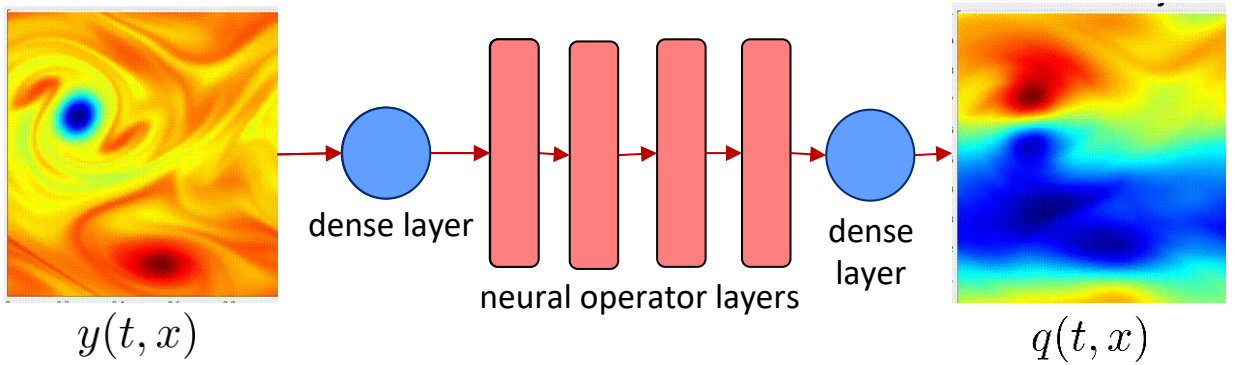
INFINITEMATHOPTAI.JL: EMBEDDING SURROGATES FOR INFINITE-DIMENSIONAL OPTIMIZATION

Dr. Joshua Pulsipher



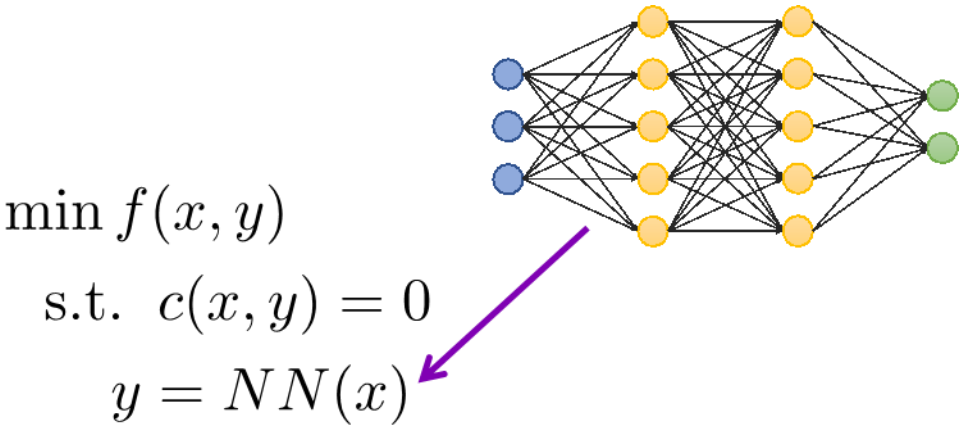
UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

OUTLINE

- Introduction and Motivation
- Infinite-Dimensional Optimization
- InfiniteMathOptAI
- Data-Driven NMPC Case Study

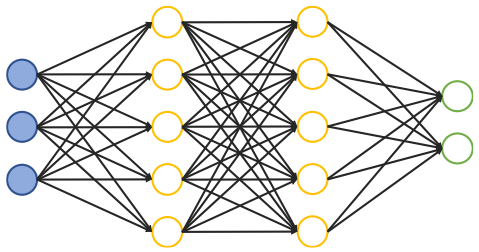
OUTLINE

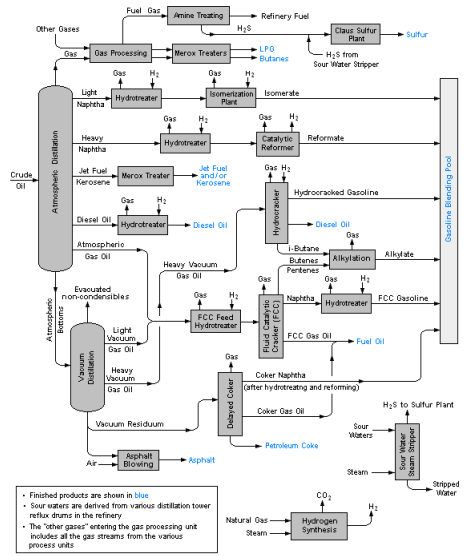
- **Introduction and Motivation**
- Infinite-Dimensional Optimization
- InfiniteMathOptAI
- Data-Driven NMPC Case Study



MACHINE LEARNING MODELS

- Two common paradigms
  - Replace computationally intensive modelling equations with **fast-to-evaluate ML model**
  - Leverage **data-driven model** to capture phenomena not fully described by equations
- Neural networks (NNs) are a popular choice
- Numerous approaches have been proposed





EMBEDDING AN NN INTO AN OPTIMIZATION PROBLEM

- **Models predict** action outcomes, **optimization prescribes** the "best" actions
- Embedding NNs into optimization problem **enables a greater breadth of models** to be optimized
- **ReLU** NNs are common → produce piecewise linear models
- Many recently released software tools

