

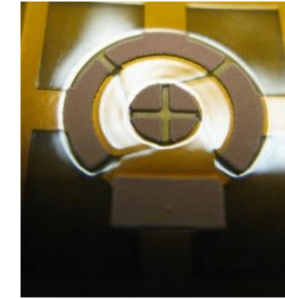
Soft Magnetic Materials and Ordinary Differential Equations: From Linear Circuits to Neural Network Models

Thomas Guillod

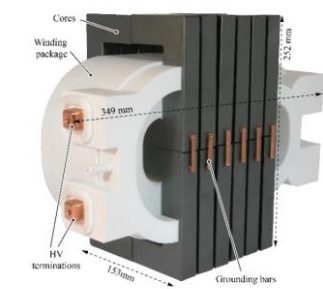
Dartmouth College, USA

IEEE MagNet Challenge Webinar
May 23, 2025

- **Magnetics** are a **bottleneck**
 - Bulky, expensive, lossy
 - Challenging design process
- **Soft magnetic core material**
 - Inductors, transformers, sensors, etc.
 - Datasheet: only sinusoidal and incomplete
 - Models: inaccurate (up to 100% deviation)
 - No accurate first principles model
- **Better models are required**



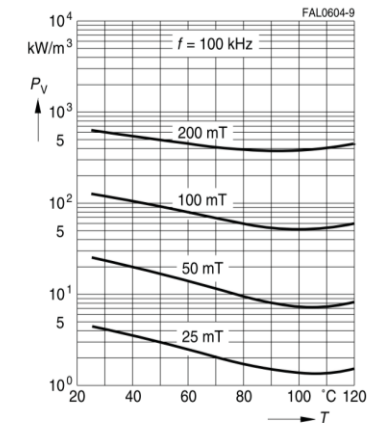
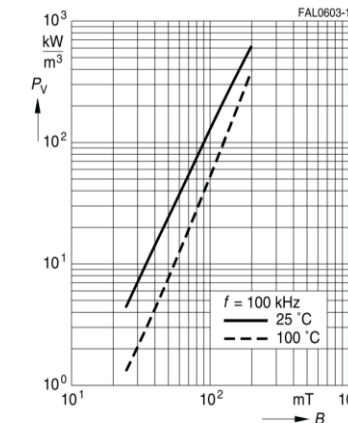
[Dartmouth]



[ETHZ]

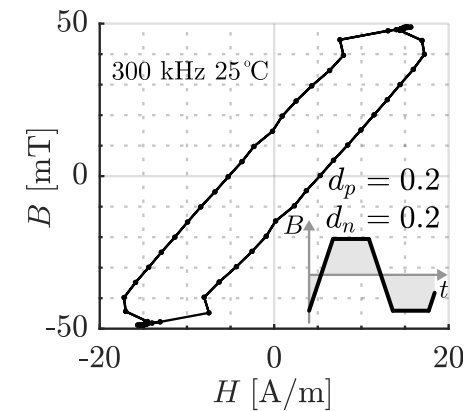
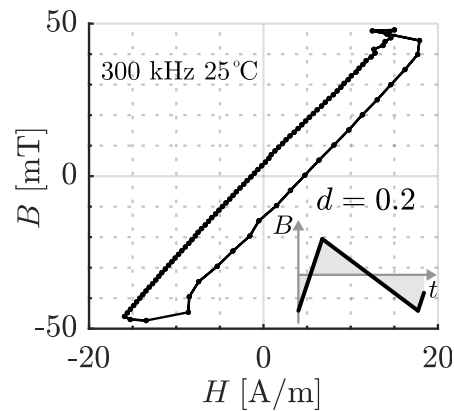
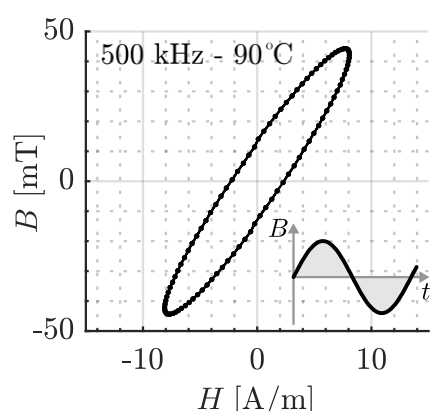
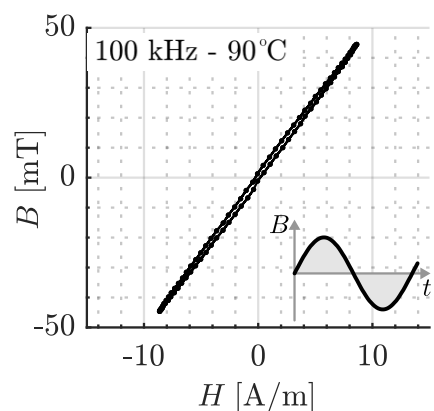
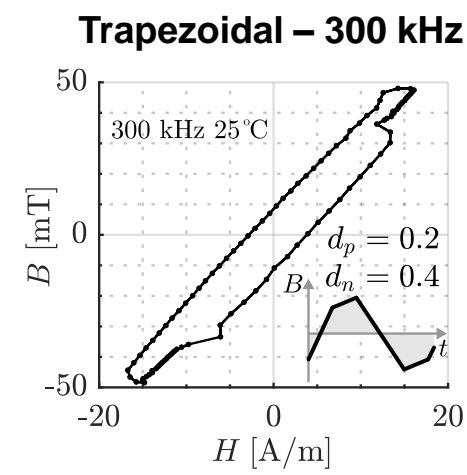
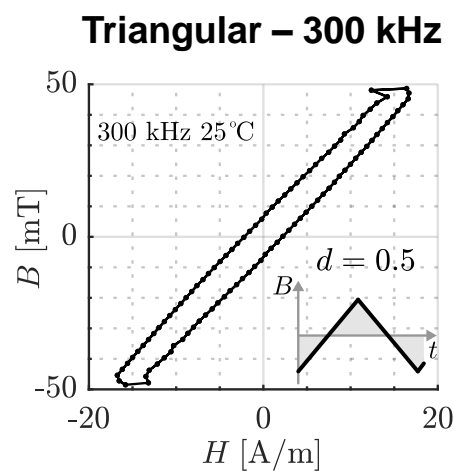
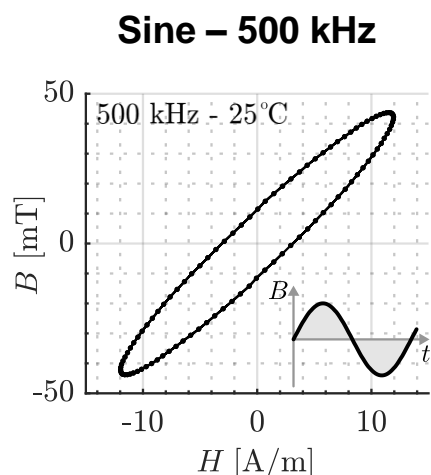
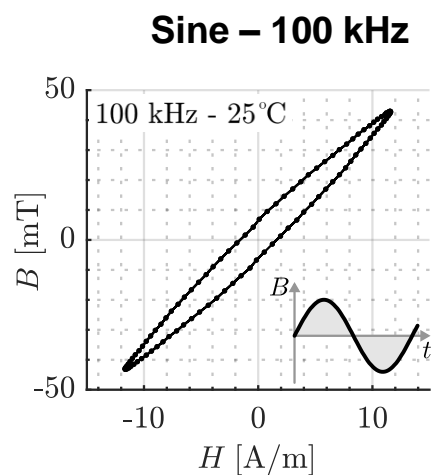
SIFERRIT materials

N87

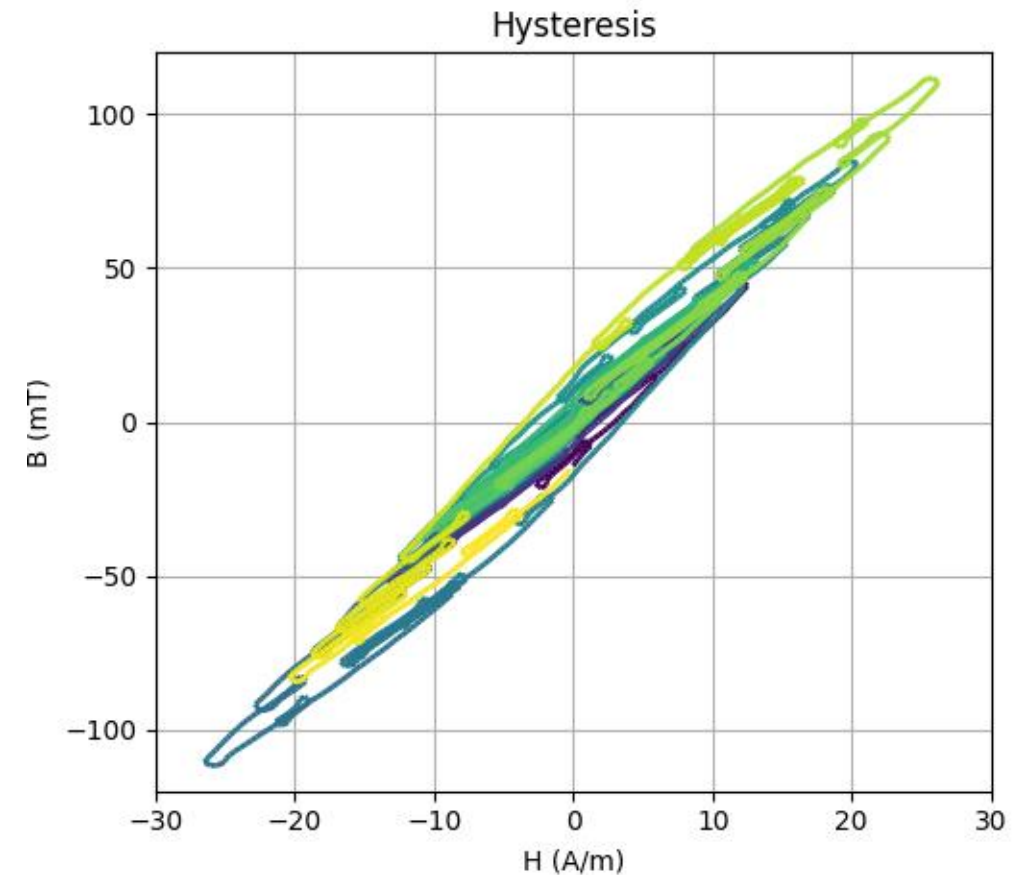
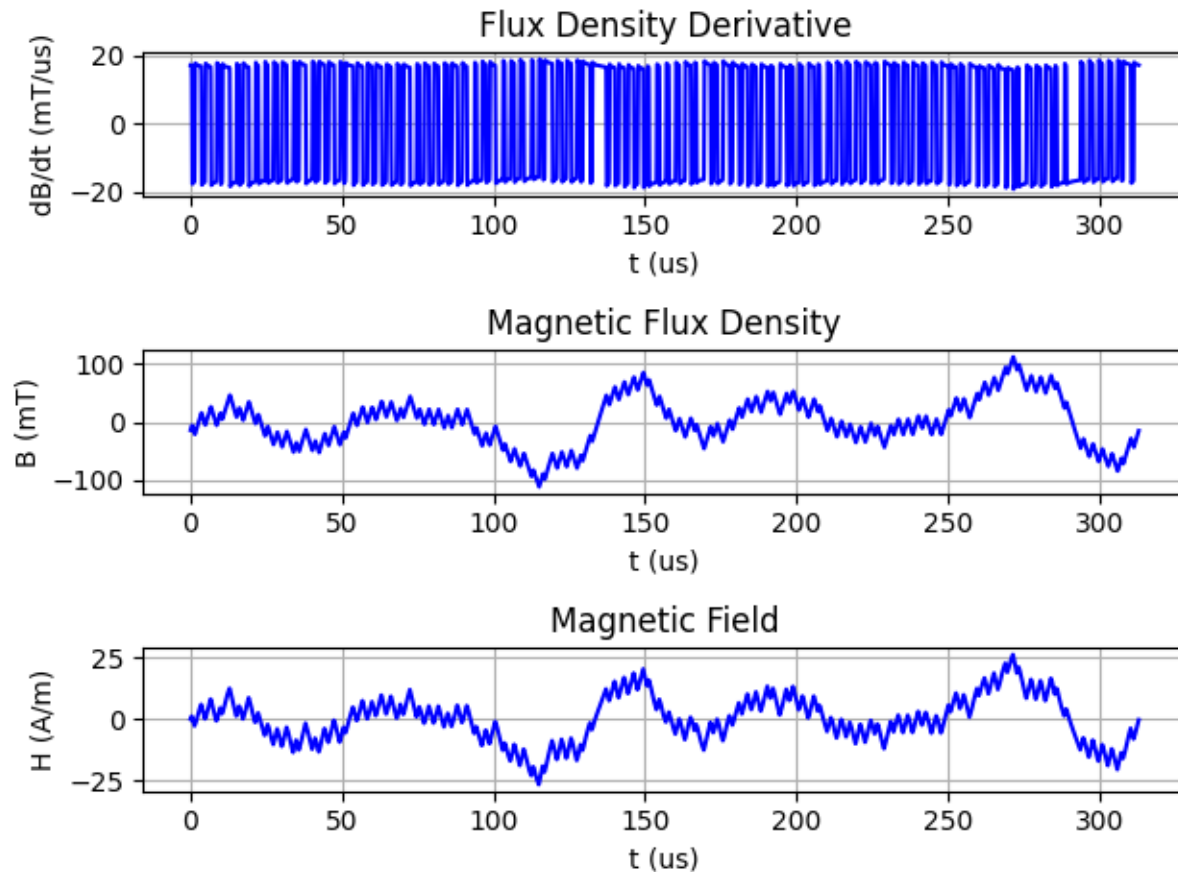


