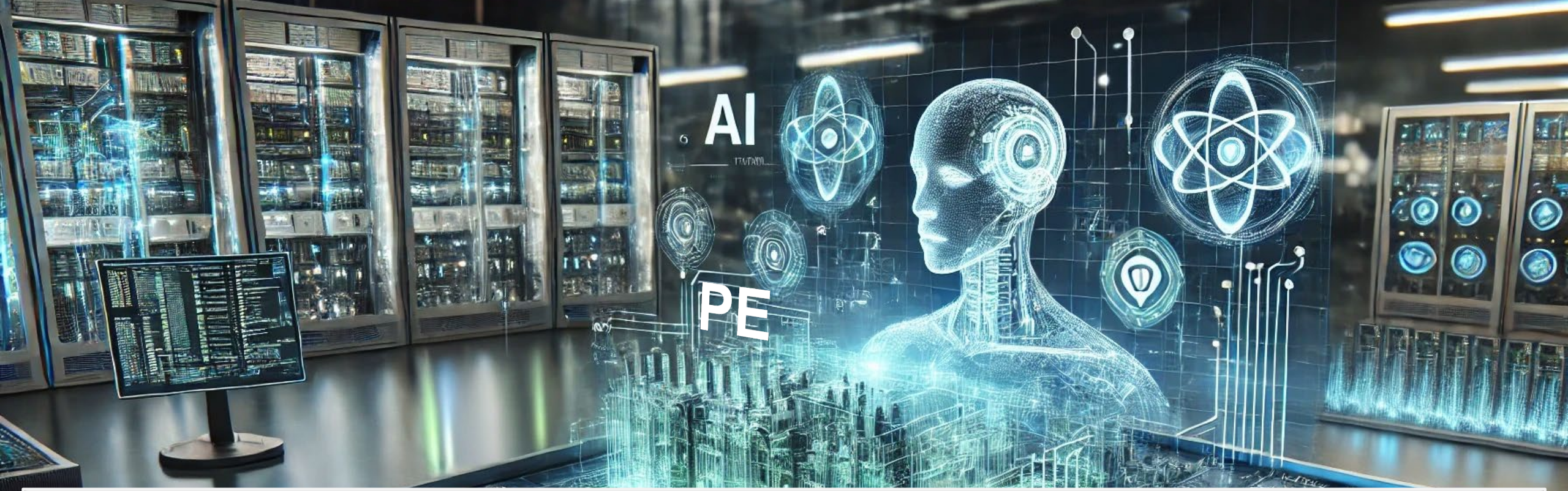



# Next Generation of AI for Power Electronics:

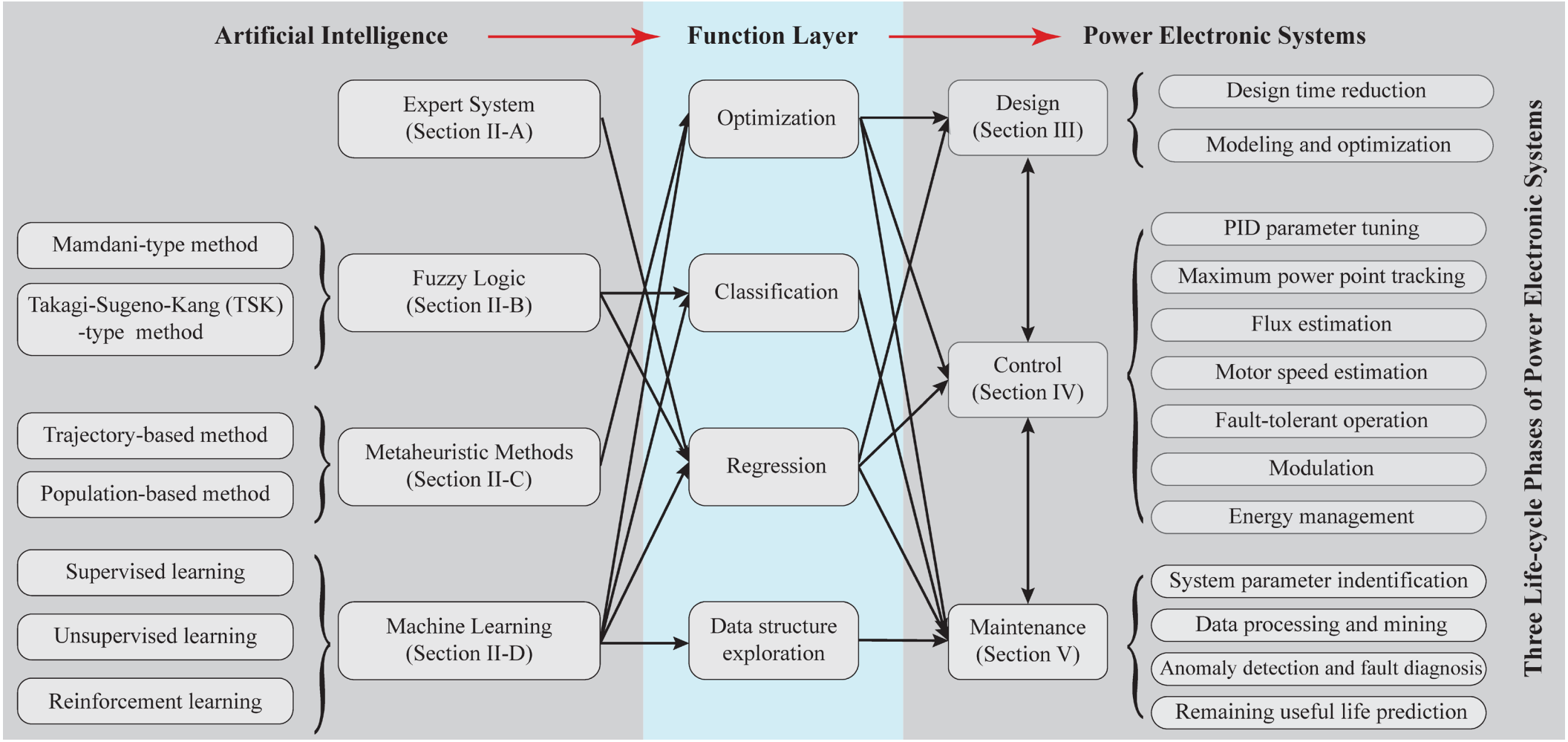
## Explainable, Light, and Flexible

Xinze Li, xinzel@uark.edu

UNIVERSITY OF ARKANSAS

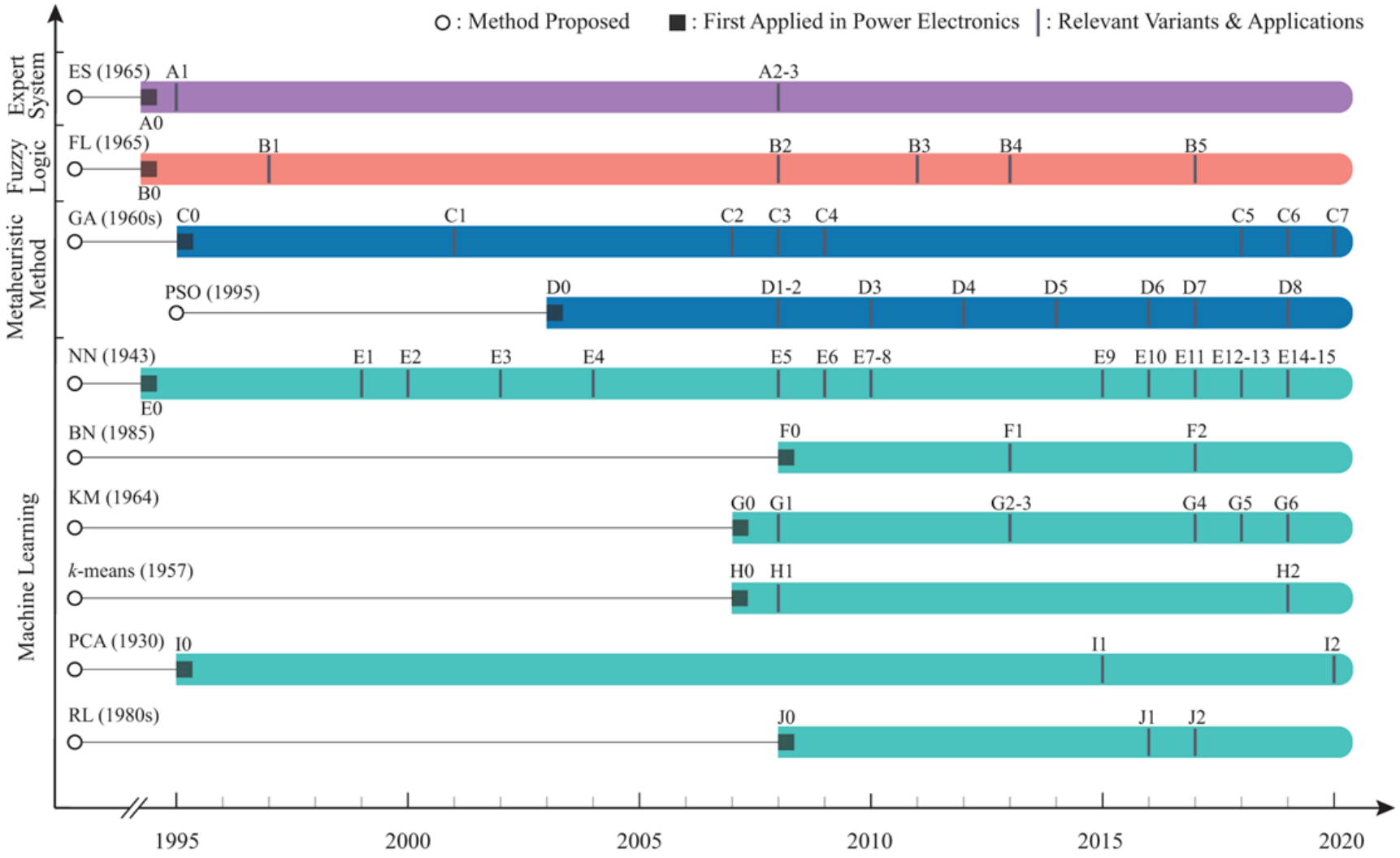
# NEXT GENERATION OF AI FOR POWER ELECTRONICS

# Applications of AI in Power Electronics



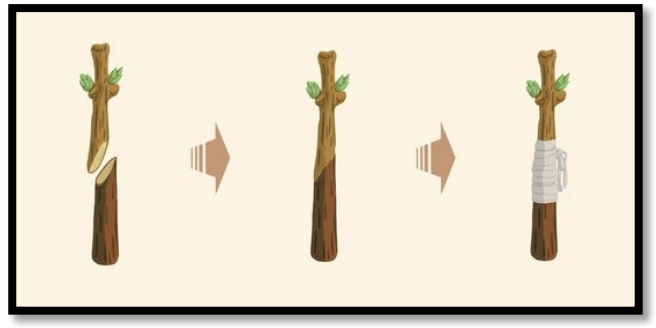

**Source**: Shuai Zhao, Frede Blaabjerg, and Huai Wang, "An overview of artificial intelligence applications for power electronics," *IEEE Transactions on Power Electronics*, vol. 36, no. 4, pp. 4633 – 4658, Apr. 2021.

# Timeline of AI Algorithms Applied in Power Electronics

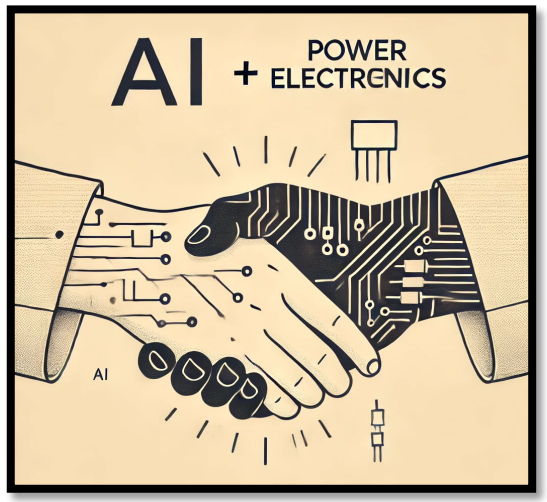



**Source**: Shuai Zhao, Frede Blaabjerg, and Huai Wang, "An overview of artificial intelligence applications for power electronics," *IEEE Transactions on Power Electronics*, vol. 36, no. 4, pp. 4633 – 4658, Apr. 2021.

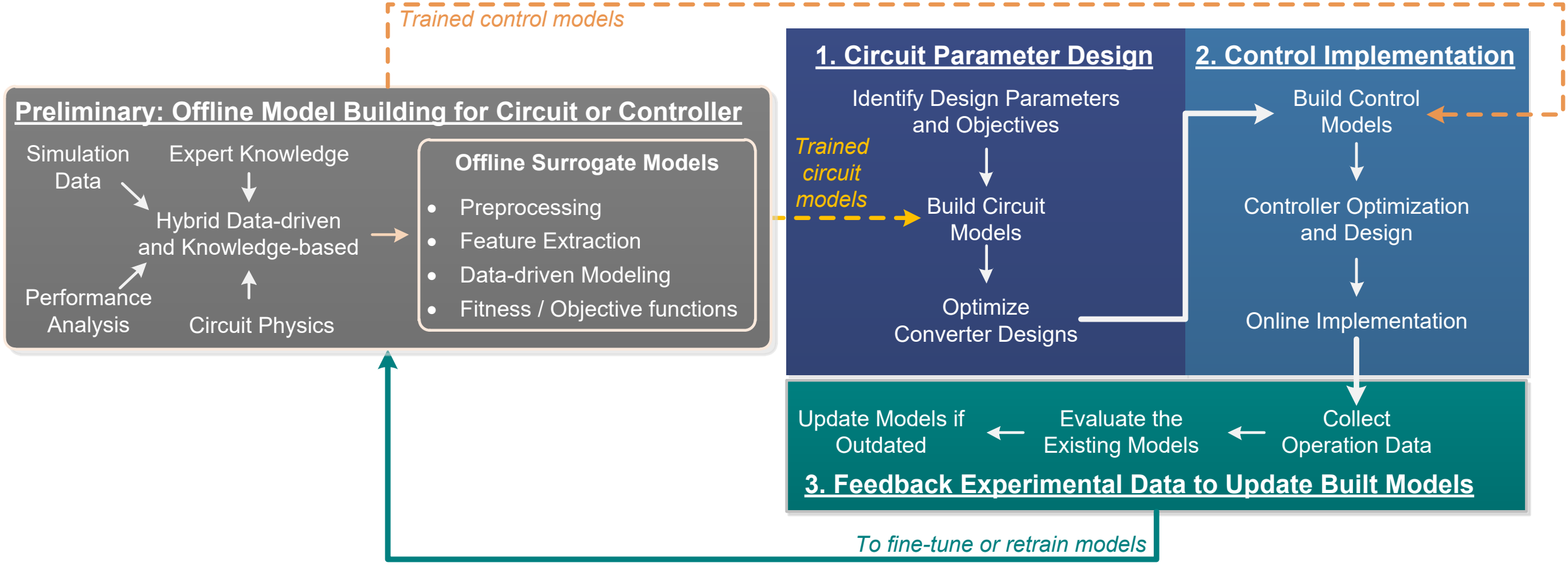
Just like **plant grafting**, where a specific branch is carefully chosen and integrated into a **compatible** rootstock



AI needs to be **adapted and enhanced** with power electronics expertise.

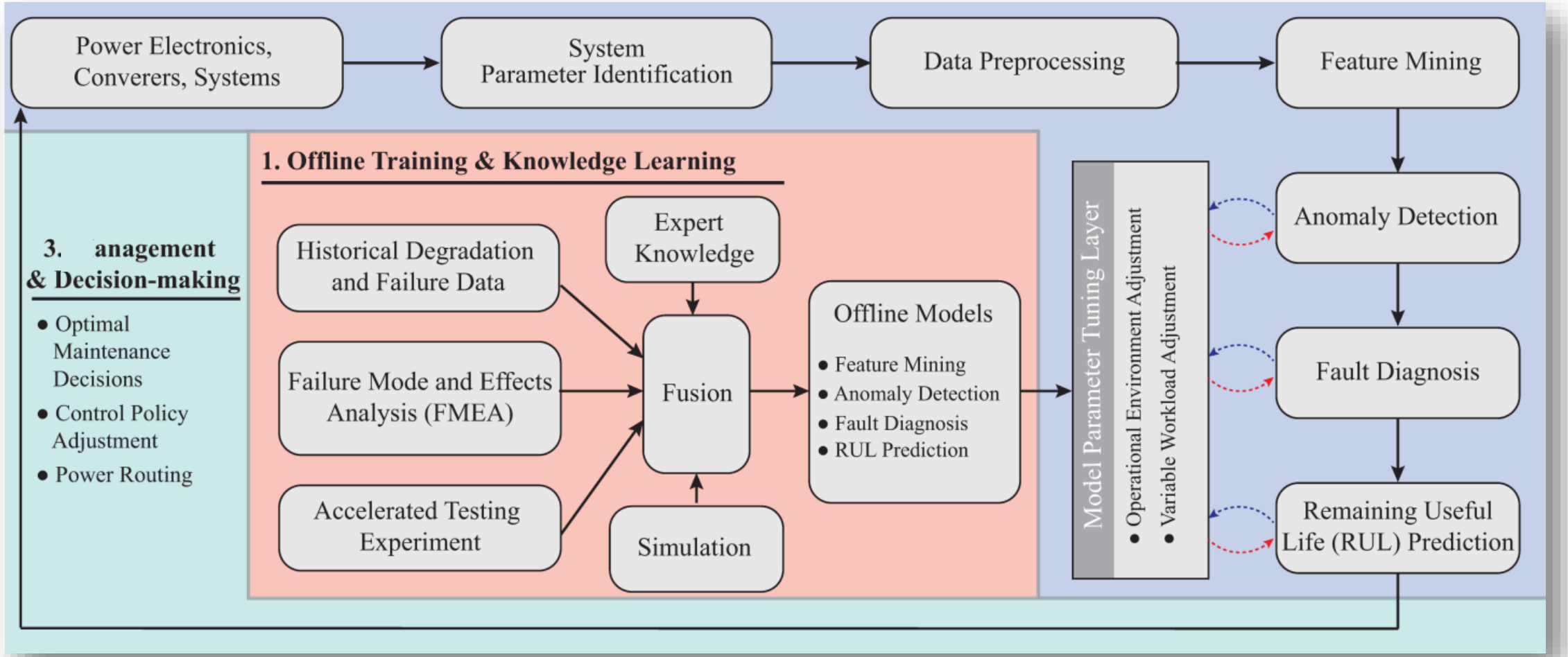


# Existing Application 1: AI-Based Circuit and Control Design




Abstract diagram of circuit and control design through AI-based methods

# Existing Application 2: Digital Twin Based Condition Monitoring




**Source**: S. Zhao, Y. Peng, Y. Zhang and H. Wang, "Parameter Estimation of Power Electronic Converters With Physics-Informed Machine Learning," in *IEEE Transactions on Power Electronics*, vol. 37, no. 10, pp. 11567-11578, Oct. 2022.

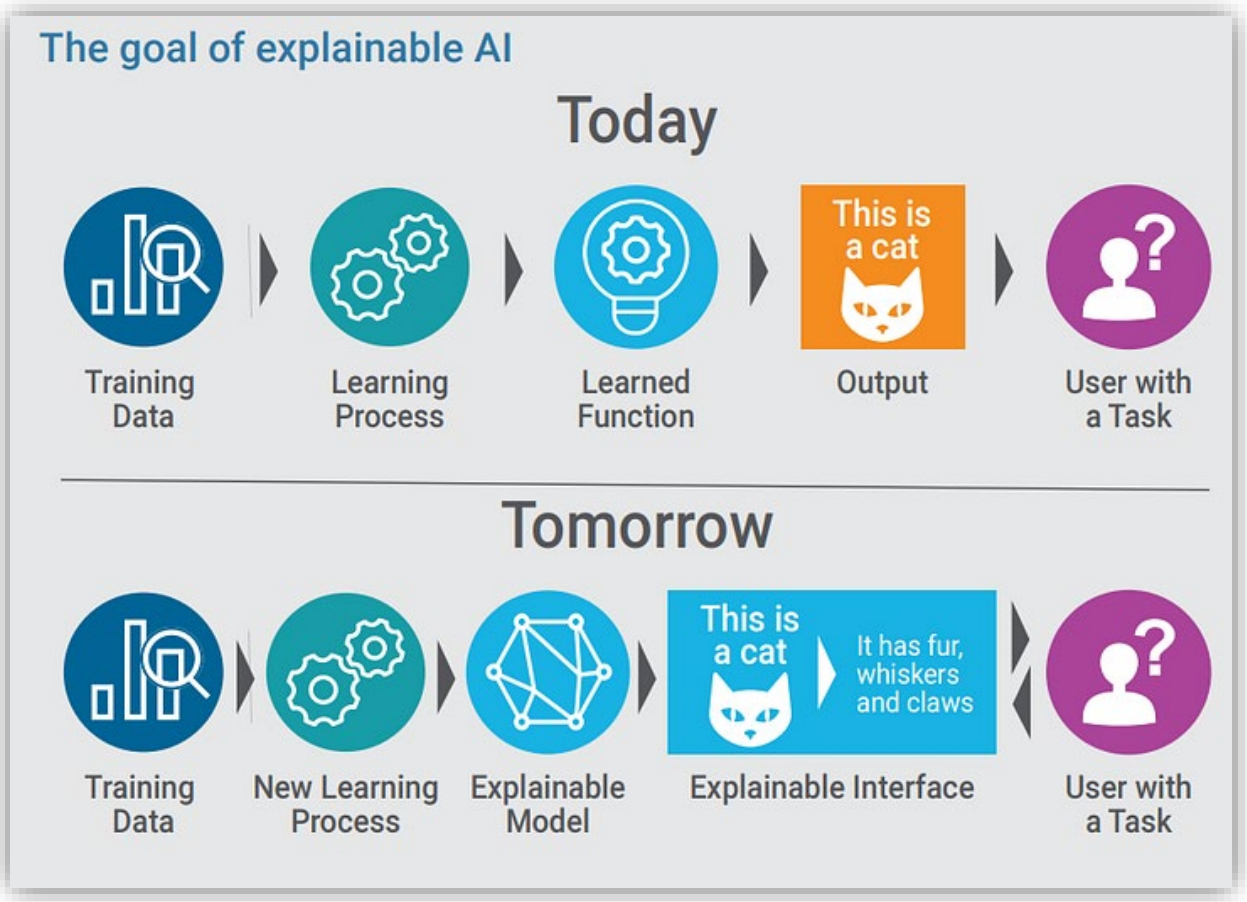
## **Challenge 1:** Power Electronics Requires **Data-Light and Computation-Light** AI

| | Conventional fields | Power electronics |
|---|---|---|
| Data | "Big data"  | Small data  |
| Accuracy | Medium  | High  |
| Computation unit | Flexible  | Edge node with limited computation power  |

**Characteristics of power electronics and other fields**

**Source**: Aalborg University, "Researchers will design cognitive power electronics", https://www.aau.dk/researchers-will-design-cognitive-power-electronics-n28286

# Challenge 2: Explainability of AI to Enhance Deployment Confidence




**Source**: https://towardsdatascience.com/need-for-explainability-in-ai-and-robotics-75dc6077c9fa; F. Lin et al., "PE-GPT: A New Paradigm for Power Electronics Design," *in IEEE Transactions on Industrial Electronics,* doi: 10.1109/TIE.2024.3454408.