GUROBI OPTIMIZATION

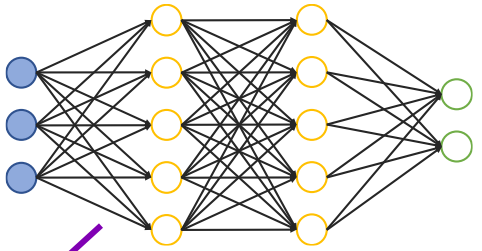
OptiCL **JANOS**

MeLOn **MathOptAI**

PySCIPOpt-ML

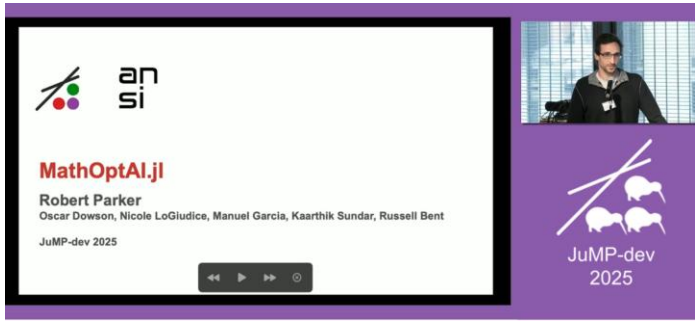
reluMIP

$$\min f(x, y)$$
$$\text{s.t. } c(x, y) = 0$$
$$y = NN(x)$$



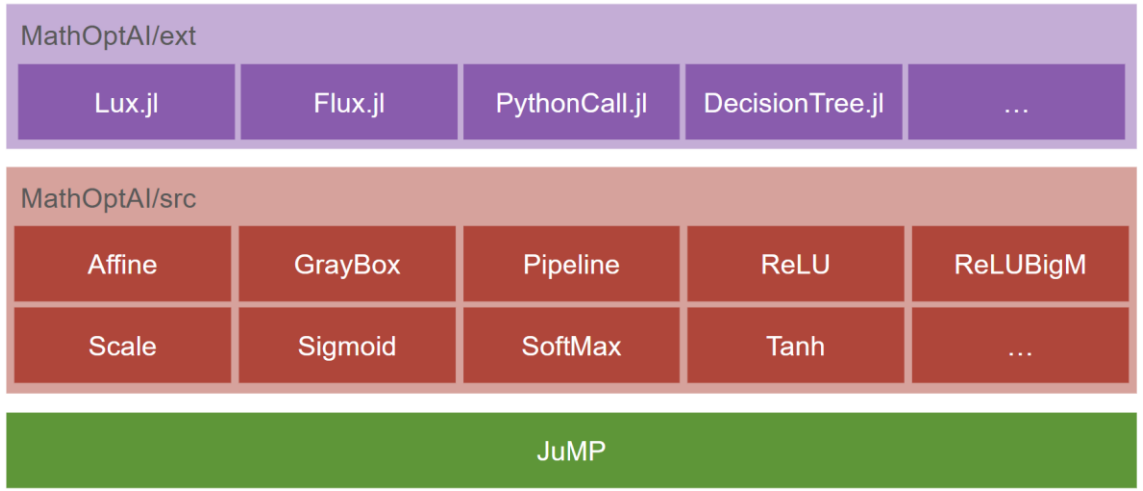
López-Flores et. al. "Process Systems Engineering Tools for Optimization of Trained Machine Learning Models: Comparative and Perspective." (2024)

MATHOPTAI.JL



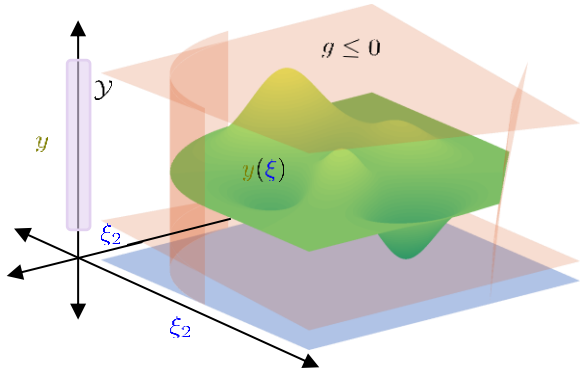
- **Embeds ML models** as expressions or oracles
  - Neural networks (Flux, Lux, PyTorch)
  - Gaussian processes (AbstractGPs)
  - Decision trees (DecisionTree, EvoTrees)
  - More
- Can also **train neural networks**
  - Replace the weights with variables
  - Uses existing weight values as initial guesses

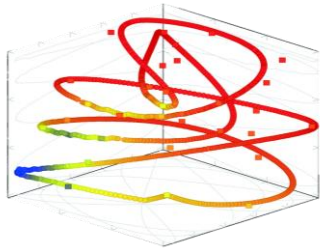
```julia
using JuMP, MathOptAI, Flux
model = Model()
@variable(model, x[1:4, 1:4]);
predictor = Flux.Chain(Flux.MaxPool((2, 2)), Flux.flatten)
y, formulation = MathOptAI.add_predictor(model, predictor, x)
```



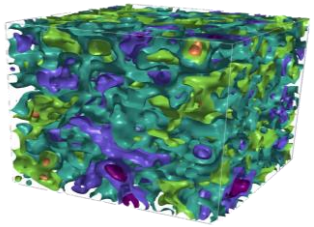
WHAT ABOUT INFINITE-DIMENSIONAL OPTIMIZATION?

- Variables and/or constraints **indexed over continuous domains**
  - Modelled and solved via InfiniteOpt

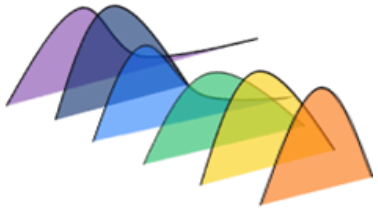




Time



Space

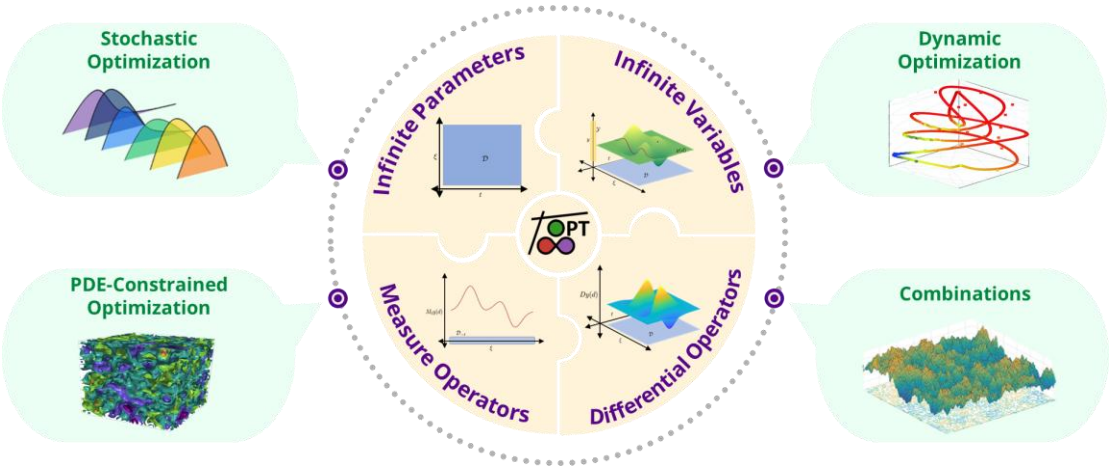


Uncertainty

- Complex systems that increasingly leverage **machine learning models**
- **No software tools** to embed infinite-dimensional models in optimization AMLs

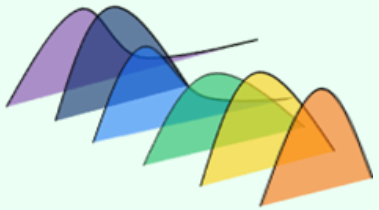
OUTLINE

- Introduction and Motivation
- **Infinite-Dimensional Optimization**
- InfiniteMathOptAI
- Data-Driven NMPC Case Study