[TDK-EPCOS]

- **Nonlinear** → Amplitude, waveshape, frequency, temperature

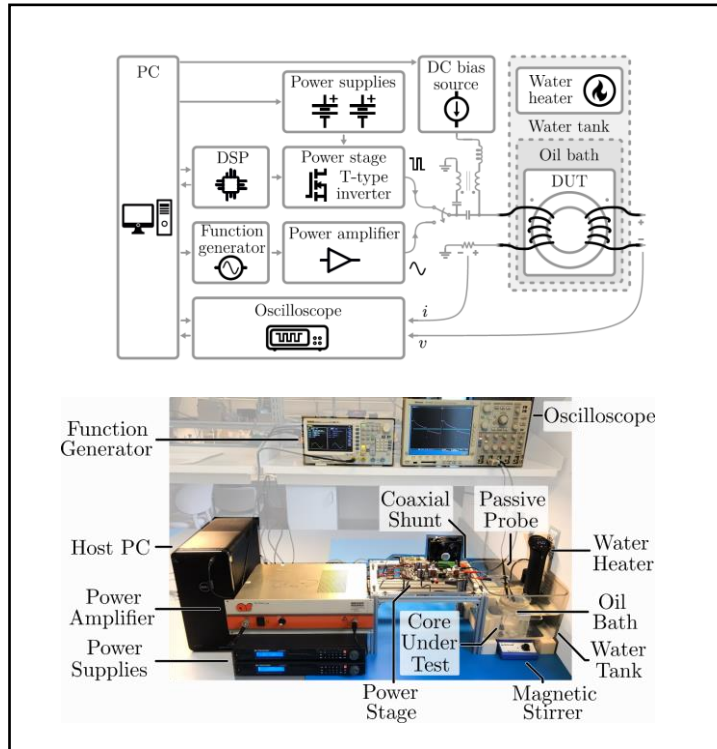


- **Arbitrary PWM** → longer waveforms with minor loops

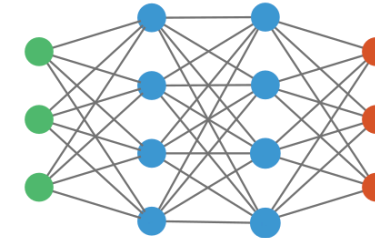


[example EPCOS/TDK N87 at 25C]

- **MagNetX Dataset**
 - 10 different materials
 - Over 10GB of measurements



- **MagNet Challenge 2**
 - Innovative models
 - Accurate & versatile
 - Usable for PE engineers



Machine Learning

$$P = \frac{1}{T} \int_0^T k \left| \frac{dB}{dt} \right|^\alpha (B_{pkpk})^{\beta-\alpha} dt$$

$$W_{hyst} = a_1 B_{pkpk} + a_2 B_{pkpk}^2 + a_3 B_{pkpk}^3$$

$$f_{eff} = f \left(1 + c \left(\frac{1}{B_{pkpk}} \int_0^T \left| \frac{d^2 B}{dt^2} \right| dt \right)^\gamma \right)$$

Analytical Model

- Part I: Modeling Workflow**
- Part II: Circuit-Based Differential Equation Models**
- Part III: Neural Differential Equation Models**
- Part IV: Conclusion and Outlooks**

Part I: Modeling Workflow

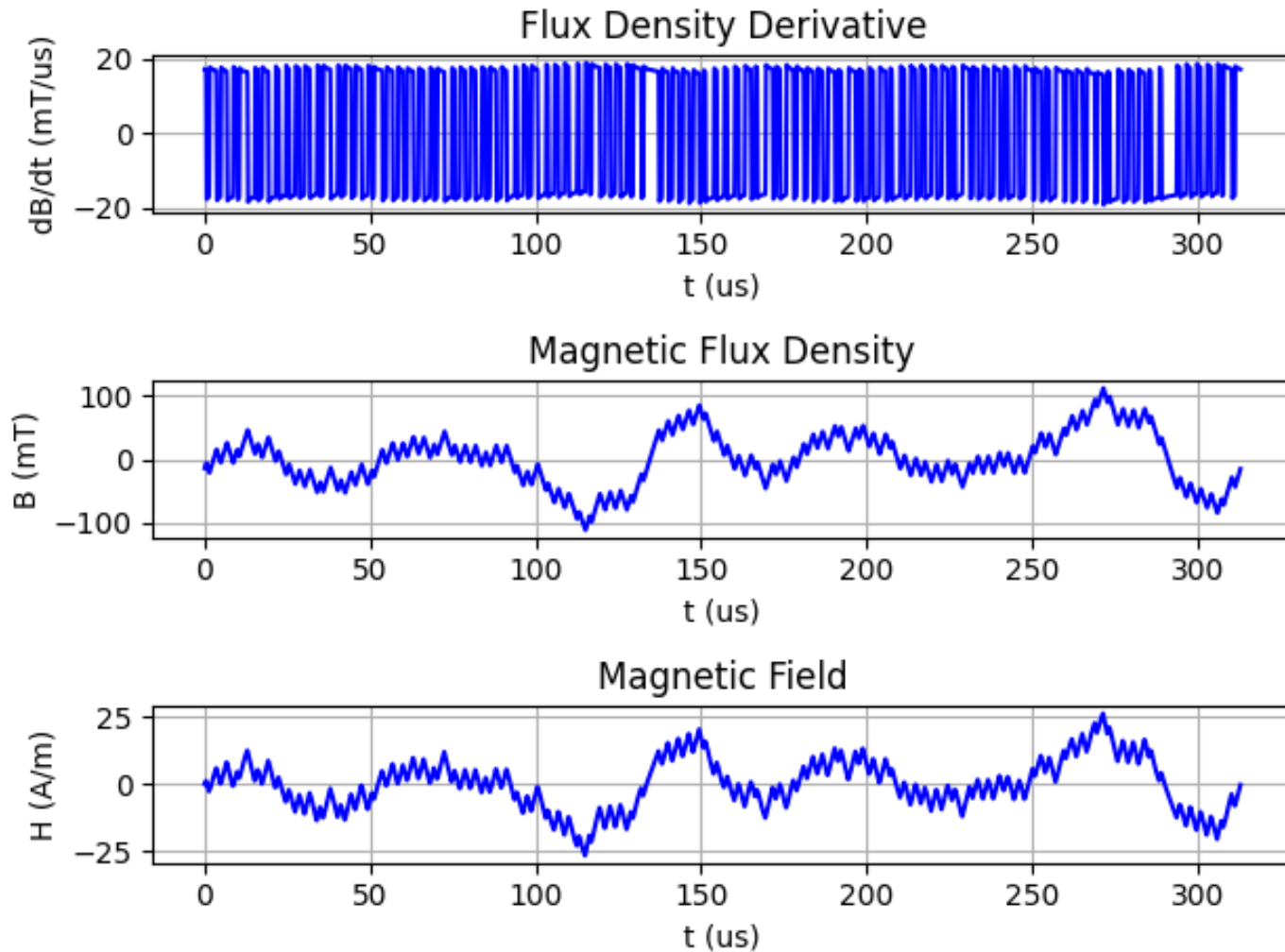
- **Load** the measurements
- **Select** the samples:
 - Select the training samples
 - Select the test samples
- **Prepare** the samples:
 - Compute the gradient of the flux density
 - Remove any offset and align the phase
 - Filter noise if required
- **Save** the resulting dataset

Data used in this
presentation

MagNetX: “N_87_5”

- EPCOS/TDK N87
- 25°C operating temp.
- 3.2 kHz rep. frequency
- 48 training samples
- 24 test samples