## Leverage PE-GPT to Provide Power Electronics Insights into a Designed Buck Converter

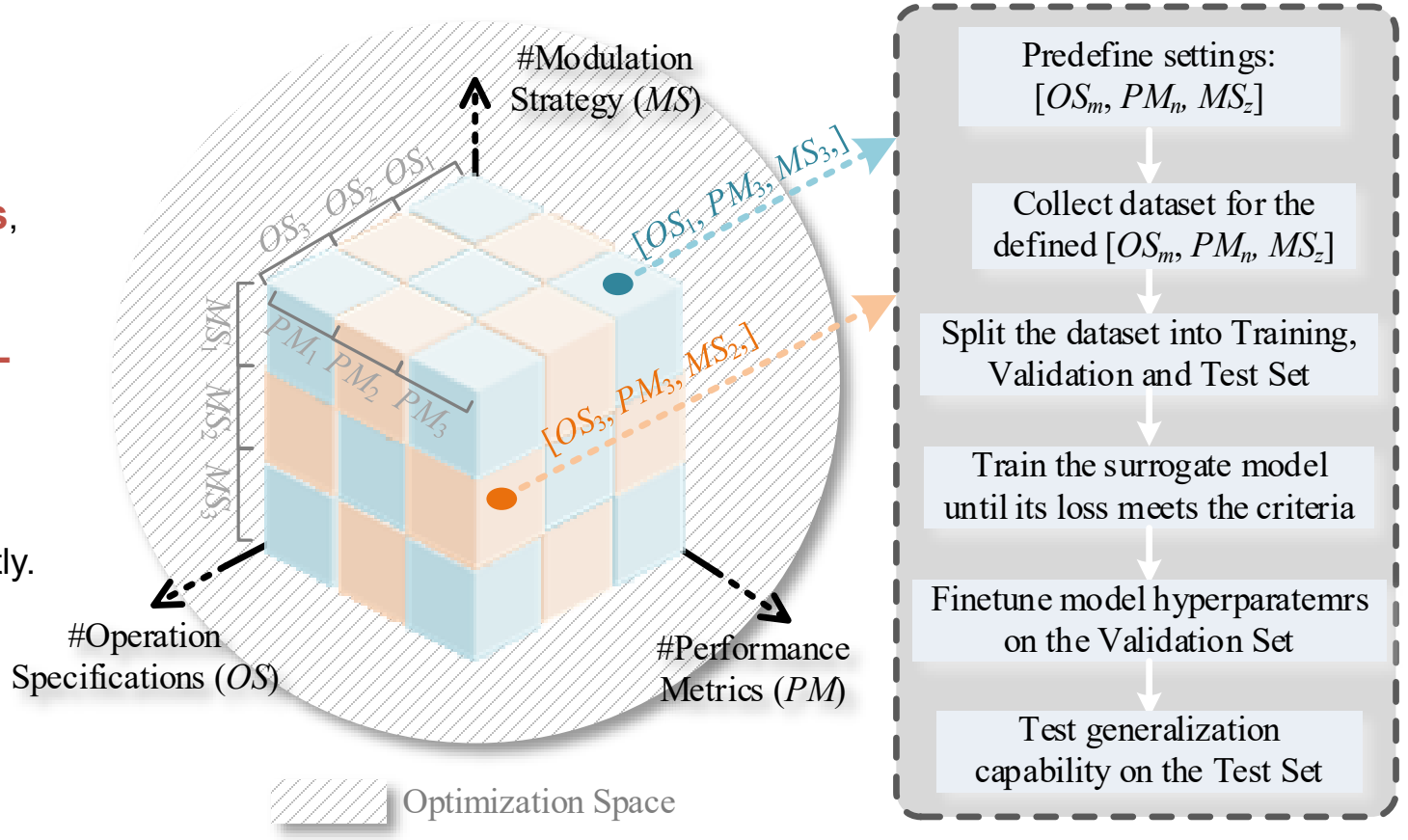
**User:** For the task to design LC filter for the buck converter which has minimum current ripple, why do you recommend this inductor value?

**PE-GPT by F. Lin et al.**

**AI:** The inductor helps to smooth out the ripple in the output current by resisting sudden changes in current flow. Lower ripple percentages typically require larger inductors.

# Challenge 3: Existing AI has NO Flexibility for Diverse Scenarios

- Traditional data-driven is **fixed in operational settings**, limiting model flexibility when settings change.
- Building separate surrogate models for all settings **is data-intensive and computationally demanding**.
- Handling multiple settings need $N_t = m \times n \times z$ **models** (imaging only 3 variables), increasing complexity significantly.
- Topology, circuit parameters, control strategies, performance metrics, operation specifications, etc., could all change.



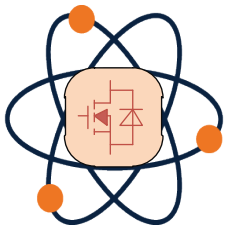

**Number of models required for variable modulation strategy, operation specification, and performance metrics**

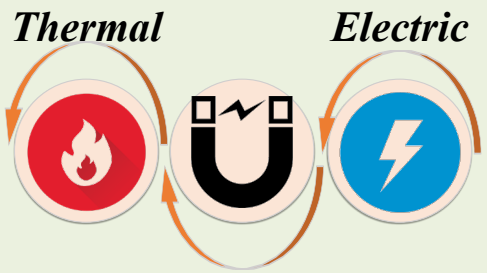
**Source**: F. Lin, X. Li, X. Zhang and H. Ma, "STAR: One-Stop Optimization for Dual-Active-Bridge Converter With Robustness to Operational Diversity," in *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 12, no. 3, pp. 2758-2773, June 2024.

# NEXT GENERATION OF AI FOR POWER ELECTRONICS

# Concept of Physics-Informed Machine Learning (PIML):

**Physics Laws** Expressed as Partial Differential Equations (PDEs)






*Circuit state-space equations:*

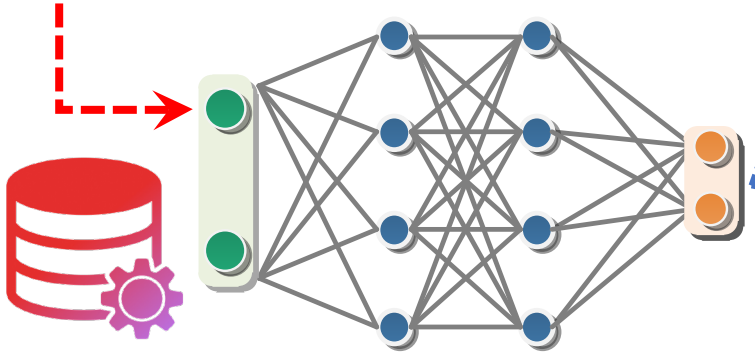
$$\frac{dx(t)}{dt} = Ax(t) + Bu(t)$$

*Heat equation in isotropic medium:*

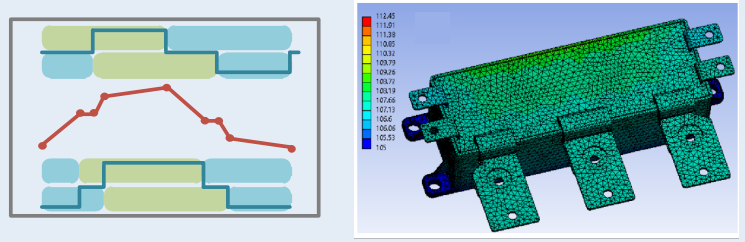
$$\frac{\partial T(t)}{\partial t} = \alpha\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right)$$

**Integrate physics**

Steer the learning of PIML towards **identifying physically consistent solutions**

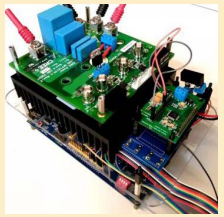


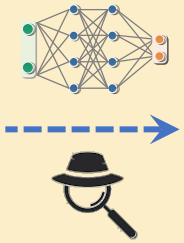
**Machine Learning Models**



- ✓ **Ill-posed scenarios**
  - Noisy or missing data
  - Deficient physical models
- ✓ **Mesh-free**

***Forward Problem*** **(Solve PDEs)**





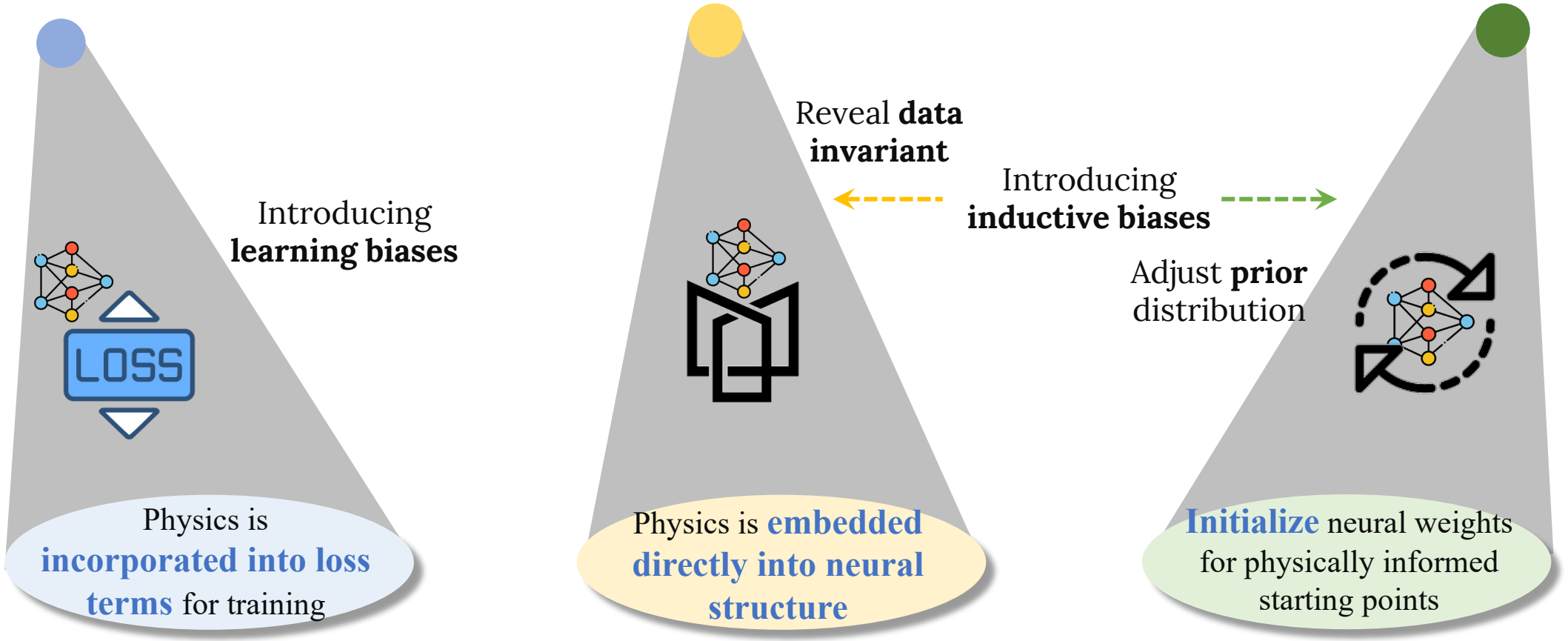
*Identify parameters*

- ✓ **Easy mathematical formulation**
- ✓ **Rich support in AI fields**

***Inverse Problem*** **(Infer Parameters)**

**Source**: G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nat Rev Phys*, vol. 3, no. 6, pp. 422–440, May 2021, doi: 10.1038/s42254-021-00314-5

# Main Types of PIML Approaches:




**Physics-in-Loss PIML**

***Most widely applied***

**Physics-in-Architecture PIML**

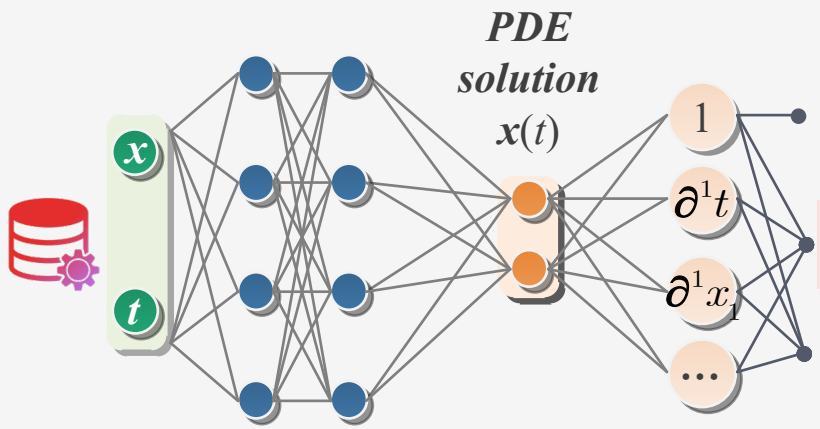
***Focus of this Session***

**Physics-in-Initialization PIML**

# Physics-in-Loss PIML and Some Applications:

Physics as PDEs: $\frac{dx(t)}{dt} = Ax(t) + Bu(t)$

Distill into Training



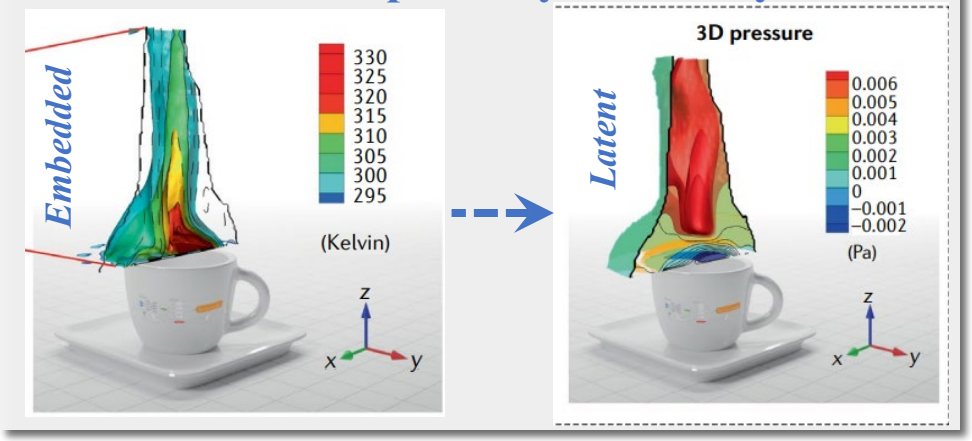

Loss Terms

$$L = w_{data} L_{data} + w_{physics} L_{physics}$$

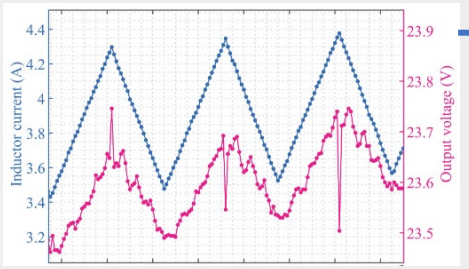
$$L_{data} = \frac{1}{N_D} \sum_{i=1}^{N_D} \left( x(t_i) - x^*(t_i) \right)^2$$

$$L_{physics} = \frac{1}{N_{PDE}} \sum_{i=1}^{N_{PDE}} \left( \frac{dx(t_i)}{dt} - Ax(t_i) - Bu(t_i) \right)^2$$

## Solve PDEs with partially hidden dynamics

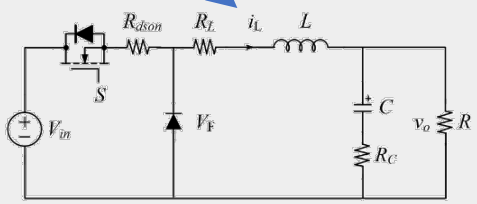



## Parameter identification for power converters




Data acquisition

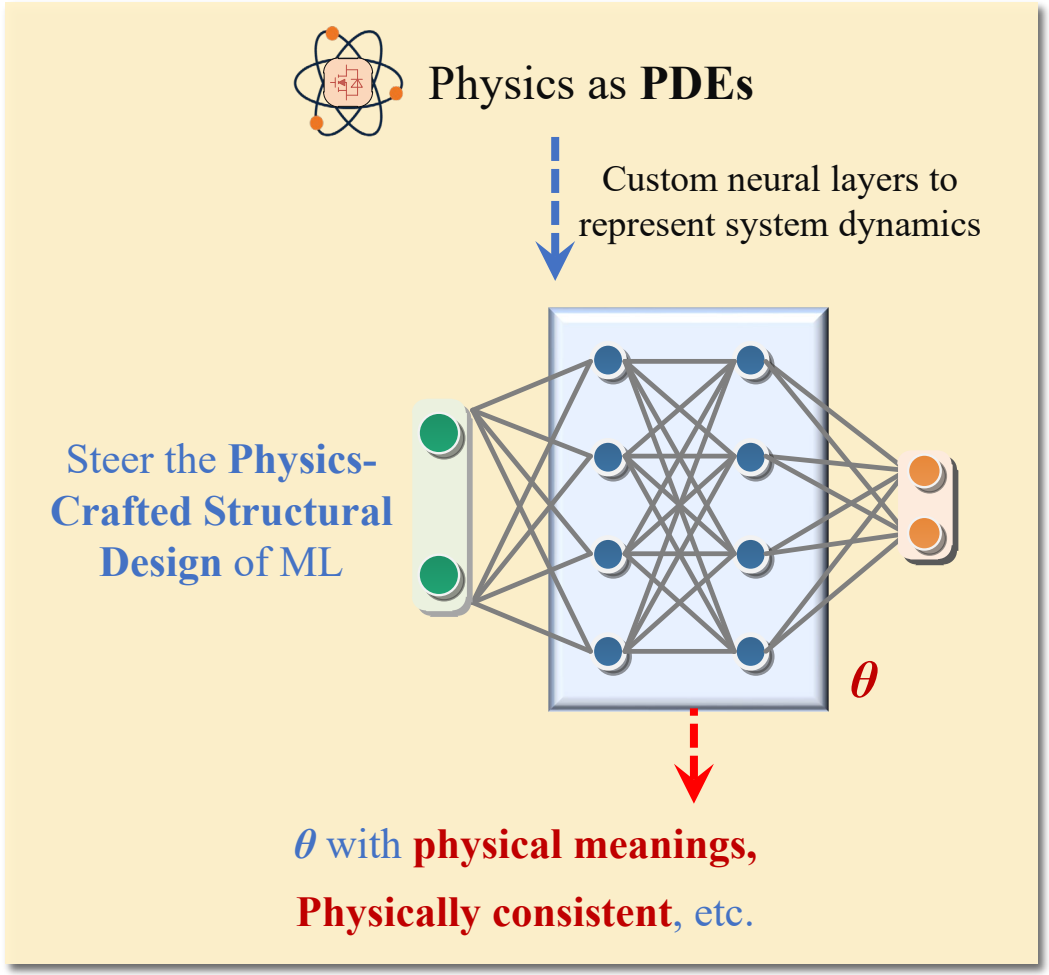
Identify



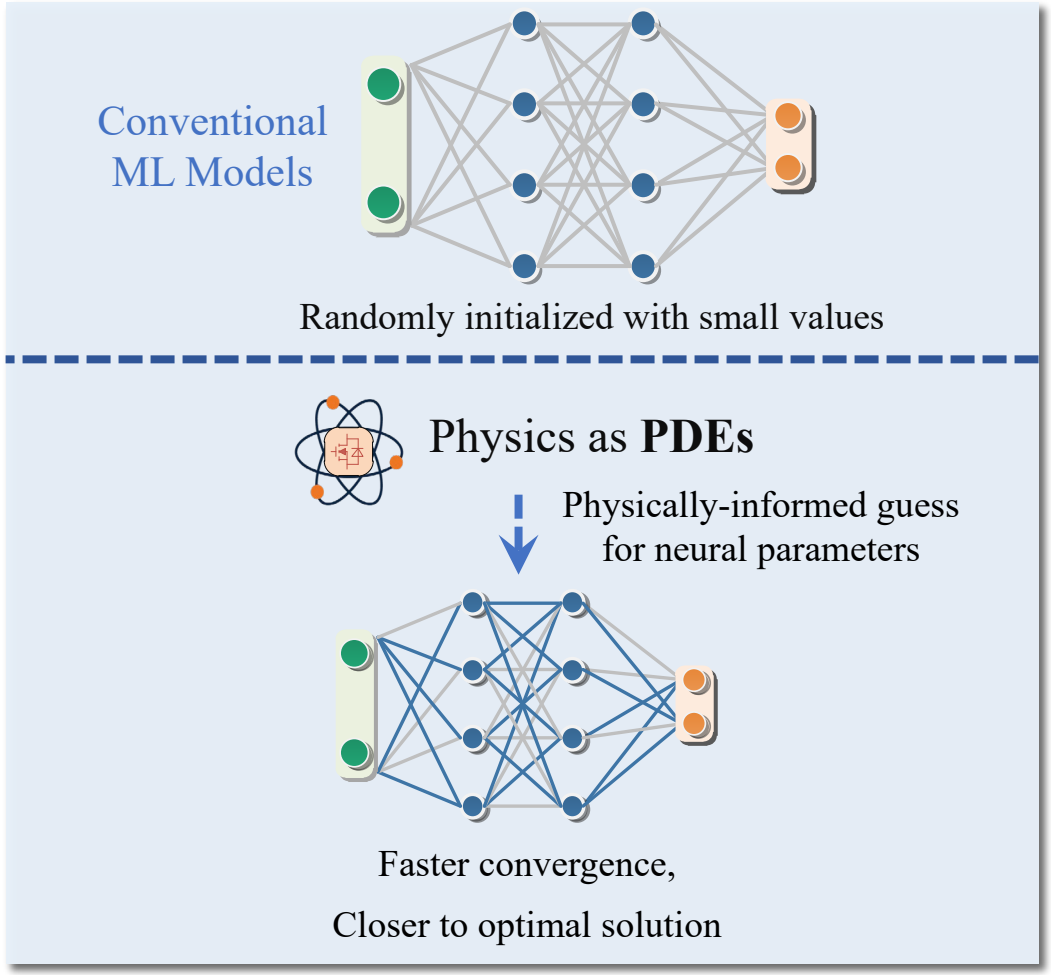

Circuit parameters

**Source**: G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nat Rev Phys*, vol. 3, no. 6, pp. 422–440, May 2021.
S. Zhao, Y. Peng, Y. Zhang, and H. Wang, "Parameter Estimation of Power Electronic Converters With Physics-Informed Machine Learning," *IEEE Transactions on Power Electronics*, vol. 37, no. 10, pp. 11567–11578, Oct. 2022.