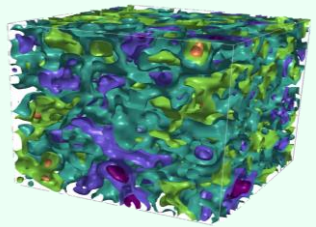


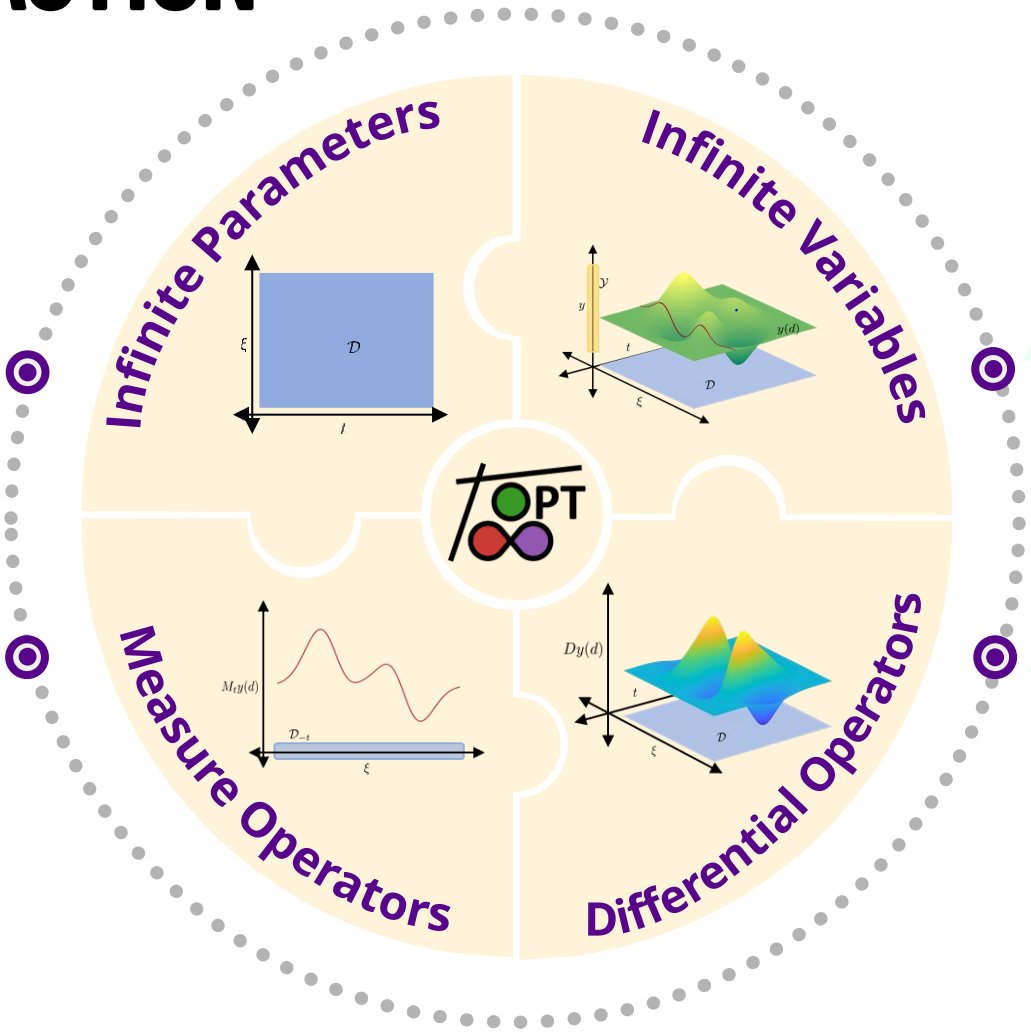
UNIFYING ABSTRACTION

Stochastic Optimization

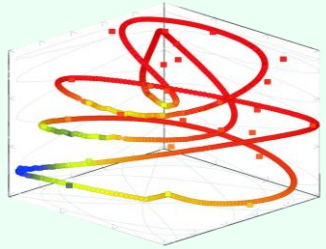


PDE-Constrained Optimization

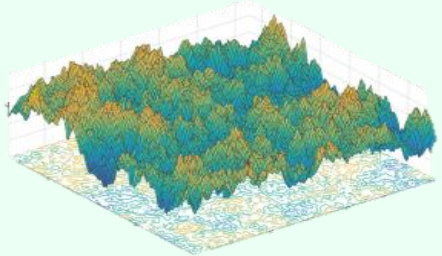




Dynamic Optimization



Combinations



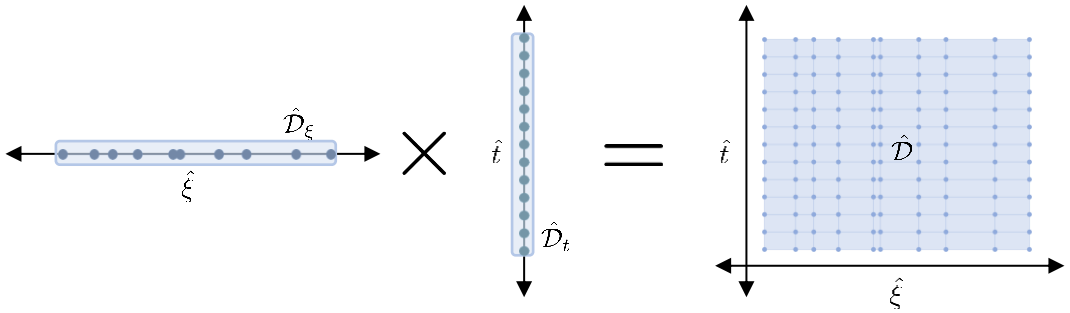
J. L. Pulsipher, W. Zhang, T. J. Hongisto, and V. M. Zavala. "A unifying modeling abstraction for infinite-dimensional optimization." 2022

SOLUTION VIA DIRECT TRANSCRIPTION

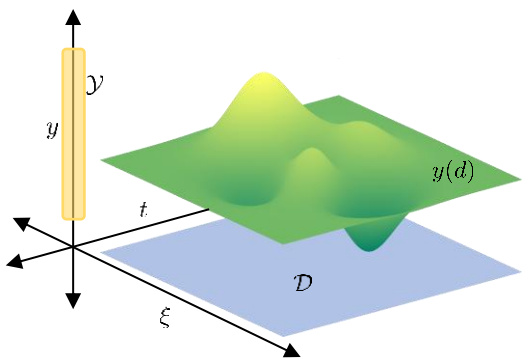
- *Idea:* Project continuous domain(s) onto a discretized point grid
  - Exhibits repeated structure

$$\sin^2(y(t)) \leq 42, \ t \in \mathcal{D}_t$$

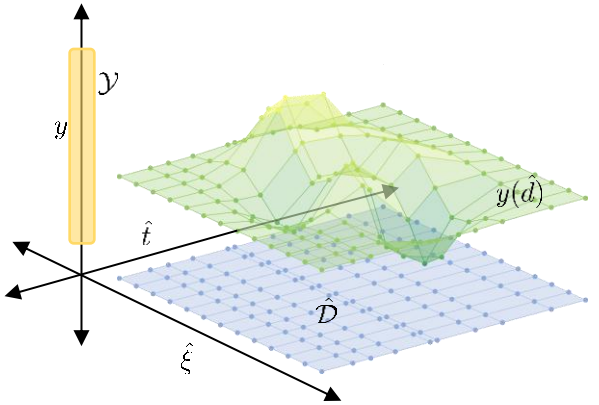
$$\sin^2(y_k) \leq 42, \ k \in \mathcal{K}$$



