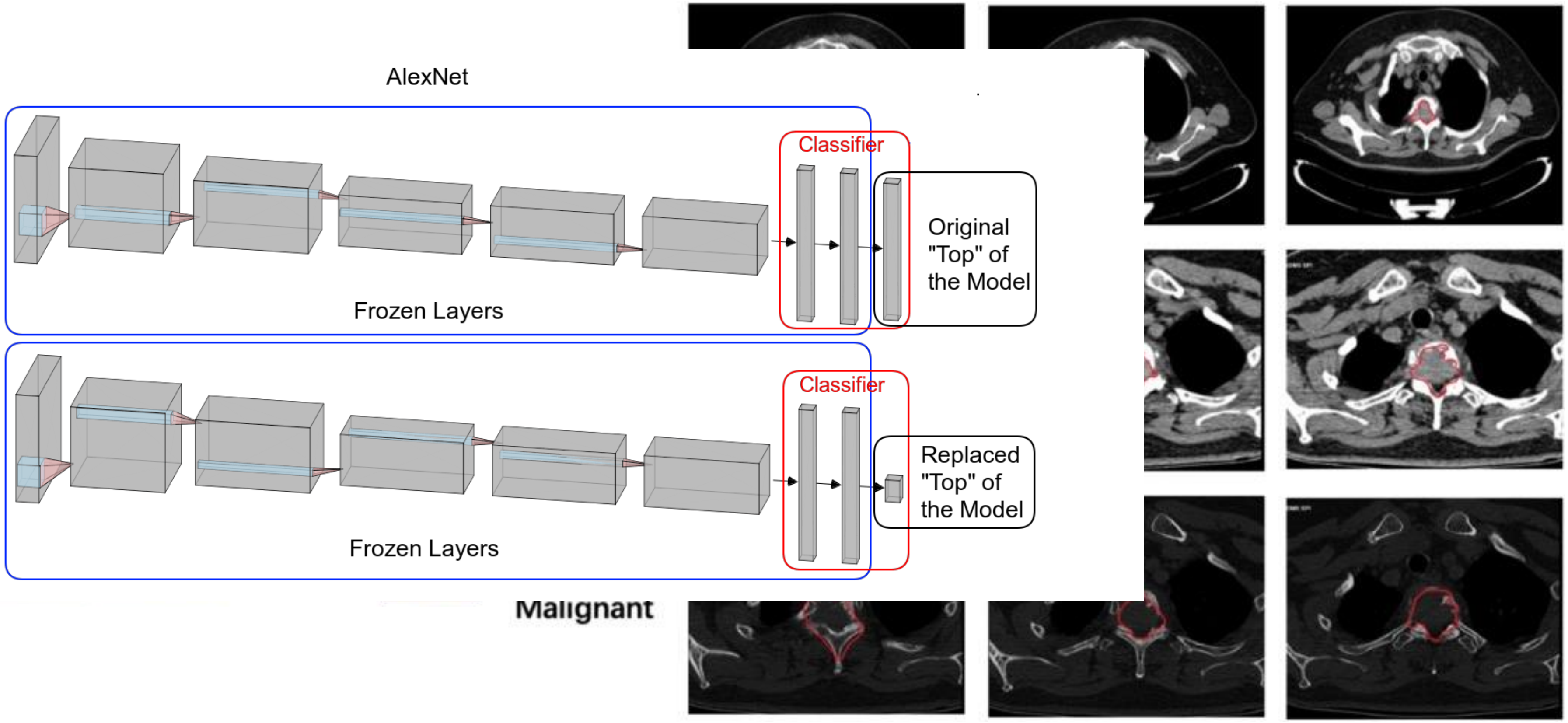
M. Baldea, T.F. Edgar, B. Stanley, A.A. Kiss, Modular Manufacturing: Status, Challenges and Opportunities, AIChE Journal, 63(10), 4262-4272, 2017

Most existing TL applications focus on image classification

Source: ImageNet Classification[1]
Image classification from 1,000 common objects

Target: Medical Diagnositics[2]
Classification of tumor type





Not dynamical systems – no governing equation constraints

[1] Russakovsky et. Al. "ImageNet Large Scale Visual Recognition Challenge" arXiv: 1409.0575 [2] Guo et. Al. "Radiographic imaging and diagnosis of spinal bone tumors: AlexNet and ResNet for the classification of tumor malignancy" 2024 Journal of Bone Oncology [3]AlexNet Visualization: Daniel Voight Godoy https://github.com/dvgodoy/dl-visuals/

Machine learning models of dynamical systems

We consider a general physical system, whose *true* dynamics are given by

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}^{\mathbf{S}}(\mathbf{x}, \mathbf{u}, \mathbf{p})$$

where the function $\mathbf{f}^{\mathbf{S}}: \mathbb{R}^{n_x \times n_u \times n_p} \rightarrow \mathbb{R}^{n_x}$ is the *governing equation(s)*, and is a function of the current state $\mathbf{x}$, inputs $\mathbf{u}$, and parameters $\mathbf{p}$.

NOTE: The true form of the function $\mathbf{f}$ is *unknown*, but data $\mathcal{D}^{\mathbf{S}} \triangleq \{\hat{\mathbf{x}}, \hat{\mathbf{u}}\}$ are available with noise that is assumed to be Gaussian.

A NN model produces discrete predictions of future states

$$\tilde{\mathbf{x}}_{t+1} = \Phi^{\mathbf{S}}(\mathbf{x}_{\mathrm{t}}, \mathbf{u}_{\mathrm{t}}, \boldsymbol{\pi})$$

where $\boldsymbol{\pi}$ is a vector of the NN parameters that are fit using the data

Research Problem Formulation

	***Source** System*	***Target** System*
System Space	$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}^{\mathbf{S}}(\mathbf{x}, \mathbf{u}, \mathbf{p}^{\mathbf{S}})$	$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}^{\mathbf{T}}(\mathbf{x}, \mathbf{u}, \mathbf{p}^{\mathbf{T}})$
Data Space	$\mathcal{D}^{\mathbf{S}} = \{\hat{\mathbf{x}}^{S}, \hat{\mathbf{u}}^{S}\}$	$\mathcal{D}_{\mathbf{T}} = \{\hat{\mathbf{x}}^{T}, \hat{\mathbf{u}}^{T}\}$ $\dim(\mathcal{D}^{\mathbf{T}}) \ll \dim(\mathcal{D}^{\mathbf{S}})$
Model Space	$\tilde{\mathbf{x}}_{t+1} = \Phi^{\mathbf{S}}(\mathbf{x}, \mathbf{u}, \boldsymbol{\pi}^{\mathbf{S}})$	$\tilde{\mathbf{x}}_{t+1} = \Phi^{\mathbf{T}}(\mathbf{x}, \mathbf{u}, \boldsymbol{\pi}^{\mathbf{T}})$

Assumption: Source and Target systems are similar but not identical

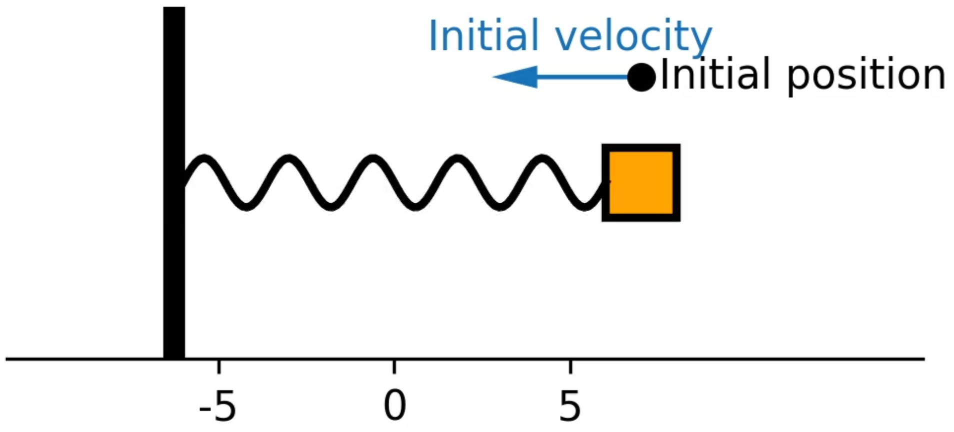
Need: minimize use of Target system data to build a ML model, starting from ML model of Source system

Example: Damped Spring

$m_s\ddot{x} + c_s\dot{x} + k_s x = u_s$
Given: $x_0, \dot{x}_0$
Predict: $x_1, x_2, \dots, x_{20}$

Source System $m_s\ddot{x} + c_s\dot{x} + k_s x = u_s$

Measurement Noise: $\mathcal{N}(0, 0.1)$
$\dim(D_S) = 100{,}000$



Use a 2-layer MLP
- 32 neurons each layer
- Sigmoid activation
- 2,868 trainable parameters

Example: Damped Spring

$m_s\ddot{x} + c_s\dot{x} + k_s x = u_s$
Given: $x_0, \dot{x}_0$
Predict: $x_1, x_2, \ldots, x_{20}$

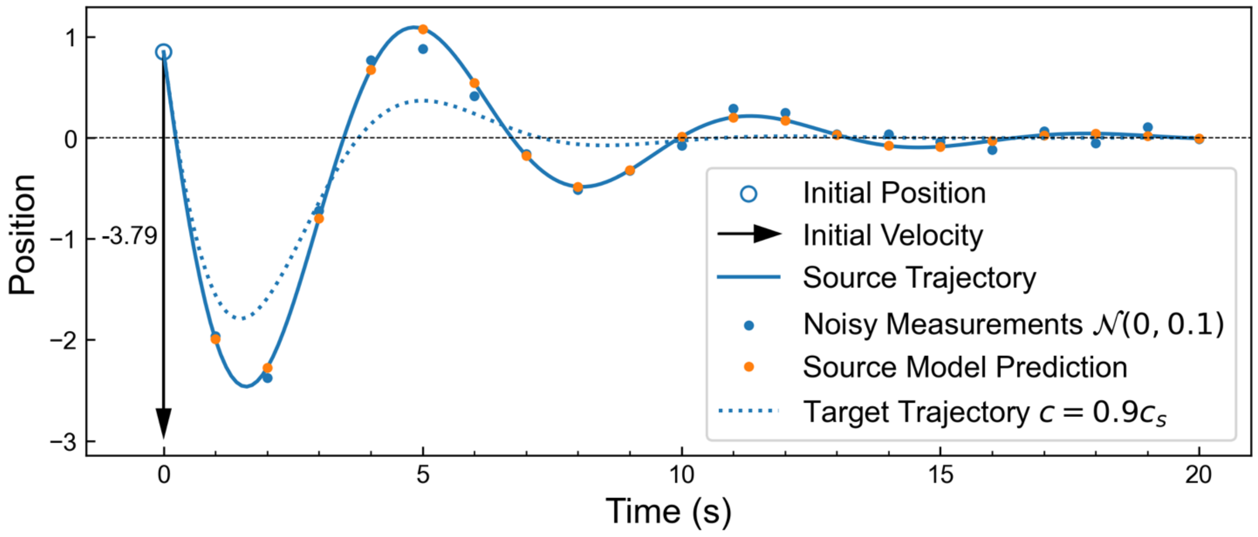
Source System $m_s\ddot{x} + c_s\dot{x} + k_s x = u_s$

Measurement Noise: $\mathcal{N}(0, 0.1)$
$\dim(D_S) = 100{,}000$

Target System (set of systems)

Change $m_s, c_s, k_s, u_s, \pm\Delta 10\%$
$\dim(D_T) = 1{,}000$

An additional 99,000 data points are used for evaluation



Use a 2-layer MLP

- 32 neurons each layer
- Sigmoid activation
- 2,868 trainable parameters

Can the NN model of the Source oscillator be easily adapted to the Target oscillator?