# Physics-in-Architecture and Physics-in-Initialization PIMLs



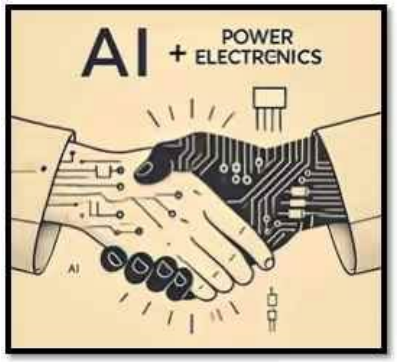

**Physics-in-architecture**




**Physics-in-initialization**

# Pros and Cons of Main PIML Methods






- Less **Data**
- More **Lightweight**
- More **Flexible**
- More **Explainable**

**Physics-in-Architecture** | **Physics-in-Loss** | **Physics-in-Init.** | **Purely Data-Driven**

| *Few Data* | *Some Data* | *Massive Data* |
|---|---|---|
| $\sim 10^0$ to $10^1$ | $\sim 10^2$ to $10^3$ | $\geq 10^4$ |
| *Data Requirements* → | | |
| Well-Posed | Allow for Hidden Physics | NO |
| ← *Physics Effectiveness* | | |
| *Many Physics* | *Some Physics* | *NO Physics* |

- Stronger **Data-Driven Capacity**

**Disadvantages**

| | | |
|---|---|---|
| ■ Low data-driven capacity<br>■ Limited to simpler physical formulation | ■ Physics inconsistent<br>■ Necessitate re-train for out-of-domain scenarios<br>■ More training epochs | ■ No physical meaning<br>■ Massive data needed<br>■ Black-box and heavy model |

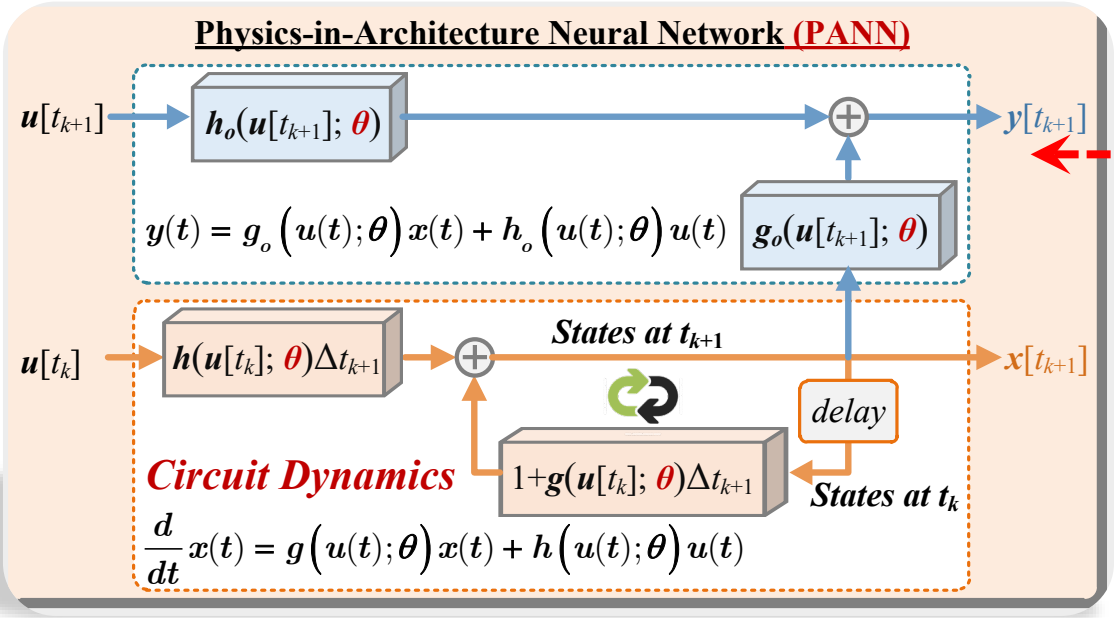
# NEXT GENERATION OF AI FOR POWER ELECTRONICS

# INTRODUCING PANN: NEXT-GENERATION OF AI FOR PE



## Concept of PANN: **Physics-in-Architecture** Neural Network (PANN)

https://github.com/XinzeLee/PANN



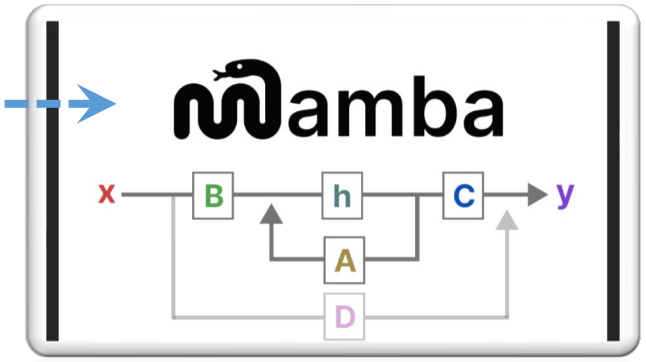

**Physics-in-Architecture Neural Network (PANN)**
**(Submitted on 21 Jun 2023)**

**Embed PE physics** in architecture. Neural $\theta$ is **converter parameters**

Neural weights **A** ~ **D** have **NO** physical meanings

*Time-domain Modeling of power converters*

*All roads lead to destination, but **PANN is a Shortcut for PE***



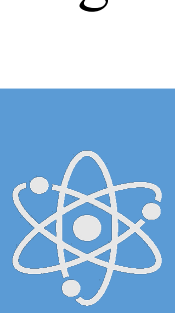

**Mamba State Space Models**
**(Submitted on 1 Dec 2023)**

**Source**:
[1] X. Li et al., "Temporal Modeling for Power Converters With Physics-in-Architecture Recurrent Neural Network," in *IEEE Transactions on Industrial Electronics*, vol. 71, no. 11, pp. 14111-14123, Nov. 2024.
[2] X. Li, F. Lin, X. Zhang, H. Ma and F. Blaabjerg, "Data-Light Physics-Informed Modeling for the Modulation Optimization of a Dual-Active-Bridge Converter," in *IEEE Transactions on Power Electronics*, vol. 39, no. 7, pp. 8770-8785, July 2024.
[3] F. Lin, X. Li, X. Zhang and H. Ma, "STAR: One-Stop Optimization for Dual-Active-Bridge Converter With Robustness to Operational Diversity," in *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 12, no. 3, pp. 2758-2773, June 2024.
[4] X. Li et al., "A Generic Modeling Approach for Dual-Active-Bridge Converter Family via Topology Transferrable Networks," in *IEEE Transactions on Industrial Electronics*, doi: 10.1109/TIE.2024.3406858.
[5] F. Lin et al., "PE-GPT: A New Paradigm for Power Electronics Design," in *IEEE Transactions on Industrial Electronics*, doi: 10.1109/TIE.2024.3454408.

# WHY PANN? EXPLAINABLE! LIGHT! FLEXIBLE!



Advantages of PANN: **Explainable, Light, and Flexible**



## Explainable

- **Physics-crafted architecture**
  - Intrinsic dynamics
- **Power electronics explainable**
  - Switching behaviors
  - Commutation loops

## Light

- **Data-light**
  - Reduce data by 3 orders of magnitudes
  - Countable in one hand
- **Lightweight**
  - Deployable on edge
  - Down to kB level

## Flexible

- **Training-free to out-of-domain scenarios**
  - Operating conditions
  - Modulation strategies
  - Performance metrics
  - Circuit parameters
  - Topological variants

**PANN is Suitable for Power Electronics Applications**

# PANN is a form of **Neural** Partial Differential Equations (***NeuralPDE***)

**Continuous** Generic **state-space equations** of power converters

$$\begin{cases} \dot{x}(t) = g\left(u(t);\theta\right)x(t) + h\left(u(t);\theta\right)u(t) \\ y(t) = g_o\left(u(t);\theta\right)x(t) + h_o\left(u(t);\theta\right)u(t) \end{cases}$$

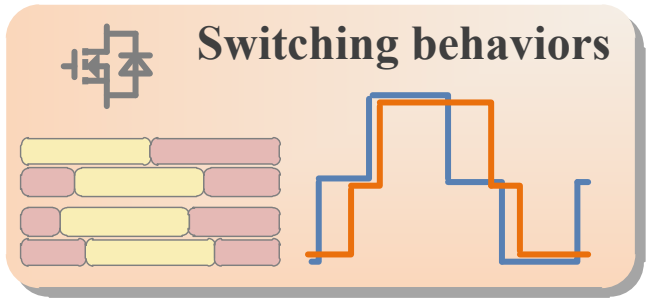
**Numerical methods** →

**Discrete, recurrent**

$$x\left[t_{k+1}\right] = x\left[t_k\right] + \phi\left(x\left[t_k\right];u\left[t_k\right]\ldots;\Delta t_{k+1};\theta\right)\Delta t_{k+1}$$

$$\begin{cases} x[t_{k+1}] = \left(1 + g\left(u[t_k];\theta\right)\Delta t_{k+1}\right)x[t_k] + h\left(u[t_k];\theta\right)u[t_k]\Delta t_{k+1} \\ y[t_{k+1}] = g_o\left(u[t_{k+1}];\theta\right)x[t_{k+1}] + h_o\left(u[t_{k+1}];\theta\right)u[t_{k+1}] \end{cases}$$

***Expert Systems***




Inputs $u[t_k]$, …

Current states $\boldsymbol{x[t_k]}$

Next states $\boldsymbol{x[t_{k+1}]}$

**Increment function $\phi$**

**Neural Euler core**

$u[t_k]$, $\theta$

$g\left(u(t);\theta\right)$ × $x[t_k]$

$h\left(u(t);\theta\right)$ + *out*

$\Delta t_{k+1}$

**PANN**

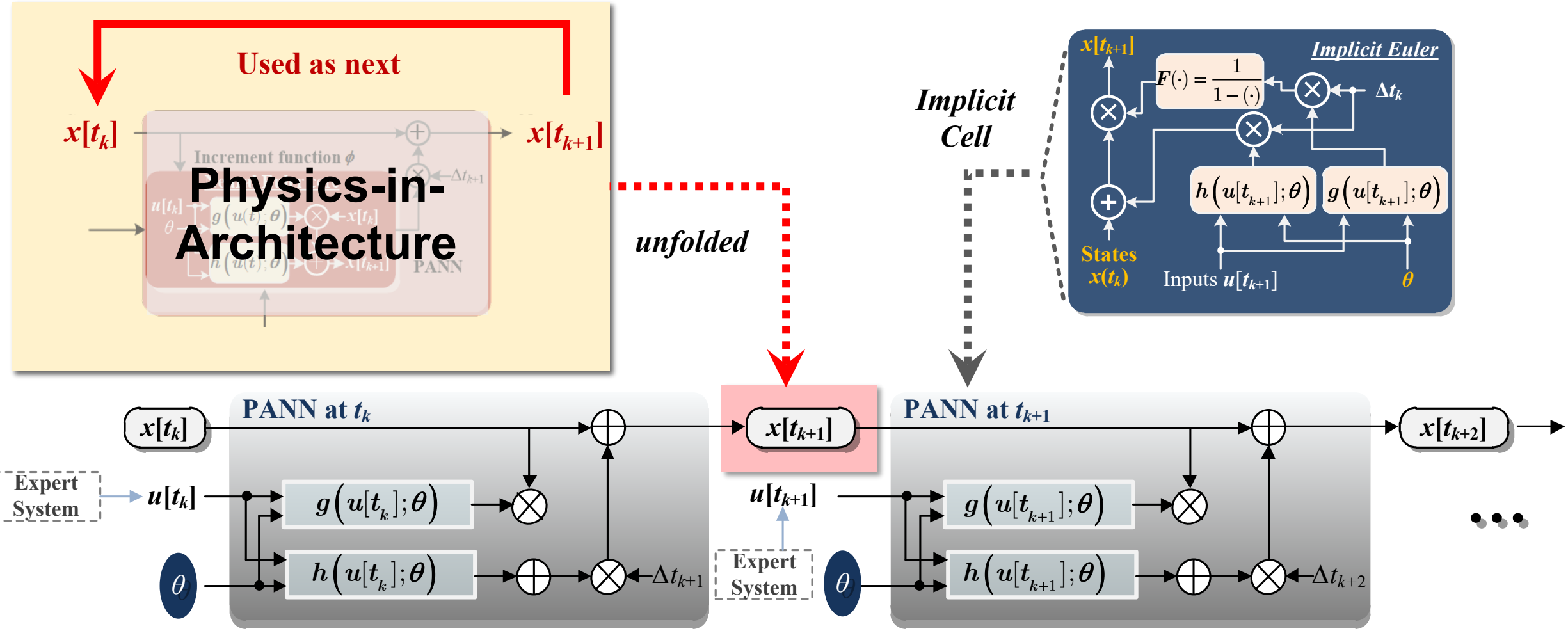
**Custom recurrent neural architecture of PANN**

**Trainable converter parameters, data-driven** ⇠ Parameters $\theta$ ⇢ **Physically explainable, direct assignment**

# PANN Inference: Predict the Next State



PANN inference: ***time-domain modeling of power converters***



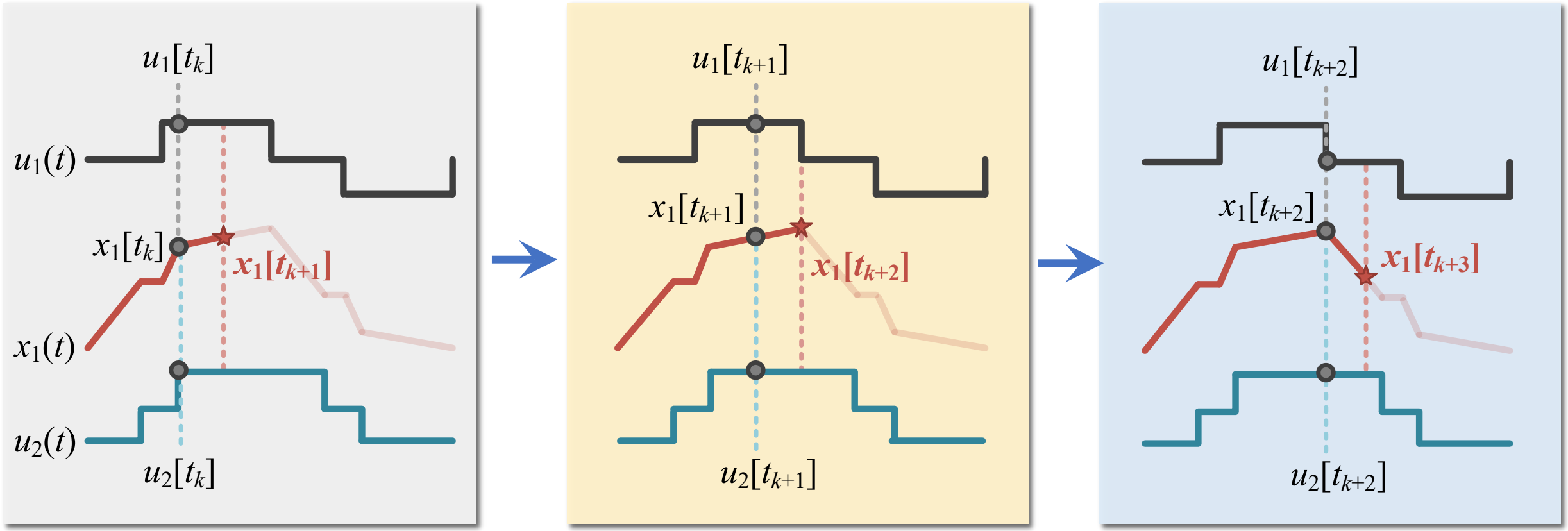

*In modeling, teacher forcing is infeasible*

**Predicted state $x[t_{k+1}]$ at $t_{k+1}$ is used to infer the following state $x[t_{k+2}]$ at $t_{k+2}$**

# PANN INFERENCE: PREDICT THE NEXT STATE



PANN inference **unfolded** over time: ***predict the next state***




Time $t_k$ Time $t_{k+1}$ Time $t_{k+2}$

# CODE STRUCTURE FOR PANN INFERENCE



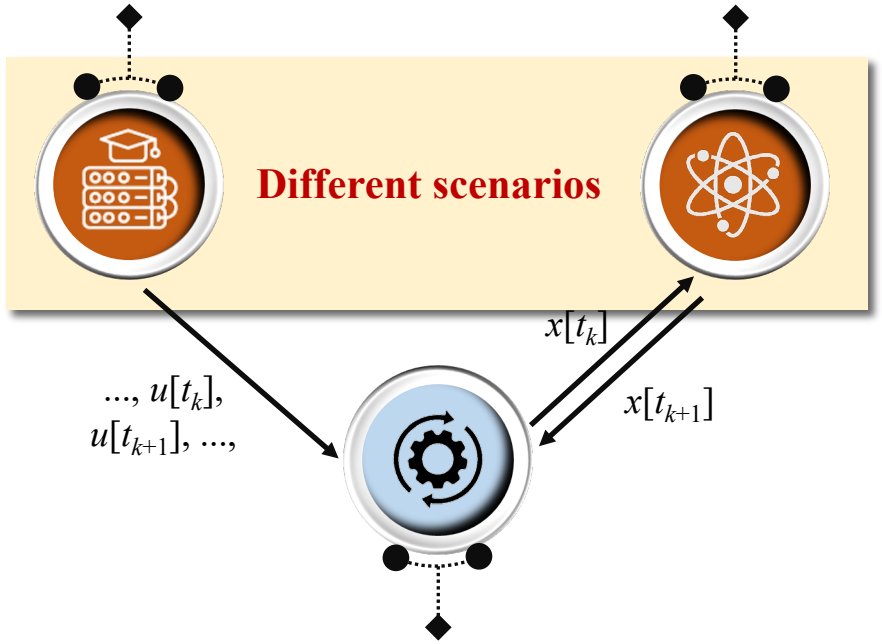
## Code Structure for PANN Inference

### Module 2. Input Blocks

Generate switching or input variables based on modulation knowledge

### Module 3. Physics Cell

Infer state variables for one timestamp based on physics




### Module 1. Recurrence Cell

Infer state variable waveforms in a recurrent way

## Module 1. Recurrence Cell

*Recurrent infer*

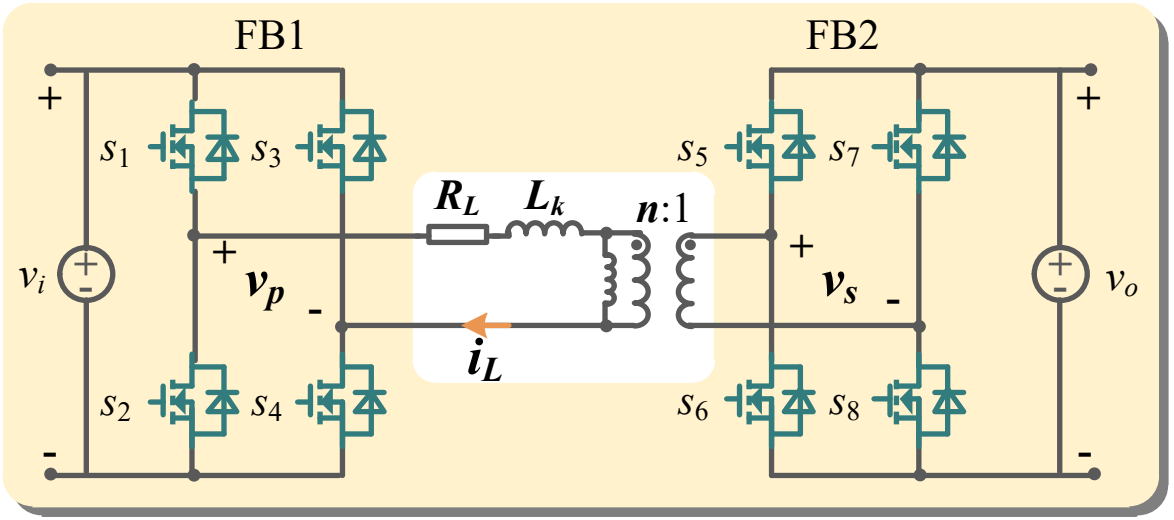
pann_net.py :: class PANN

```python
class PANN(nn.Module):
    """
        Define the generic physics-in-architecture neural network (PANN)
    """

    def __init__(self, cell, **kwargs):
        super(PANN, self).__init__(**kwargs)
        self.cell = cell

    def forward(self, inputs, x):
        outputs = []
        _x = x[:, 0] # initialize the state
        for t in range(inputs.shape[1]):
            state_next = self.cell.forward(inputs[:, t, :], _x) # infer
            _x = state_next # the predicted state (state_next) as the cu
            outputs.append(_x) # append the predicted states to outputs
        return torch.stack(outputs, dim=1)
```

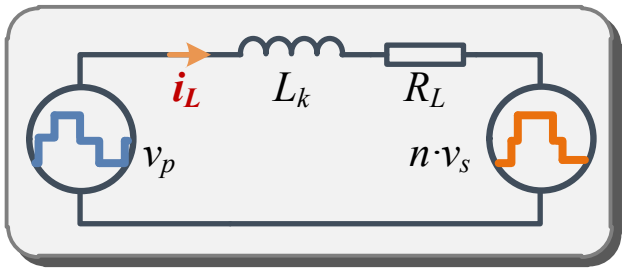
■ Physics Cell

■ Recurrent Inference

## Case Study 1: PANN for *DAB Converters*




Equivalent Circuit of Inductor




| | Specs |
|---|---|
| States $\boldsymbol{x}(t)$ | $i_L$ |
| Inputs $\boldsymbol{u}(t)$ | $v_p$, $v_s$ |
| Circuit $\boldsymbol{\theta}$ | $R_L$, $L_k$, $n$ |