Workflow Step 2: Training

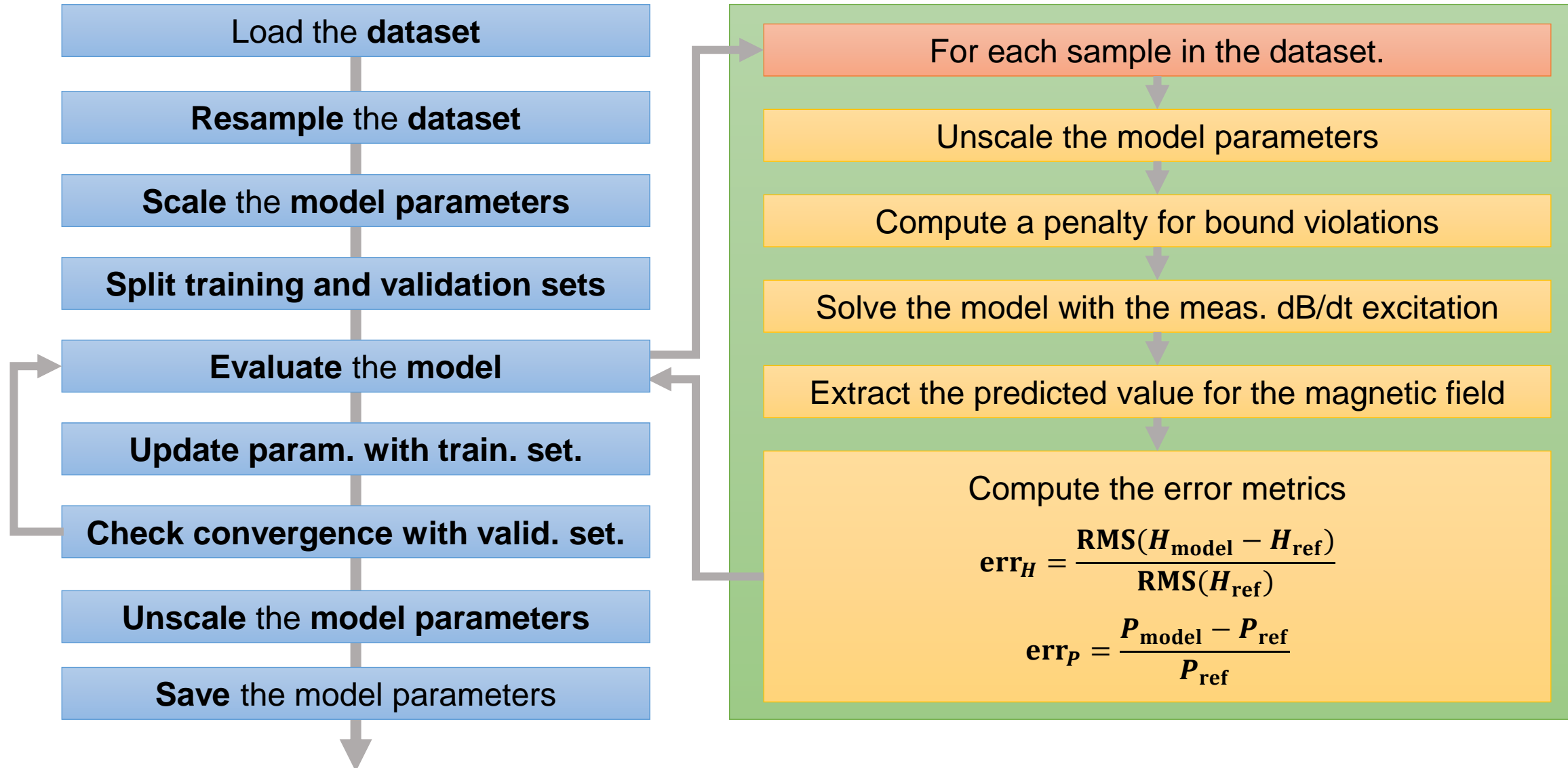


← Applied excitation

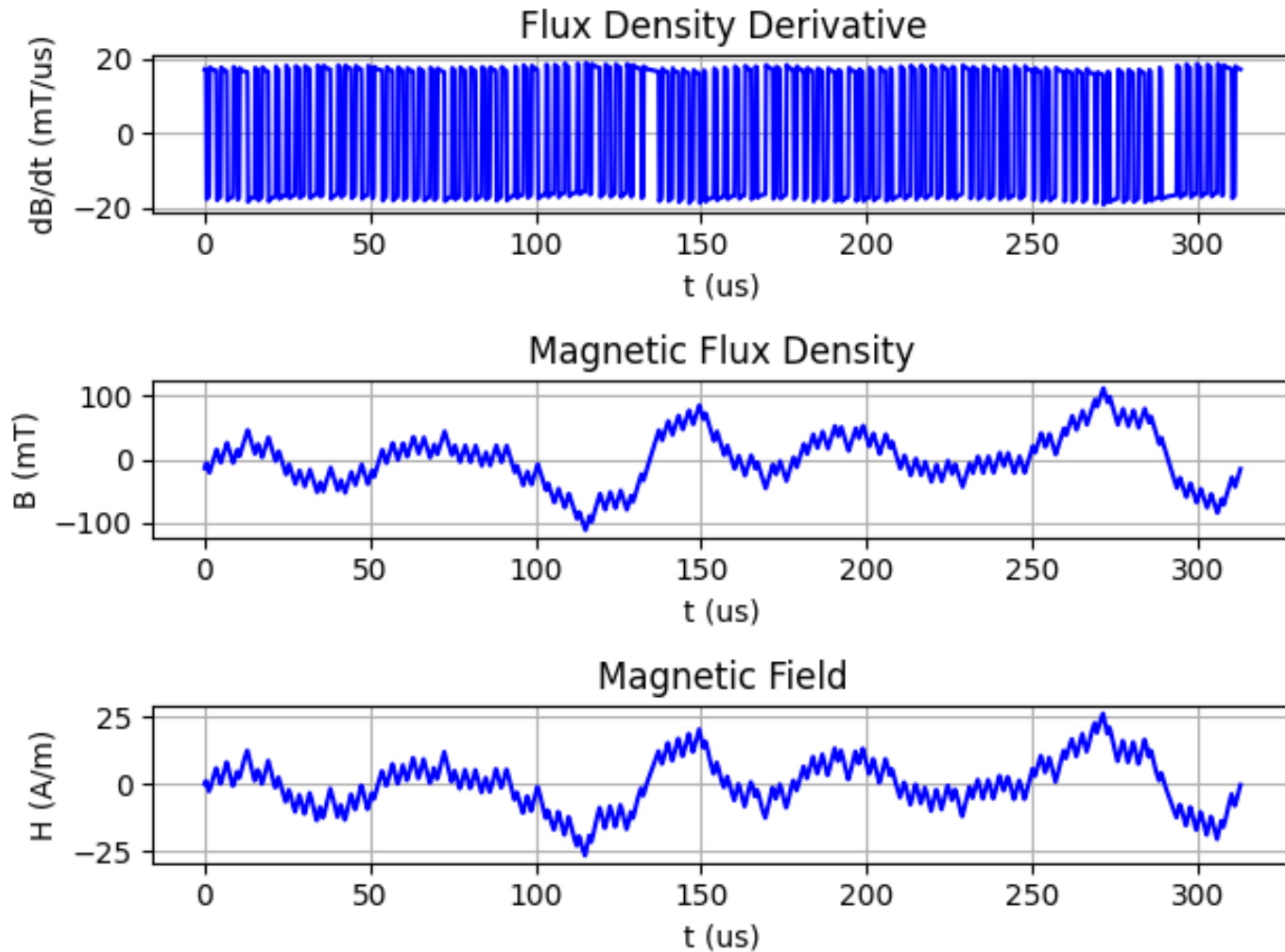
← Applied excitation

↑ differentiate

← Used to fit the model



Workflow Step 3: Inference



← Applied excitation

← Applied excitation

↑ differentiate

→ Predicted by the model



Part II:

Circuit-Based Differential Equation Models

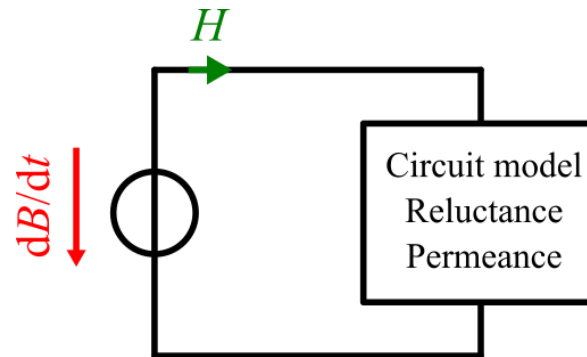
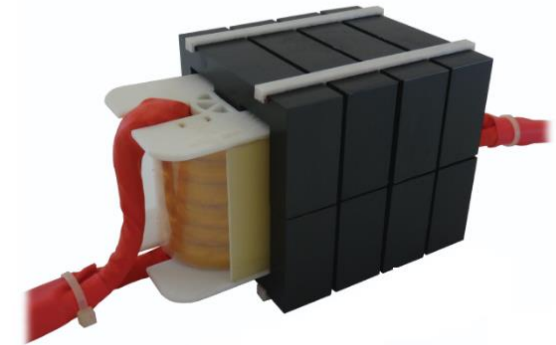
- **Basic equations**

- $V = N \cdot \int \frac{dB}{dt} dA \Rightarrow V \propto \frac{dB}{dt}$

- $I = \frac{1}{N} \cdot \int H dl \Rightarrow I \propto H$

- **Circuit model with ODEs**

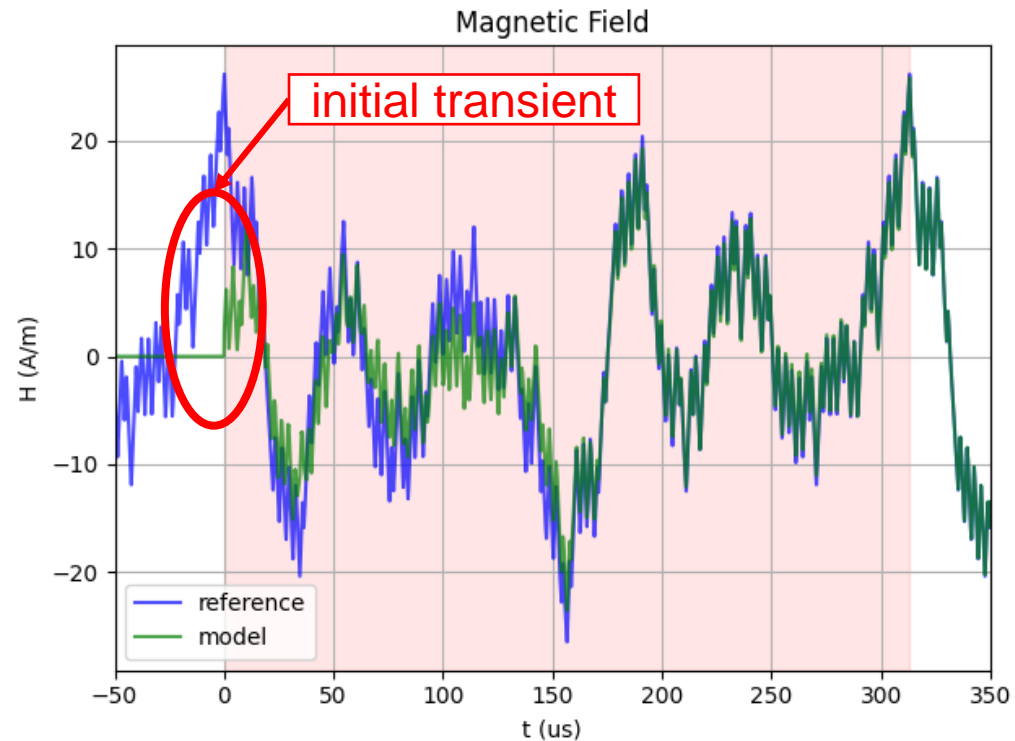
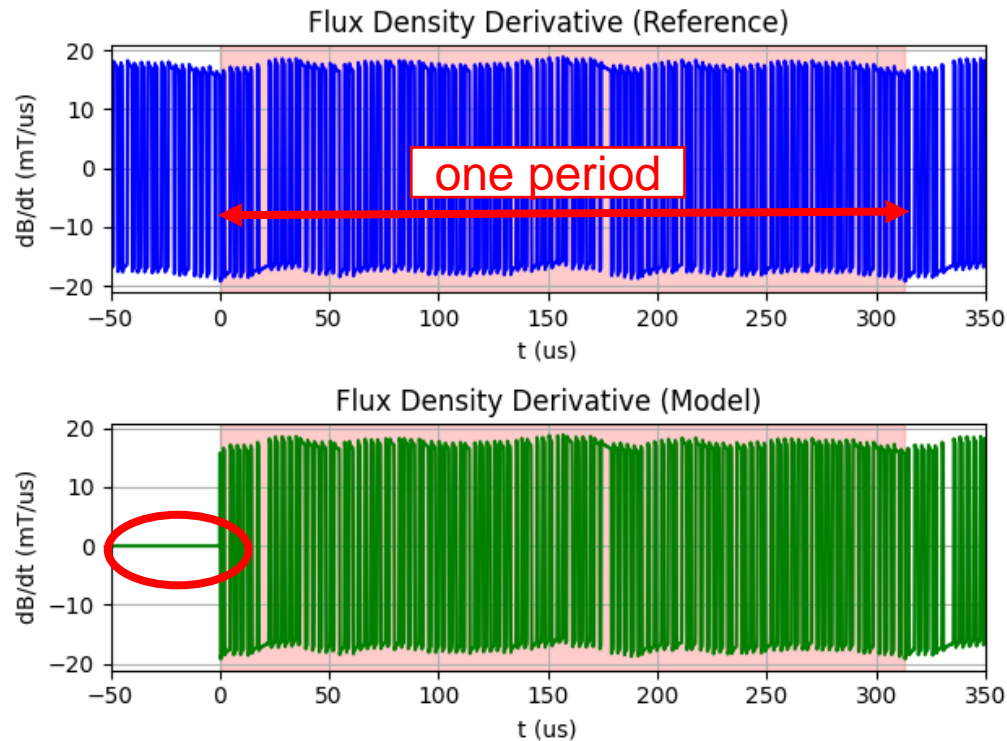
- Common for dynamic systems
 - Causality is enforced
 - Non-periodic waveforms
 - Compatible with SPICE
 - Compatible with FEM