Transfer Learning in Dynamical Systems

	***Source* System**		***Target* System**
System Space	$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}^{\mathbf{S}}(\mathbf{x},\mathbf{u},\mathbf{p}^{\mathbf{S}})$		$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}^{\mathbf{T}}(\mathbf{x},\mathbf{u},\mathbf{p}^{\mathbf{T}})$
Data Space	$\mathcal{D}^{\mathbf{S}} = \{\hat{\mathbf{x}}^{S}, \hat{\mathbf{u}}^{S}\}$	❯	$\mathcal{D}_{\mathbf{T}} = \{\hat{\mathbf{x}}^{T}, \hat{\mathbf{u}}^{T}\}$ $\dim(\mathcal{D}^{\mathbf{T}}) \ll \dim(\mathcal{D}^{\mathbf{S}})$
Model Space	$\tilde{\mathbf{x}}_{t+1} = \Phi^{\mathbf{S}}(\mathbf{x},\mathbf{u},\boldsymbol{\pi}^{\mathbf{S}})$		$\tilde{\mathbf{x}}_{t+1} = \Phi^{\mathbf{T}}(\mathbf{x},\mathbf{u},\boldsymbol{\pi}^{\mathbf{T}})$

Assumption

- The two systems are similar but not identical
  - ❑ Quantify similarity

Hypotheses

- The structure of the network $\Phi^{\mathbf{T}}$ is the same as $\Phi^{\mathbf{S}}$
- $\boldsymbol{\pi}^{\mathbf{S}}$ is a good starting point for finding $\boldsymbol{\pi}^{\mathbf{T}}$

Cosine Similarity

Given two vectors a and b:

$$\theta(a,b) \triangleq \frac{a \cdot b}{|a||b|}$$

$\theta = 1$ Perfectly correlated
$\theta = 0$ Uncorrelated
$\theta = -1$ Negatively correlated

Some benefits:

- Scale, translation independent
- Robust to different lengths, high dimensionality
- Interpretability

Data Similarity

Done on a variable by variable basis

Functional Similarity

For a given set of inputs $(\mathbf{x}, \mathbf{u})$ to two dynamical systems

$$\mathrm{f}^{\mathbf{S}}(\mathbf{x}, \mathbf{u}, \boldsymbol{p}^{\mathbf{S}})$$
$$\mathrm{f}^{\mathbf{T}}(\mathbf{x}, \mathbf{u}, \boldsymbol{p}^{\mathbf{T}})$$

Simulate/sample in discrete time the input and output trajectories and then evaluate similarity using the data

NN Training

Prediction

$$\tilde{\boldsymbol{y}}_k = \Phi(\mathbf{x}_\mathrm{k}, \mathbf{u}_\mathrm{k}, \boldsymbol{w}_k) + \boldsymbol{v}_k$$

Gradient Descent

$$e_k = \boldsymbol{y}_k - \tilde{\boldsymbol{y}}_k$$

$$\mathcal{L}_k = Loss\ Function(e_k)$$

$$\frac{d\mathcal{L}}{d\hat{\boldsymbol{\pi}}_k} \leftarrow Backpropagation(\mathcal{L}, \hat{\boldsymbol{\pi}}_k)$$

$$\hat{\boldsymbol{\pi}}_{k+1} = \hat{\boldsymbol{\pi}}_k - \eta \frac{d\mathcal{L}}{d\hat{\boldsymbol{\pi}}_k}$$

Repeat across all datapoints 1 … k until convergence

k – Discrete time index
$\boldsymbol{\pi}_k$ – Neural network parameters $(N_p, 1)$
$\hat{\boldsymbol{\pi}}_k$ – NN parameter estimate $(N_p, 1)$
$\mathbf{y}_k$ – Discrete measurement $(N_{out}, 1)$
$\tilde{\boldsymbol{y}}_k$ – NN output $(N_{out}, 1)$
Φ – NN model
$\boldsymbol{w}_k$ - Model uncertainty
$\boldsymbol{v}_k$ - Measurement uncertainty
$\boldsymbol{j}$ – Selected parameter indices
$\boldsymbol{e}_k$ – Error vector $(N_{out}, 1)$
$\mathbf{H}_k$ – Gradients of network parameters w.r.t. outputs (Jacobian) (N_{params}, N_{out})
$\boldsymbol{P}_k$ – Covariance matrix (N_{params}, N_{params})
$\boldsymbol{Q}$ – Process noise matrix (N_{params}, N_{params}) $E[\boldsymbol{w}_k \boldsymbol{w}_{k+1}] = \delta_{k,k+1} \boldsymbol{Q}_k$
$\boldsymbol{R}_k$ - Measurement Noise (N_{out}, N_{out}) $E[\boldsymbol{v}_k \boldsymbol{v}_{k+1}] = \delta_{k,k+1} \boldsymbol{R}_k$

Strategies for updating NN models

Hammond et. al "Staying Alive: Online Neural Network Maintenance and Systemic Drift" arXiv:2503.17681

Idea: If models are similar, and their parameters will be close, we can use model-updating techniques to evolve model parameters to reflect the target system

The Subset Extended Kalman Filter (SEKF)

- Uses well-established Kalman Filtering techniques to update a subset of model parameters
- Probabilistic methods for state-estimation in the presence of noise ideally limit spurious correlations and overfitting

$$\boldsymbol{e}_k = \boldsymbol{y}_k - \widetilde{\boldsymbol{y}}_k$$
$$\boldsymbol{H}_k = \nabla_{\boldsymbol{\pi}} \boldsymbol{y}_k$$
$$\boldsymbol{j} = \{\text{Prop.}\, q, \text{Mag.}\, q\}$$
$$\boldsymbol{H}'_k = \boldsymbol{H}_k[:, \boldsymbol{j}]; \boldsymbol{P'}_k = \boldsymbol{P}_k[\boldsymbol{j}, \boldsymbol{j}]$$

Parameter selection

$$\boldsymbol{A}_k = \left[\boldsymbol{R}_k + \boldsymbol{H}'^T_k \boldsymbol{P'}_k \boldsymbol{H'}_k\right]^{-1}$$
$$\boldsymbol{K}_k = \boldsymbol{P'}_k \boldsymbol{H'}_k \boldsymbol{A}_k$$
$$\widehat{\boldsymbol{\pi}}[\boldsymbol{j}]_{k+1} = \widehat{\boldsymbol{\pi}}[\boldsymbol{j}]_k + \boldsymbol{K}_k \boldsymbol{e}_k$$
$$\boldsymbol{P}[\boldsymbol{j}, \boldsymbol{j}]_{k+1} = \boldsymbol{P'}_k - \boldsymbol{K}_k \boldsymbol{H'}^T_k \boldsymbol{P'}_k + \boldsymbol{Q'}$$

k – Discrete time index
$\boldsymbol{\pi}_k$ – Neural network parameters (N_p,1)
$\widehat{\boldsymbol{\pi}}_k$ – NN parameter estimate (N_p,1)
$\mathbf{y}_k$ – Discrete measurement (N_{out},1)
$\widetilde{\boldsymbol{y}}_k$– NN output ($N_{out}$,1)
Φ – NN model
$\boldsymbol{w}_k$ - Model uncertainty
$\boldsymbol{v}_k$ - Measurement uncertainty
$\boldsymbol{j}$ – Selected parameter indices
$\boldsymbol{e}_k$ – Error vector (N_{out},1)
$\mathbf{H}_k$ – Gradients of network parameters w.r.t. outputs (Jacobian) (N_{params}, N_{out})
$\boldsymbol{P}_k$ – Covariance matrix (N_{params}, N_{params})
$\boldsymbol{Q}$ – Process noise matrix (N_{params}, N_{params})
$E[\boldsymbol{w}_k \boldsymbol{w}_{k+1}] = \delta_{k,k+1} \boldsymbol{Q}_k$
$\boldsymbol{R}_k$ - Measurement Noise (N_{out}, N_{out}) $E[\boldsymbol{v}_k \boldsymbol{v}_{k+1}] = \delta_{k,k+1} \boldsymbol{R}_k$

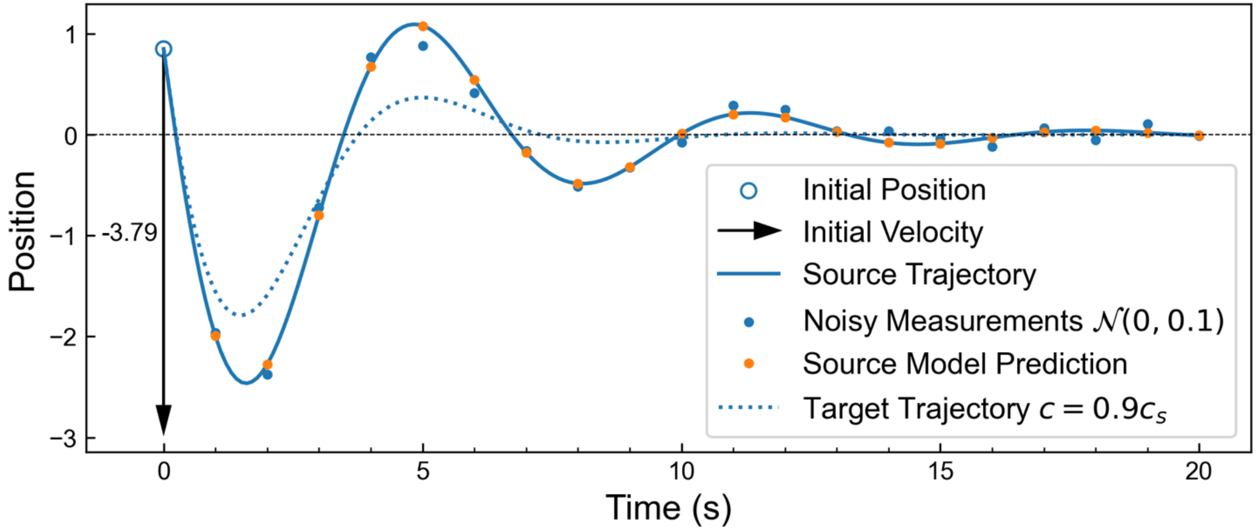
Transfer Learning for Damped Spring

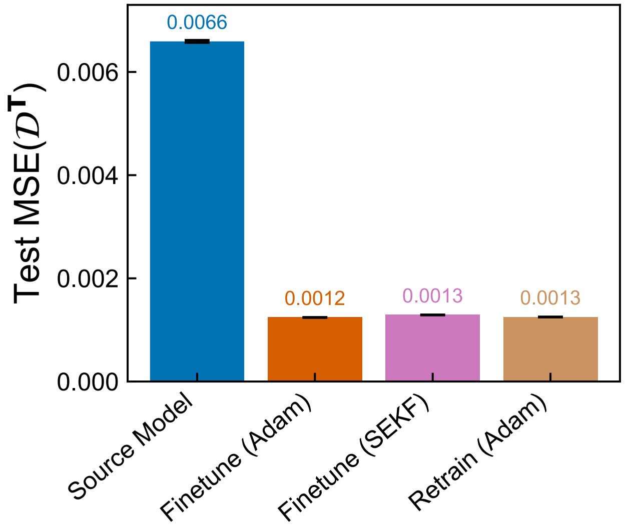
$m_s\ddot{x} + c_s\dot{x} + k_s x = u_s$
Given: $x_0, \dot{x}_0$
Predict: $x_1, x_2, \dots, x_{20}$