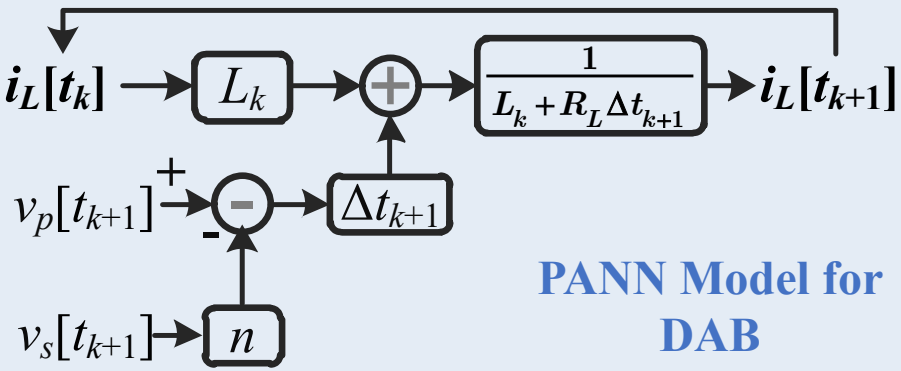
Continuous $L_k \frac{di_L(t)}{dt} + R_L i_L(t) = v_p(t) - n v_s(t)$

Discretize via Implicit Euler Algorithm

$$i_L[t_{k+1}] - i_L[t_k] = \left(v_p[t_{k+1}] - n v_s[t_{k+1}] - R_L i_L[t_{k+1}]\right) \frac{\Delta t_{k+1}}{L_k}$$

Simplify

$$i_L[t_{k+1}] = \frac{\left(\Delta t_{k+1}\left(v_p[t_{k+1}] - n v_s[t_{k+1}]\right) + L_k i_L[t_k]\right)}{L_k + R_L \Delta t_{k+1}}$$




PANN Model for DAB

# CASE STUDY: PANN FOR DAB CONVERTERS



## Case Study 1: PANN for *DAB Converters*

```
D0 = D0+D1-D2
# create a time array with the sampling period dt
t = np.linspace(0, Ts, round(Ts/dt), endpoint=False)

# create switching functions
s_pri = signal.square(2*np.pi/Ts*t, D1_cycle)
s_pri2 = deque(-s_pri)
s_pri2.rotate(int(np.ceil(np.round(D1*Ts/2/dt, 5))))
s_sec = deque(signal.square(2*np.pi/Ts*t, D2_cycle))
s_sec.rotate(int(np.ceil(np.round(D0*Ts/2/dt, 5))))
s_sec2 = deque(-np.array(s_sec))
s_sec2.rotate(int(np.ceil(np.round(D2*Ts/2/dt, 5))))

# create high-frequency ac waveforms
vp = (np.array(s_pri)+np.array(s_pri2)).clip(-1, 1)*Vin
vs = (np.array(s_sec)+np.array(s_sec2)).clip(-1, 1)*Vref
```

■ Gate drive signal for $s_1$

■ Gate drive signal for $s_4$

■ Gate drive signal for $s_5$

■ Gate drive signal for $s_8$

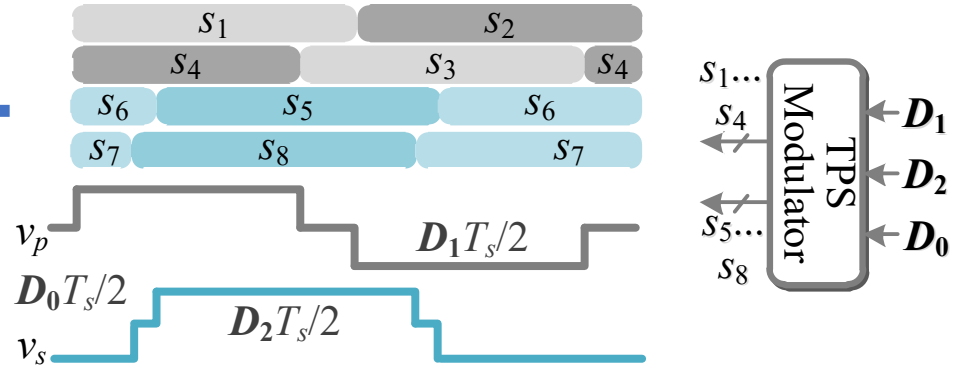
■ Generate $v_p$ and $v_s$

**Module 2. Input Blocks**

pann_utils.py :: def create_vpvs

*To generate* $v_p$ *and* $v_s$

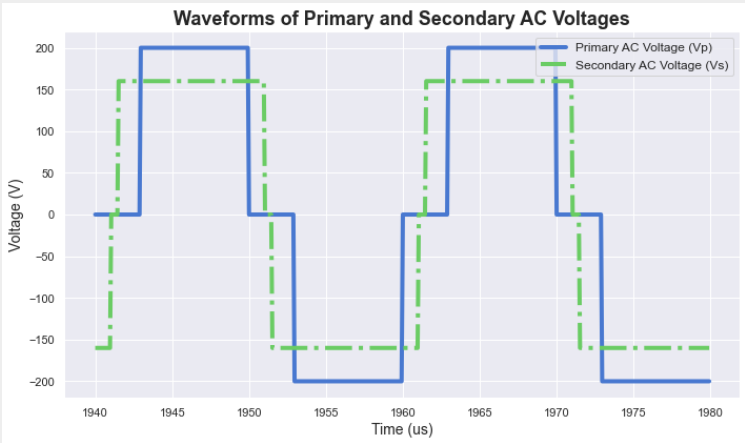
*embed*




*args*: ($D_0$, $D_1$, $D_2$, $D_{1\_cycle}$, $D_{2\_cycle}$, $v_i$, $v_o$)

*args*: (0.1, 0.7, 0.95, 50%, 50%, 200, 160)

*return*:




**Exemplary $v_p$ and $v_s$ waveforms**

# Case Study 1: PANN for *DAB Converters*

```python
class EulerCell_DAB(nn.Module):

    def __init__(self, dt, Lr, RL, n, **kwargs):
        super(EulerCell_DAB, self).__init__(**kwargs)
        self.dt = dt # define a constant parameter time step dt
        self.Lr = nn.Parameter(torch.Tensor([Lr])) # define trainable
        self.RL = nn.Parameter(torch.Tensor([RL])) # define trainable
        self.n = nn.Parameter(torch.Tensor([n])) # define trainable ne
    def forward(self, inputs, states):
        # inputs : represent vp, vs respectively
        # states : represent iL, respectively
        iL_next = (self.Lr/(self.Lr+self.RL*self.dt))*states[:, 0]+\
            (self.dt/(self.Lr+self.RL*self.dt))*(inputs[:, 0]-
                                                 self.n*inputs[:, 1])
        return iL_next[:, None]
```

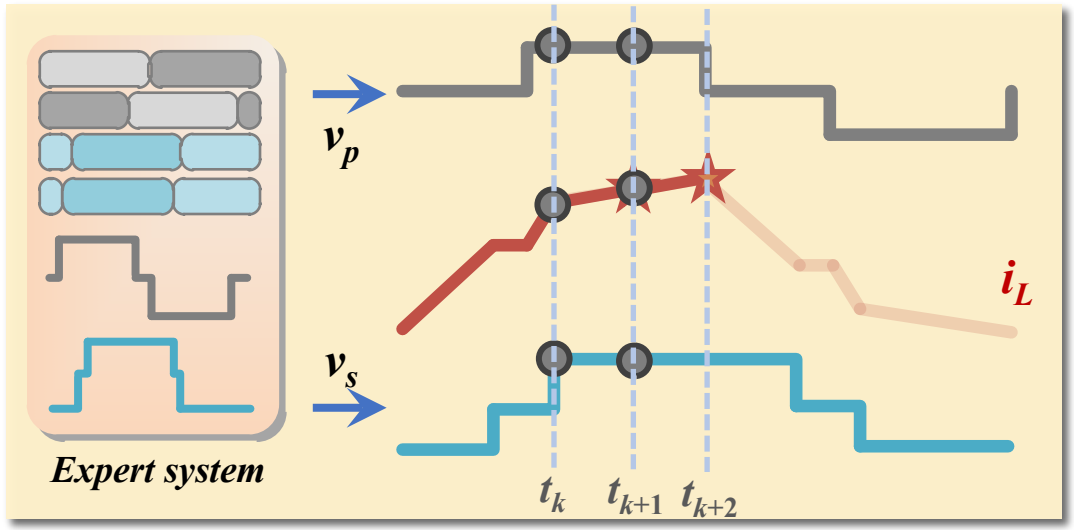
- Define neural circuit parameters $\theta$
- Physics of inductor

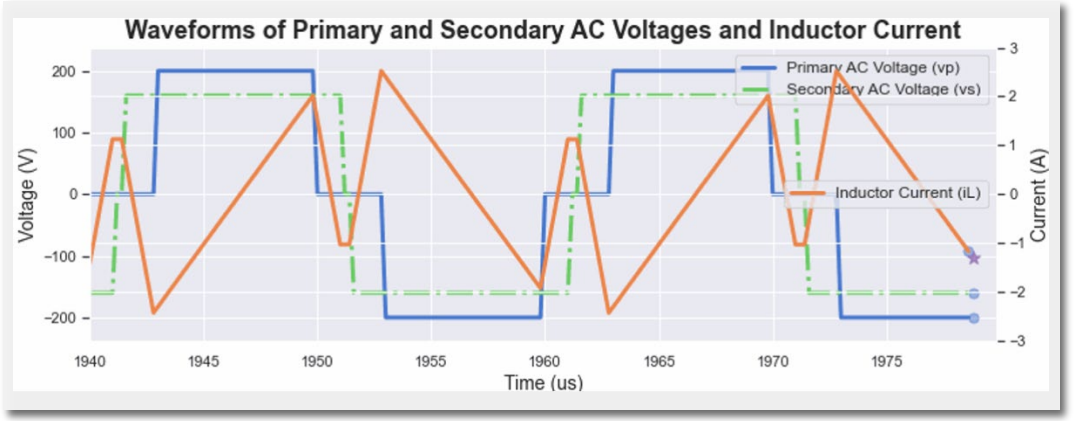
*Embed physics*

**Module 3. Physics Cell**

pann_net.py :: class EulerCell_DAB

**Animated inference breakdown**



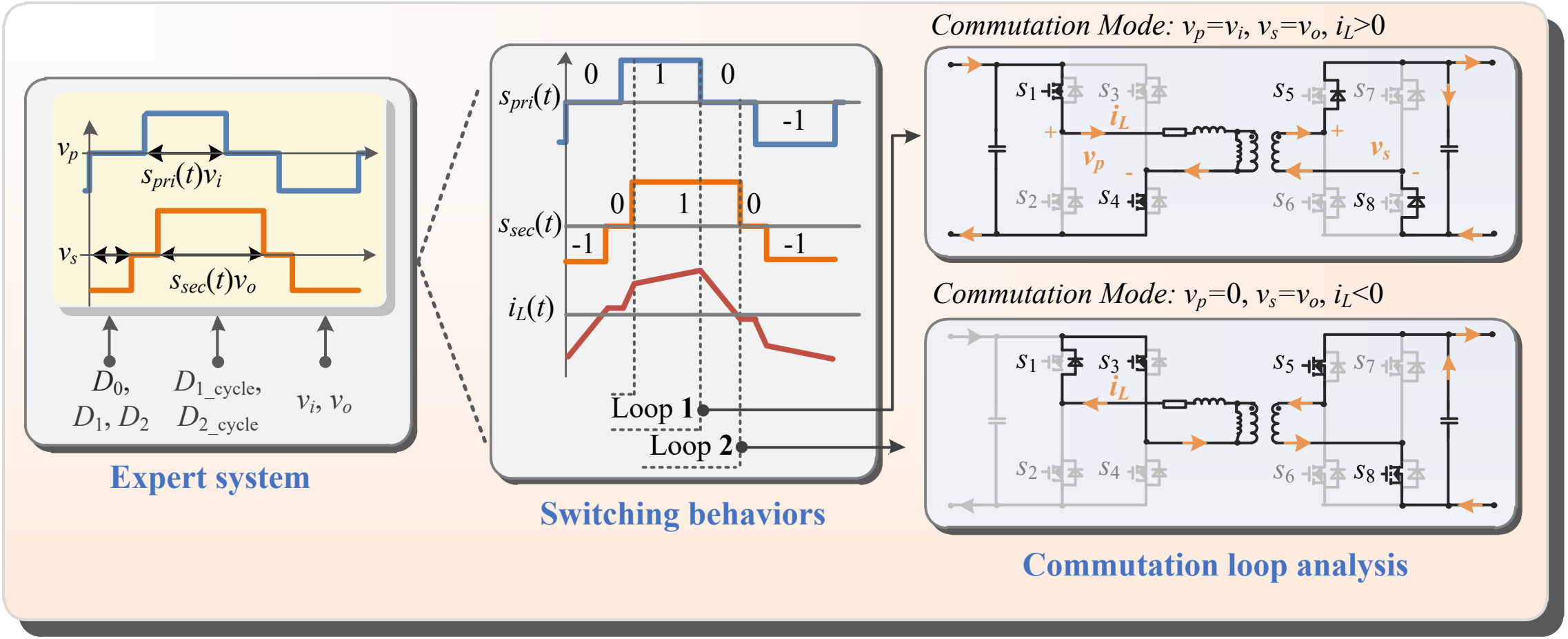




**Inference Breakdown** (Code Demo*)

"DAB-inference and training.ipynb"

https://github.com/XinzeLee/PANN

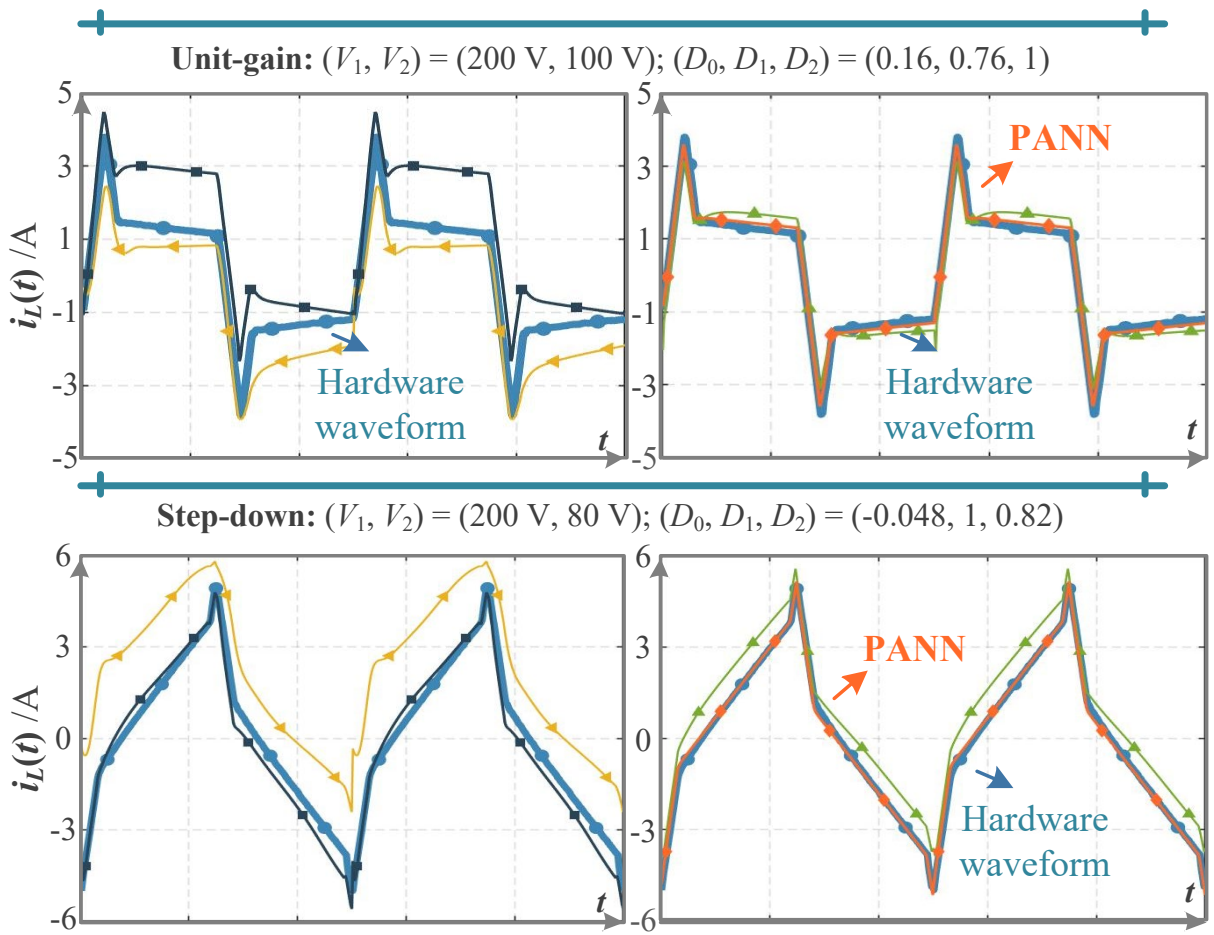
# Circuit Insights Revealed in PANN



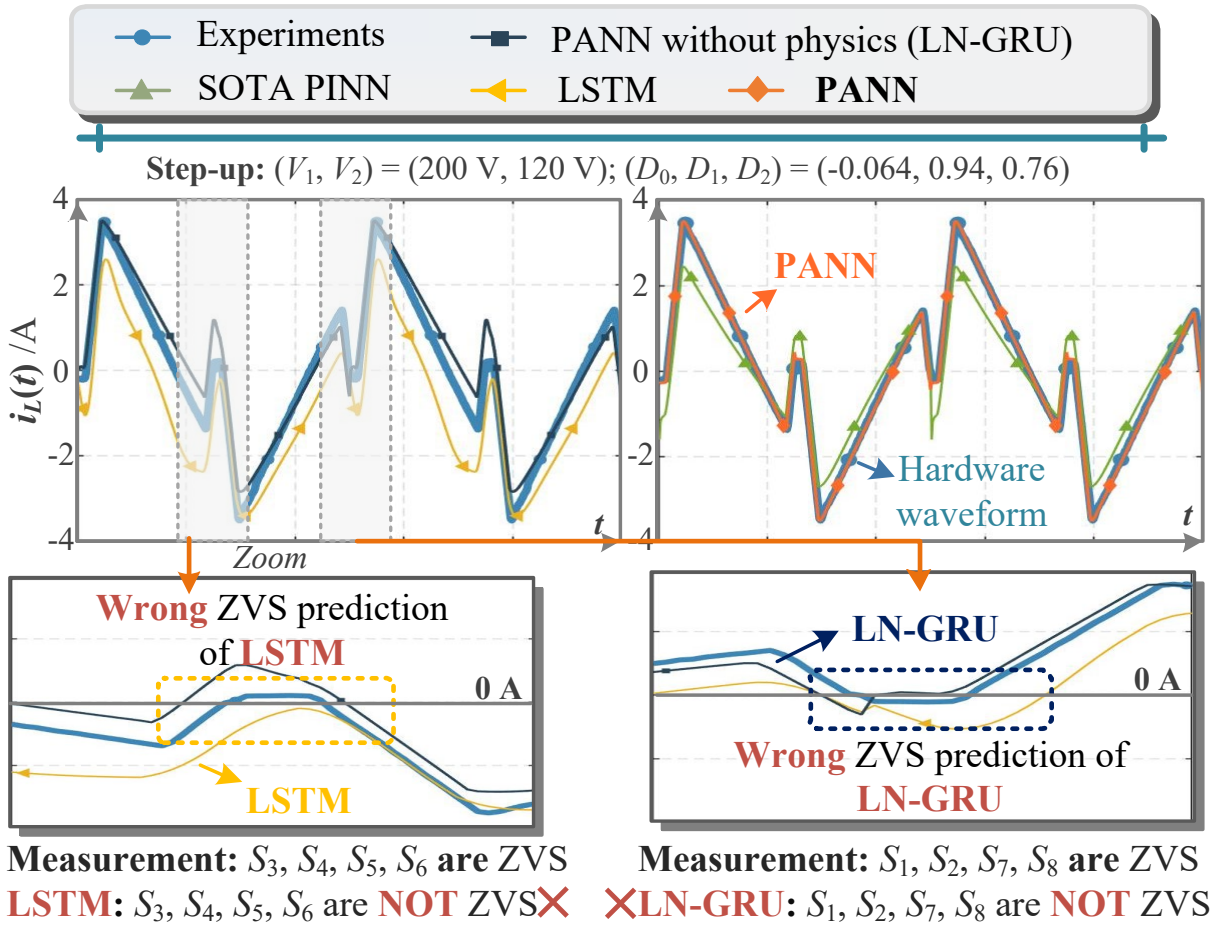

**PANN pushes the definition of AI explainability to a NEW Height:**

**Power Electronics Explainable**

# Exemplary waveforms of PANN for DAB




*Models for converter time-domain modeling*

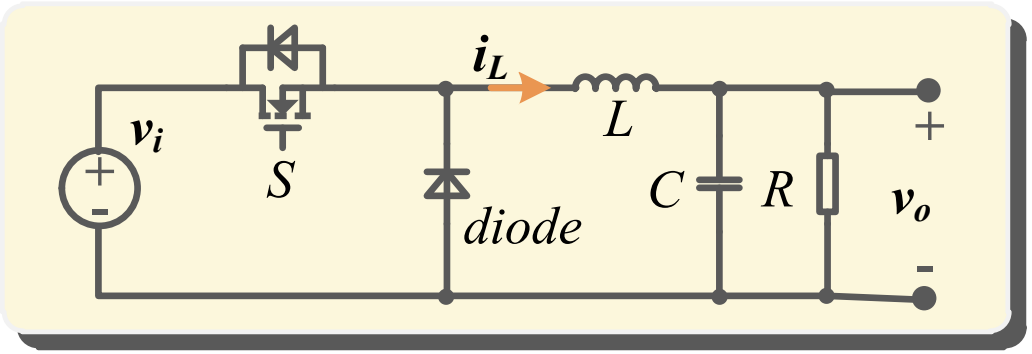



**Measurement:** $S_3$, $S_4$, $S_5$, $S_6$ **are** ZVS
**LSTM:** $S_3$, $S_4$, $S_5$, $S_6$ are **NOT** ZVS✕

**Measurement:** $S_1$, $S_2$, $S_7$, $S_8$ **are** ZVS
✕**LN-GRU:** $S_1$, $S_2$, $S_7$, $S_8$ are **NOT** ZVS

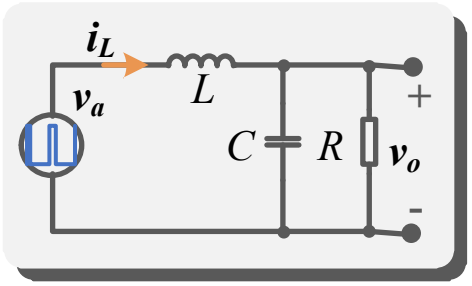
**A DAB Variant (NPC-DAB) under TPS Modulation**