$$\frac{d\vec{y}(t)}{dt} = f\left(\vec{y}(t), \frac{dB(t)}{dt}\right)$$

$$H(t) = g\left(\vec{y}(t), \frac{dB(t)}{dt}\right)$$

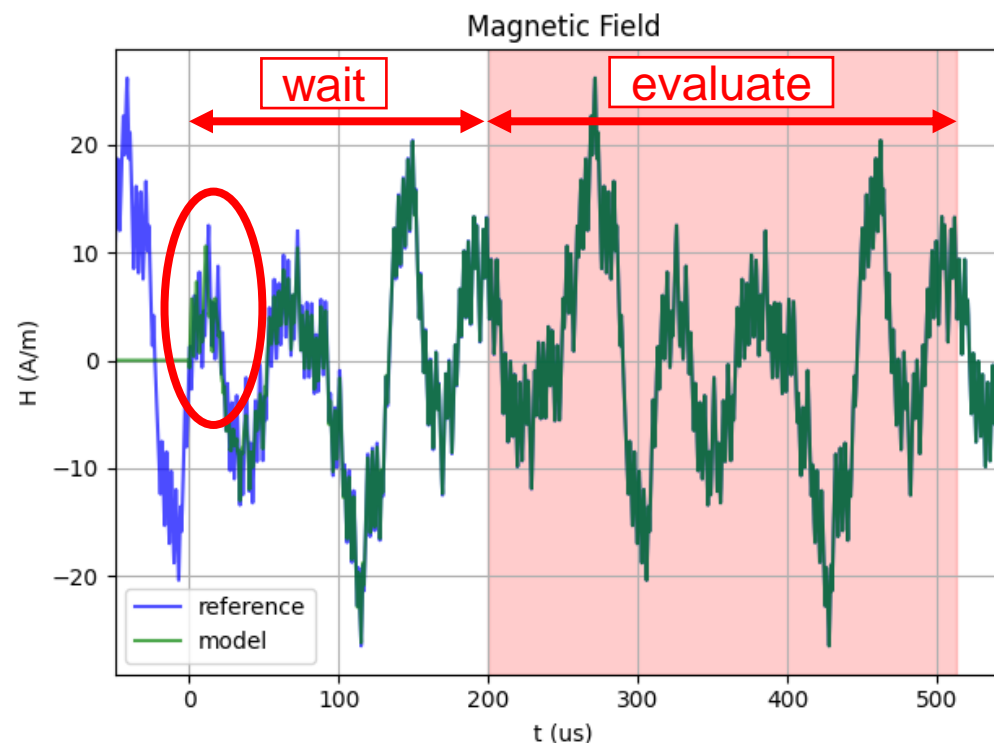
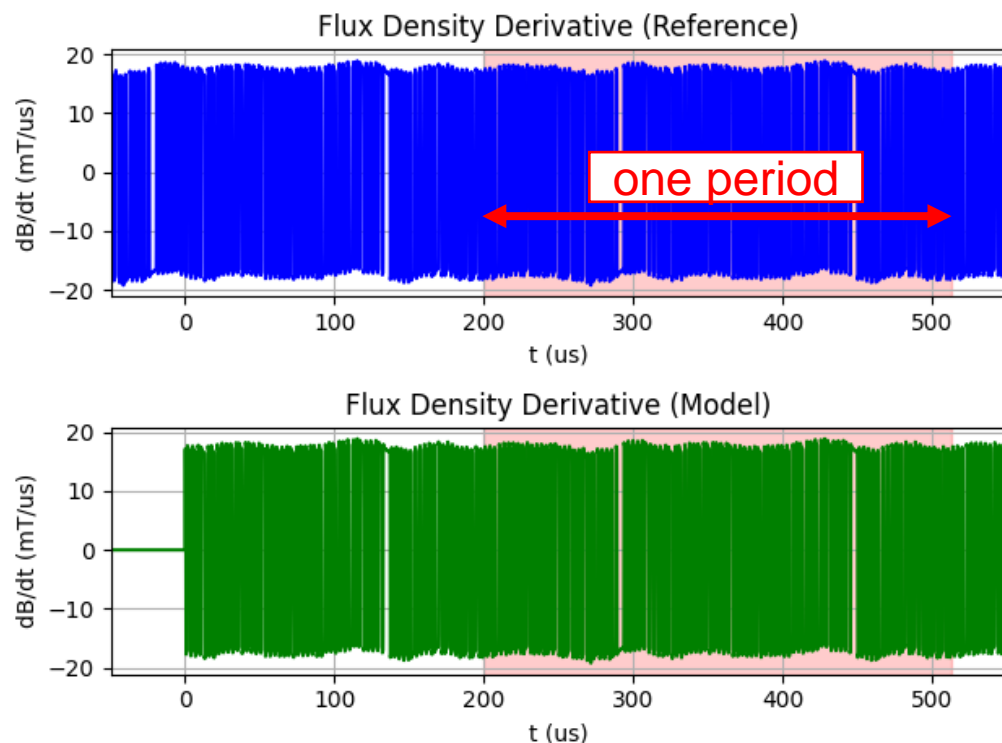
- The MagNetX dataset contains **periodic steady-state data**



- The **initial state** of the material is **not known!**

Problem: Steady-State

- The MagNetX dataset contains **periodic steady-state data**



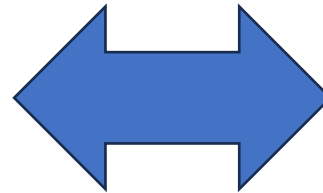
- Align the phase** with zero crossings and **wait** for steady-state!

Problem: Computational Cost

- Integrate **ODEs** for **thousand of time steps** with a measured excitation
- Repeat for the **hundreds of signals** composing the dataset
- Repeat **hundred of times** until the **optimal parameters** are found



[Oak Ridge, CC BY]



[Personal Laptop, CC BY]



=

Python matrix / vector / array computation library
Similar to NumPy/SciPy or MATLAB

+

Just-In-Time **compiler, auto-vectorization, and parallelism**

+

Transparent usage with **CPUs, GPUs, and TPUs**

+

Extremely **powerful automatic differentiation** engine

■

“The Sharp Bits” - pure functions, immutable arrays, etc.

```
import jax
import jax.numpy as jnp

@jax.value_and_grad
def my_fun(val):
    return 1 / jnp.sqrt(val + 1.0)
```

```
# evaluate the function with a scalar
x = jnp.array(1.0)
(y, dydx) = my_fun(x)
```

```
# evaluate the function with vectorization
x = jnp.array([1.0, 2.0])
(y, dydx) = jax.vmap(my_fun)(x)
```

```
y = 0.7071067690849304
dydx = -0.1767767071723938
```

```
y = [0.70710677 0.57735026]
dydx = [-0.1767767 -0.09622505]
```

- Define a simple **mathematical function**.
- Compute the **value and gradient**.
- **Vectorize the function call** for arrays.
- **Possible with very complex operations.**

- **Neural networks:**

- **Flax:** flexible neural networks
- **Equinox:** lightweight neural networks
- **Optax:** first-order gradient optimizers
- **Leventer:** scalable foundation models

- **Other tools:**

- **Optimistix:** root finding, minimization, etc.
- **DiffraX:** differential equation solvers
- **Lineax:** linear equation solvers
- **BlackJAX:** probabilistic sampling

- In this work: JAX, DiffraX, Equinox, Optax, and Optimistix

```
import jax
import jax.numpy as jnp
import diffrax as dfx

@jax.jit
def eval_ode(R, L):
    # definition of the ODE: voltage step applied to a RL network
    term = dfx.ODETerm(lambda t, i, args: (1.0 - R * i) / L)

    # solve the ODE with the specified initial value
    sol = dfx.diffeqsolve(term, dfx.Dopri5(), t0=0e-6, t1=5e-6, dt0=50e-9, y0=0.0)

    # return the final value of the current
    return sol.ys[0]

# compute the ODE solution
R = jnp.array(5.0)
L = jnp.array(20e-6)
i = eval_ode(R, L)

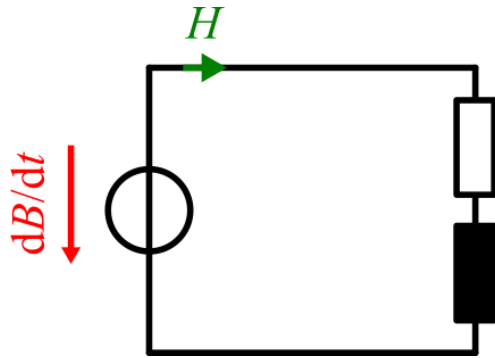
# compute the derivative of the ODE solution
didR_auto = jax.grad(eval_ode, argnums=0)(R, L)

# compute the derivation with an approximation (slower, less accurate)
didR_diff = (eval_ode(R + 0.01, L) - eval_ode(R - 0.01, L)) / 0.02
```

```
i = 0.14269903302192688
didR_auto = -0.014214569702744484
didR_diff = -0.014217197895050049
```

- Define a **simple ODE**:
 - RL step response with 1V.
 - The ODE state is the current.
 - Return the current (last step).
- Evaluate the **solution of the ODE**.
- Evaluate the **derivative of the solution**.

- Simplest possible circuit



$$V(t) = R \cdot i(t) + L \frac{di(t)}{dt}$$

$$\frac{dB(t)}{dt} = r \cdot H(t) + \mu_0 \mu_r \frac{dH(t)}{dt}$$

- Single (linear) ordinary differential equation
 - Two free parameters (resistance and permeability)
 - Cannot model saturation behavior
 - Cannot model nonlinear losses

```
@eqx.filter_jit
def get_ode(t, H, param, interp):
    # extract the variables
    r_lin = param["r_lin"]
    mu_lin = param["mu_lin"]

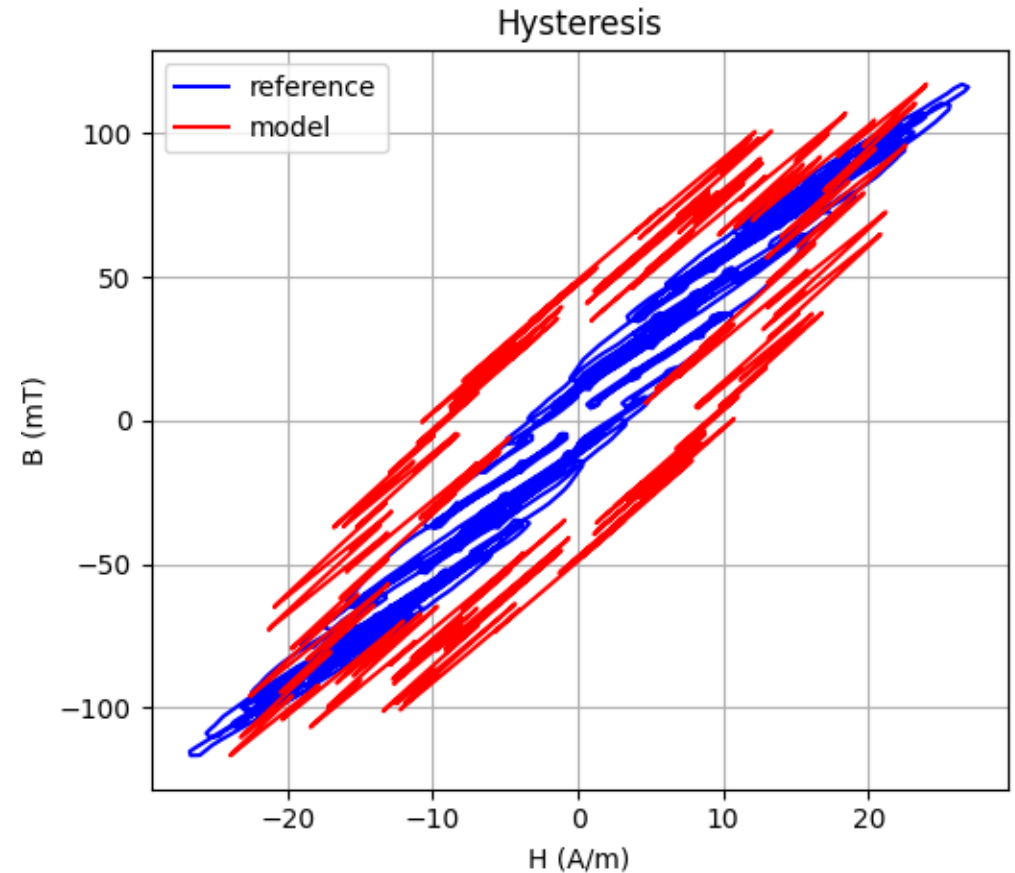
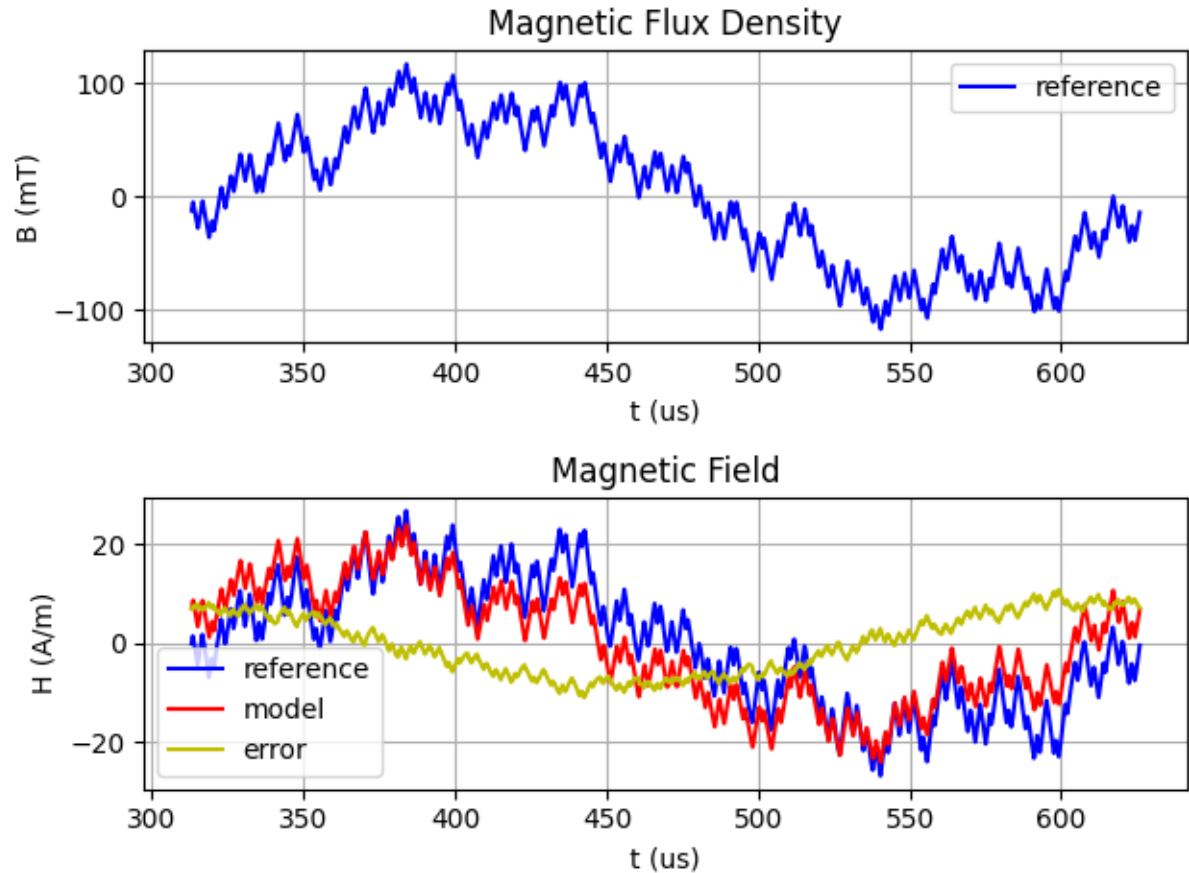
    # obtain the applied excitation
    dBdt = interp(t)

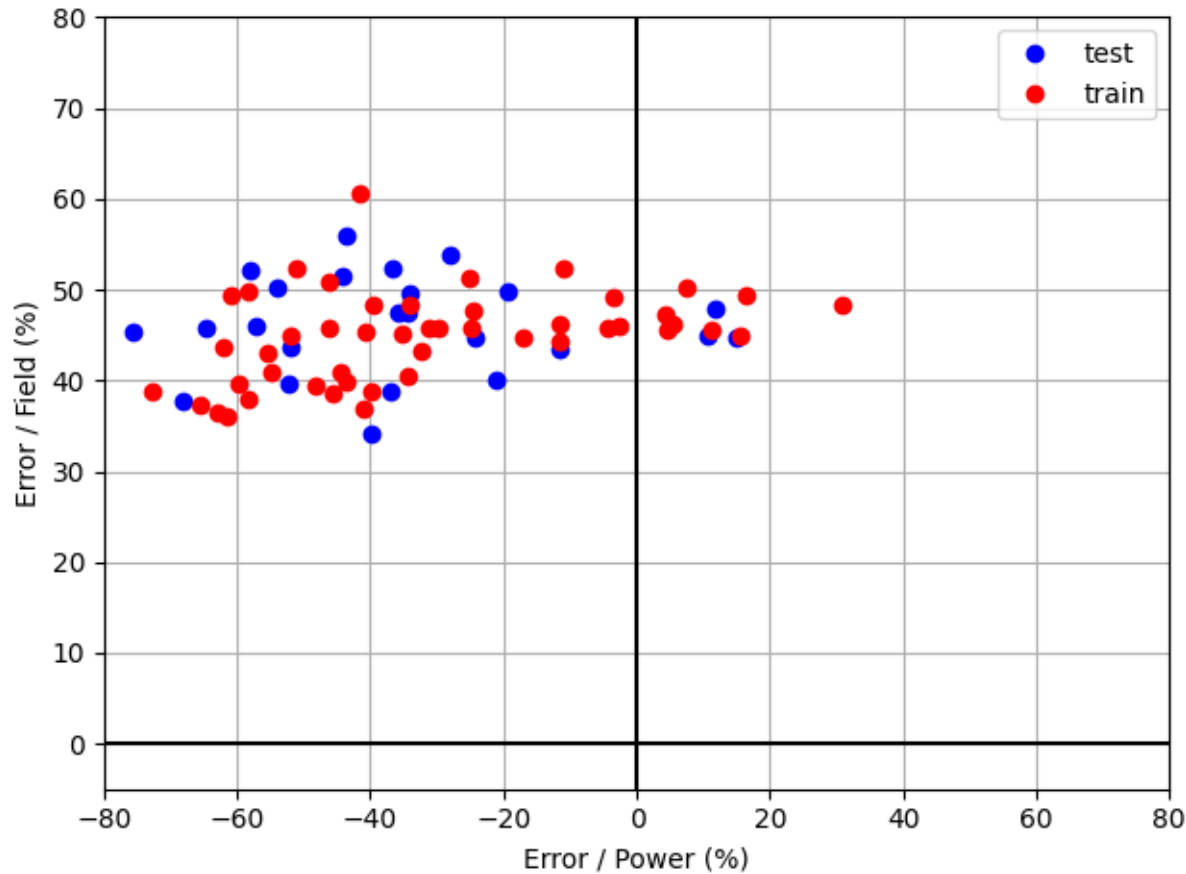
    # define the permeability
    mu0 = 4 * jnp.pi * 1e-7

    # compute the dynamic term
    dHdt = (dBdt - r_lin * H) / (mu0 * mu_lin)

    return dHdt
```