- Optimize the **large discretized problem** with an optimization solver



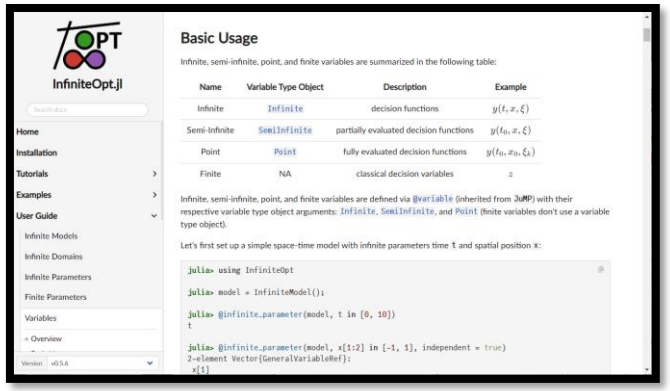
Direct transcription



InfiniteOpt

Why is it Different?

- Implements **unifying abstraction**
  - Models a wide range of problems
  - Leverages structure to **accelerate solutions**
- Implemented in julia
  - Enables **intuitive** symbolic expressions
  - Highly **performant**
- **Extensive resources**
  - Documentation, tutorials, examples, forum, short courses, videos



Intuitive Modeling API

$$\frac{\partial y_b(t,\xi)}{\partial t} = 2y_b(t,\xi)^2 + y_a(t) - z_1$$
$$\mathbb{E}_\xi\left[y_c(t,\xi)\right] \geq \alpha$$
$$y_a(0) + z_2 = \beta$$

```
@constraint(m, ∂(yb, t) == 2yb^2 + ya - z[1])
@constraint(m, 𝔼(yc, ξ) ≥ α)
@constraint(m, ya(0) + z[2] == β)
```

Impact

- 1000s of downloads
- Use cases in **diverse disciplines**
  - e.g., evolutionary biology, rocketry, economics, autonomous vehicles

Try it @ https://github.com/infiniteopt/InfiniteOpt.jl

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

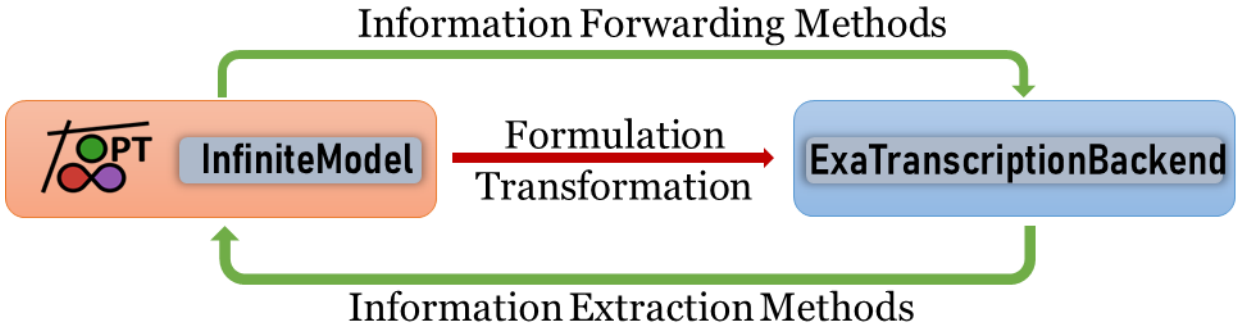
INFINITEEXAMODELS.JL

- Bridges the gap between InfiniteOpt & ExaModels
- **Automates transcription** via established transformation interface
- Leverages repeated structure to **drastically reduce model creation time**

```
1   using InfiniteOpt, InfiniteExaModels, MadNLPGPU, CUDA
2   transform_backend = ExaTranscriptionBackend(MadNLPSolver, backend = CUDABackend())
3   model = InfiniteModel(transform_backend)
```

- Enables **GPU** AD & solution

MadNLP



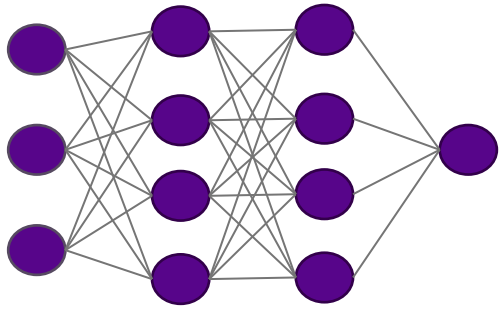
Evelyn Gondosiswanto and Joshua L. Pulsipher. “Advances to Modelling and Solving Infinite-Dimensional Optimization Problems in InfiniteOpt.jl." 2025. Under Review

NEURAL DIFFERENTIAL & ALGEBRAIC EQUATIONS (DAES)

- Many engineering systems are dynamic (e.g., transient balances)
- Neural DAEs learn RHS of ODEs via NN
- **Reduces NN model complexity** vs. learning y(t) directly
  - Less overfitting
  - Decreased data requirement

$$\frac{dy}{dt} = \frac{t+y}{t-y} \Rightarrow \frac{1}{2}\log\left(\frac{y^2}{t^2}+1\right) - \tan^{-1}\left(\frac{y}{t}\right) = C - \log(t)$$

$$\frac{dy(t)}{dt} =$$

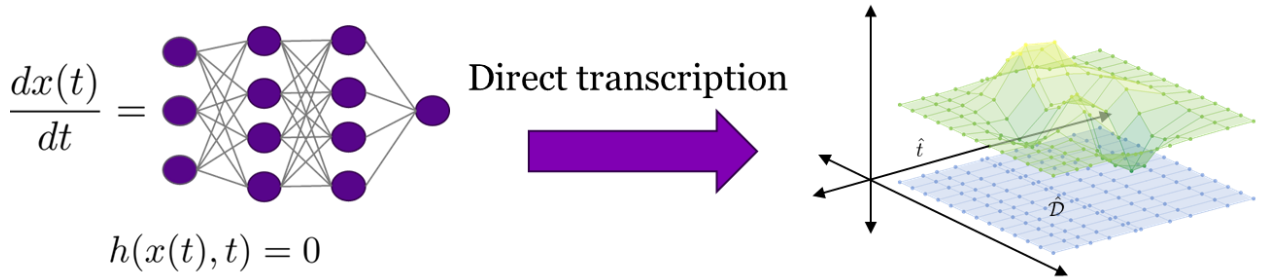


$$h(y(t), t) = 0$$

- Neural DAEs are **discretization agnostic**
- **Difficult to train** with traditional sequential methods

OUTLINE

- Introduction and Motivation
- Infinite-Dimensional Optimization
- **InfiniteMathOptAI**
- Data-Driven NMPC Case Study