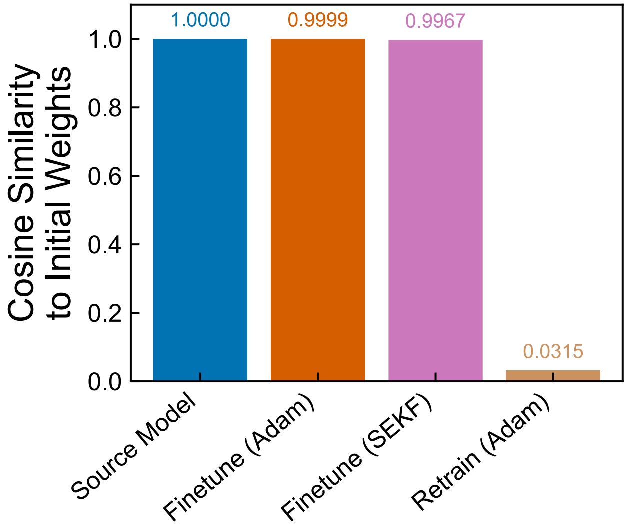
Competing Techniques:

- **Apply source model**
- **Finetune (train NN starting from source model parameters) using Adam**
- **Finetune using SEKF**
- **Retrain (reinitialize NN weights) using ADAM**

Use a 2-layer MLP
- 32 neurons each layer
- Sigmoid activation
- 2,868 trainable parameters





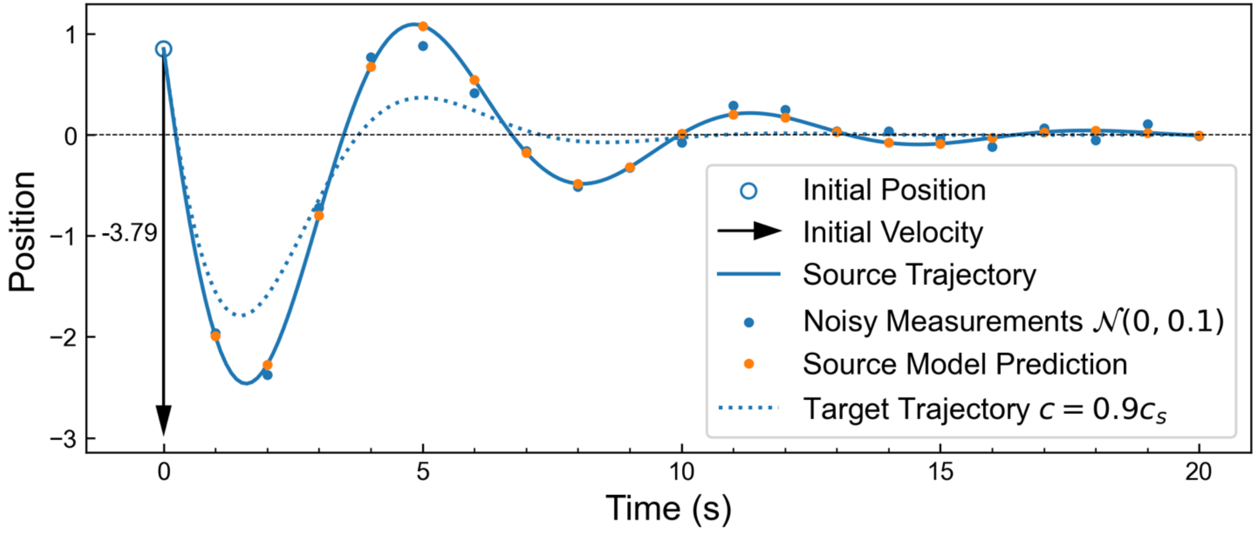


Transfer Learning for Damped Spring

$m_s\ddot{x} + c_s\dot{x} + k_s x = u_s$
Given: $x_0, \dot{x}_0$
Predict: $x_1, x_2, \ldots, x_{20}$

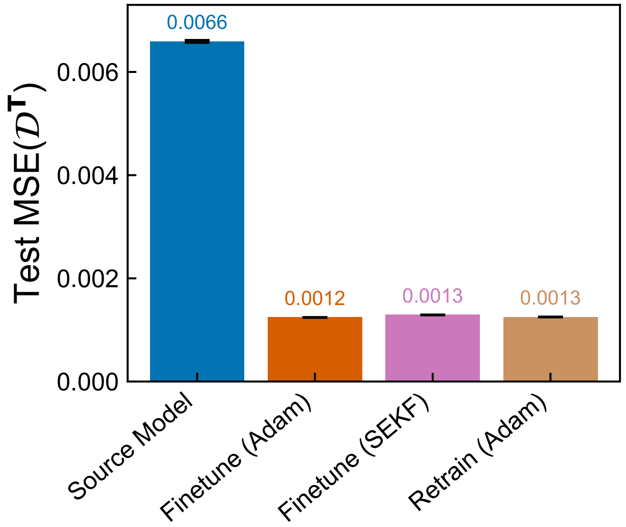
Competing Techniques:

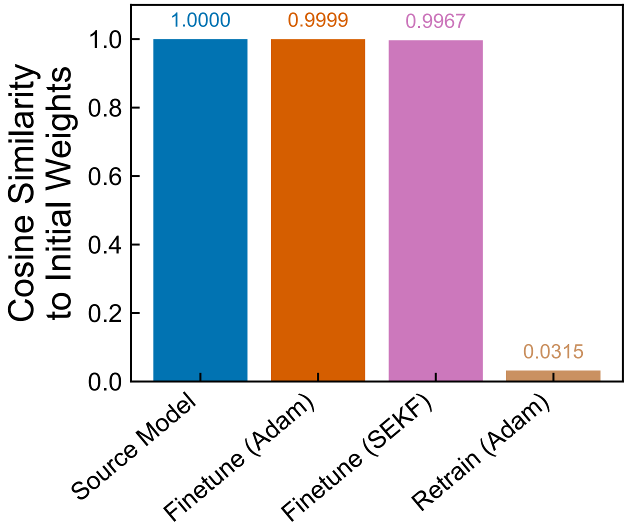
- Apply source model
- Finetune (train NN starting from source model parameters) using Adam
- Finetune using SEKF
- Retrain (reinitialize NN weights) using ADAM



Preliminary findings

- Finetuning results in similar model parameters between source and target
- Retraining may change NN model parameters substantially
- Loss is comparable