## Case Study 2: PANN for *Buck Converters*




| | Specs |
|---|---|
| States $\boldsymbol{x}(t)$ | $i_L$, $v_o$ |
| Inputs $\boldsymbol{u}(t)$ | $v_a$ |
| Circuit $\boldsymbol{\theta}$ | $R$, $L$, $C$ |

**Equivalent Circuit**



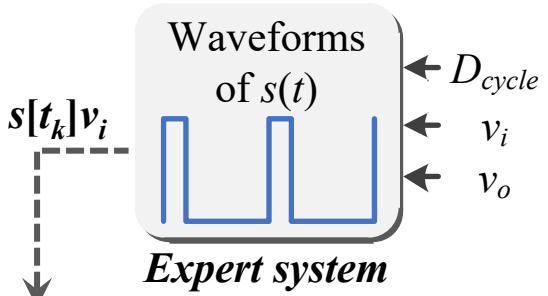

**Continuous**

$$\begin{cases} L\dfrac{d}{dt}i_L(t) = v_a(t) - v_o(t) \\ C\dfrac{d}{dt}v_o(t) = i_L(t) - \dfrac{v_o(t)}{R} \end{cases}$$

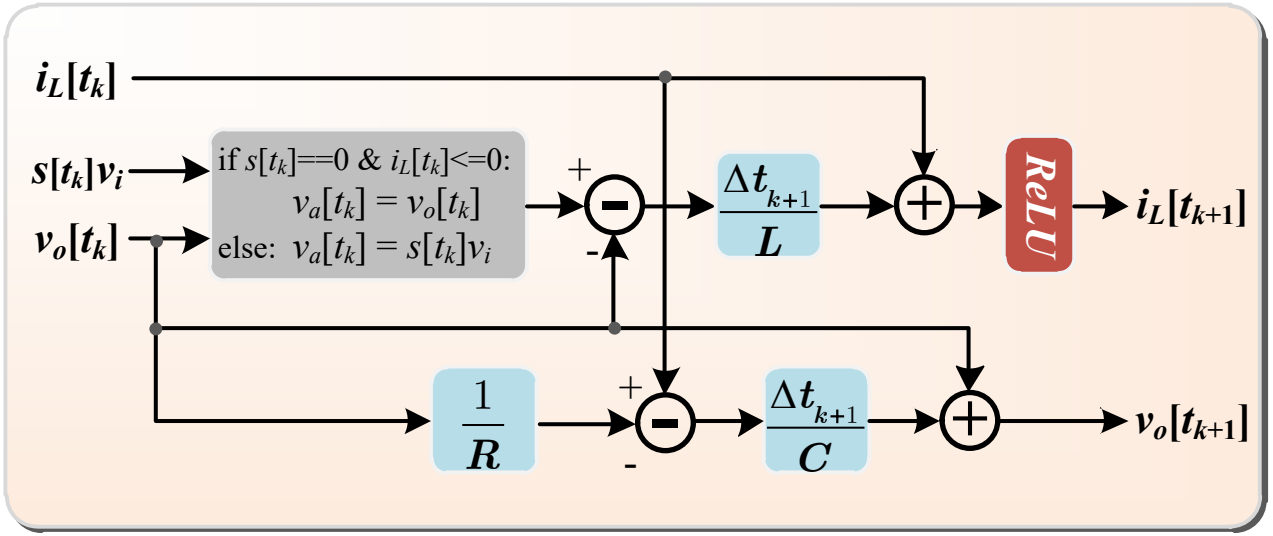
**Discretize via Explict Euler Algorithm**

$$L\left(i_L[t_{k+1}] - i_L[t_k]\right) = \left(v_a[t_k] - v_o[t_k]\right)\Delta t_{k+1}$$

$$C\left(v_o[t_{k+1}] - v_o[t_k]\right) = \left(i_L[t_k] - \frac{v_o[t_k]}{R}\right)\Delta t_{k+1}$$




**Simplify**




**PANN Model for Buck**

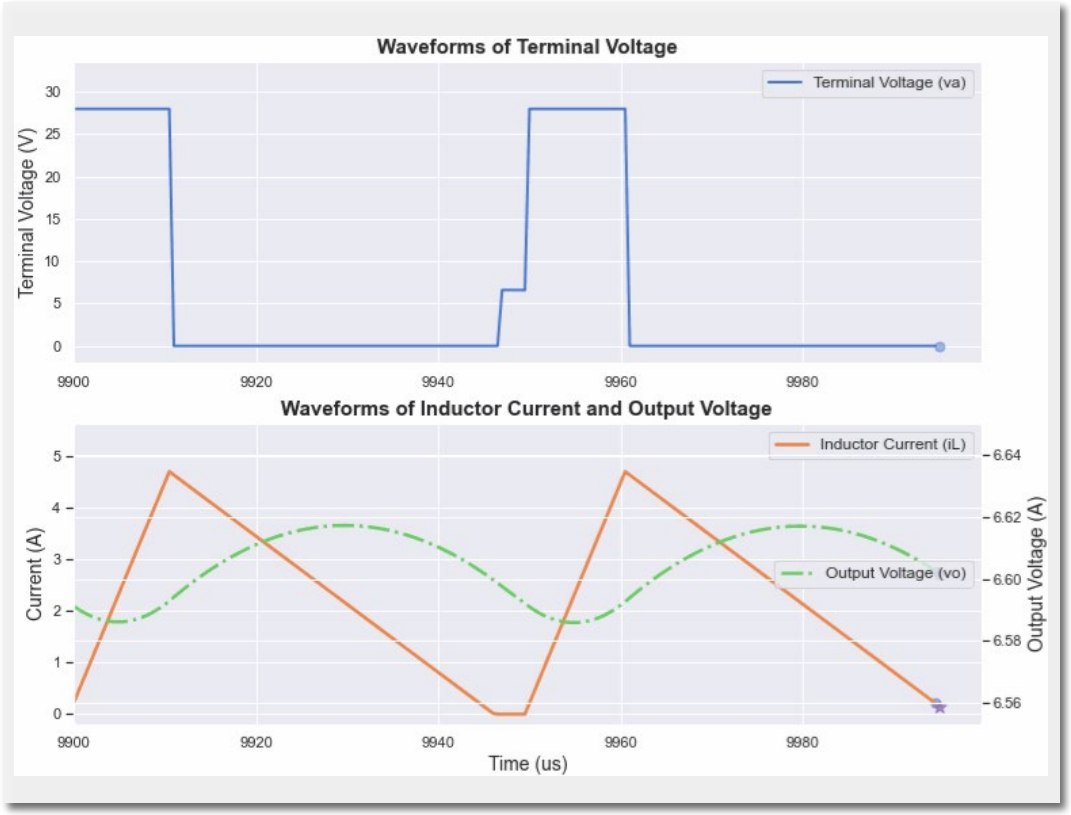
# Case Study 2: PANN for *Buck Converters*

```python
def forward(self, inputs, states):
    # explict Euler method
    # inputs : represent s_pri*Vin, respectively
    # states : represent iL, vo, respectively
    vo = states[:, 1]
    va = inputs[:, 0] # terminal voltage
    idx = (inputs[:, 0] == 0) & (states[:, 0] <= 0) # evaluate dis
    va[idx] = vo[idx]
    iL_next = states[:, 0] + self.dt / self.L * (va - vo) # physi
    iL_next = torch.relu(iL_next) # torch.relu to consider DCM
    vC_next = states[:, 1] + self.dt / self.Co * (states[:, 0]
                                                  - vo / self.Ro)
    return torch.stack((iL_next, vC_next), dim=1)
```

■ Determine DCM

■ Physics of inductor

■ Physics of capacitor




*Embed physics*

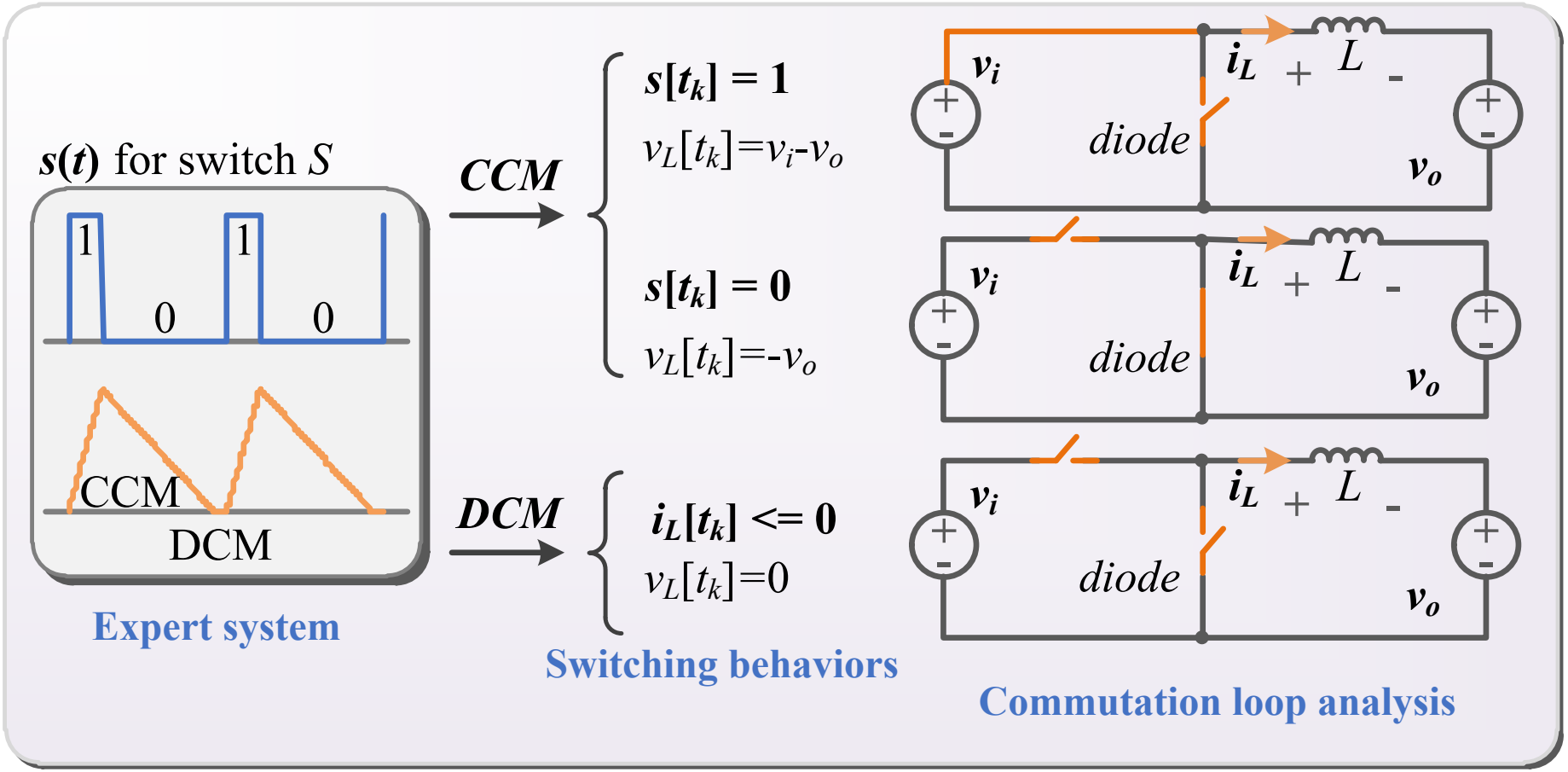
**Module 3. Physics Cell**

**pann_net.py :: class EulerCell_Buck**

**Inference Breakdown (Code Demo*)**

**"Buck-inference.ipynb"**

# Circuit Insights Revealed in PANN




**PANN pushes the definition of AI explainability to a NEW Height:**

**Power Electronics Explainable**

# KEY TAKEAWAYS OF PANN INFERENCE



Takeaways

PANN Inference

## 1. To Customize PANN for ANY Power Converter

Define states, inputs, outputs variables

↓

Equivalent circuit

↓

**Continuous state-space circuit equations**

↓ *Numerical*

**Discretized state-space**

↓

Construct PANN

## 2. Code Structure of PANN

*Generate Input Signals* → **II. Input Blocks**

*Embed Physics* → **III. Physics Cell**

**I. Recurrence Cell**

*Loop through timestamps*

## 3. Main Features of PANN Inference

- **Neural ODE/PDE**
- **Time-domain** modeling
- Present predictions are used to **predict next states**
- **Infer state variables** $\boldsymbol{x}(t)$ based on input variables $\boldsymbol{u}(t)$ from expert systems

## 4. Explain-ability of PANN

- **Physics-crafted** structure: state-space circuit physics
- **Physical compliance**
- Explainability in power electronics: switching behaviors, commutation loops, operating modes

# NEXT GENERATION OF AI FOR POWER ELECTRONICS

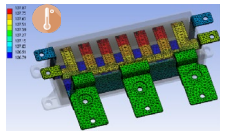
# PANN Training: Adjust Neural Parameters to Minimize Loss

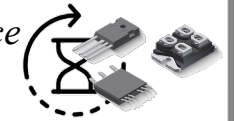


Training is ***Imperative*** to minimize losses

- Manufacturing tolerance

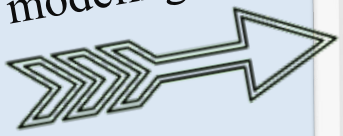
- Ambient variations
  - *Thermal drift*
  - *Operation conditions*
  - *Electromagnetic interference*





- Aging and degradation

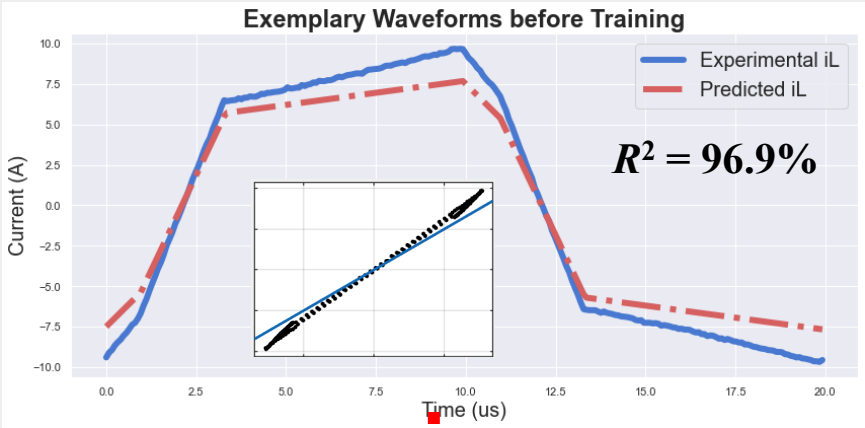
**Deviated** theoretical modeling



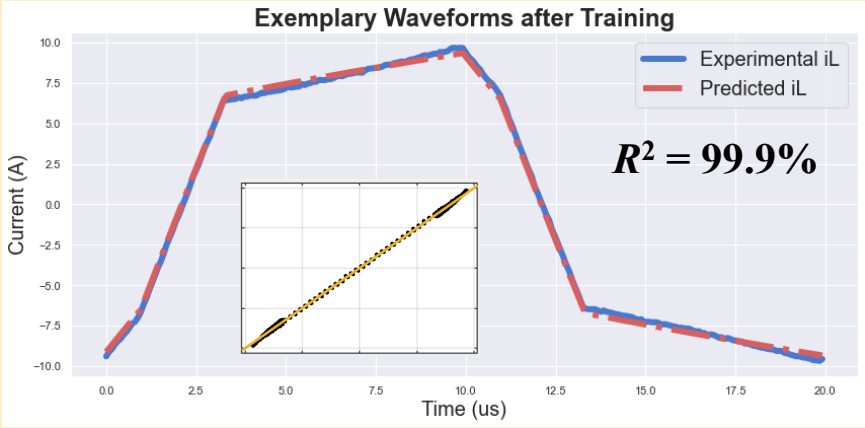
**With Training**

## Before Training

***Neural θ before Training***: $R_L = 10\ \text{m}\Omega$, $L_k = 80\ \mu\text{H}$, $n = 1.1$




***Neural θ after Training***: $R_L = 1.8\ \Omega$, $L_k = 64\ \mu\text{H}$, $n = 1.0$

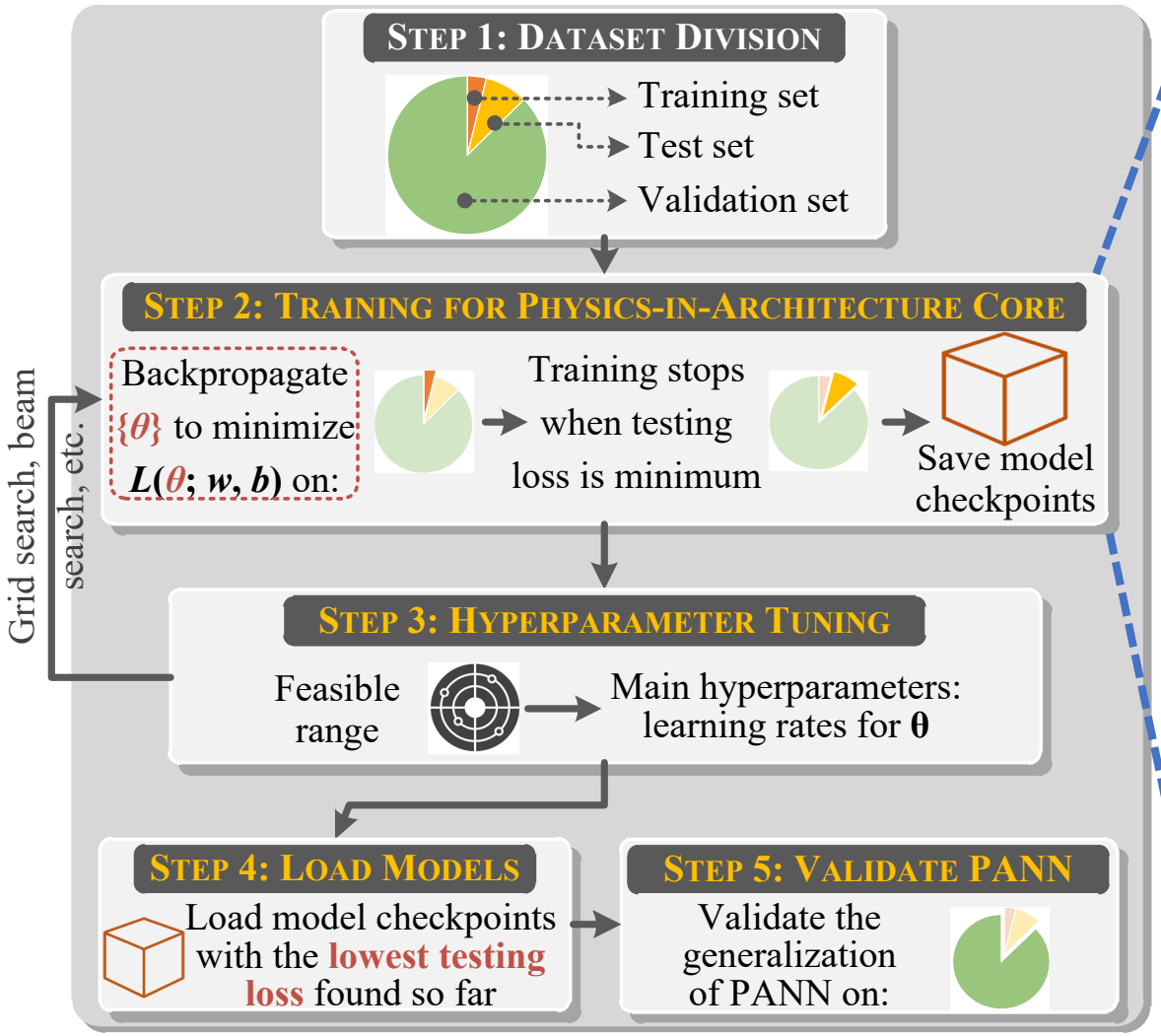



**After Proper Training**

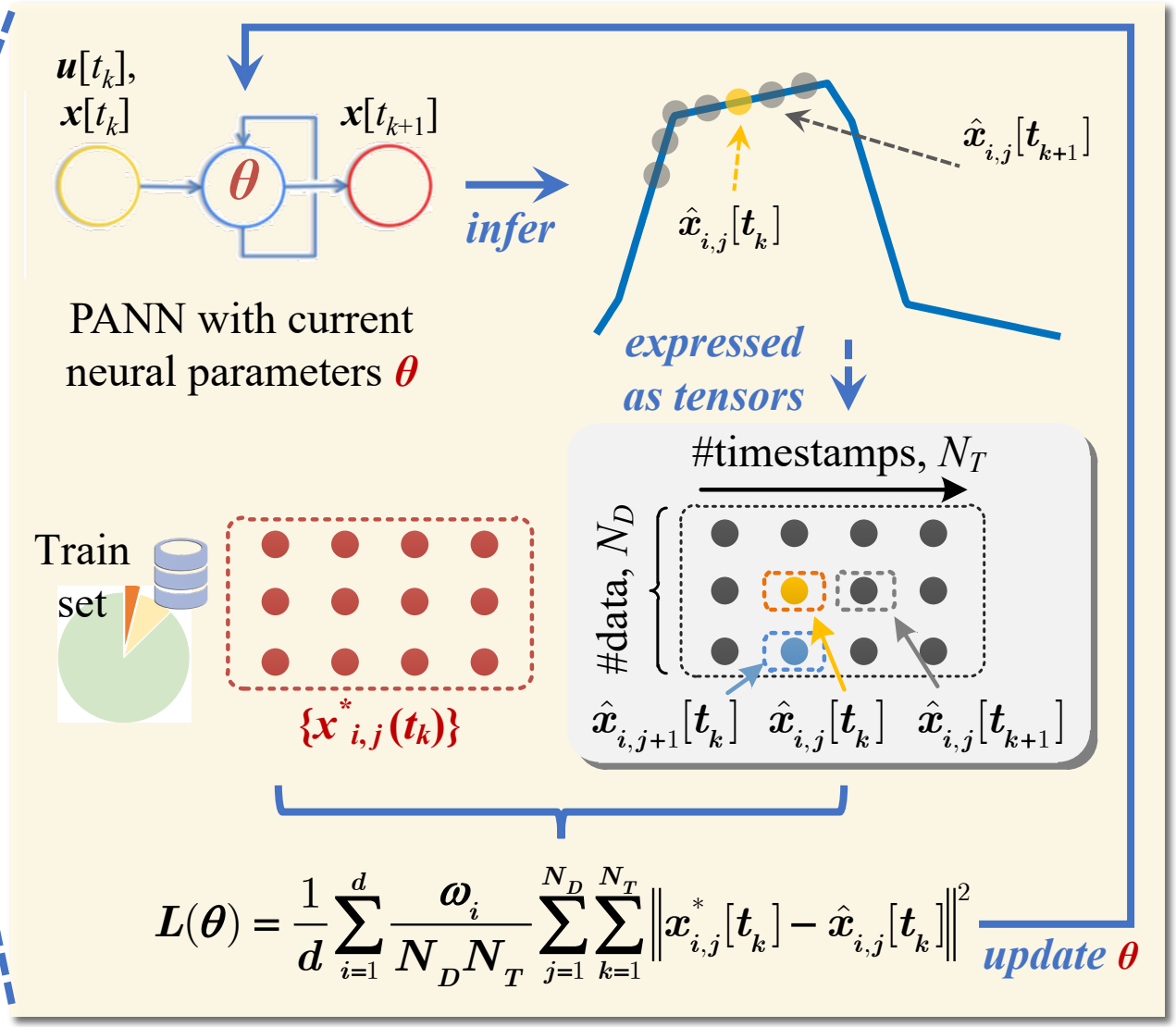
# PANN Training: Flowchart and Breakdown