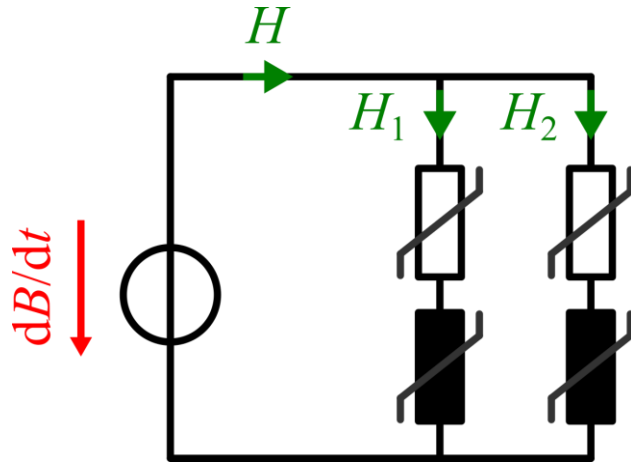
- $\frac{dB(t)}{dt} = r \cdot H(t) + \mu_0 \mu_r \frac{dH(t)}{dt}$
- Define a **circuit ODE**.
- **RL series** connection.
- The ODE **state** is the **mag. field**.
- The **excitation** is the **flux derivative**.
- The excitation is interpolated.





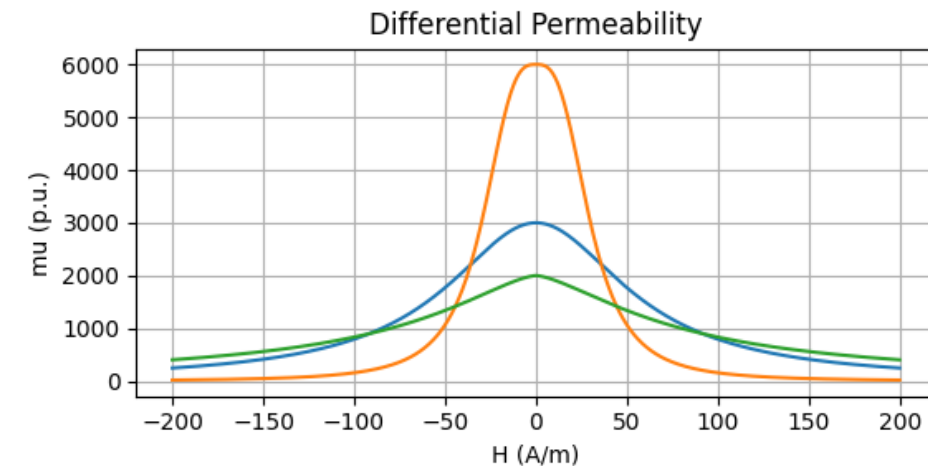
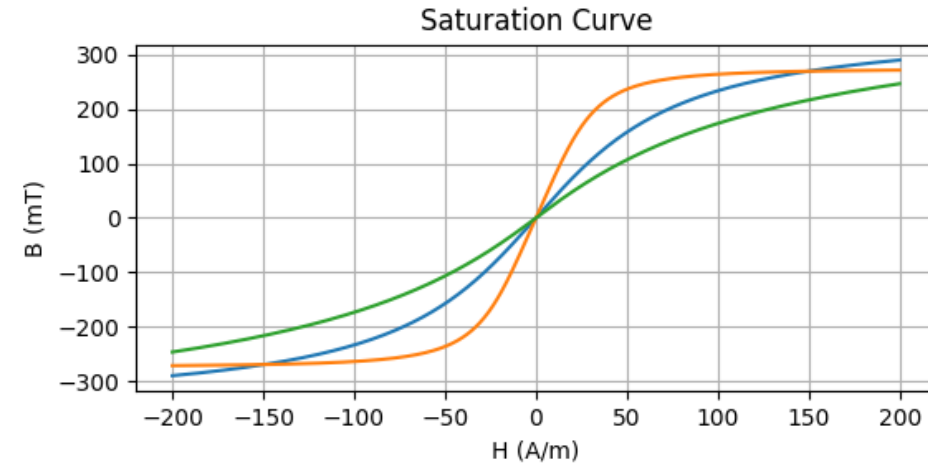
- RMS error on the mag. field: 46%
- RMS error on the losses: 41%
- Only two parameters...
- Magnetics are non-linear...

- Transition to a nonlinear circuit

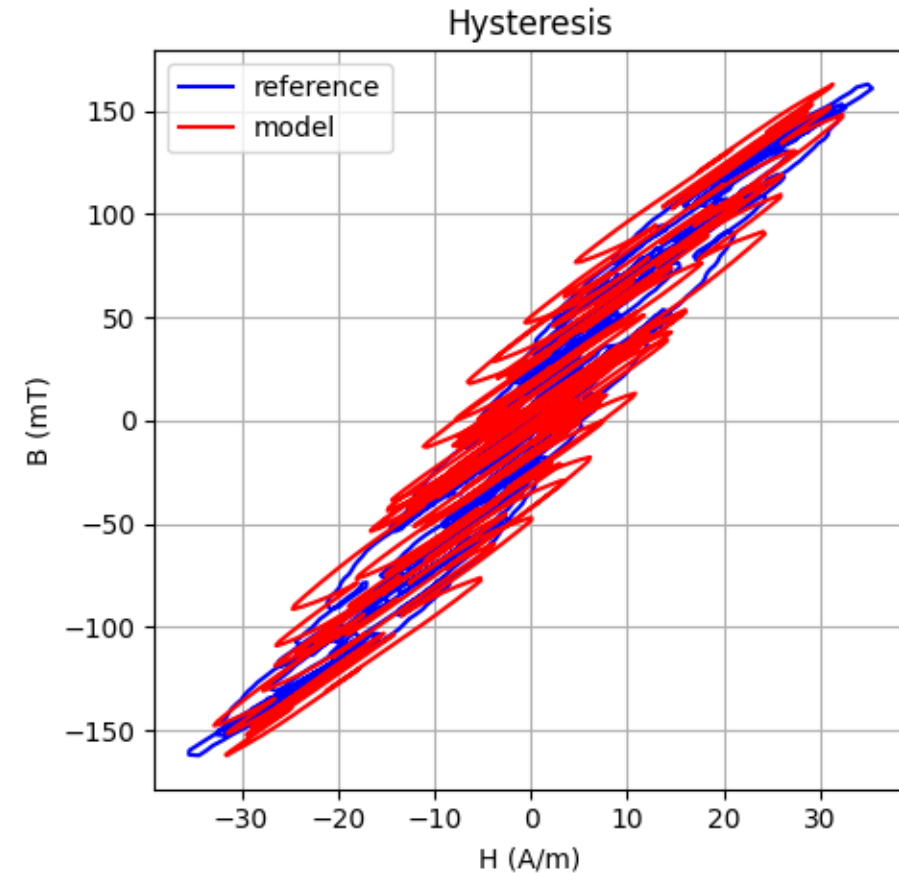
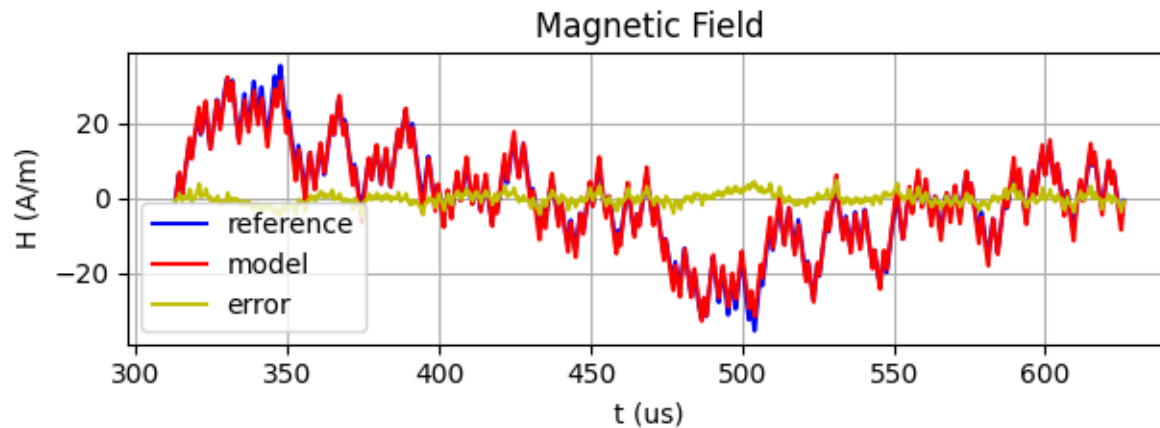
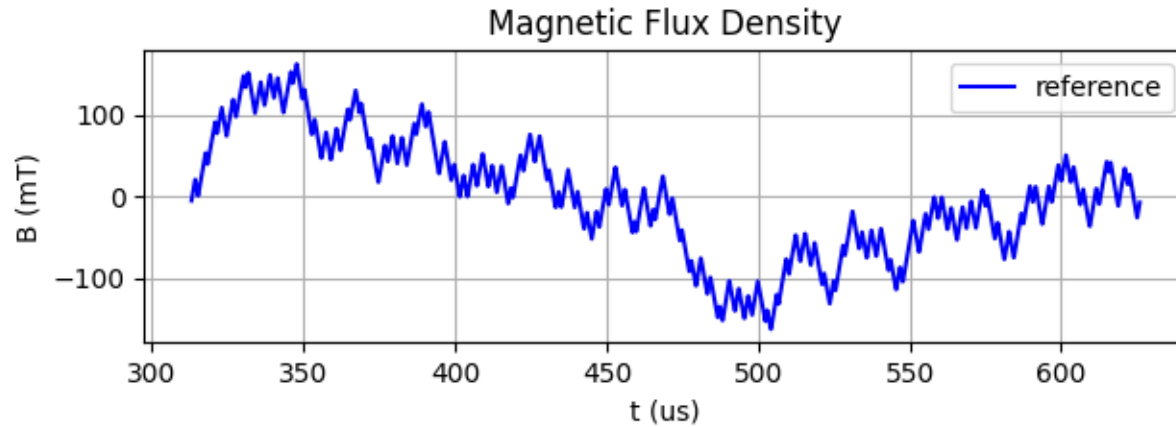