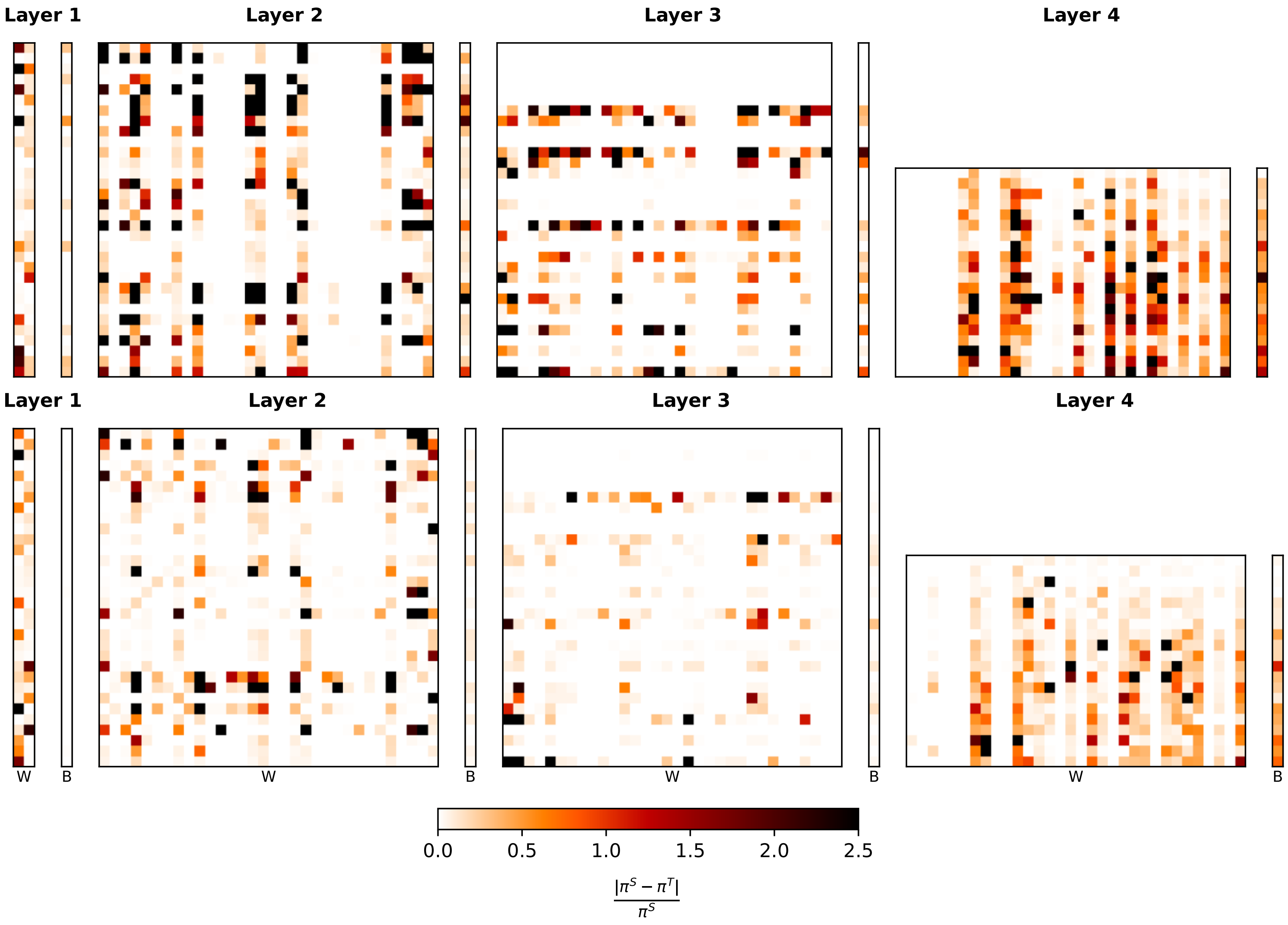




Changes to the NN parameters are distributed through the model

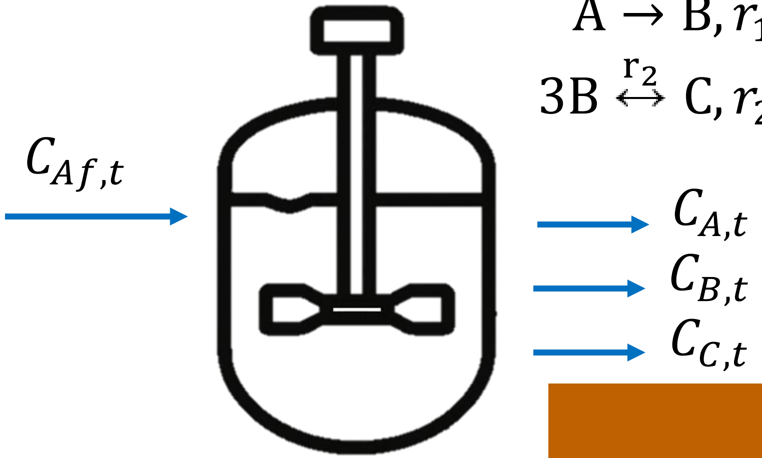
SEKF Weights

Adam Weights



Example: Official TWCCC CSTR ☺

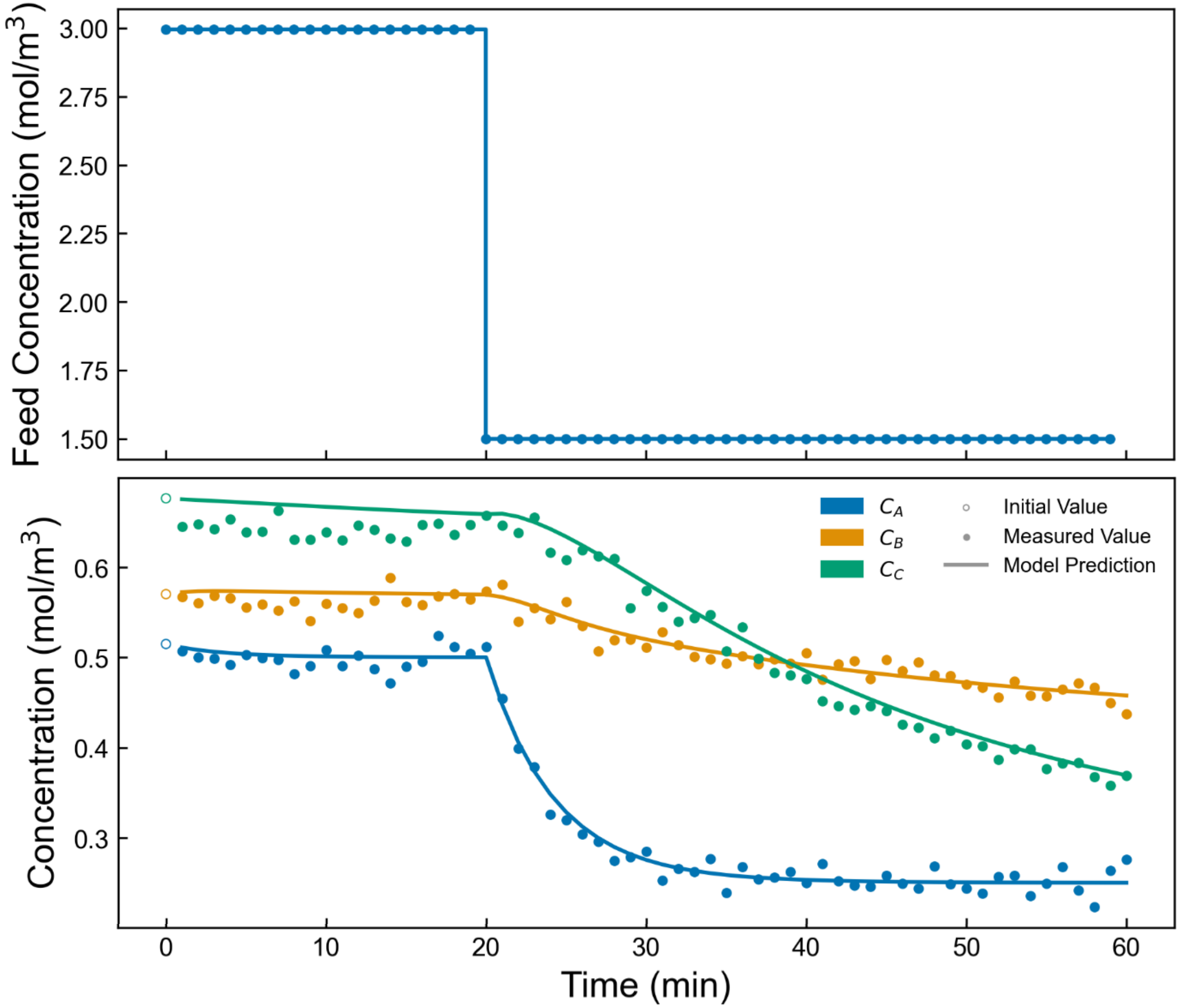
Given: $C_{A0}, C_{B0}, C_{C0}, U_{0,\ldots,59}$
Predict: $C_{i1}, \ldots, C_{i,60}$
Source→Target: Reactor Volume Δ20%



$$A \xrightarrow{r_1} B, r_1 = k_1 c_A$$
$$3B \overset{r_2}{\leftrightarrow} C, r_2 = k_{2f} c_B^3 - k_{2b} c_C$$

Source System
$\dim(D_S)$ =1 year of data

Use a 2-layer Neural ODE
- 16 neurons each layer
- Sigmoid activation
- 419 trainable params



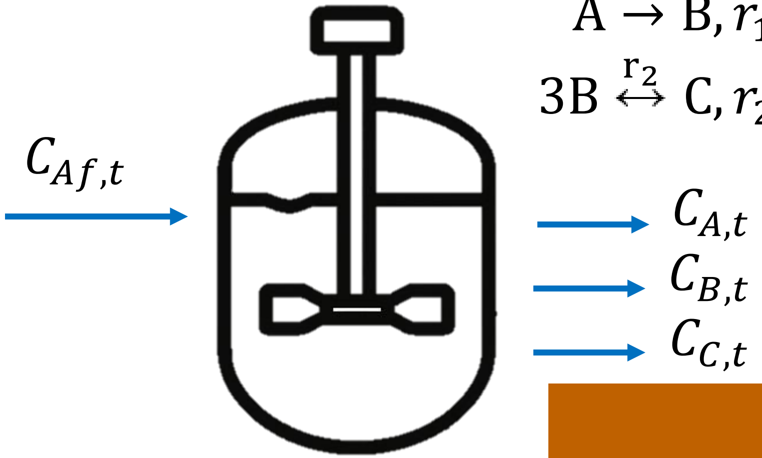
[1] Kumar, P., & Rawlings, J. B. (2023). Structured nonlinear process modeling using neural networks and application to economic optimization. *Computers and Chemical Engineering*, *177*. https://doi.org/10.1016/j.compchemeng.2023.108314

Example: Official TWCCC CSTR ☺

Given: $C_{A0}, C_{B0}, C_{C0}, U_{0,\ldots,59}$
Predict: $C_{i1}, \ldots, C_{i,60}$
Source→Target: Reactor Volume Δ20%

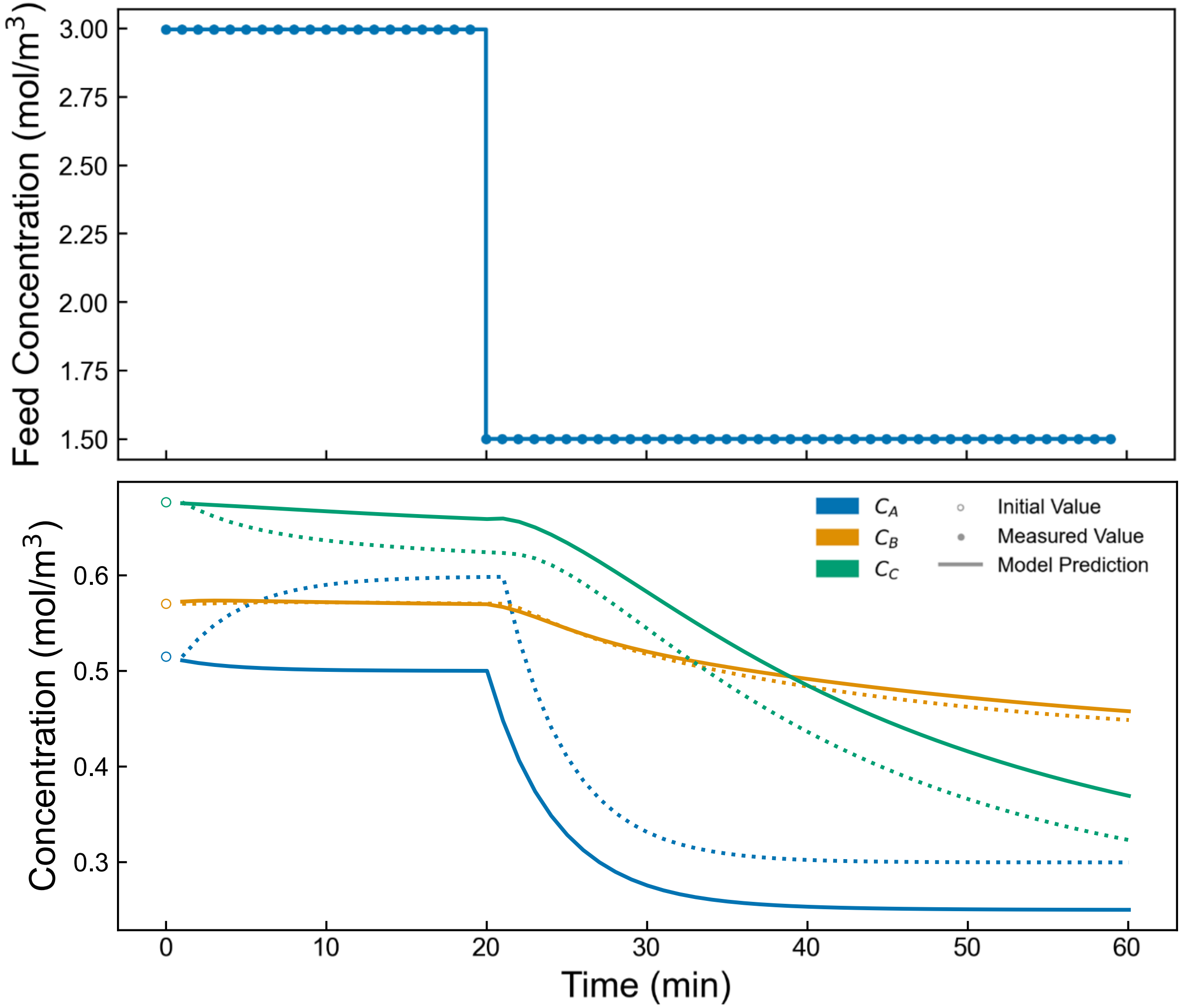
$$A \xrightarrow{r_1} B, r_1 = k_1 c_A$$

$$3B \overset{r_2}{\leftrightarrow} C, r_2 = k_{2f} c_B^3 - k_{2b} c_C$$



Use a 2-layer Neural ODE
- 16 neurons each layer
- Sigmoid activation
- 419 trainable params

Source System
$\dim(D_S)$ =1 year of data
Target System
Reactor Volume Δ20%
$\dim(D_T)$ = One day of training data
Evaluated on remaining 363 days