## PANN training flowchart



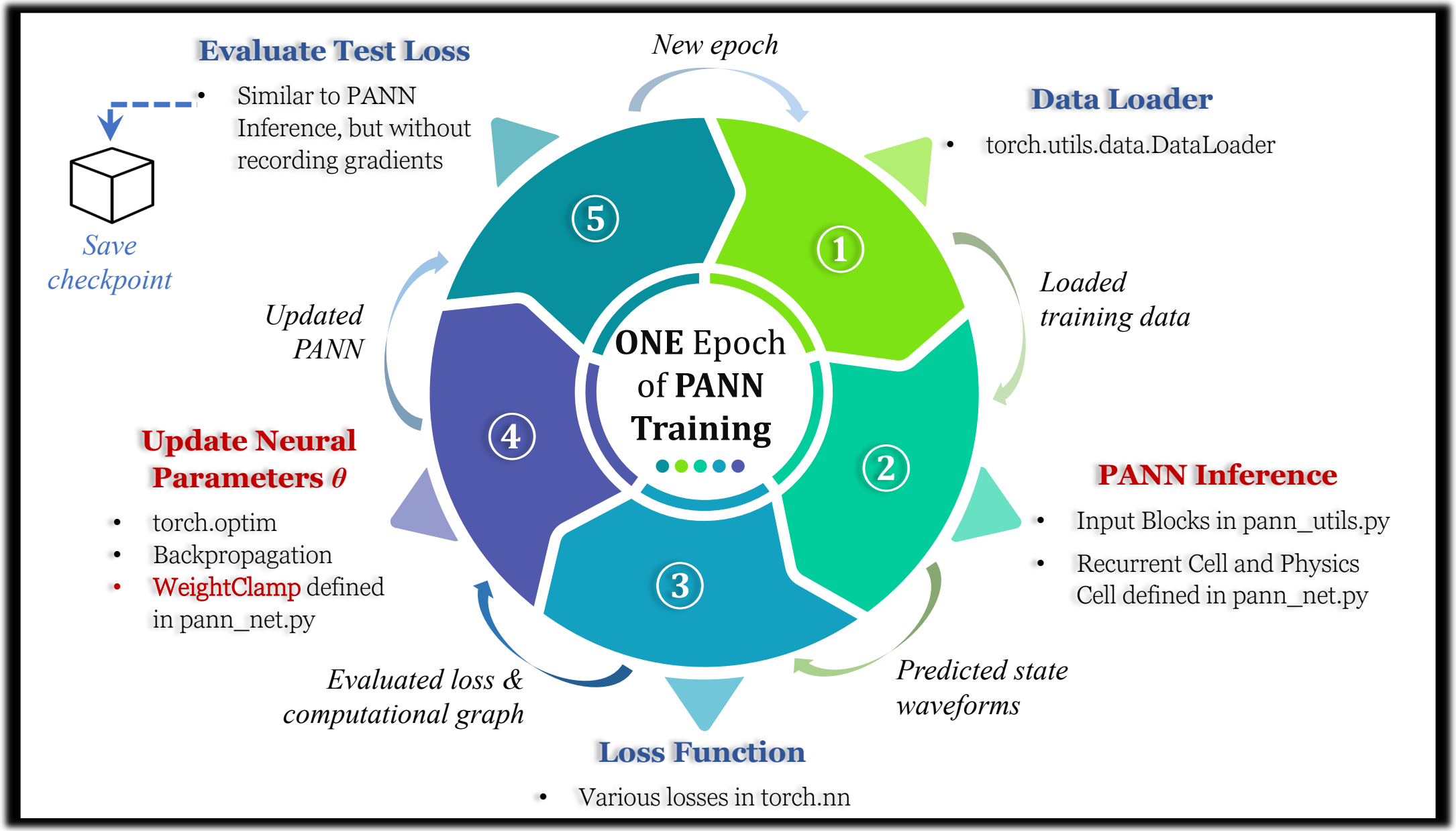

Flowchart of PANN Training and Hyperparameter Tuning




Training of One PANN under One Set of Hyperparams.

# Training Loop of PANN





# PANN Training Code Analysis



```
optimizer_pann = torch.optim.Adam([{"params": [param[1] for param in params], "lr":2e-6},
                                   {"params": [param[1] for param in params2], "lr":1e-1},
                                   {"params": [param[1] for param in params3], "lr":5e-1},])
```

Define an optimizer with different learning rates for neural $\theta$

*Defined in*

```
state0 = torch.zeros(state.shape) # should be zero to avoid learnin
pred = model_pann.forward(input_, state0)
pred, _ = transform(input_[:, -2*Tslen:], pred[:, -2*Tslen:], Vin,
                    Tslen, convert_to_mean=convert_to_mean)

loss_train = loss_pann(pred, target)
optimizer_pann.zero_grad()
loss_train.backward()
optimizer_pann.step()
clamper(model_pann)
```

■ PANN Inference

■ Evaluate Loss

■ Backpropagate Loss

■ Clamp Neural Parameters $\theta$

**Main Loop of PANN Training**

**pann_train.py :: def train**

*Define training process*

*Defined in*

```
clamper = WeightClamp(['cell.Lr', 'cell.RL', 'cell.n'],
                      [(10e-6, 200e-6), (1e-3, 5e0), (0.8, 1.2)])
```

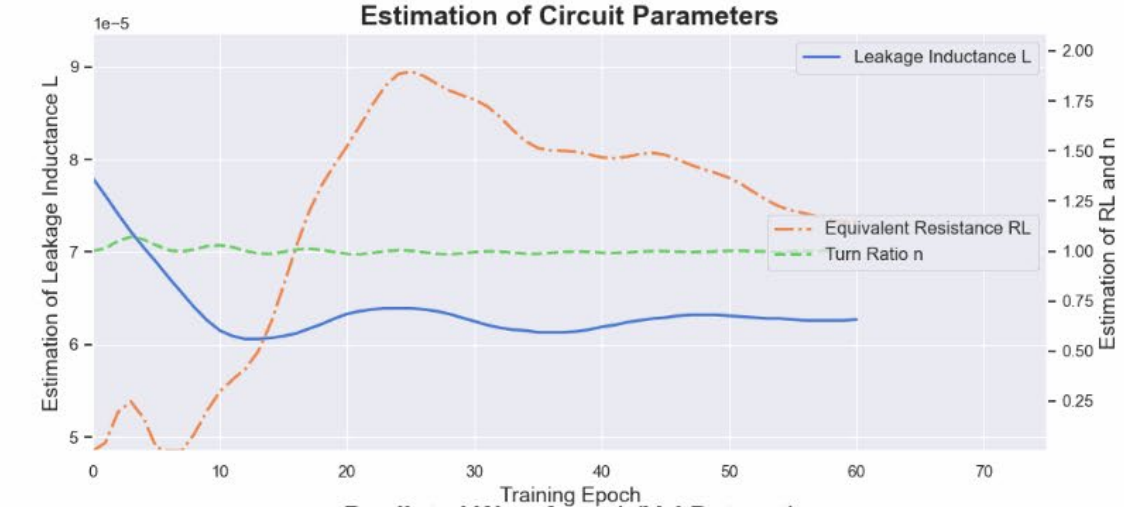
**Define a clamper of class WeightClamp**

**Bound the weight during training to avoid unrealistic neural $\theta$**

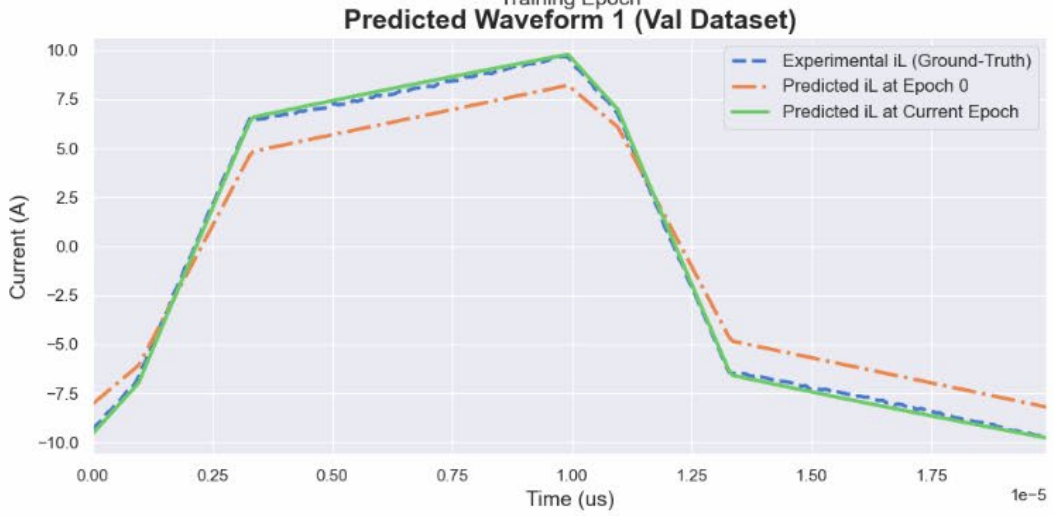
# CASE STUDY: PANN TRAINING OF DAB

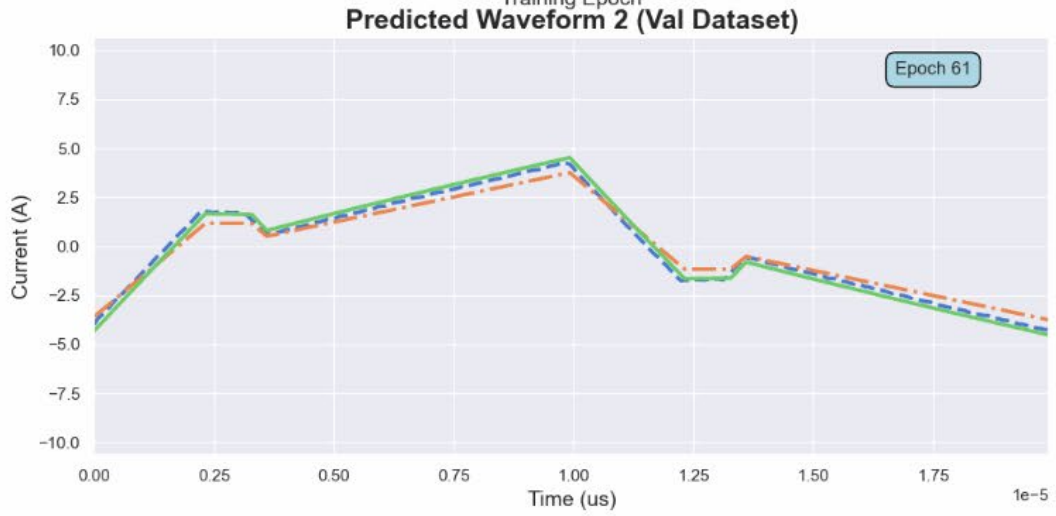


## PANN Training (Code Demo*, "DAB-inference and training.ipynb")





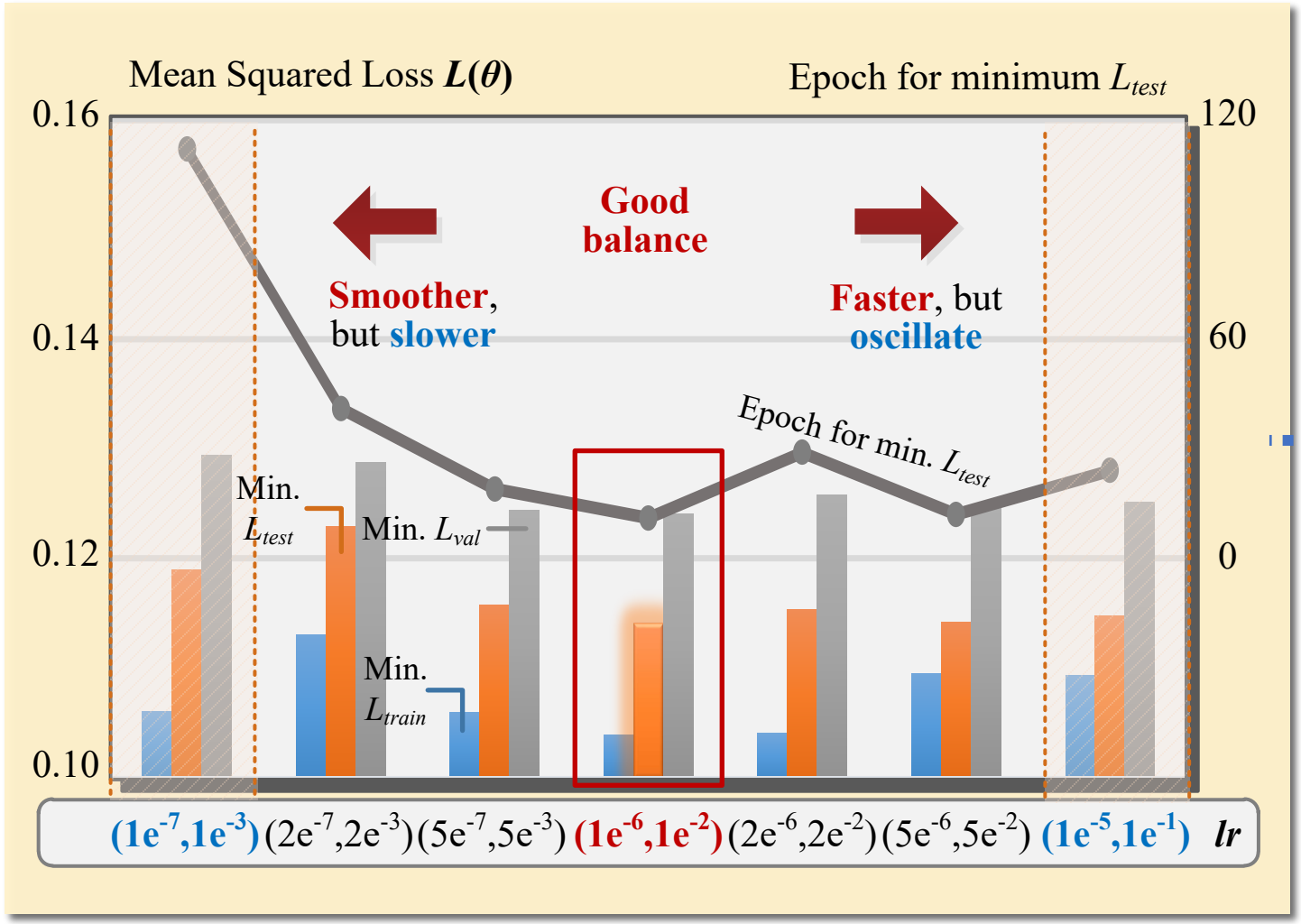


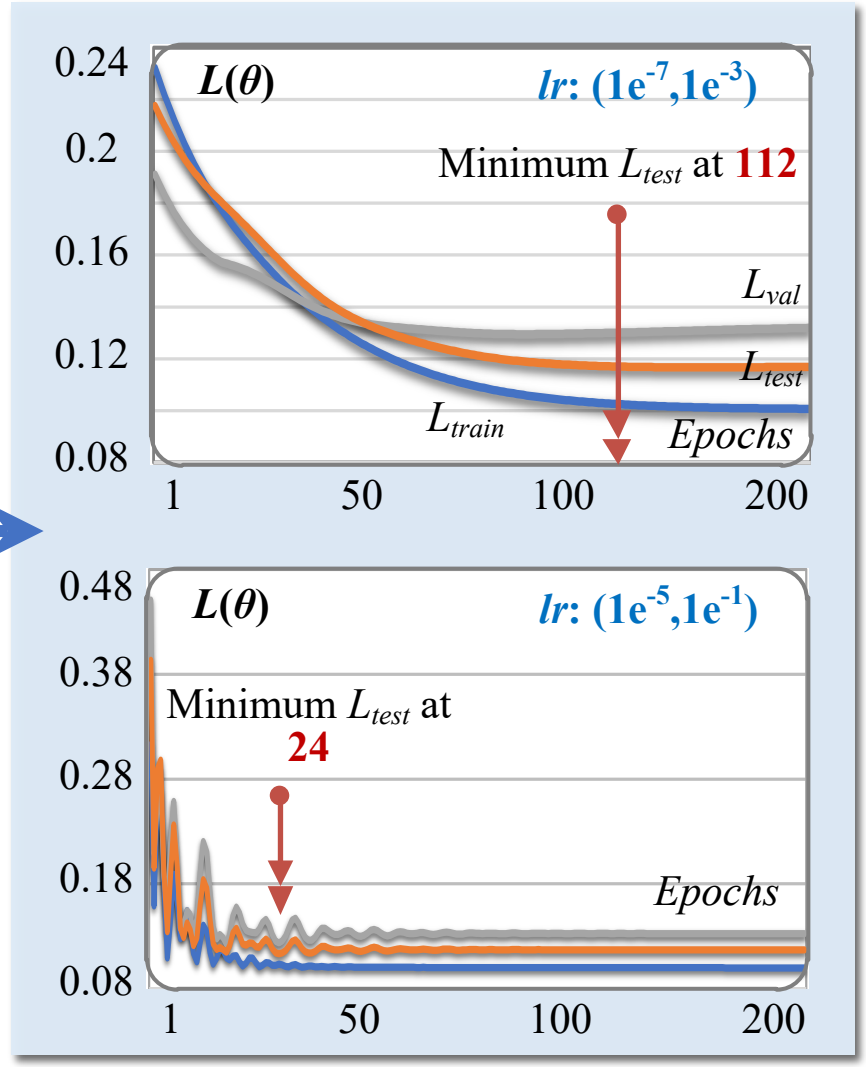
# TUNING OF LEARNING RATES FOR PANN TRAINING



## Tuning of learning rates




Training Performances w.r.t. Learning Rates




Training Curves

# NEXT-GENERATION OF LIGHT AI FOR POWER ELECTRONICS



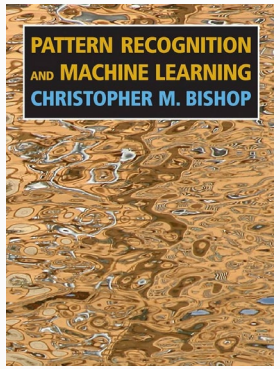
## PANN is Data-Light.

| MAEs / A (mean±std.) | Data Sizes for (Train, Test, Validation) Sets | | |
|---|---|---|---|
| | (10, 90, 900) | (50, 90, 860) | (100, 90, 810) |
| Piecewise | | 0.311±0 | |
| SVR | 2.286±0 | 2.052±0 | 1.991±0 |
| LSTM | $1.560\pm9.5E^{-2}$ | $1.340\pm5.6E^{-2}$ | $1.193\pm9.9E^{-2}$ |
| TCN | $1.371\pm3.6E^{-1}$ | $0.726\pm6.1E^{-2}$ | $0.511\pm7.2E^{-2}$ |
| PL-PINN | $0.310\pm1.3E^{-2}$ | $0.259\pm1.1E^{-2}$ | $0.234\pm7.3E^{-3}$ |
| LN-GRU | $1.257\pm1.2E^{-1}$ | $0.991\pm1.2E^{-1}$ | $0.527\pm8.6E^{-2}$ |
| PANN | $0.160\pm4.1E^{-3}$ | $0.145\pm3.6E^{-3}$ | $0.132\pm3.3E^{-3}$ |

Popular ML: Thousands, Millions, and MORE

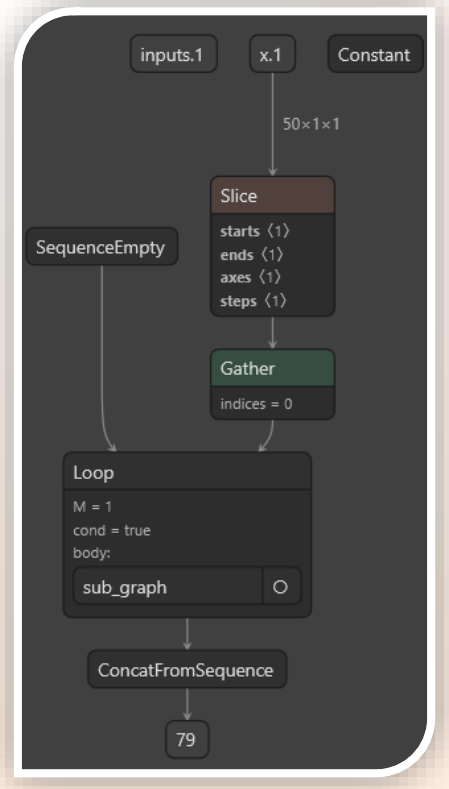
***PE Physics*** ↓ *Embed Data Invariant*