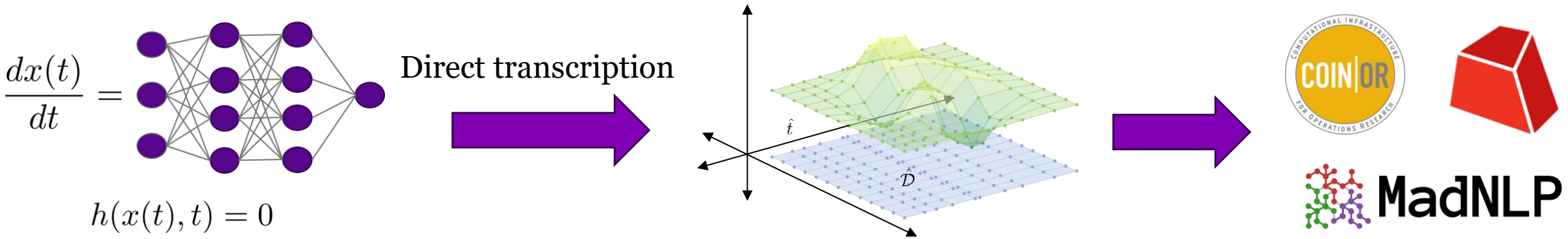
INFINITEMATHOPTAI.JL

- Bridges InfiniteOpt & MathOptAI
- Enables infinite ML models
  - Embed ML models in DAEs
    - E.g., Neural DAEs
    - Anything supported by MathOptAI
  - Opens door for new ML models
    - E.g., Neural operators
- Stable **training of neural DAEs**

```julia
using InfiniteOpt, MathOptAI, Flux, Ipopt, DelimitedFiles

t_data = collect(0:2:100)
y_data = vec(readdlm("mydata.txt", Float64))
u_input = 42

NN = Flux.Chain(
    Flux.Dense(2 => 4, tanh),
    Flux.Dense(4 => 1)
)

model = InfiniteModel(Ipopt.Optimizer)
@infinite_parameter(model, t in [0, 100], supports = t_data)

@variable(model, y, Infinite(t))

moai_nn = MathOptAI.build_predictor(NN)
moai_nn = MathOptAI.replace_weights_with_variables(model, moai_nn)
RHS, form = MathOptAI.add_predictor(model, moai_nn, [y, u_input])

@constraint(model, ∂(y, t) == RHS[1])
@constraint(model, y(0) == y_data[1])

@objective(model, Min, sum((y(td) - yd)^2 for (td, yd) in zip(t_data, y_data)))

optimize!(model)
```

TRAINING NEURAL DAES VIA DIRECT TRANSCRIPTION

- *Idea:* Convert neural ODE/DAE into **algebraic equations** via numerical method and solve using a **2nd-order NLP solver** with a sparse linear solver.
  - Initial work explored by Shapovalova et al. (2025) and Lueg et al. (2025)
- More numerically stable and enables **hard constraints** (no more soft constraints)
- InfiniteOpt accelerates this on **GPU via InfiniteExaModels**



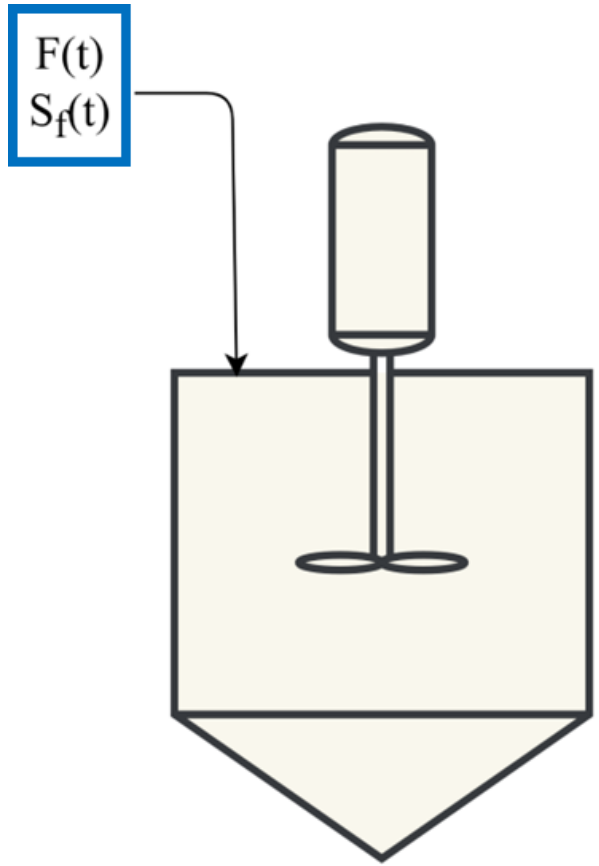
EMBEDDING TRAINED MODELS

- Use `MathOptAI.add_predictor`
  - Trained via InfiniteOpt or externally
- Solve like any InfiniteOpt model
- **GPU acceleration** for differentiable models