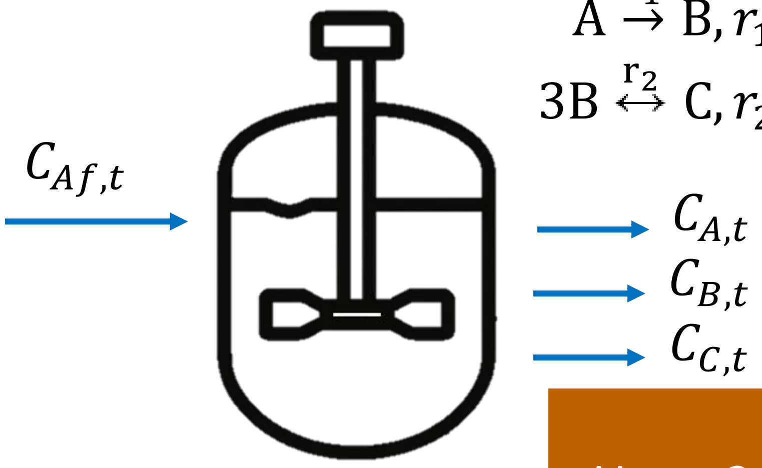


[1] Kumar, P., & Rawlings, J. B. (2023). Structured nonlinear process modeling using neural networks and application to economic optimization. *Computers and Chemical Engineering*, *177*. https://doi.org/10.1016/j.compchemeng.2023.108314

Example: Official TWCCC CSTR ☺

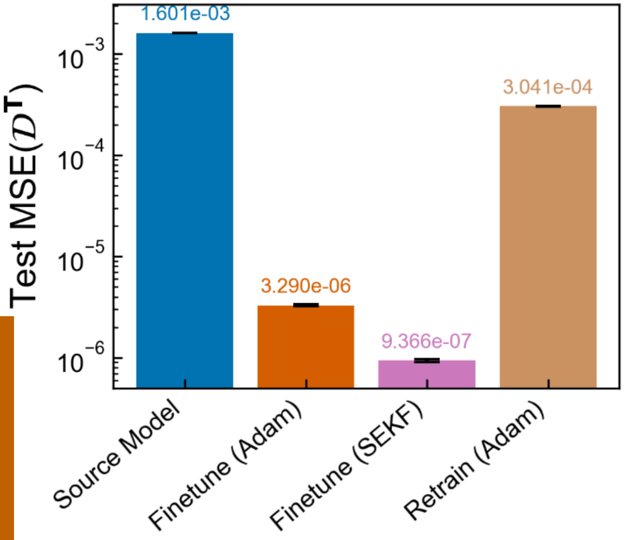
Given: $C_{A0}, C_{B0}, C_{C0}, U_{0,\ldots,59}$
Predict: $C_{i1}, \ldots, C_{i,60}$
Source→Target: Reactor Volume Δ20%

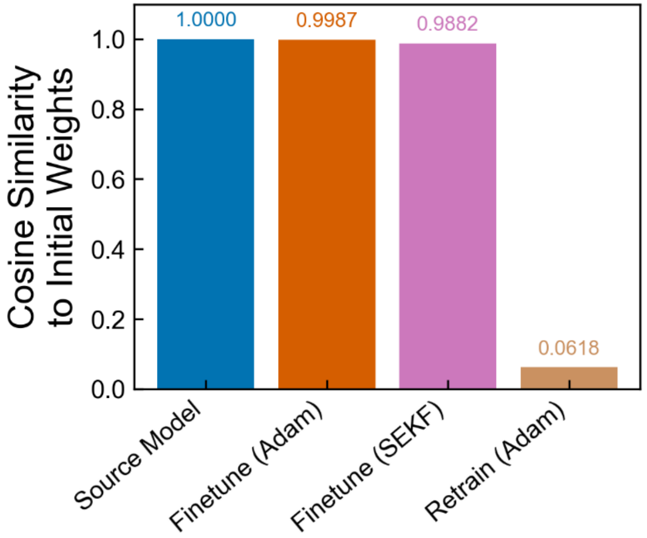
$$A \xrightarrow{r_1} B, r_1 = k_1 c_A$$
$$3B \overset{r_2}{\leftrightarrow} C, r_2 = k_{2f} c_B^3 - k_{2b} c_C$$



Source System
$\dim(D_S)$ =1 year of data
Target System
Reactor Volume Δ20%
$\dim(D_T)$ = One day of training data
Evaluated on remaining 363 days

Use a 2-layer Neural ODE
- 16 neurons each layer
- Sigmoid activation
- 419 trainable params



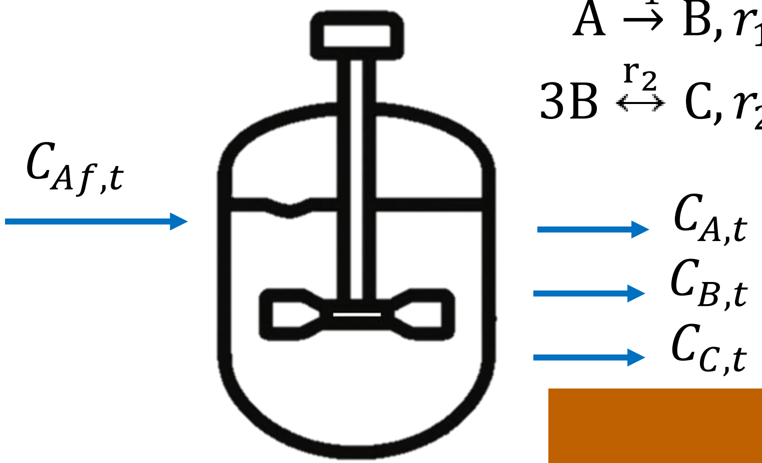


[1] Kumar, P., & Rawlings, J. B. (2023). Structured nonlinear process modeling using neural networks and application to economic optimization. *Computers and Chemical Engineering*, *177*. https://doi.org/10.1016/j.compchemeng.2023.108314

Example: Official TWCCC CSTR ☺

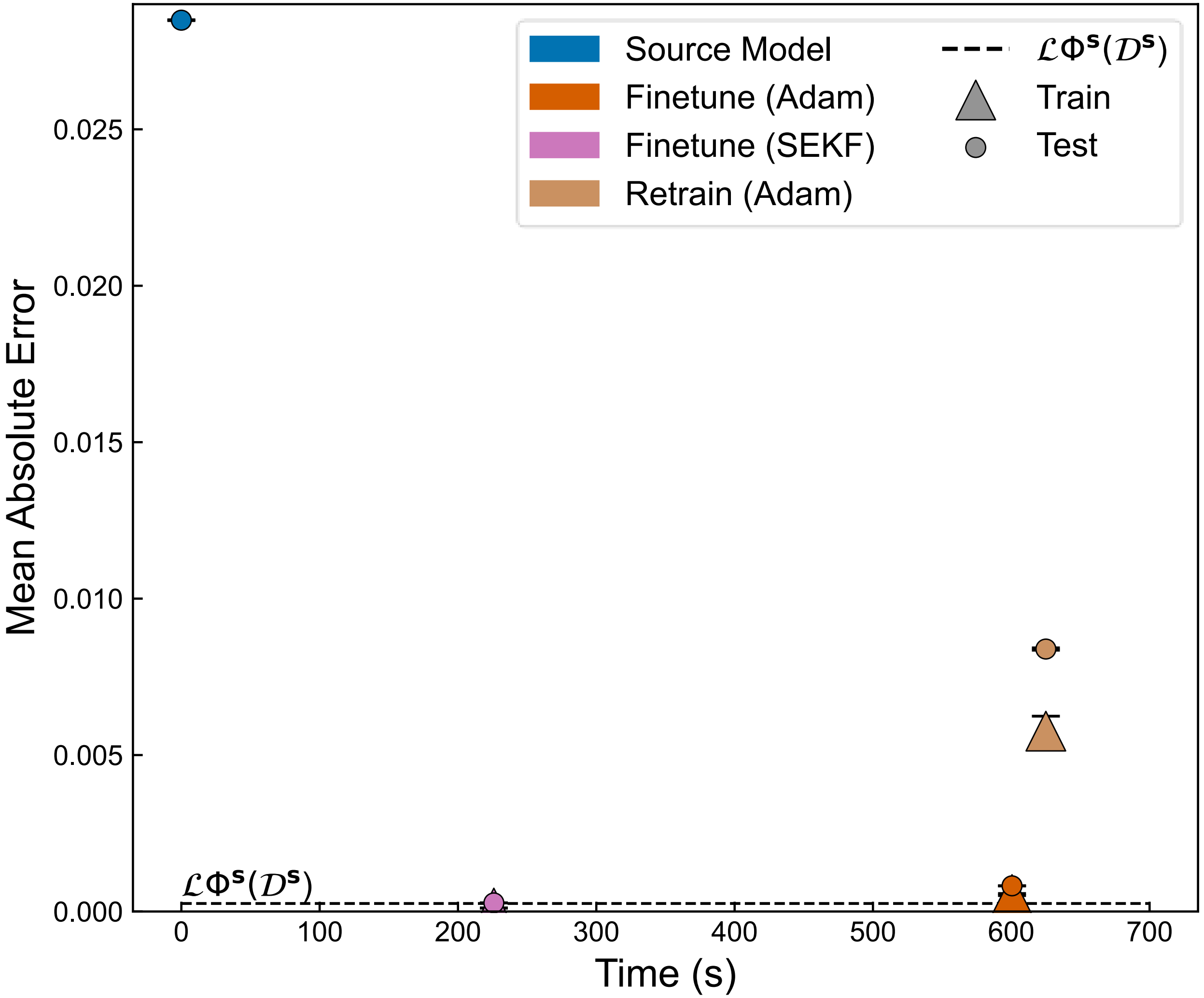
Given: $C_{A0}, C_{B0}, C_{C0}, U_{0,\ldots,59}$
Predict: $C_{i1}, \ldots, C_{i,60}$
Source→Target: Reactor Volume $\Delta 20\%$

$$A \xrightarrow{r_1} B, r_1 = k_1 c_A$$
$$3B \overset{r_2}{\leftrightarrow} C, r_2 = k_{2f} c_B^3 - k_{2b} c_C$$



Use a 2-layer Neural ODE
- 16 neurons each layer
- Sigmoid activation
- 419 trainable params

Source System
$\dim(D_S)$ =1 year of data
Target System
Reactor Volume $\Delta 20\%$
$\dim(D_T)$ = One day of training data
Evaluated on remaining 363 days