PANN: a **handful** of data, reducing by **3 orders of magnitudes**




***Data Size*** ≥ ***#Params.***

## PANN is Lightweight.

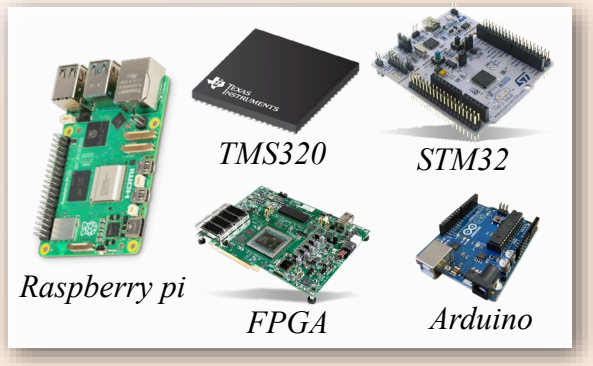



**One-layer Recurrent Net**

**1.72 kB** of a PANN for DAB


1762 strings, Approx. 160 words

***Deployable on Edge***




**PANN pioneers the next-generation of LIGHT AI for power electronics in:**

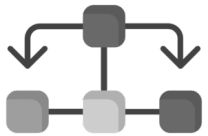
1. Data: **countable on ONE HAND**, 2. Model size: **deployable on EDGE DEVICE.**

# KEY TAKEAWAYS OF PANN TRAINING

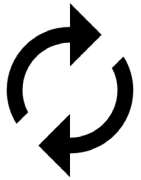


Dataset division
↓
Training of PANN
↓
Hyperparameter tuning
↓
Load model checkpoint
↓
Validate PANN

## 1. Training and Hyper-parameter Tuning of PANN



## 2. Training Loop of PANN



Load training data
↓
PANN Inference
↓
Evaluate training loss
↓
**Update neural $\theta$: backpropagate & clamp**
↓
Evaluate test loss

***Takeaways***

***PANN Training***

- **Learning rates *lr*** of PANN should be carefully tuned
- Low *lr*: **slow** convergence
- High *lr*: large **oscillation**
- **Rule of thumb**: *lr* to be **1%** of parameter values

## 3. Hyper-Parameter Tuning of PANN

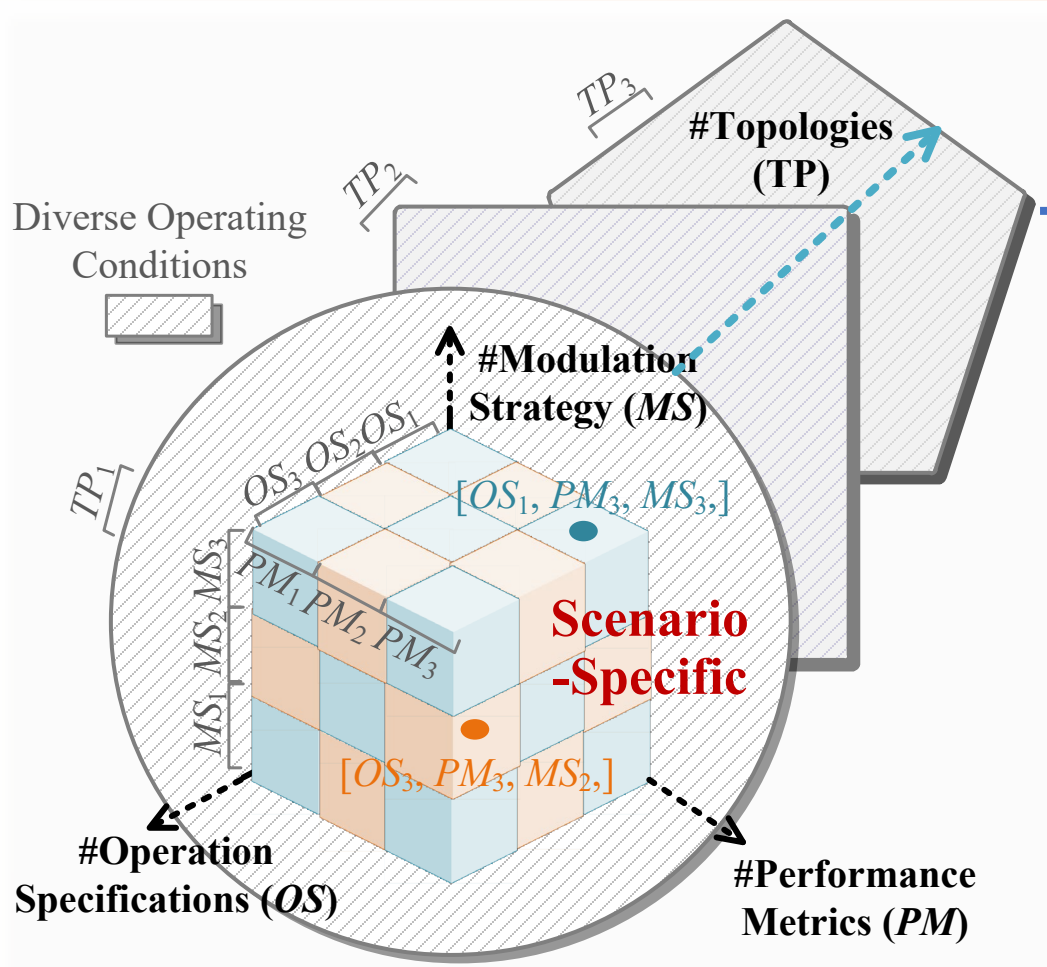
## 4. PANN is a "Light" AI Model

- **Data-light:** training data required for PANN is **countable on ONE HAND**
- **Lightweight:** few kBytes, deployable on resource-constrained **EDGE** devices

# NEXT GENERATION OF AI FOR POWER ELECTRONICS

# PANN – Next Level of Flexibility: Training-Free

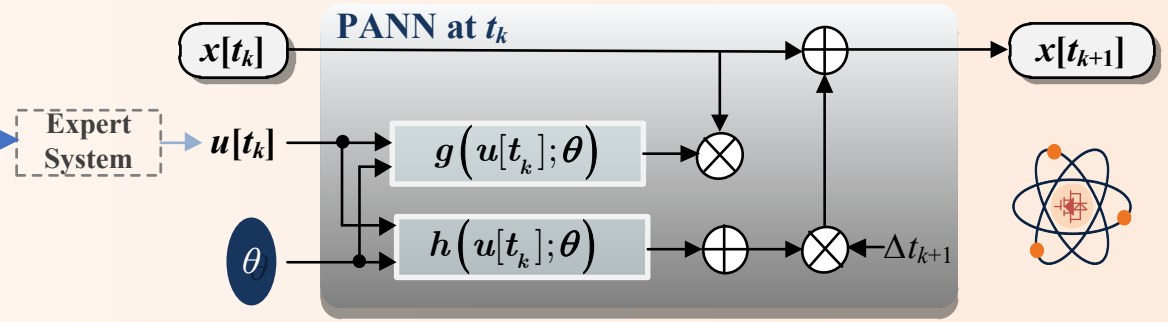





**Total Number of Conventional ML Models required:**

$$MS \times PM \times OS \times TP$$

## ONE PANN for Diverse Conditions and Topologies




**Training-FREE**

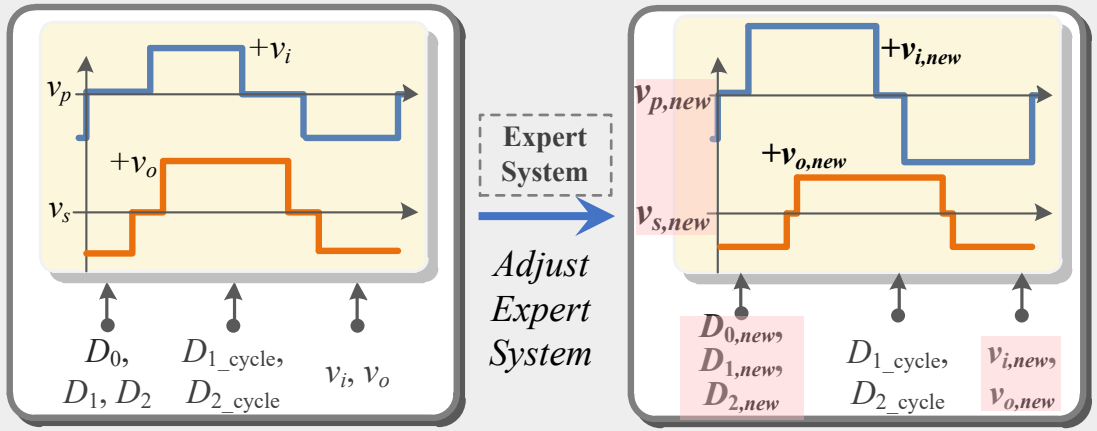

**Four Angles of PANN's Flexibility**

# PANN – Flexibility across Conditions

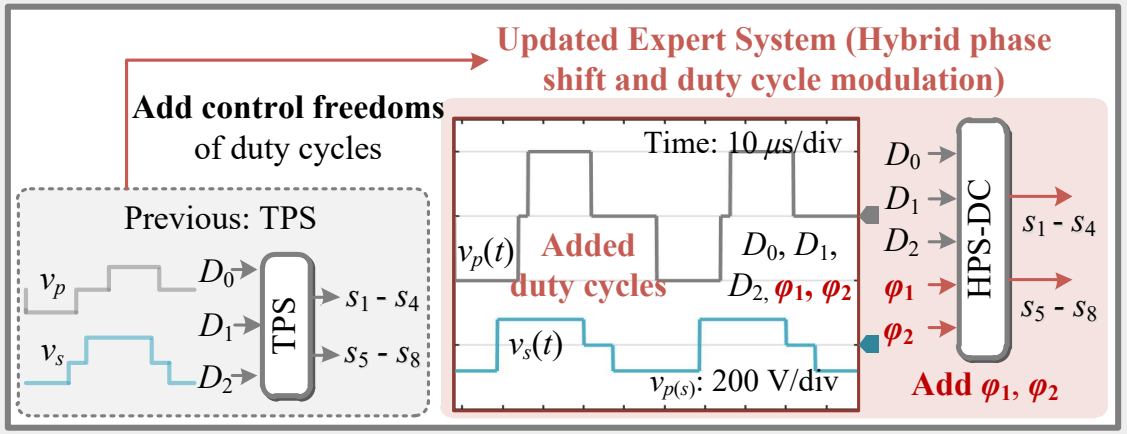


## Flexibility across Conditions - DAB

### *Flexibility* 1: *Out-of-Domain Operating Conditions*




### *Flexibility* 2: *Diverse Modulation Strategies*




## Code Implementation

### Adjust the Knowledge in Input Blocks

```
# Example 3: Vin = 300 V, Vo = 220 V
D0, D1, D2 = 0.16, 0.78, 0.88
Vin, Vref = 300, 220
Tsim = 100*Ts # simulation time, considering 100 times of sw
# generate voltage waveforms (vp and vs) given one set of mo
vp3, vs3 = create_vpvs(D0, D1, D2, Vin, Vref, Tslen, dt,
                       Tsim, Ts, D1_cycle=0.5, D2_cycle=0.5)
inputs3 = np.hstack([vp3[:, None], vs3[:, None]])[None]
# convert to torch FloatTensor
inputs3 = torch.FloatTensor(inputs3)
```

- Adjust $v_i$ and $v_o$
- Adjust expert system

```
# Example 1: Vin = 200 V, Vo = 140 V, (D0, D1, D2) = (0.1, 0.9
D0, D1, D2 = 0.04, 0.84, 0.64
D1_cycle, D2_cycle = 0.5, 0.32
Vin, Vref = 200, 140
Tsim = 100*Ts # simulation time, considering 100 times of swit
# generate voltage waveforms (vp and vs) given one set of modu
vp, vs = create_vpvs(D0, D1, D2, Vin, Vref, Tslen, dt, Tsim,
                     Ts, D1_cycle=D1_cycle, D2_cycle=D2_cycle)
inputs = np.hstack([vp[:, None], vs[:, None]])[None]
# convert to torch FloatTensor
inputs = torch.FloatTensor(inputs)
```

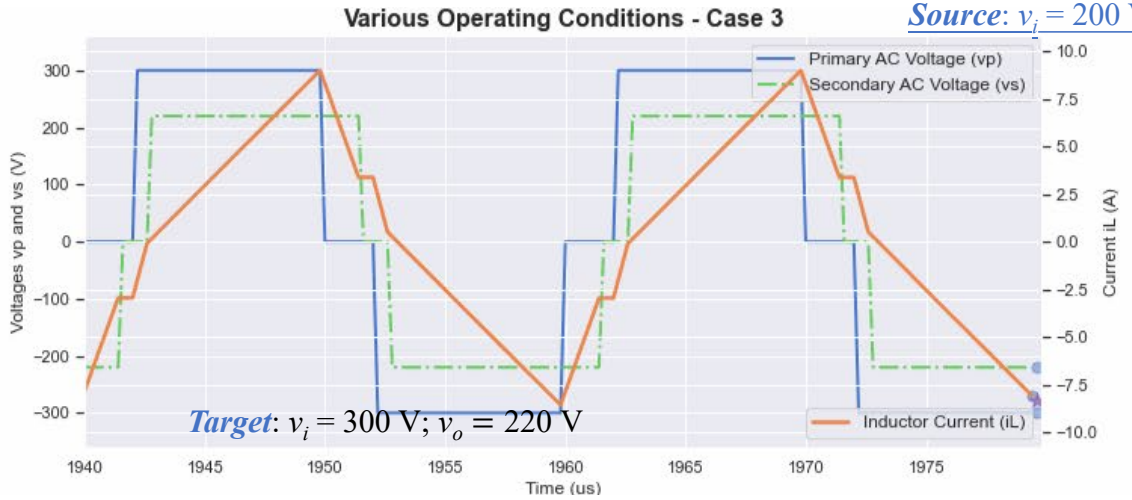
- Adjust duty cycles
- Adjust expert system

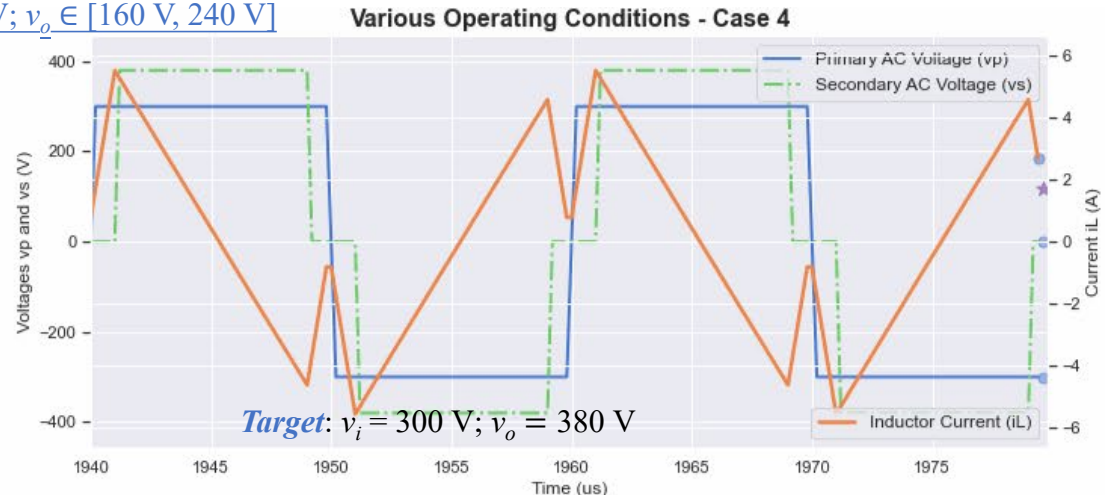
# CASE STUDY: PANN'S FLEXIBILITY ACROSS CONDITIONS



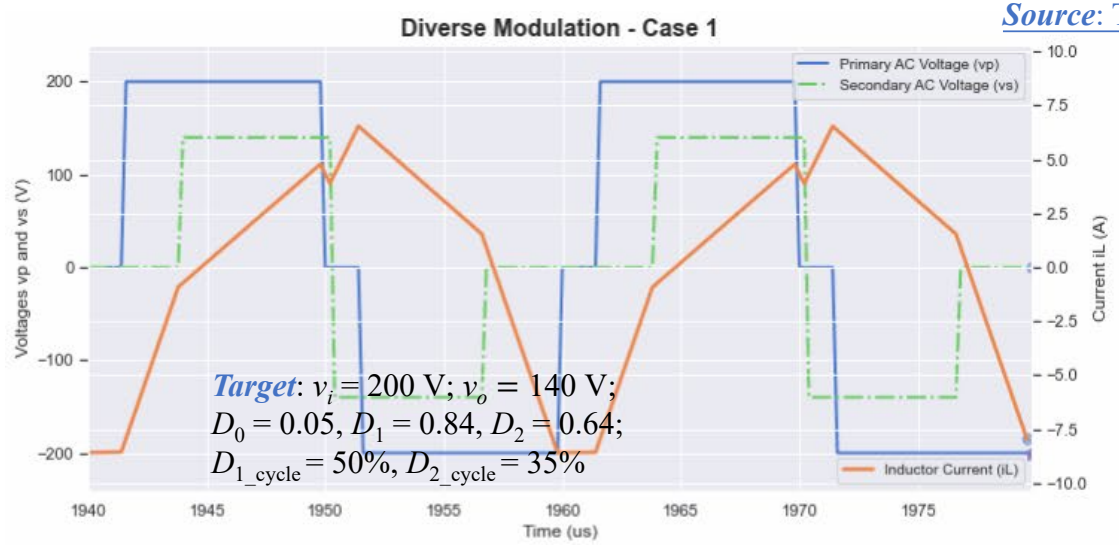
## PANN's Flexibility across Conditions (Code Demo*, "DAB-Operational Transfer.ipynb")

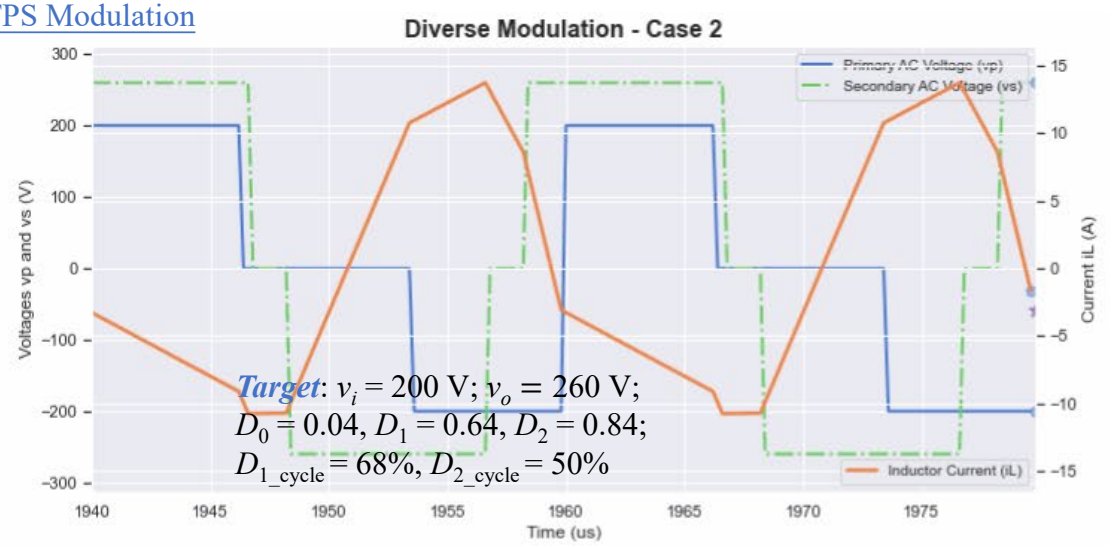
*Source*: $v_i$ = 200 V; $v_o \in$ [160 V, 240 V]




*Source*: TPS Modulation



# PANN – Flexibility across Performance Metrics

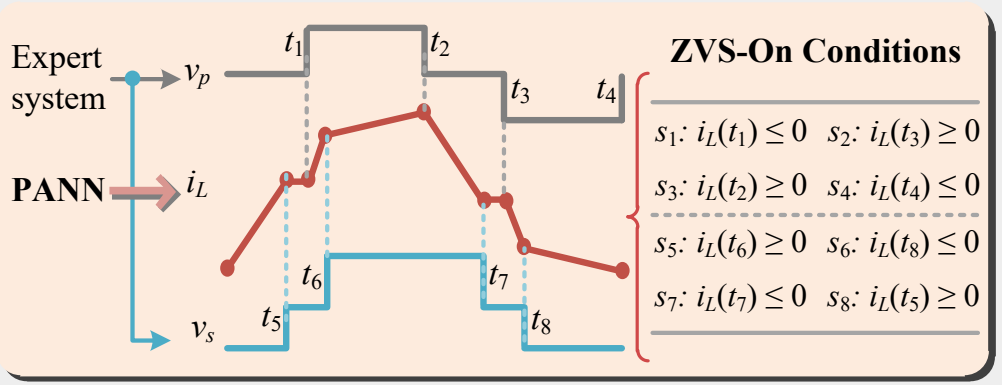


## Flexibility across Performance Metrics - DAB

***Flexibility* 3: *Various Performance Metrics***

Expert System → PANN

↓ *Switching behaviors* $v_p$, $v_s$, ⋯ ↓ *Circuit behaviors* $i_L$, ⋯




**ZVS-On Conditions**

$s_1$: $i_L(t_1) \le 0$ $s_2$: $i_L(t_3) \ge 0$

$s_3$: $i_L(t_2) \ge 0$ $s_4$: $i_L(t_4) \le 0$

$s_5$: $i_L(t_6) \ge 0$ $s_6$: $i_L(t_8) \le 0$

$s_7$: $i_L(t_7) \le 0$ $s_8$: $i_L(t_5) \ge 0$

➢ *ZVS turning-on*

$$v_{L,rms} = \sqrt{\frac{1}{N_T}\sum_{k=1}^{N_T}\left(v_L[t_k]\right)^2} \qquad i_{L,rms} = \sqrt{\frac{1}{N_T}\sum_{k=1}^{N_T}\left(i_L[t_k]\right)^2}$$

$$Q = v_{L,rms} \cdot i_{L,rms} \qquad i_{pp} = i_{L,max} - i_{L,min}$$

↓ ➢ *Reactive power* ↓ ➢ *Current stress*

➢ *Conduction loss*

➢ *Transferred power*

➢ *Circulating current, etc.*

**One-Stop Metric Adapter for Diverse Performances**

### Code Implementation

star_metric_adapter.py

```
def soft_switching(iL, vp, vs, Vin, Vo, threshold=0.):
    # Evaluate soft-switching performances of conventional DAB co
    indices = locate(vp, Vin)
    indices2 = locate(vs, Vo)
    ZVS = np.zeros(8, )
    ZCS = np.zeros(8, )
    ##### ZVS of Primary Side Switches
    ZVS[0] = (iL[indices[1]]<=threshold).astype(int) # ZVS: S1
    ZVS[1] = (iL[indices[3]]>=-threshold).astype(int) # ZVS: S2
    ZVS[2] = (iL[indices[2]]>=-threshold).astype(int) # ZVS: S3
    ZVS[3] = (iL[indices[0]]<=threshold).astype(int) # ZVS: S4
    ##### ZVS of Secondary Side Switches
    ZVS[4] = (iL[indices2[1]]>=-threshold).astype(int) # ZVS: S5
    ZVS[5] = (iL[indices2[3]]<=threshold).astype(int) # ZVS: S6
    ZVS[6] = (iL[indices2[2]]<=threshold).astype(int) # ZVS: S7
    ZVS[7] = (iL[indices2[0]]>=-threshold).astype(int) # ZVS: S8
    return ZVS, indices, indices2
```