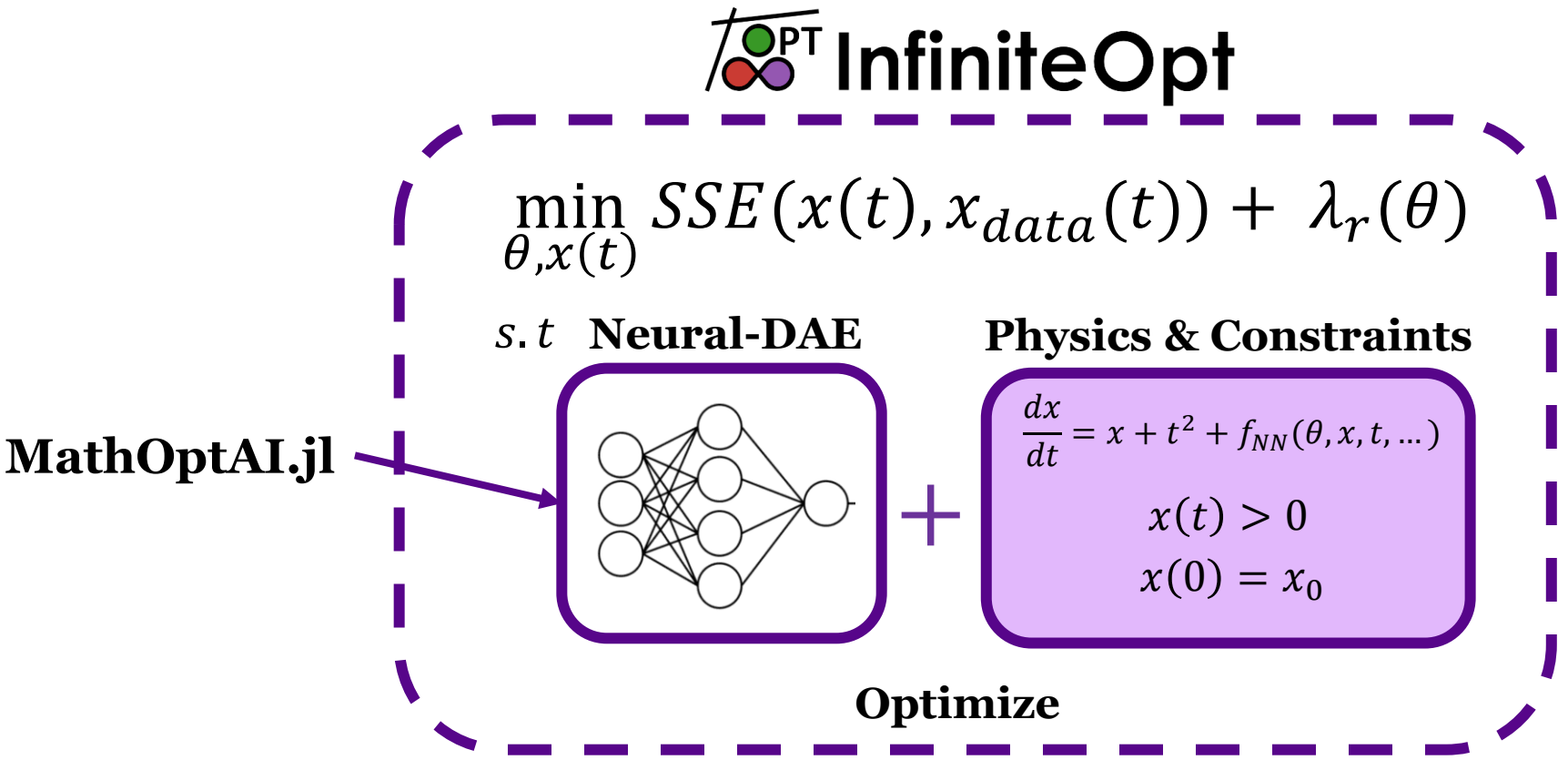
```julia
using InfiniteOpt, Flux, MathOptAI
NN = predictor(
    Flux.Chain(
        Flux.Dense(2, 16, Flux.relu),
        Flux.Dense(16, 16, Flux.relu),
        Flux.Dense(16, 2)
    )
)
model = InfiniteModel()
@infinite_parameter(model, t ∈ [0, 10])
@variable(model, x[1:2], Infinite(t))
y, formulation = add_predictor(model, NN, x)
@constraint(model, [i in 1:2], ∂(x[i], t) == y[i])
```

OUTLINE

- Introduction and Motivation
- Infinite-Dimensional Optimization
- InfiniteMathOptAI
- **Data-Driven NMPC Case Study**



PROPOSED NEURAL-DAE TRAINING



Model Optimization

- Feasibility solve to warm start training
- Utilize InfiniteExaModels to perform training on GPU
- Iteratively increase data set until N-DAE is robust

MadNLP

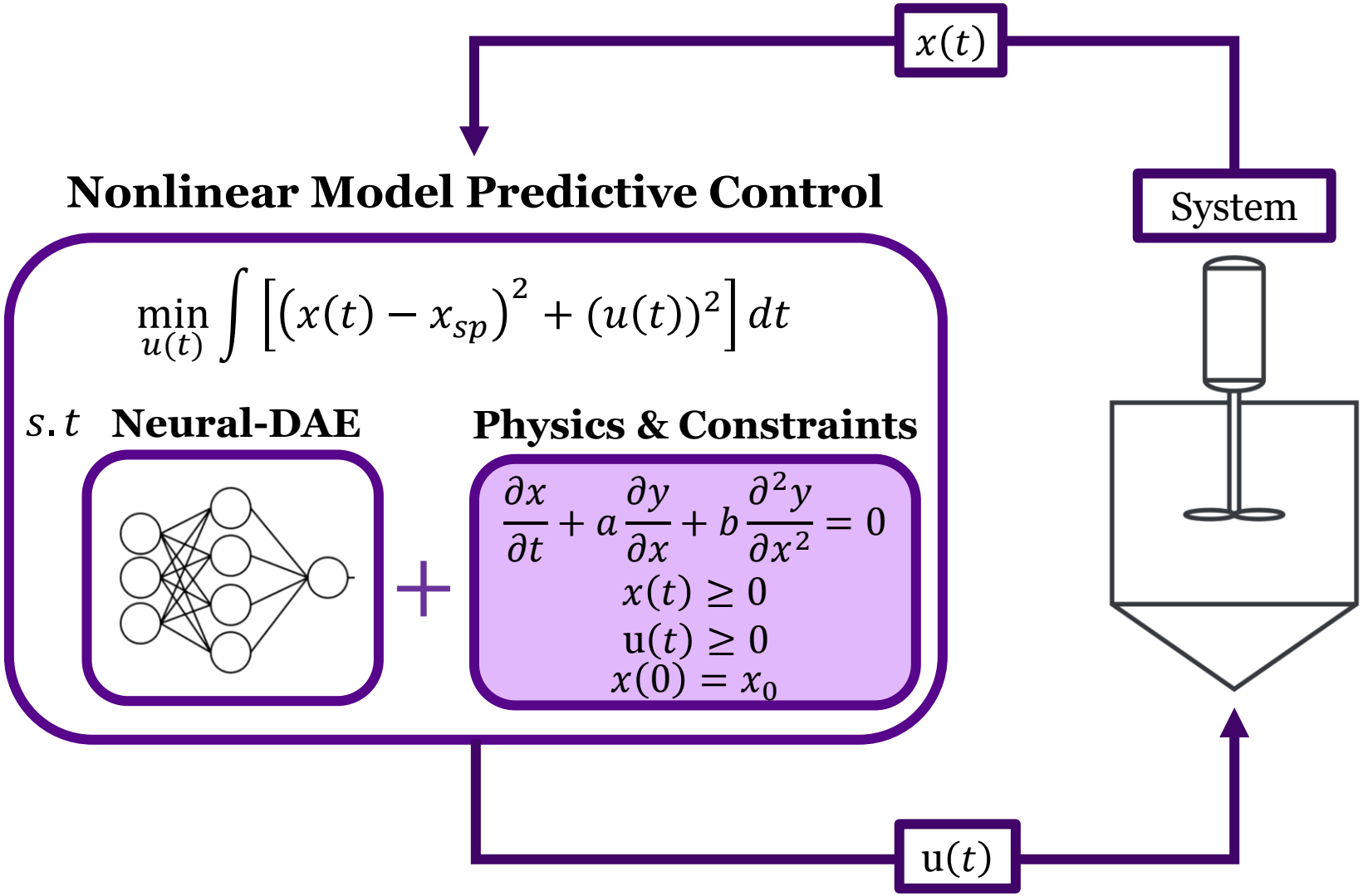
NVIDIA CUDA

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

E. Gondosiswanto and J. L. Pulsipher, "Advances to modeling and solving infinite-dimensional optimization problems in InfiniteOpt.jl," Digital Chemical Engineering, vol. 15, p. 100236, Jun. 2025, doi: https://doi.org/10.1016/j.dche.2025.100236.