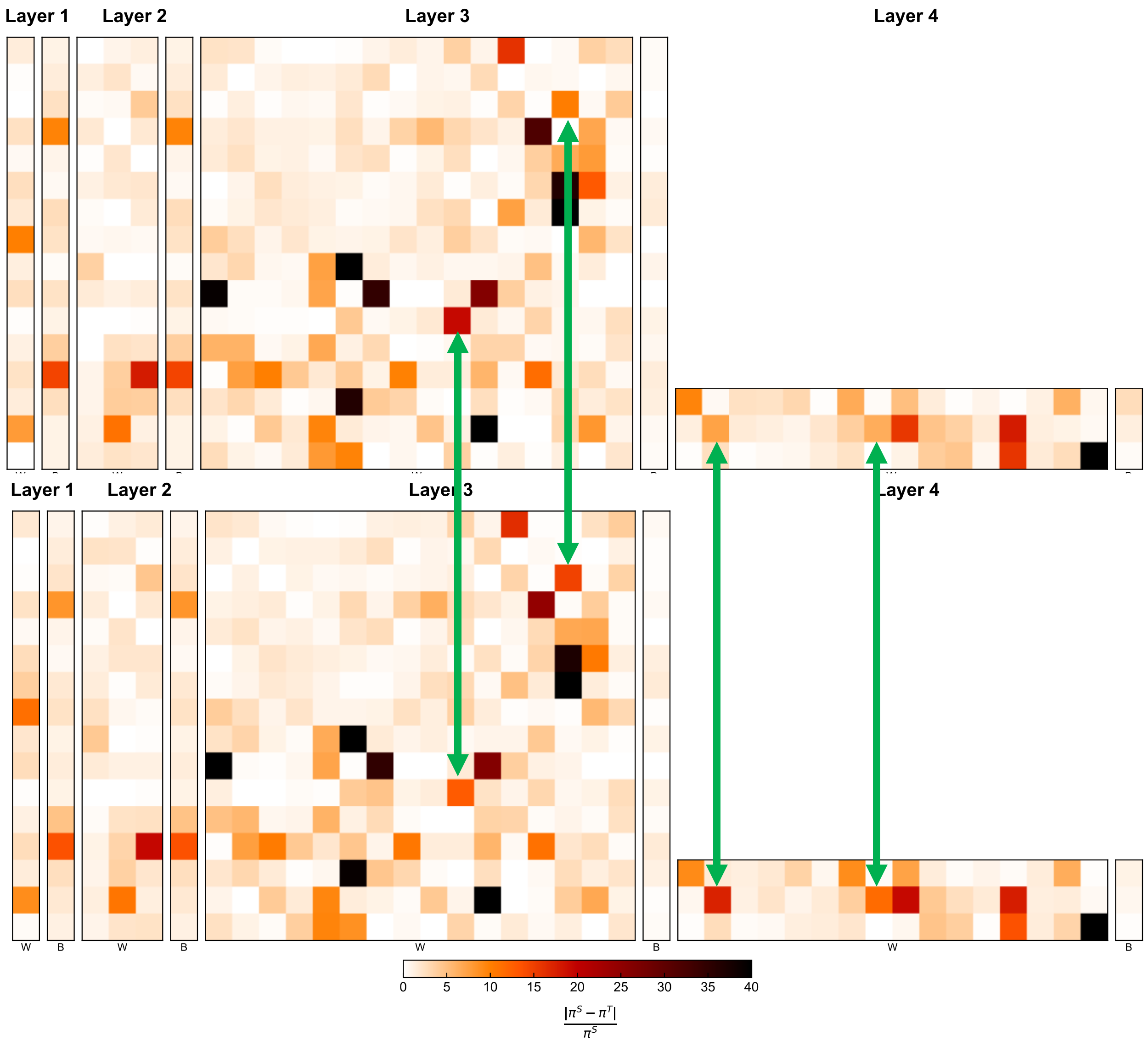


[1] Kumar, P., & Rawlings, J. B. (2023). Structured nonlinear process modeling using neural networks and application to economic optimization. *Computers and Chemical Engineering*, *177*. https://doi.org/10.1016/j.compchemeng.2023.108314

Changes to the NN parameters are distributed through the model

SEKF Weights

Adam Weights

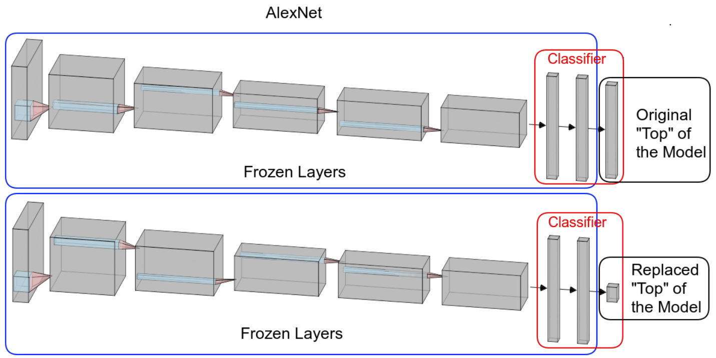


Similar but not identical

Small Changes in NN Parameters Adapt the Model (Source → Target)

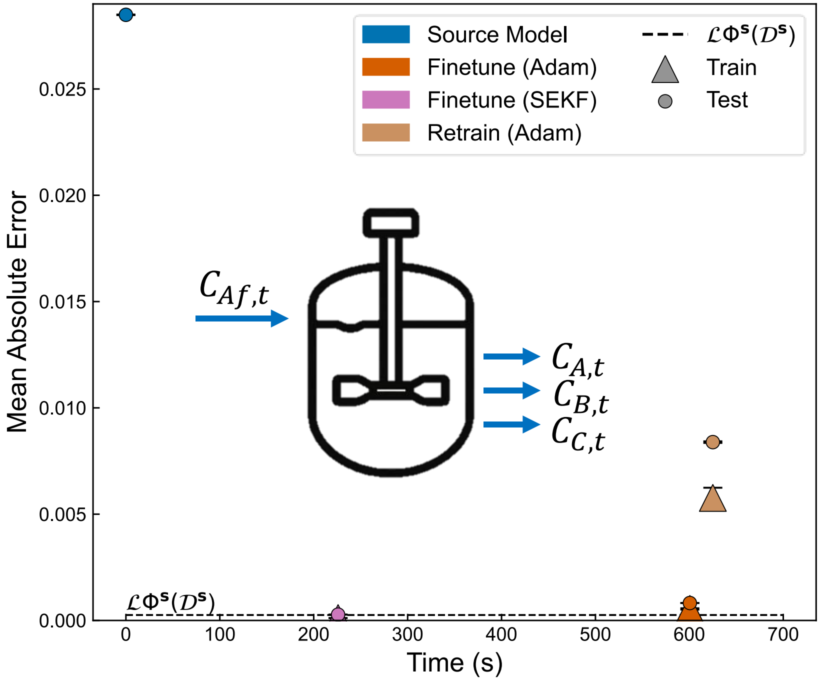
Identical processes vary in behavior.

Transfer Learning is most common in image processing. Determining **which parameters to modify** for dynamical regression is an open question.

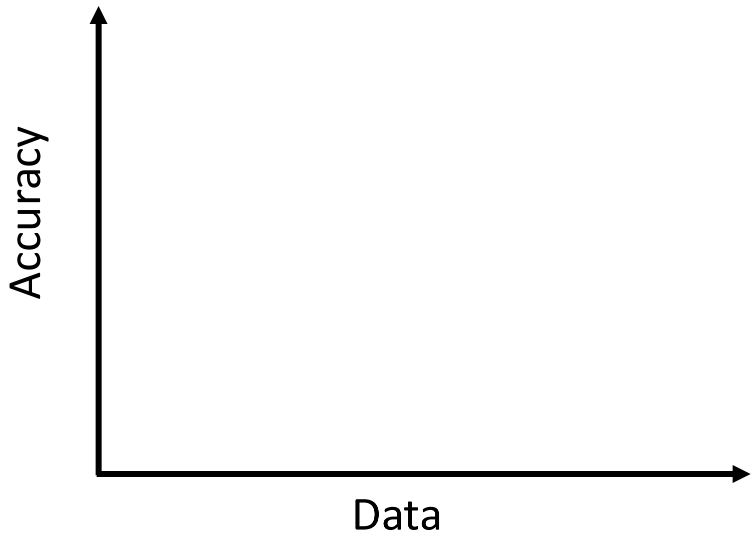


Small changes in the source model parameters can adapt the model to the target system.

The SEKF may reduce the generalization error associated with the limited target dataset



Future Work


Accuracy
Data


Temperature Control Lab
https://apmonitor.com/heat.htm

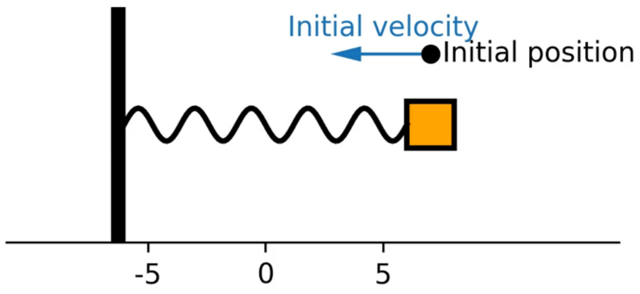
Thank you

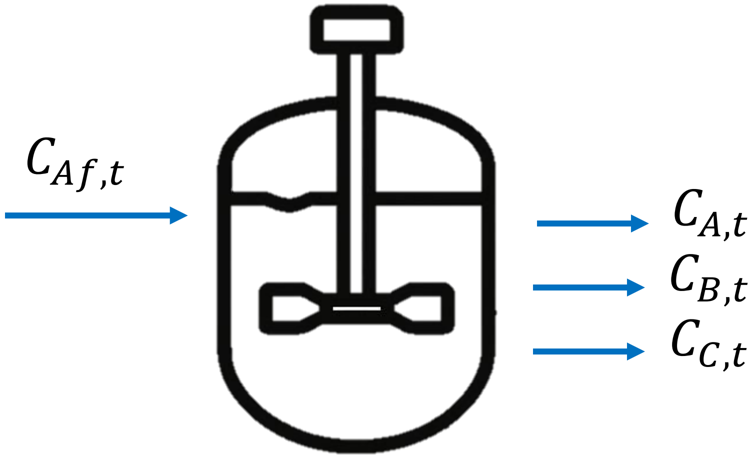
Baldea Group

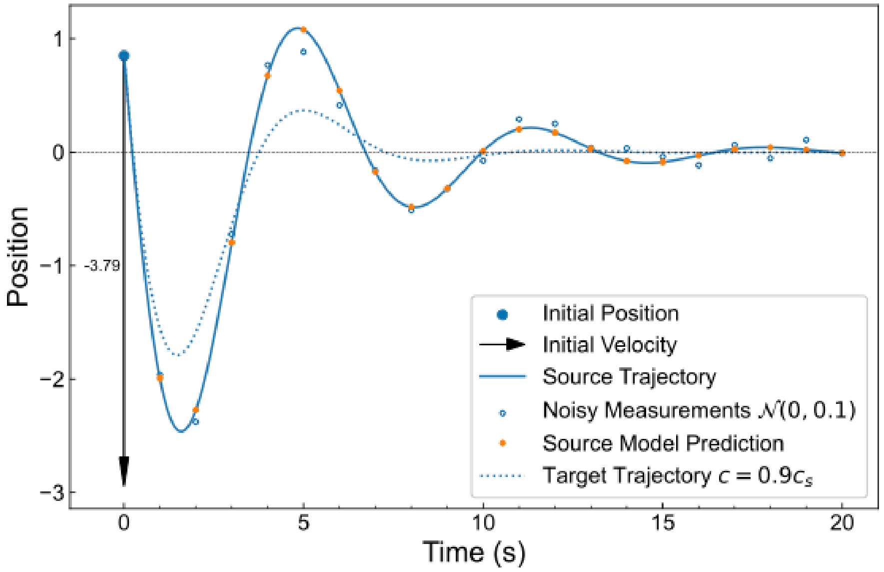
Korgel Group

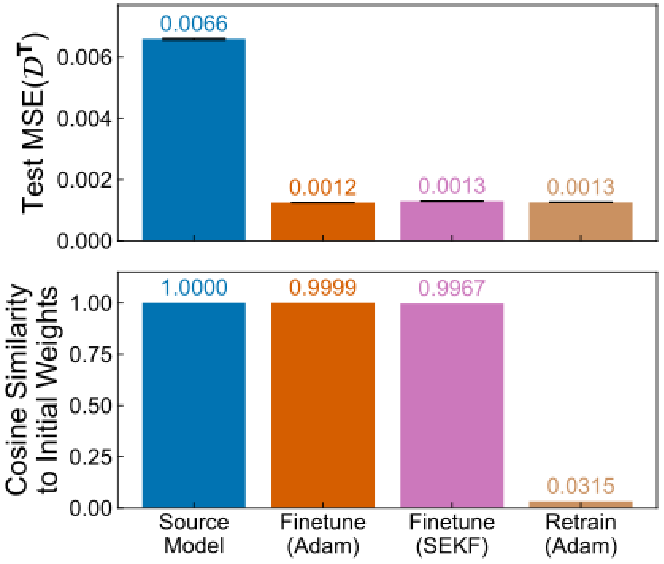
Thank you to Tyler Soderstrom and ExxonMobil for the support