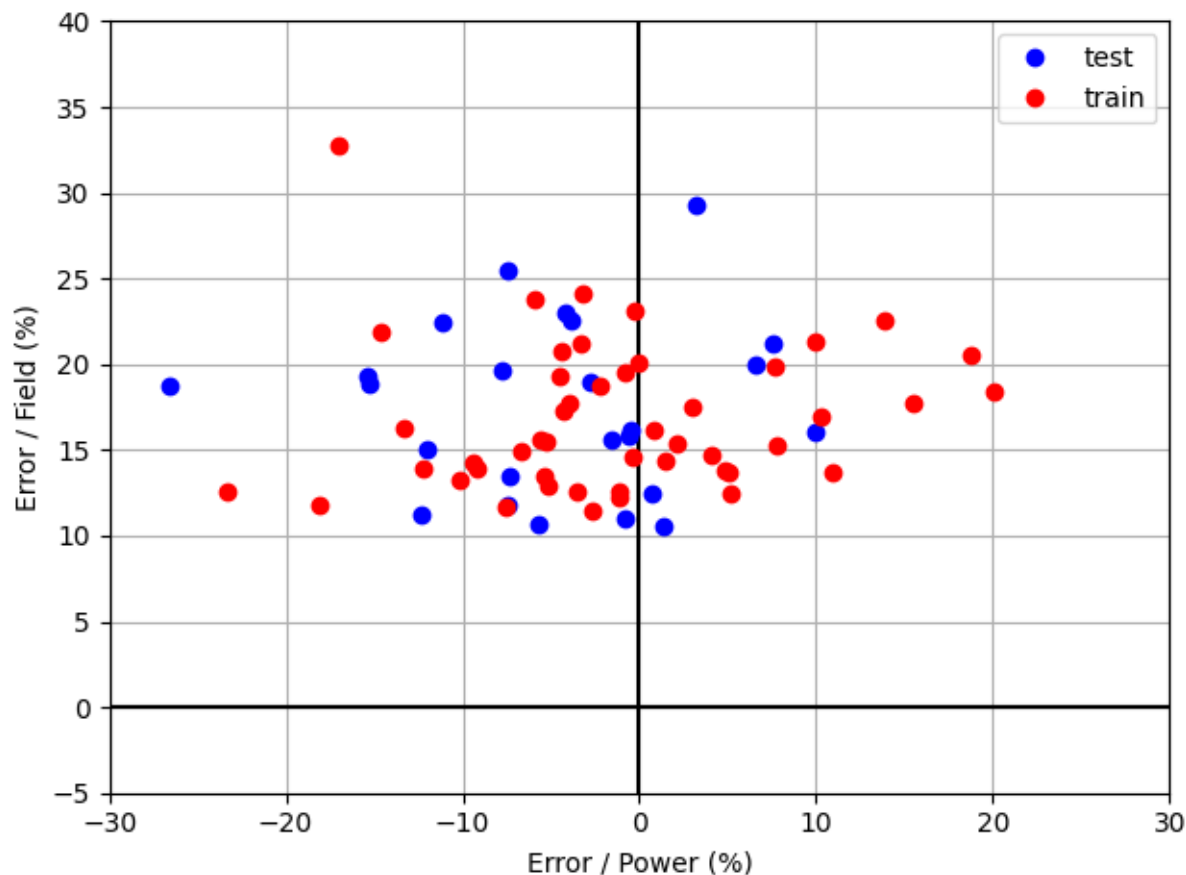
- Two decoupled ODEs

- Two resistances ($R_i = R_i(H_i)$)
- Two inductances ($L_i = L_i(H_i)$)
- Non-linear equations
- 10 free parameters



differentiate

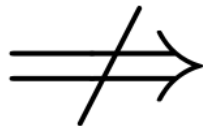




- RMS error on the mag. field: 18%
- RMS error on the losses: 9%
- No overfitting (test set)
- Number of parameters: 2 x 5
- Parameters bounds are enforced
- Initial values: latin hypercube
- Optimizer: AdaBelief grad. descent

- **Equation-based analytical models have potential**
- **Physics-based (or inspired) models**
 - Steinmetz-based model (e.g. iGSE)
 - Landau–Lifshitz–Gilbert equation
 - Preisach Hysteresis model
 - Jiles–Atherton model
- **A last warning about such models**

**Physics-based
material model**



**Fitted parameters have
a physical meaning**

- **Loss Models**

- K. Venkatachalam et. al., "Accurate Prediction of Ferrite Core Loss with Nonsinusoidal Waveforms using only Steinmetz Parameters", 2002, <https://doi.org/10.1109/CIPE.2002.1196712>
- J. Mühlethaler et al., "Improved Core-Loss Calculation for Magnetic Components Employed in Power Electronic Systems", 2012, <https://doi.org/10.1109/TPEL.2011.2162252>
- E. Stenglein et al., "Core Loss Model for Arbitrary Excitations With DC Bias Covering a Wide Frequency Range", 2021, <https://doi.org/10.1109/TMAG.2021.3068188>
- T. Dimier et al, "Non-Linear Material Model of Ferrite to Calculate Core Losses With Full Frequency and Excitation Scaling," 2023, <https://doi.org/10.1109/TMAG.2023.3277492>

- **Hysteresis models**

- D. C. Jiles et al., "Theory of Ferromagnetic Hysteresis". 1984, [https://doi.org/10.1016/0304-8853\(86\)90066-1](https://doi.org/10.1016/0304-8853(86)90066-1)
- H., K. Tanaka, Nakamura et al. "Calculation of Iron Loss in Soft Ferromagnetic Materials using Magnetic Circuit Model Taking Magnetic Hysteresis into Consideration", 2015, <https://doi.org/10.3379/msjmag.1501R001>
- M. Luo et al., "Modeling Frequency Independent Hysteresis Effects of Ferrite Core Materials Using Permeance–Capacitance Analogy for System-Level Circuit Simulations," 2018, <https://doi.org/10.1109/TPEL.2018.2809704>
- G. Mörée et al., "Review of Play and Preisach Models for Hysteresis in Magnetic Materials", 2023, <https://doi.org/10.3390/ma16062422>

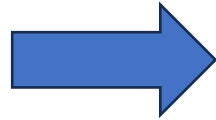
Part III:

Neural Differential Equation Models

- **Neural ODEs**

$$\frac{d\vec{y}(t)}{dt} = f\left(\vec{y}(t), \frac{dB(t)}{dt}\right)$$

$$H(t) = g\left(\vec{y}(t), \frac{dB(t)}{dt}\right)$$



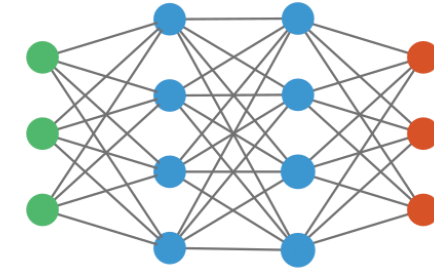
$$\frac{d\vec{y}(t)}{dt} = \text{Neural Network}\left(\vec{y}(t), \frac{dB(t)}{dt}\right)$$

$$H(t) = \text{Neural Network}\left(\vec{y}(t), \frac{dB(t)}{dt}\right)$$

- **Neural networks are defining the ODE functions**

- Coupled system of nonlinear differential equations
- Very similar problem to circuit-based ODEs
- Most of the code / libraries are the same

- Standard **multilayer perceptron** (MLP) network
- Neural networks are **small and shallow**
- **Scaling** of the variables is **required**



State dynamics: single network:

$$\begin{bmatrix} dH_1/dt \\ dH_2/dt \\ \vdots \\ dH_n/dt \end{bmatrix} = \text{MLP} \left(\begin{bmatrix} H_1(t) \\ H_2(t) \\ \vdots \\ H_n(t) \\ dB(t)/dt \end{bmatrix} \right)$$

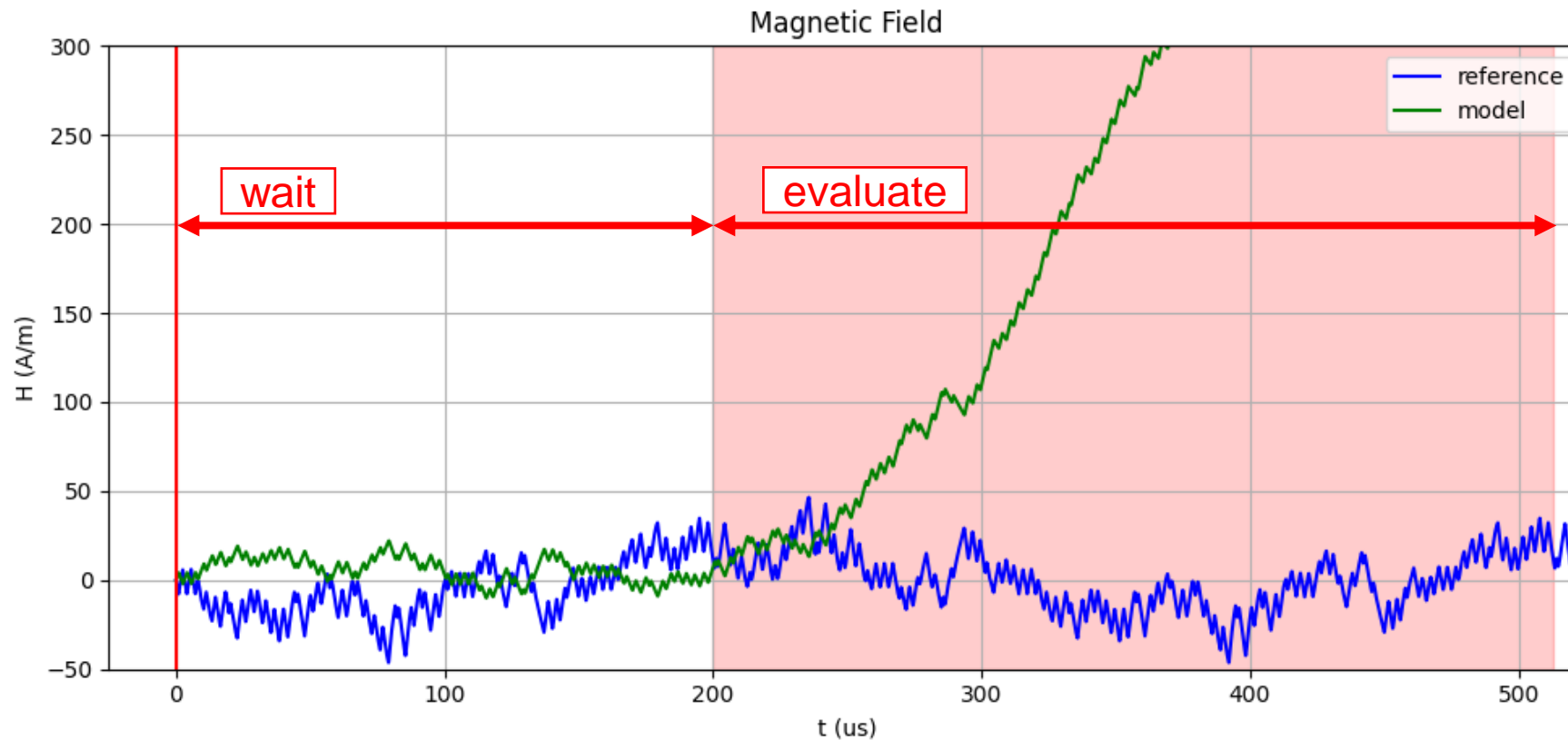
State dynamics: two networks:

$$\begin{bmatrix} dH_1/dt \\ dH_2/dt \\ \vdots \\ dH_n/dt \end{bmatrix} = \text{MLP}_1 \left(\begin{bmatrix} |H_1(t)| \\ |H_2(t)| \\ \vdots \\ |H_n(t)| \end{bmatrix} \right) \begin{bmatrix} H_1(t) \\ H_2(t) \\ \vdots \\ H_n(t) \end{bmatrix} + \text{MLP}_2 \left(\begin{bmatrix} |H_1(t)| \\ |H_2(t)| \\ \vdots \\ |H_n(t)| \end{bmatrix} \right) dB(t)/dt$$