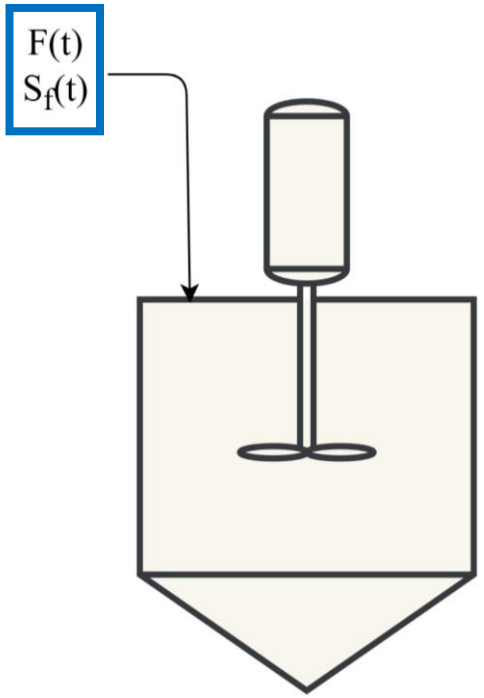
NMPC IMPLEMENTATION

- NMPC predicts system behaviour x(t)
- NMPC provides optimal control actions u(t) to reach setpoint x_{sp}
- Neural-DAE is embedded into the NMPC framework as constraints



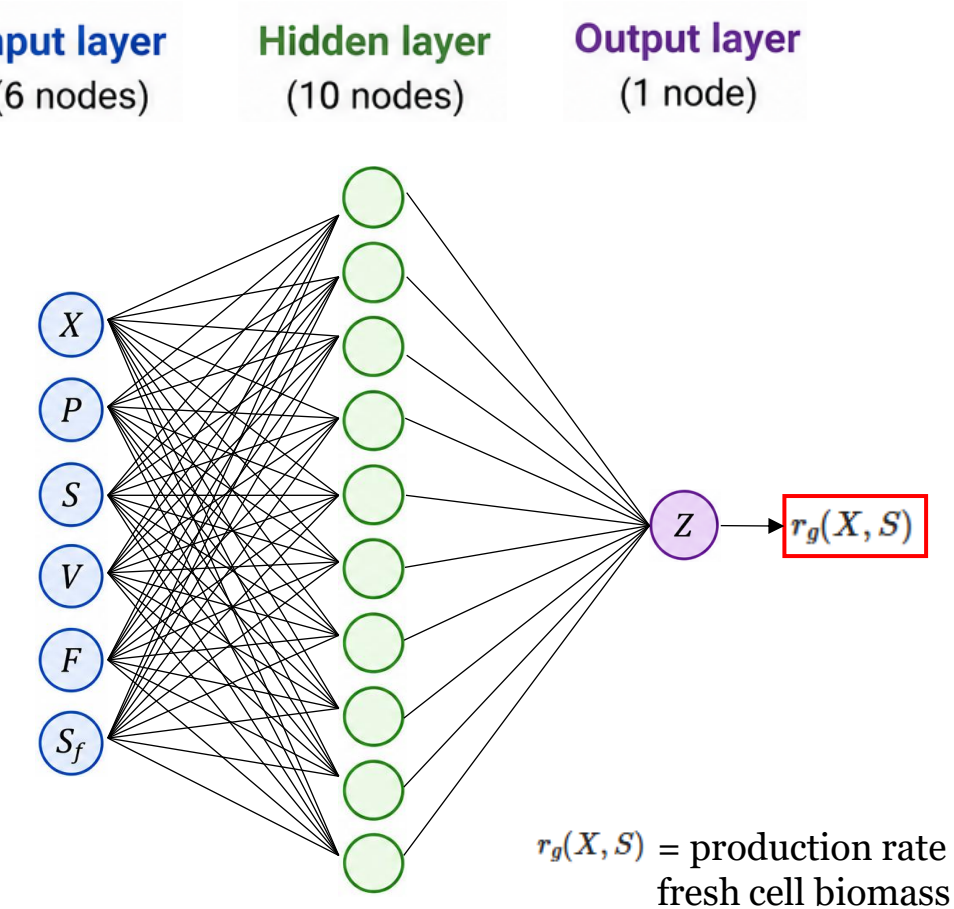
CASE STUDY: BIO-REACTOR MODEL PREDICTIVE CONTROL

Control product concentration by manipulating feed flow rate, and feed substrate concentration



System Dynamics

$$\frac{dX}{dT} = -\frac{F(t)}{V}X - r_g(X,S)$$
$$\frac{dP}{dT} = -\frac{F(t)}{V}P - Y_{P/X}r_g(X,S)$$
$$\frac{dS}{dT} = -\frac{F(t)}{V}(S_f - S) - \frac{1}{Y_{X/S}}r_g(X,S)$$
$$\frac{dV}{dT} = F(t)$$
$$X(t), P(t), S(t), V(t) \geq 0$$



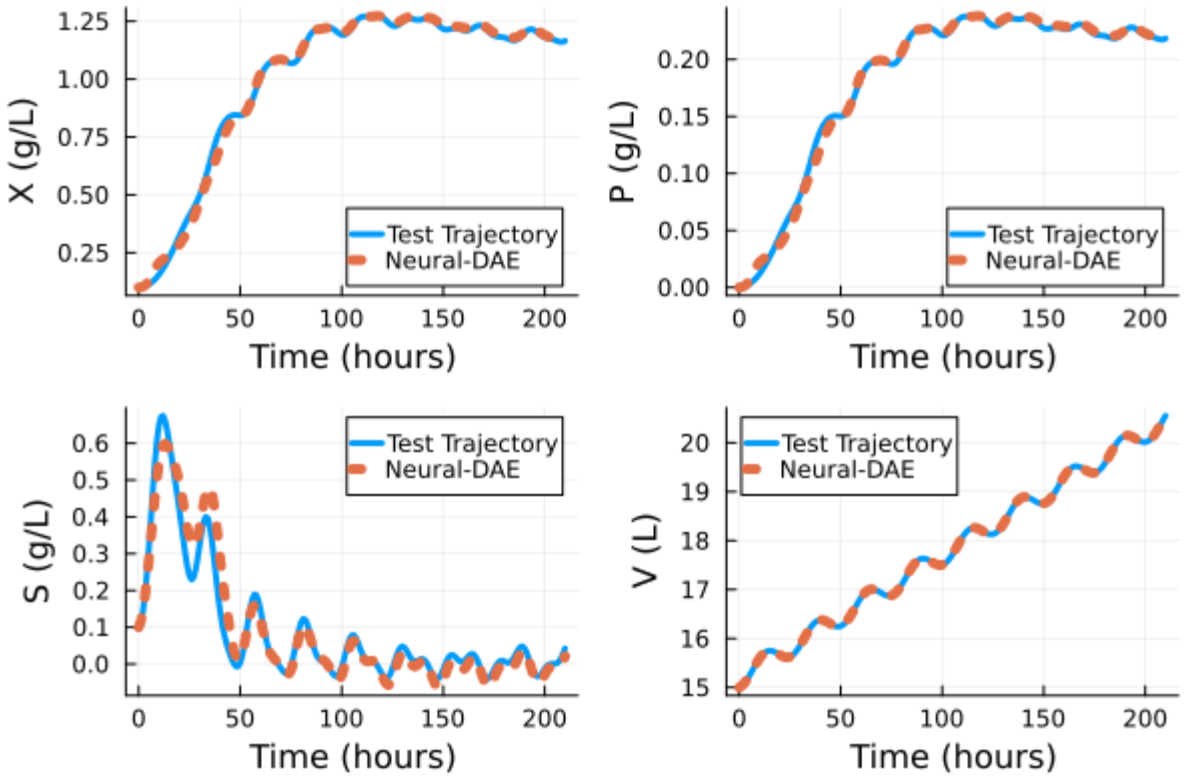
$r_g(X,S)$ = production rate of fresh cell biomass