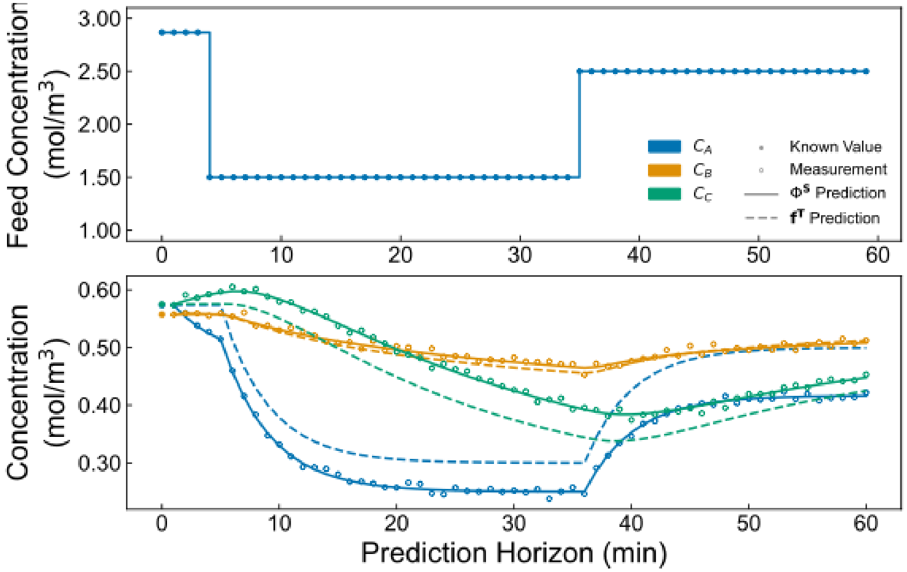
Questions?

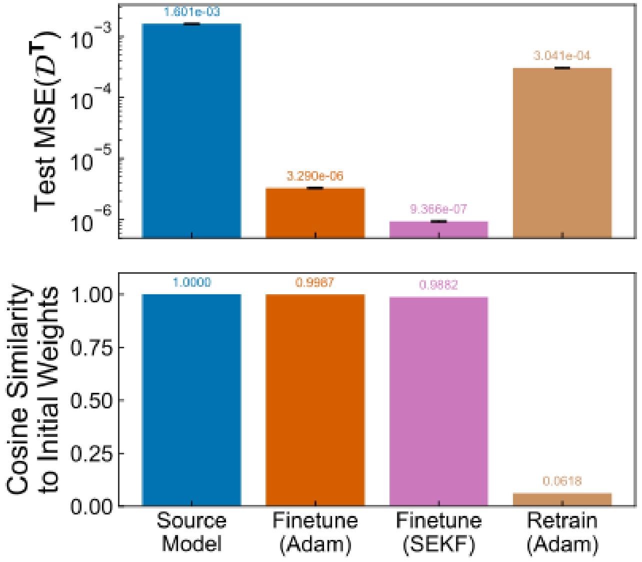

Initial velocity
Initial position
-5
0
5


$C_{Af,t}$
$C_{A,t}$
$C_{B,t}$
$C_{C,t}$


Position
Time (s)
-3.79
Initial Position
Initial Velocity
Source Trajectory
Noisy Measurements $\mathcal{N}(0, 0.1)$
Source Model Prediction
Target Trajectory $c = 0.9c_s$


Test MSE($\mathcal{D}^T$)
0.0066
0.0012
0.0013
0.0013
Cosine Similarity to Initial Weights
1.0000
0.9999
0.9967
0.0315
Source Model
Finetune (Adam)
Finetune (SEKF)
Retrain (Adam)


Feed Concentration (mol/m^3)
Concentration (mol/m^3)
Prediction Horizon (min)
C_A
C_B
C_C
Known Value
Measurement
Φ^S Prediction
f^T Prediction


Test MSE($\mathcal{D}^T$)
1.601e-03
3.290e-06
9.366e-07
3.041e-04
Cosine Similarity to Initial Weights
1.0000
0.9987
0.9882
0.0618
Source Model
Finetune (Adam)
Finetune (SEKF)
Retrain (Adam)

Supplemental Slides

Damped Spring System: Source Model Training Details

Training Data

- 100,000 Examples:
  - 10,000 for training
  - 10,000 for validation
  - 80,000 for testing
- $x_0, \dot{x}_0$ for each examples drawn from uniform distribution [-5,5]
- AHSA Hyperparameter Tuning w/ 100 samples
- Max 200 Epochs
- Hyperparameters
  - Batch size (16, 32, 64, 128, 256, 1028, 4096)
  - Decrease LR on Plateau Scheduler
    - Initial learning rate (1×10^{-6} - 1×10^{-1})
    - Learning rate patience(10, 20, 30, 40, 50)
    - Learning rate factor (0.1 - 0.9)

Best Model:

- Batch Size: 16
- Initial learning rate: 0.03182
- Patience: 10 epochs
- LR factor: 0.3442

Damped Spring System: Transfer Learning Details

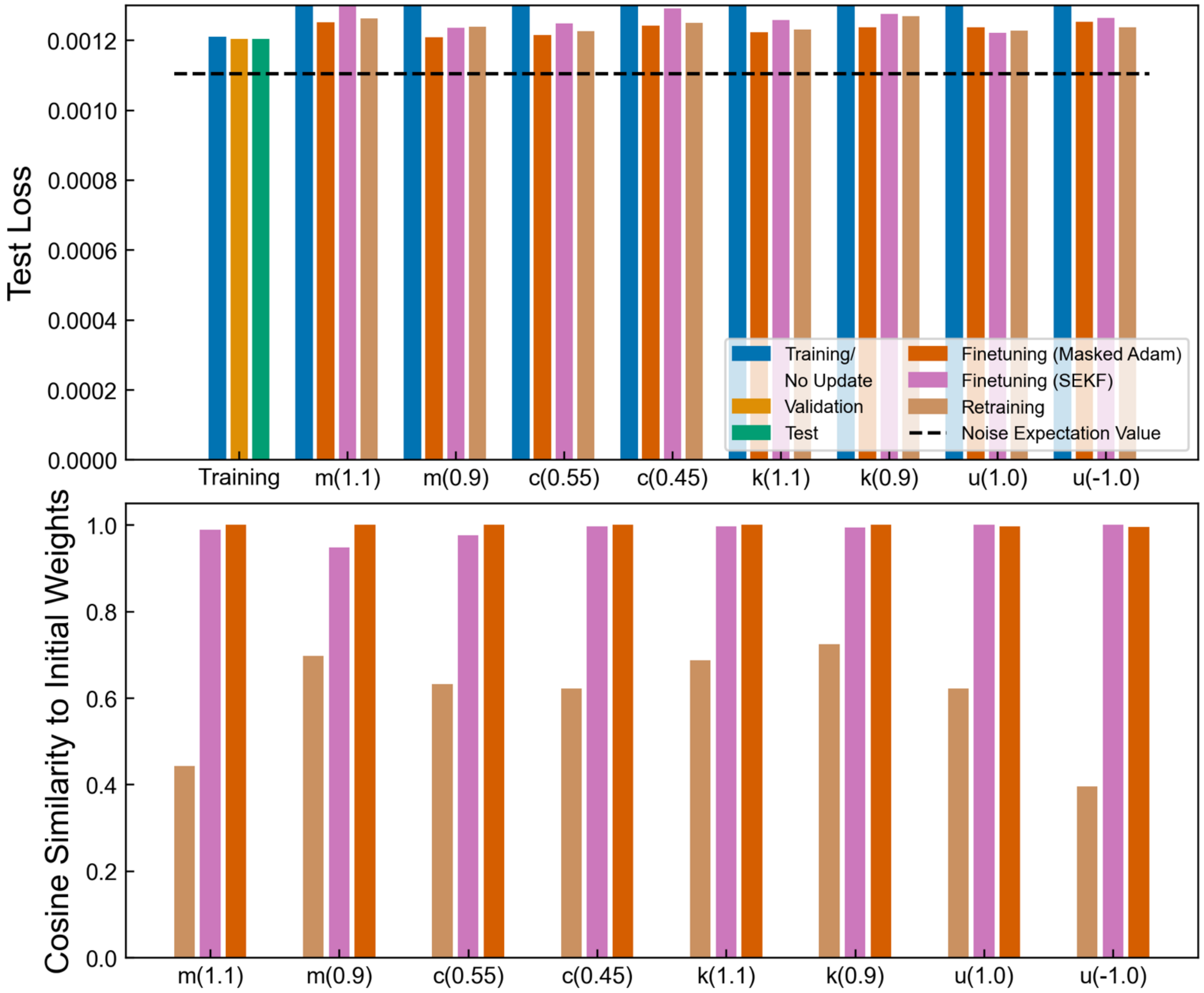
Training Data

- 100,000 Examples:
  - 10,000 for training
  - 10,000 for validation
  - 80,000 for testing
- $x_0, \dot{x}_0$ for each examples drawn from uniform distribution [-5,5]
- AHSA Hyperparameter Tuning w/ 100 samples
- Max 200 Epochs
- Hyperparameters
  - Batch size (16, 32, 64, 128, 256, 1028, 4096)
  - Decrease LR on Plateau Scheduler
    - Initial learning rate (1×10^{-6} - 1×10^{-1})
    - Learning rate patience(10, 20, 30, 40, 50)
    - Learning rate factor (0.1 - 0.9)

Best SEKF Parameters:

- Batch Size: 32
- Initial learning rate: 100
- Patience: 100 epochs
- LR factor: -
- Q = $1 \times 10^{-6}\ \mathbf{I}$
- Initial P = 100 $\mathbf{I}$
- % parameters updated: 54.5 %