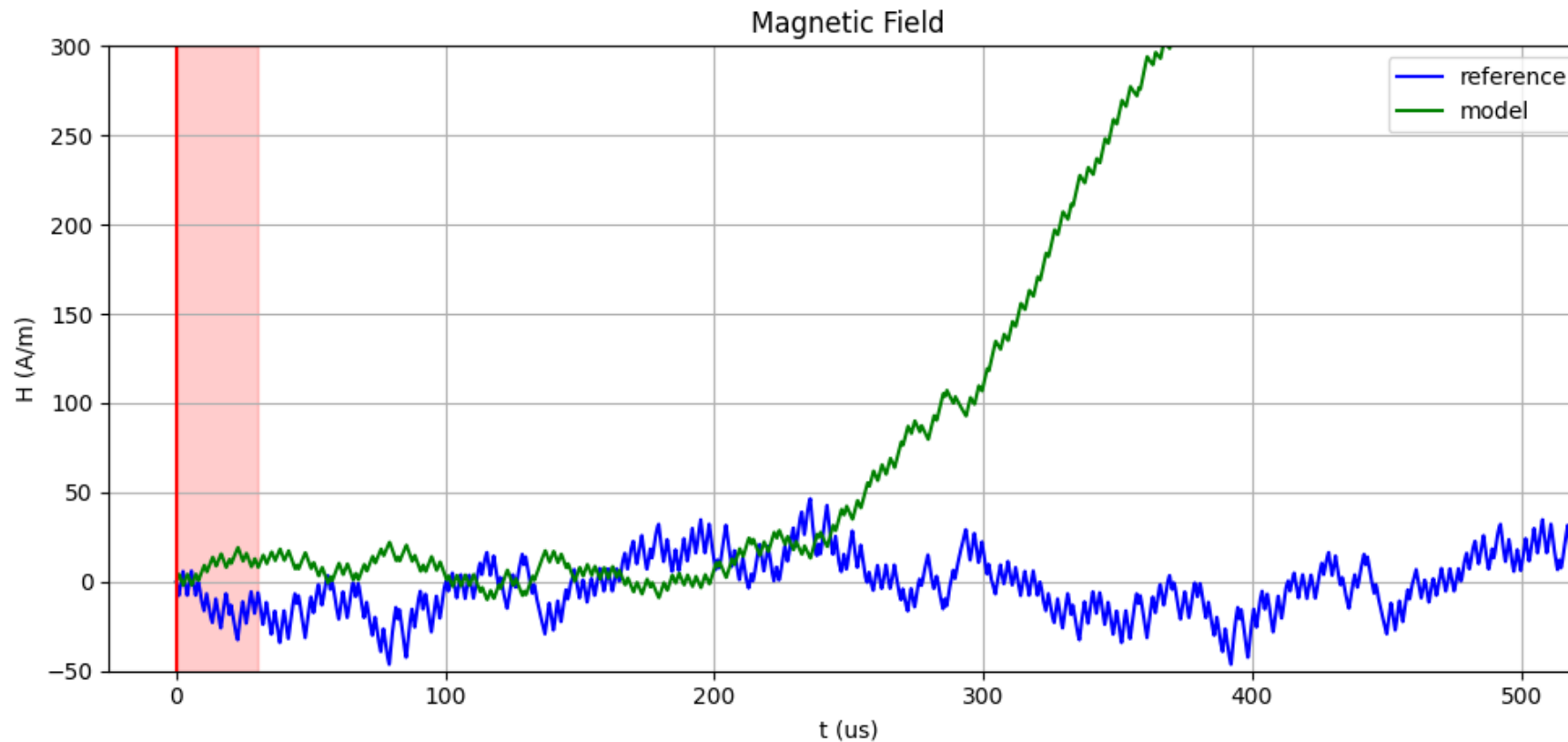
Output functions: parallel connections:

$$H(t) = \sum_i H_i$$

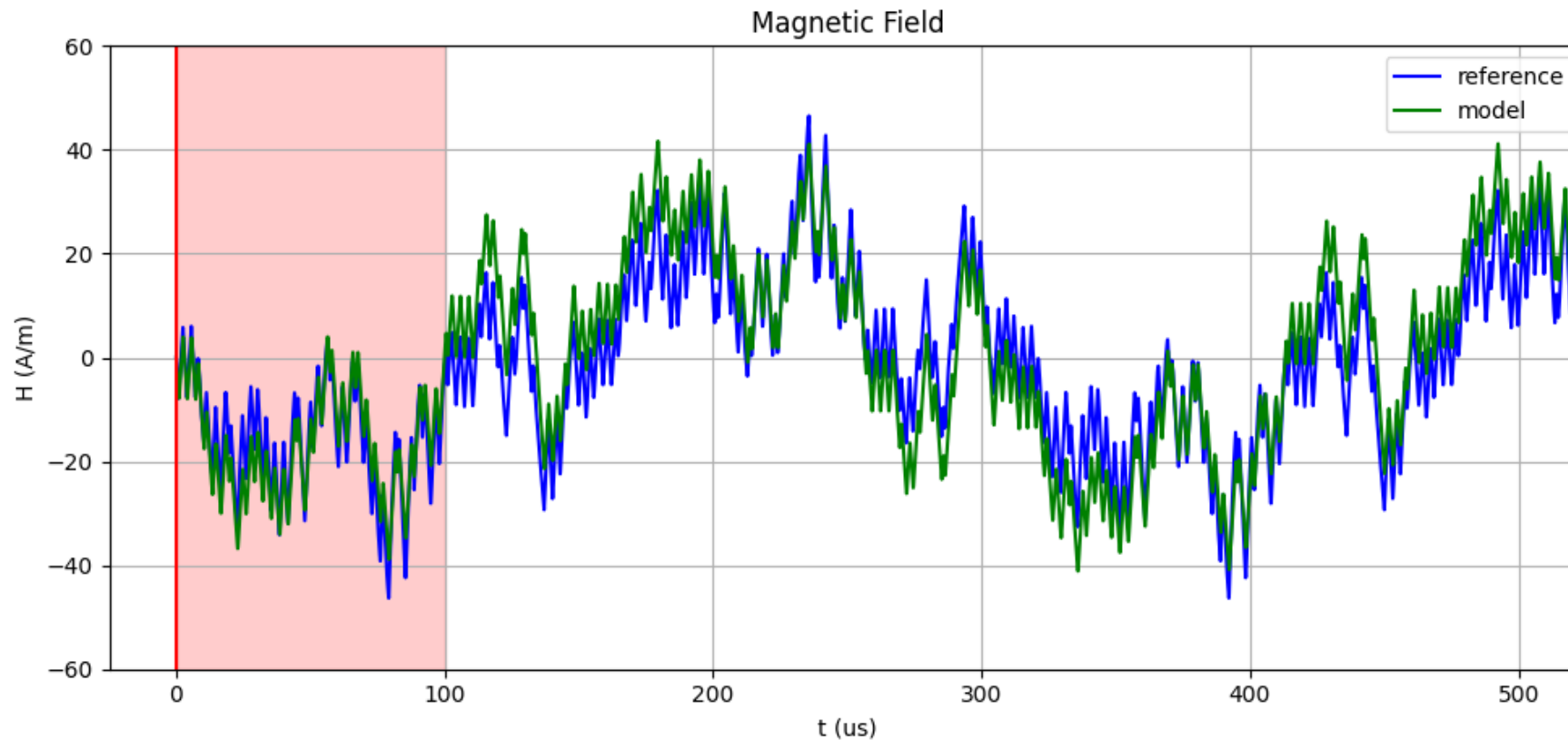
- The **initial** random networks **parameters** are a **bad model**
- This would imply a **very slow** and/or **unstable** training



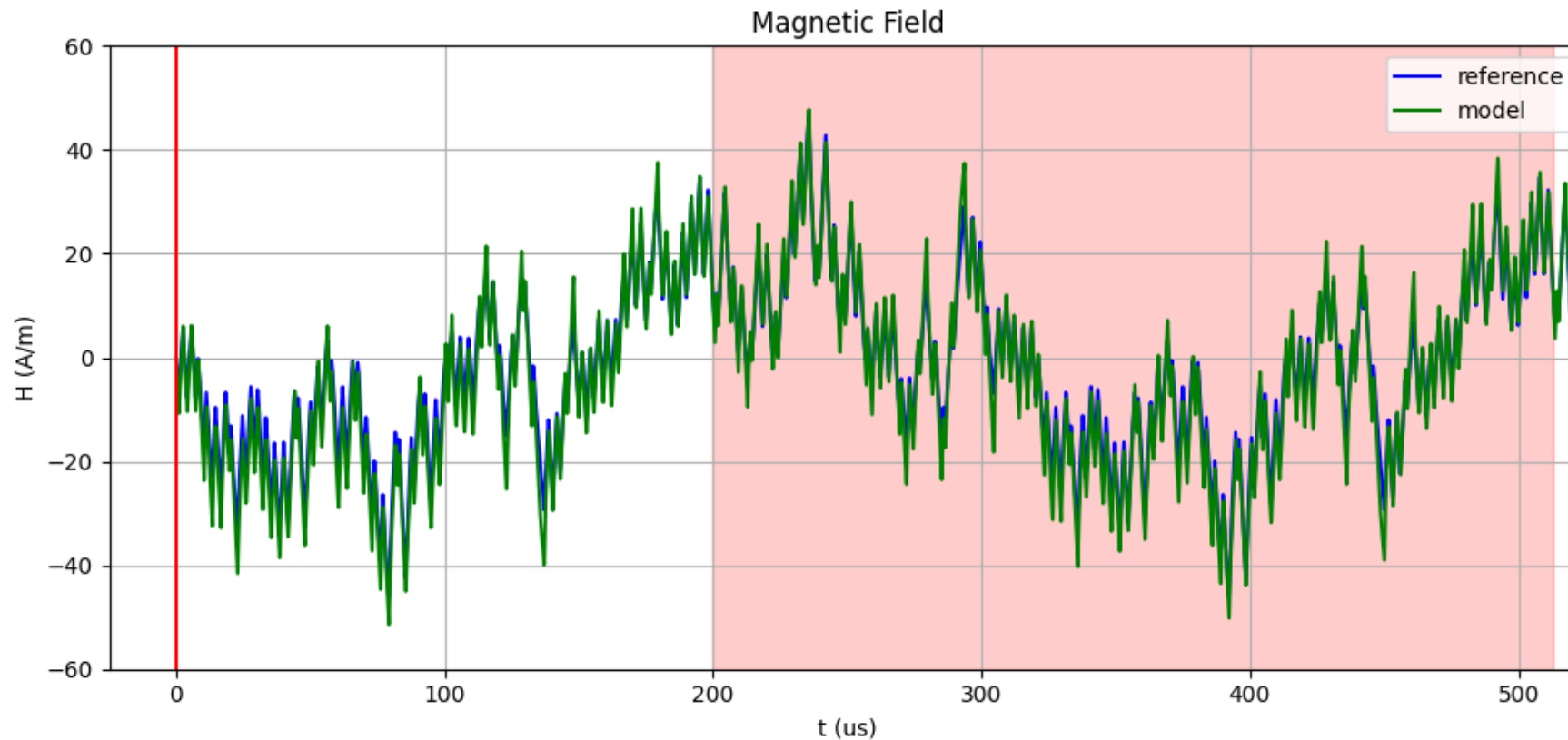
- **Solution:** start the training with a very **short window**
- **Drawback:** initial transient and hysteresis loop is not closed