A Simultaneous Approach for Training Neural Differential-Algebraic Systems of Equations," *Arxiv.org*, 2019. https://arxiv.org/html/2504.04665v1

NEURAL-DAE TRAINING AND VALIDATION

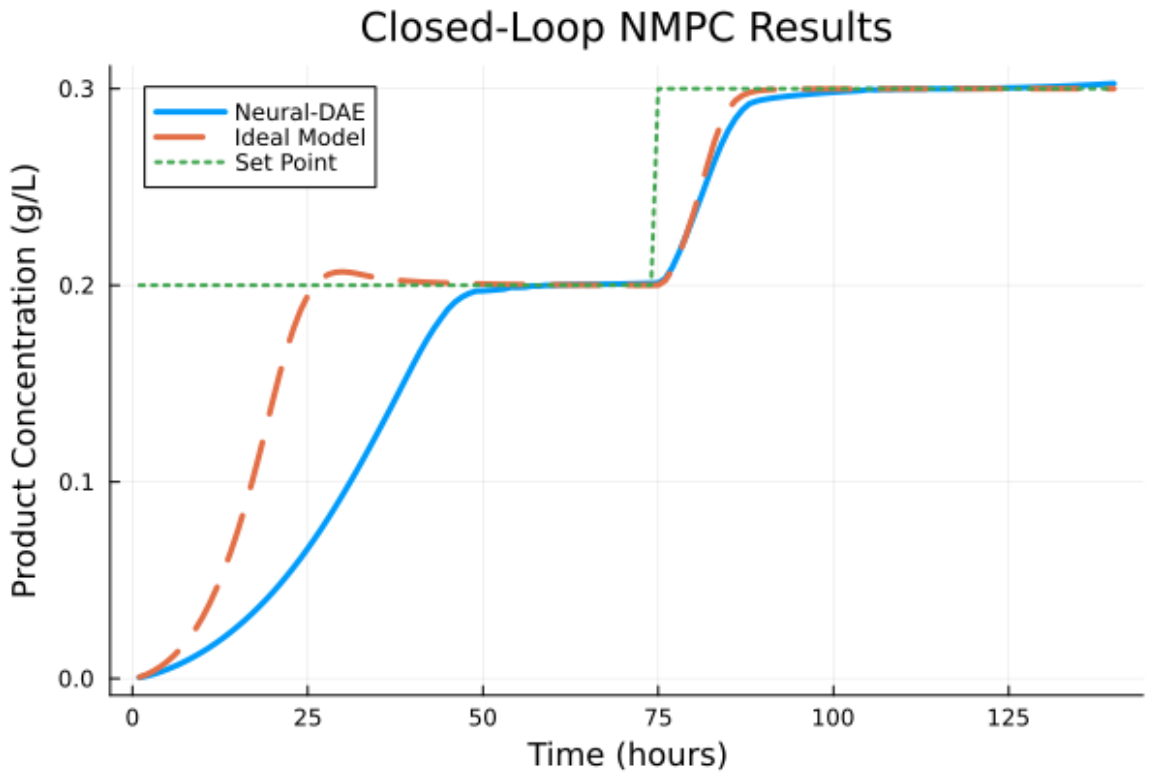
- Using RMSE, hybrid N-DAE training is over 99% accurate
- CPU training solve time ~ 40 seconds
- **GPU** training solve time ~ **3 seconds**
- Using RMSE, the validation results are over 95% accurate

Sinusoidal Input Comparison



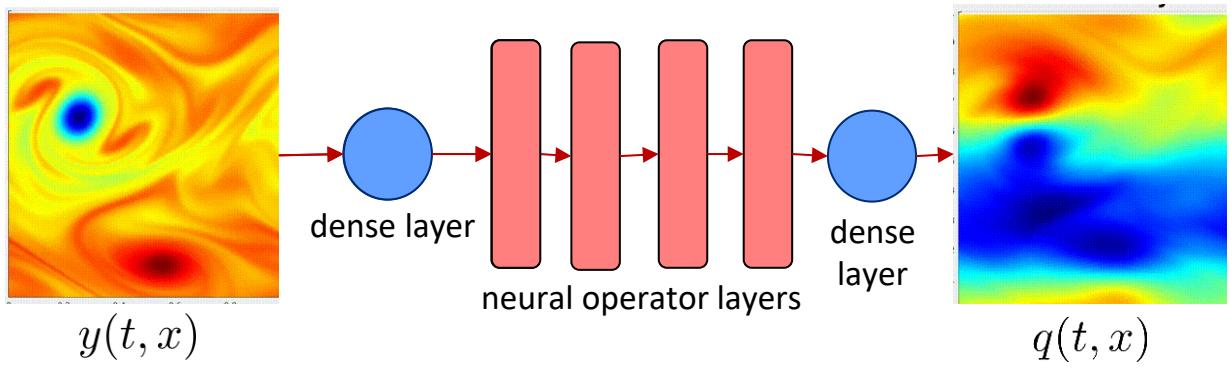
NEURAL-DAE NMPC RESULTS

- Open-Loop **CPU solve time ~ 15.07 sec**
- Open-Loop **GPU solve time ~ 0.5 sec**



WHAT'S NEXT?

- Develop toolbox for training models
  - Automatically convert model parameters
  - Improve warmstart generation



- Expand support for **ML operators**
  - Investigate full space embeddings

ACKNOWLEDGEMENTS

Oscar Dowson
JuMP-dev
Main Developer

Luis Ricardez-Sandoval
UWaterloo
Professor

Manvir Banwait
UWaterloo
MASc Student

UNIVERSITY OF WATERLOO
Department of Chemical Engineering

NSERC CRSNG

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

CHECK IT OUT