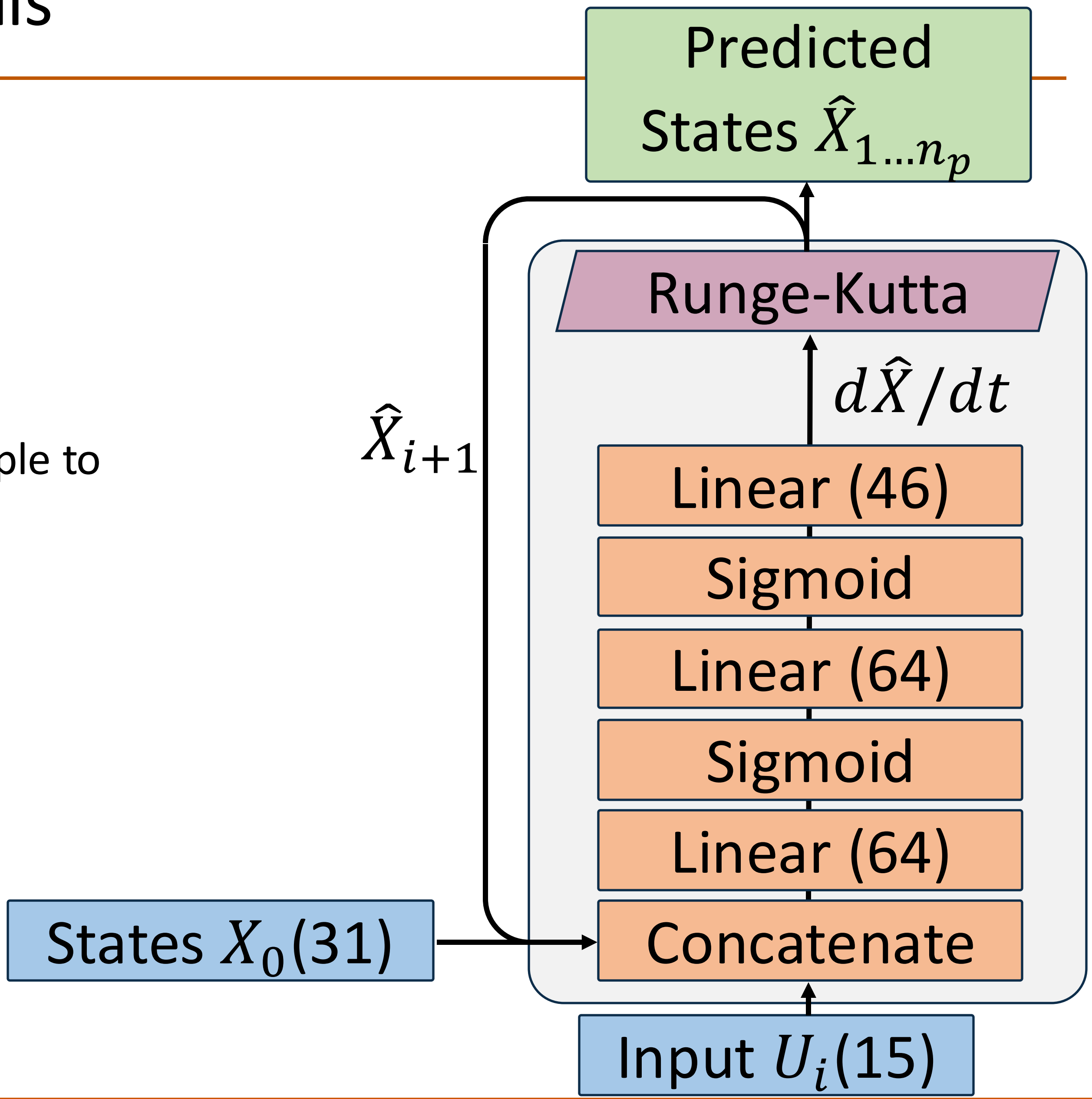
Damped Spring System: Full Results



CSTR System: Initial Training Details

Training Data

- 1 Year of data
  - 80% for training
  - 10% for validation
  - 10% for testing
- 5 minute interval between each input-output example to maximize throughput while minimizing redundant information
- Hyperparameters
  - Epochs: 10
  - Learning Rate: 0.1
  - Batch Size: 256



Multi-stream SEKF

Problem: The SEKF performs updates on single datapoints. When performing this across large datasets, this becomes time-consuming to iterate across all datapoints individually.

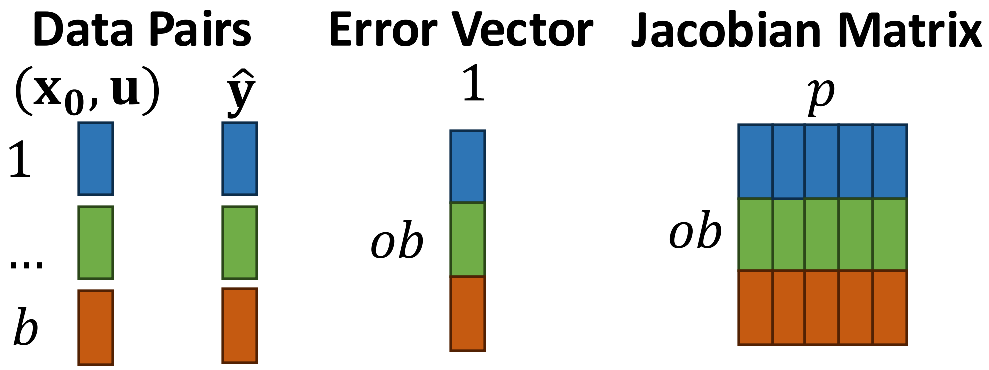
For a **batch size** $\boldsymbol{b}$, where each prediction includes $\boldsymbol{o}$ **outputs** produced from a model with $\boldsymbol{p}$ **parameters**:

Standard Batched Operations

- Individual losses averaged into a single scalar value
- Update calculated with respect to that single averaged loss

Multi-Stream Operations

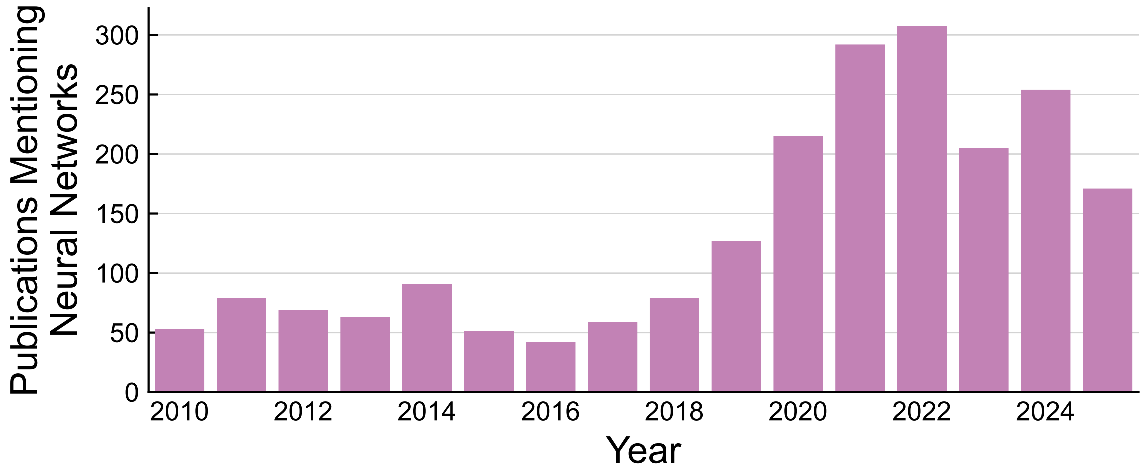
- Individual losses and Jacobians concatenated
- Update is the sum of each individual update calculated in parallel

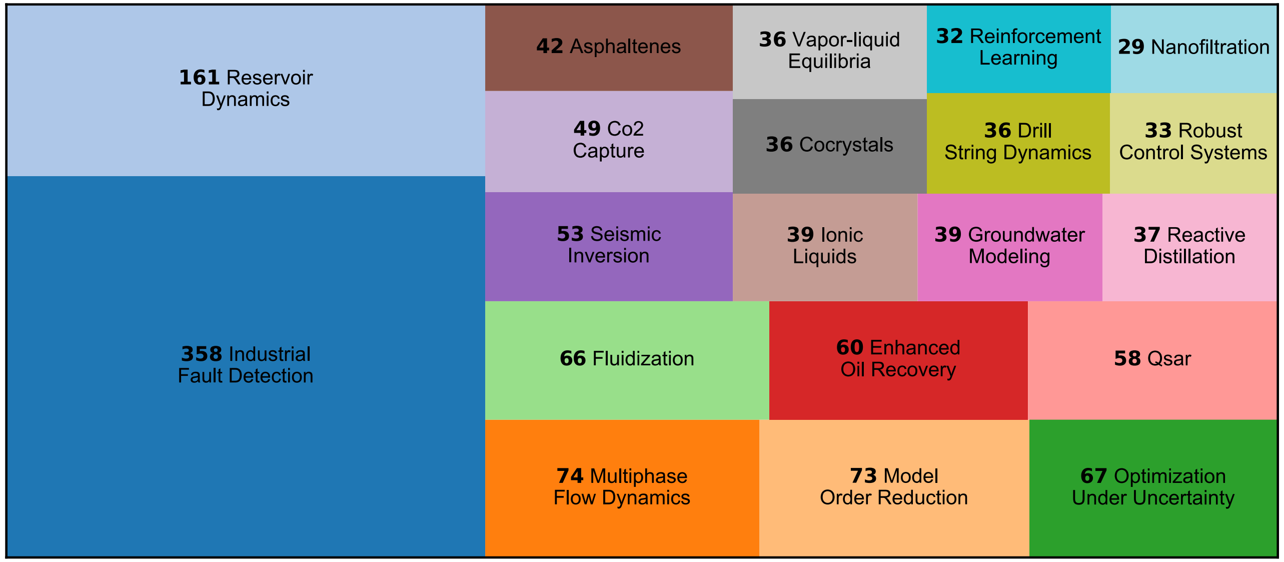


$$\boldsymbol{e}_{ob,1} = \boldsymbol{y}_{ob,1} - \widetilde{\boldsymbol{y}}_{ob,1}$$
$$\boldsymbol{H}_{ob,p} = \nabla_{\boldsymbol{\pi}} \boldsymbol{y}_{ob,1}$$
$$\boldsymbol{A}_{ob,ob} = \left[\boldsymbol{R}_{ob,ob} + \boldsymbol{H}_{ob,p} \boldsymbol{P}_{p,p} \boldsymbol{H}^{T}{}_{p,ob}\right]^{-1}$$
$$\boldsymbol{K}_{p,ob} = \boldsymbol{P}_{p,p} \boldsymbol{H}^{T}{}_{p,ob} \boldsymbol{A}_{ob,ob}$$
$$\widehat{\boldsymbol{\pi}}_{p,1} = \widehat{\boldsymbol{\pi}}_{p,1} + \boldsymbol{K}_{p,ob} \boldsymbol{e}_{ob,1}$$
$$\boldsymbol{P}_{p,p} = \boldsymbol{P}_{p,p} - \boldsymbol{K}_{p,ob} \boldsymbol{H}_{ob,p} \boldsymbol{P}_{k} + \boldsymbol{Q}_{p,p}$$

Hot topic: neural networks in Process Systems Engineering

- Data science, machine learning, and artificial intelligence research and tools have been increasing at an exponential rate.
  - ***Neural networks*** have been the "hot topic"
- Most research assumes that there is sufficient **data coverage and quantity**, that the training data is drawn from the **same system** as the application, and that the ***system is stationary***
- Data-driven models have no generalization or extrapolation guarantees





Web of Science Query Topic: "Neural Network" Dates: 2010 – 2025 Journals: Computers & Chemical Engineering, Industrial & Engineering Chemistry, Scientific Reports, Renewable Sustainable Energy Reviews, Journal of Process Control, Chemical Engineering Science, Chemical Engineering Research Design, AICHE Journal