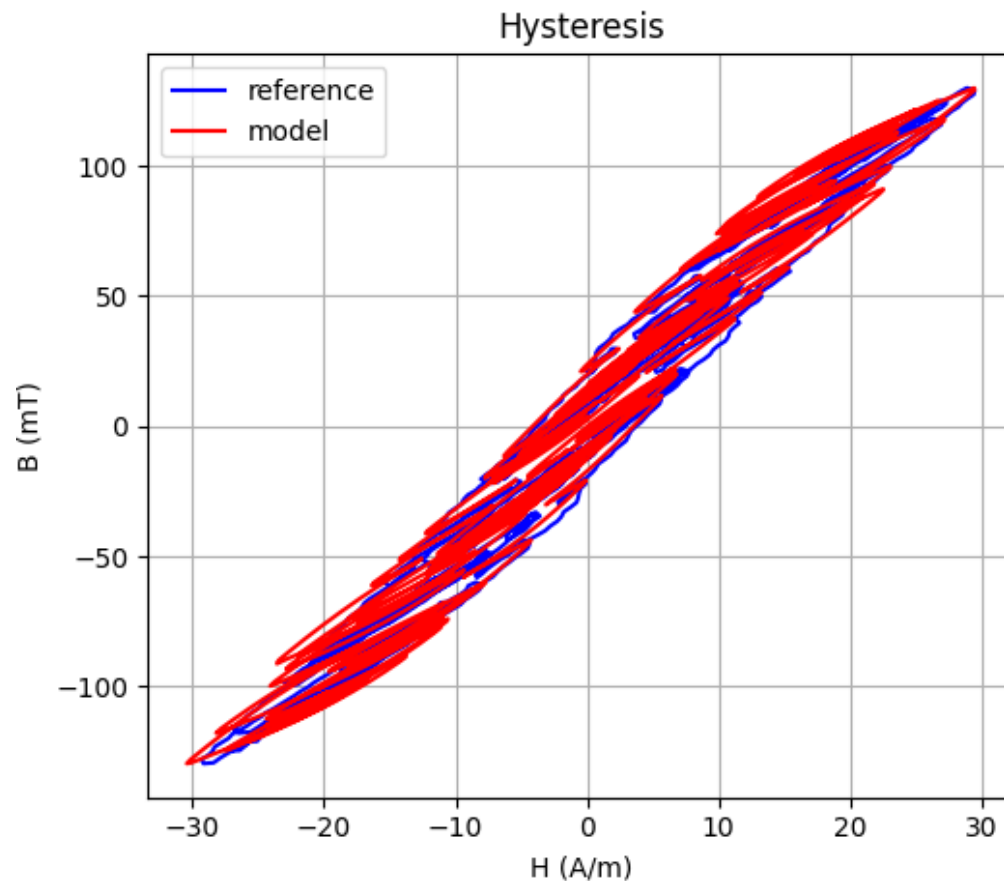
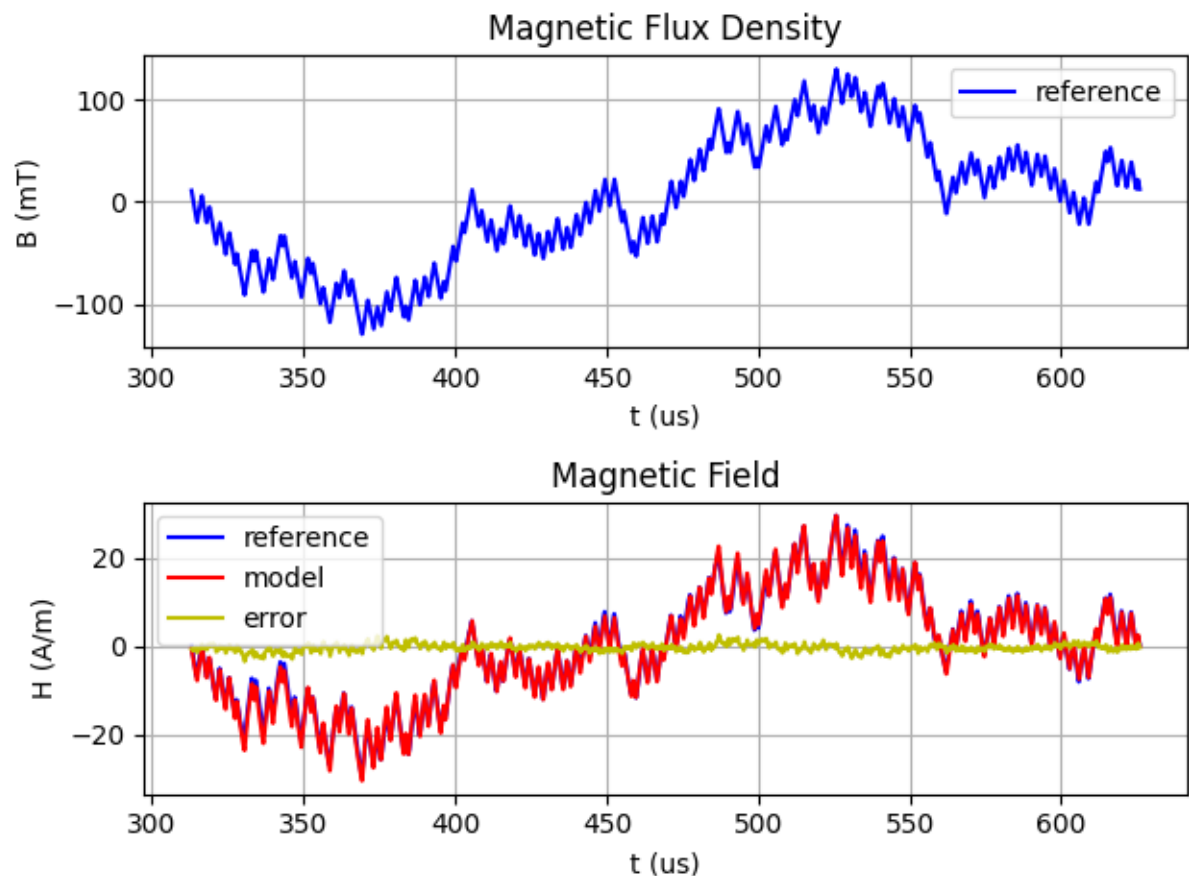


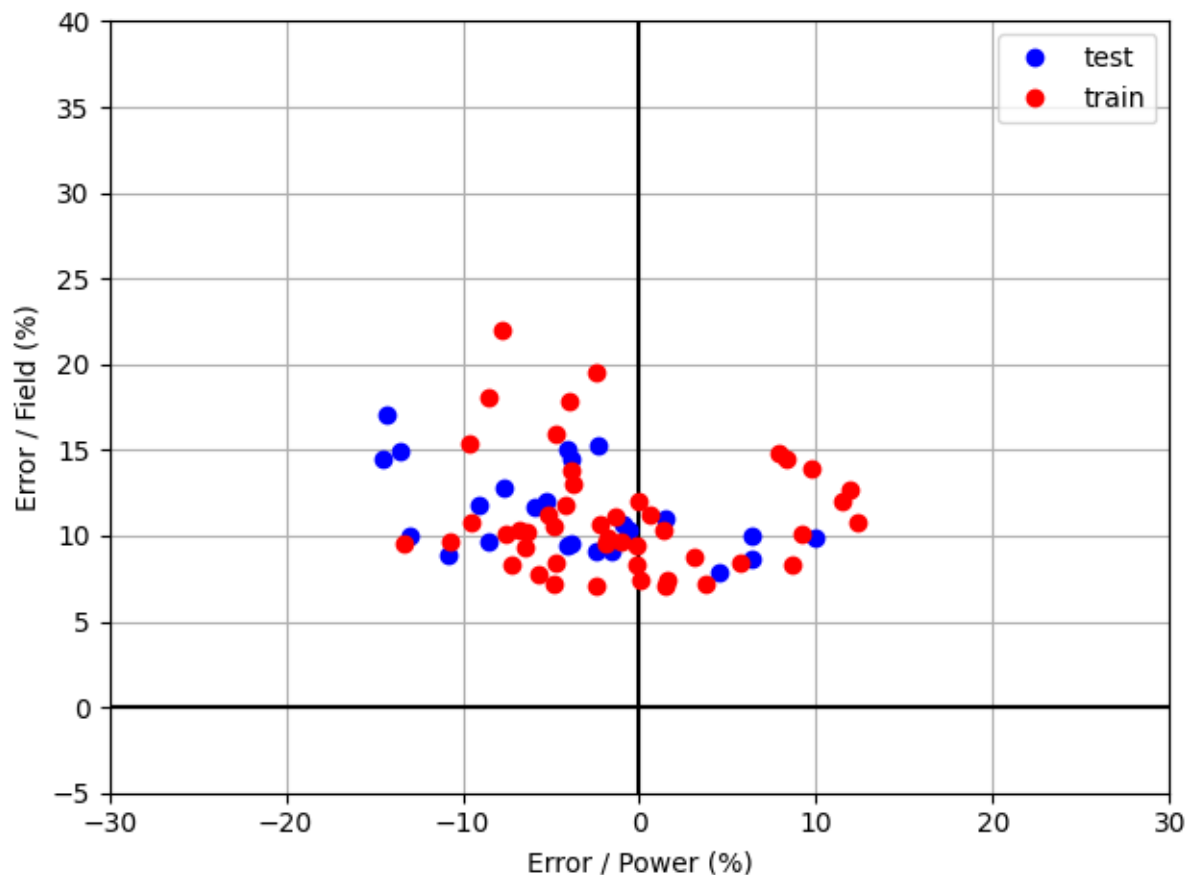
- **Solution:** start the training with a very **short window**
- **Goal:** slowly increase the size of the window



- **Solution:** start the training with a very **short window**
- **Goal:** training a full period with the model in steady-state







- RMS error on the mag. field: 13%
- RMS error on the losses: 9%
- No overfitting (test set)
- Network: MLP with tanh activation
- Network size: 4 inputs and 3 outputs
- Hidden layers: 2 with 16 neurons
- Optimizer: AdaBelief grad. descent

- **Neural ODEs**

- No clear physical meaning
- Opportunity for **compact models**
- Compatible with **existing tooling**

- **Endless possibilities**

- Neural SDE, neural CDE, latent ODE, etc.
- System with discrete states (e.g. Preisach-based)
- Possibility to implement a physics-informed network

- **Some references**

- P. Kidger, “On Neural Differential Equations”, 2021, <https://doi.org/10.48550/arXiv.2202.02435>
- P. Kidger, “Neural Differential Equations in Machine Learning”, 2021, https://kidger.site/links/KItsP45n2UP5/NDE_presentation.pdf

Part IV:

Conclusion and Outlooks

What is the ideal model?

- **Parameters and dataset**
 - Small number of parameters
 - Small dataset for the parametrization
 - Robust parametrization of the model
- **Model performance**
 - Accurate prediction for the field and the losses
 - Extrapolation outside the training/fitting range
 - Predicting waveshapes that are not in the training/fitting data
 - Transition to special cases (small signal, saturation, static curves, etc.)
 - Low computational cost (training and inference)
- **Other qualities**
 - Link with physical phenomena
 - Model debuggability and interpretability
 - Transfer of knowledge between materials
 - Possibility to extend the model (core geometry, DC bias, etc.)
 - Compatible with existing tooling (SPICE, FEM, etc.)

Advantage for analytical models?

Advantage for machine learning?

Unclear advantage ?

- **Used** to generate all the **presented results**
- Can be used as a **starting point** for the challenge
- **Implementation features**
 - **Training** and **inference** of **ODE models**
 - **Management** of the **dataset** with dataframes
 - Using **Python**, **JAX**, **Diffax**, **Pandas**, etc.
 - Does not require a GPU
 - Open source (MPL 2.0)
- **Disclaimers**
 - The goal of this code is to demonstrate basic ODE models
 - The implementation is neither comprehensive nor optimized
- **https://github.com/otvam/magnetic_ode_models**

- **The dataset is quite large**
 - Don't hesitate to create a "simplified fake dataset"
 - Don't hesitate to start with a subset of the dataset
 - Don't hesitate to downsample / truncate the signals
- **Several factors are critical**
 - The model (structure, parameters, etc.)
 - The training/fitting process (optimizer, scaling, etc.)
 - The error metrics used as an objective
- **Programming tips**
 - Use clear interfaces between the training and inference
 - Use vectorized instructions (avoid loops and branches)
 - Nice data structures for storing the results
 - Make tools for displaying the results

Thank you! Questions?

Slides and Python source code:

https://github.com/otvam/magnetic_ode_models

Magnet Challenge GitHub:

<https://github.com/minjiechen/magnetchallenge-2>