■ **Locate switching timestamp**

■ **Primary ZVS**

■ **Secondary ZVS**

```
def current_stress(iL, metric="ipp"):
    assert metric in ["ipp", "irms"]
    if metric == "ipp":
        return max(iL)-min(iL)
    elif metric == "irms":
        return np.sqrt((iL**2).mean())
```

■ **Peak-peak current**

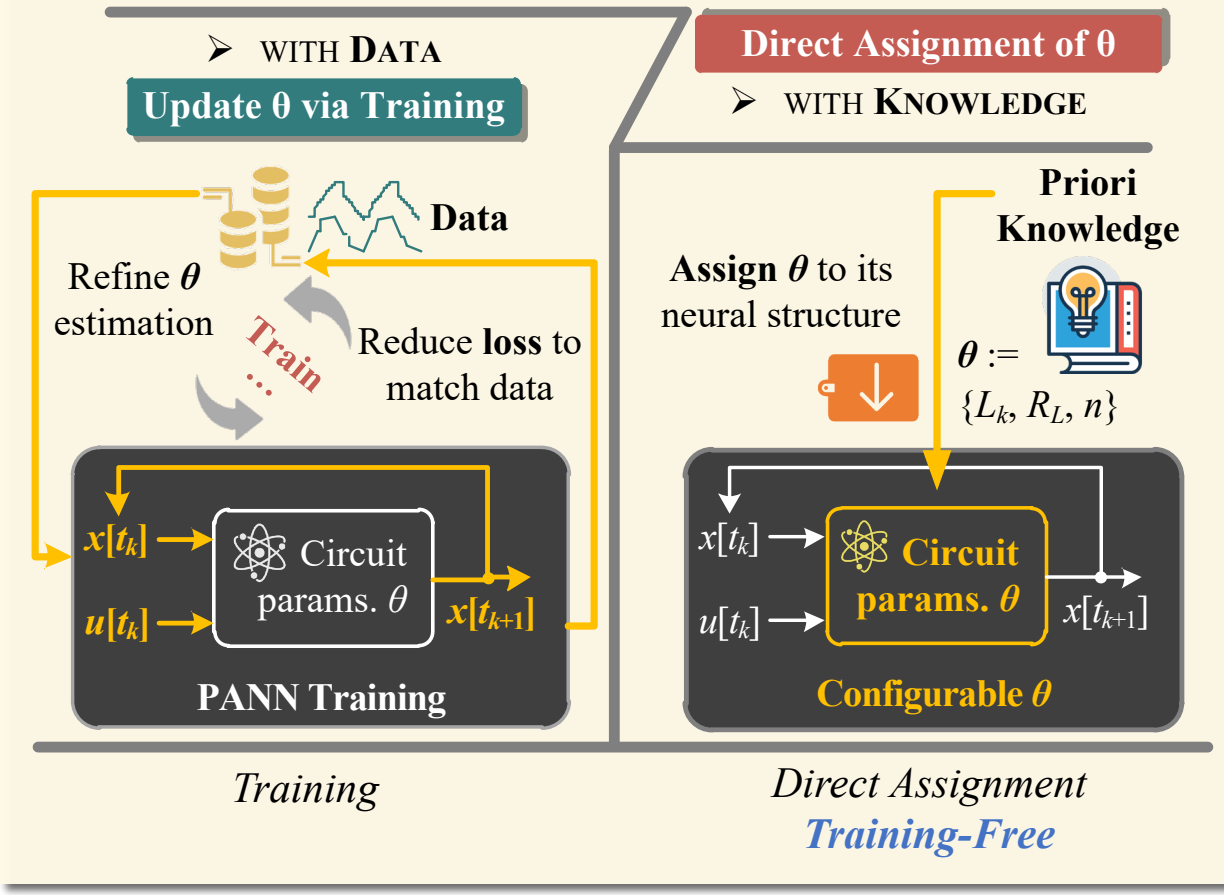
■ **RMS current**

# PANN – Flexibility across Circuit Parameters



## Flexibility across Circuit Parameters - DAB

***Flexibility* 4-1: *Different Circuit Parameters***




**Flexibility across Different Circuit Parameters** by Training or Direct $\theta$ Assignment

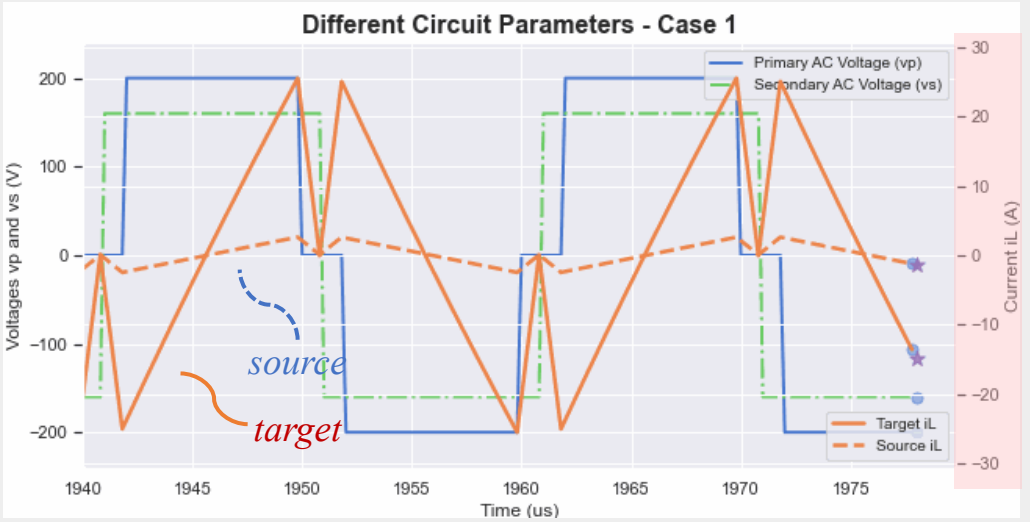
### Code Implementation

**Adjust Neural θ in Physics Cell Module**

```
# Example: Directly assign new circuit parameters to neural θ
pann_dab.cell.n.data = torch.FloatTensor([1.0])
pann_dab.cell.RL.data = torch.FloatTensor([120e-3])
pann_dab.cell.Lr.data = torch.FloatTensor([6.3e-6])
```

***source***: $R_L$ = 120 mΩ; $L_k$ = 63 μH; $n$ = 1

***target***: $R_L$ = 120 mΩ; $L_k$ = 6.3 μH; $n$ = 1




**Circuit Parameter Transfer** (Code Demo*)

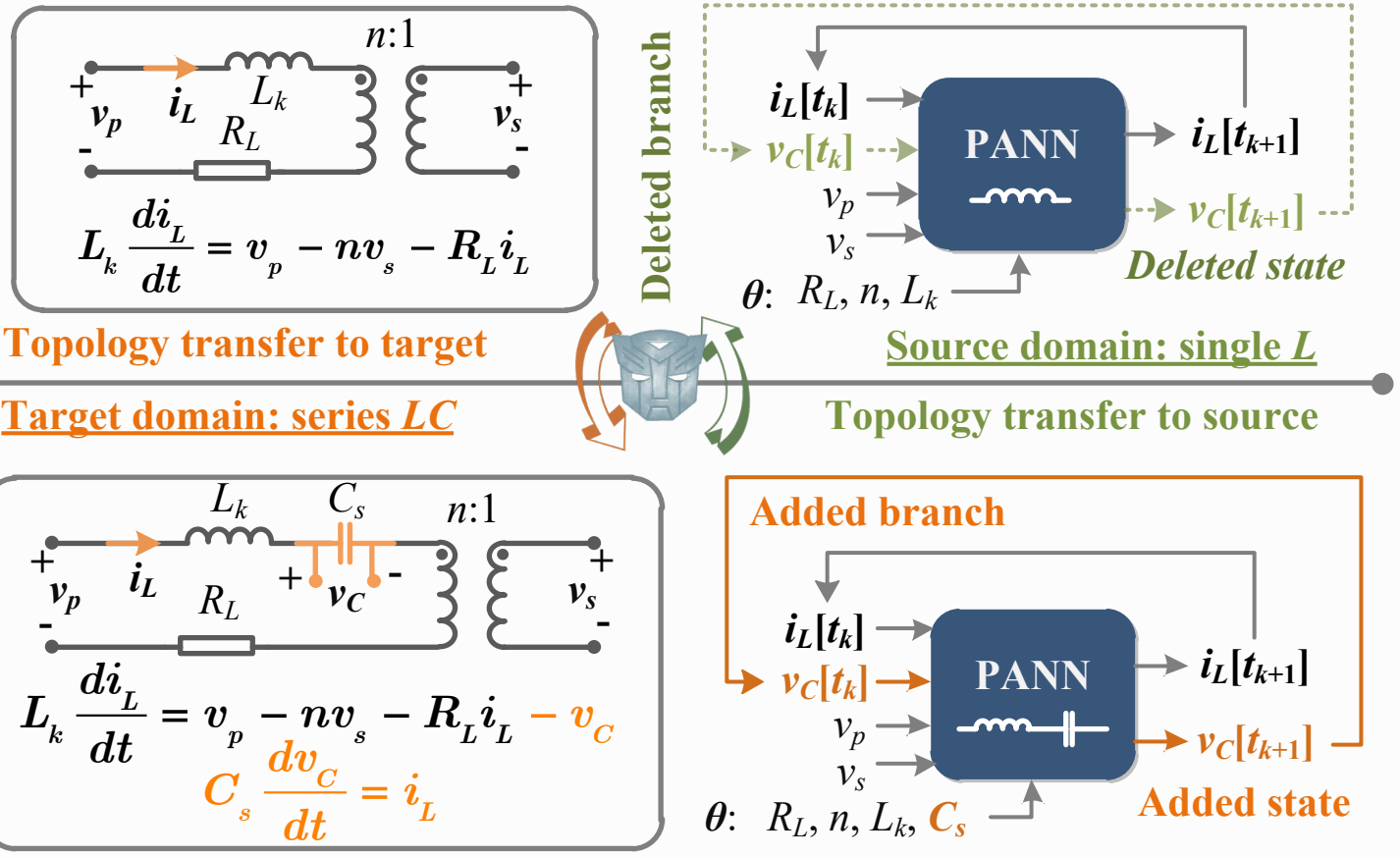
"DAB-Operational Transfer.ipynb"

# PANN – Flexibility across Topologies



## Flexibility across Topologies - DAB Family

***Flexibility* 4-2: *Different Topological Variants***




Discretized by Implicit Euler:

$$i_L[t_{k+1}] = \frac{L_k i_L[t_k] + \left(v_p[t_k] - nv_s[t_k]\right)\Delta t_{k+1}}{L_k + R_L \Delta t_{k+1}}$$

*Delete Branch* ⇅ *Add Branch*

Discretized by Implicit Euler:

$v_C$ is added into the voltage loop

$$\begin{cases} i_L[t_{k+1}] = \dfrac{L_k i_L[t_k] + \left(v_p[t_{k+1}] - nv_s[t_{k+1}] - v_C[t_{k+1}]\right)\Delta t_{k+1}}{L_k + R_L \Delta t_{k+1}} \\ C_s v_C[t_{k+1}] = C_s v_C[t_k] + i_L[t_{k+1}]\Delta t_{k+1} \end{cases}$$

Added physics of $v_C$ of resonant capacitor

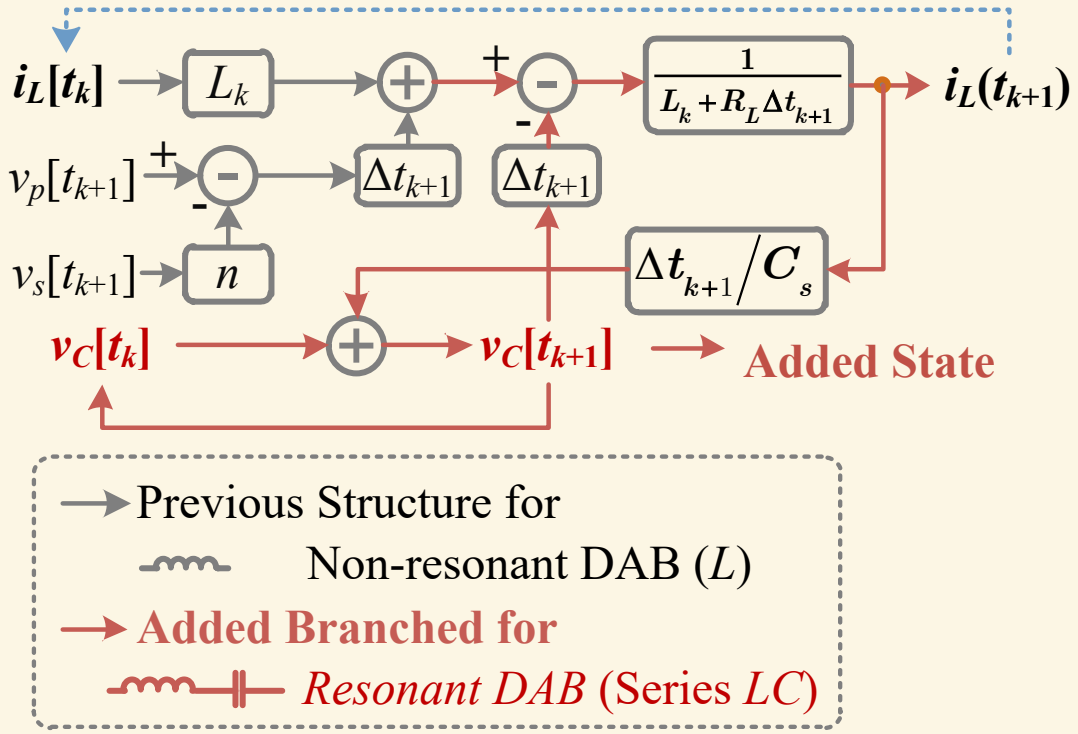
**Topology Transfer** between Non-Resonant and Resonant DAB

# PANN – Flexibility across Topologies



## Topology Transfer between Non-Resonant and Resonant DAB

*Schematic Diagram of PANN for Topology Transfer*




**PANN's Neural Link Transformation**

### Code Implementation

DAB-Topology Transfer.ipynb

■ **Non-resonant DAB with Single *L***

```
iL_next = (states[:, 0]+(inputs[:, 0]-
            inputs[:, 1]*self.n)*self.dt/self.Lr)\/
           (1+self.RL*self.dt/self.Lr) # physics of iL
```

- Add $v_C$ into the voltage loop
- Add physics of $v_C$

```
# physics of resonant inductor iL
iL_next = (states[:, 0]+(inputs[:, 0]-
            inputs[:, 1]*self.n
           -states[:, 1])*self.dt/self.Lr)/\
          (1+self.RL*self.dt/self.Lr+self.dt**2/self.Lr/self.Cs)
# physics of resonant capacitor vC
vC_next = iL_next/self.Cs*self.dt+states[:, 1]
```

■ Add $v_c$ into voltage loop

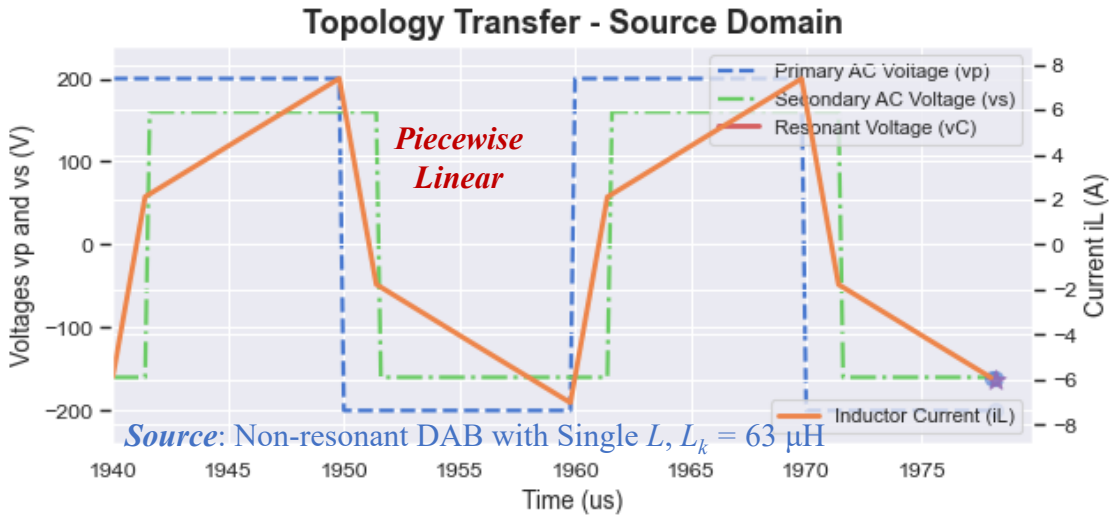
■ Add physics of $v_c$

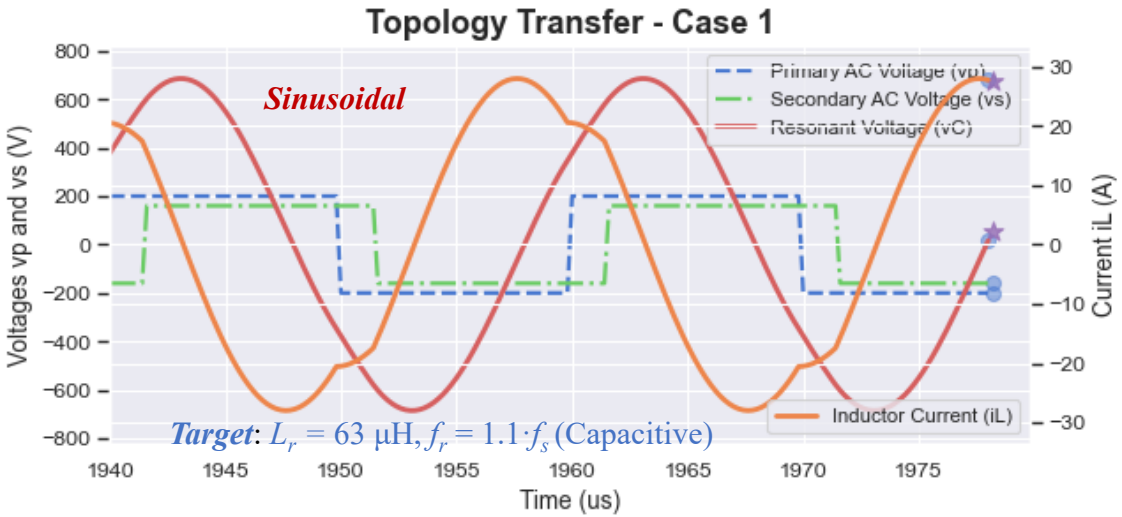
■ **Resonant DAB with Series *LC***

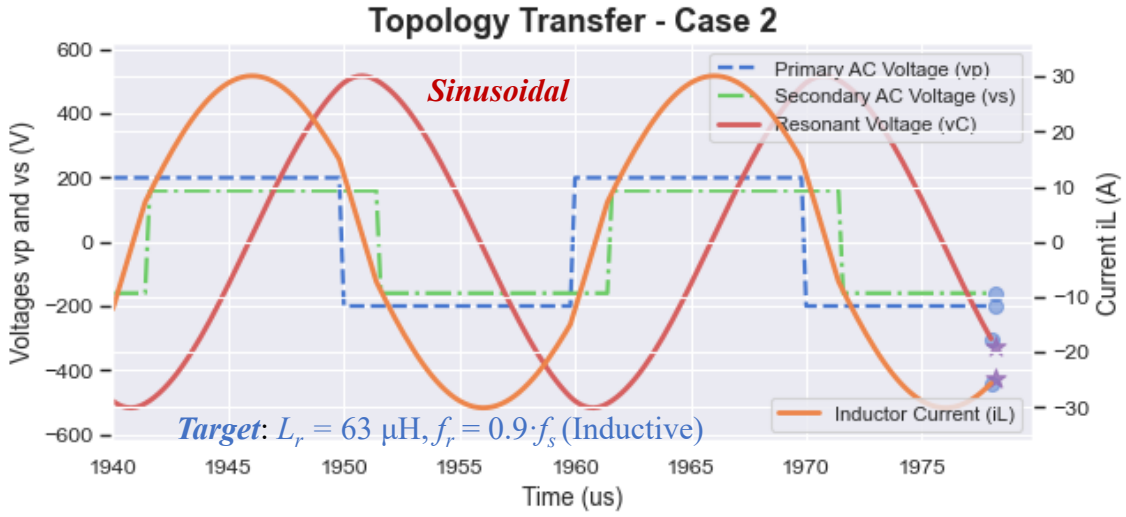
# CASE STUDY: PANN'S FLEXIBILITY ACROSS TOPOLOGIES

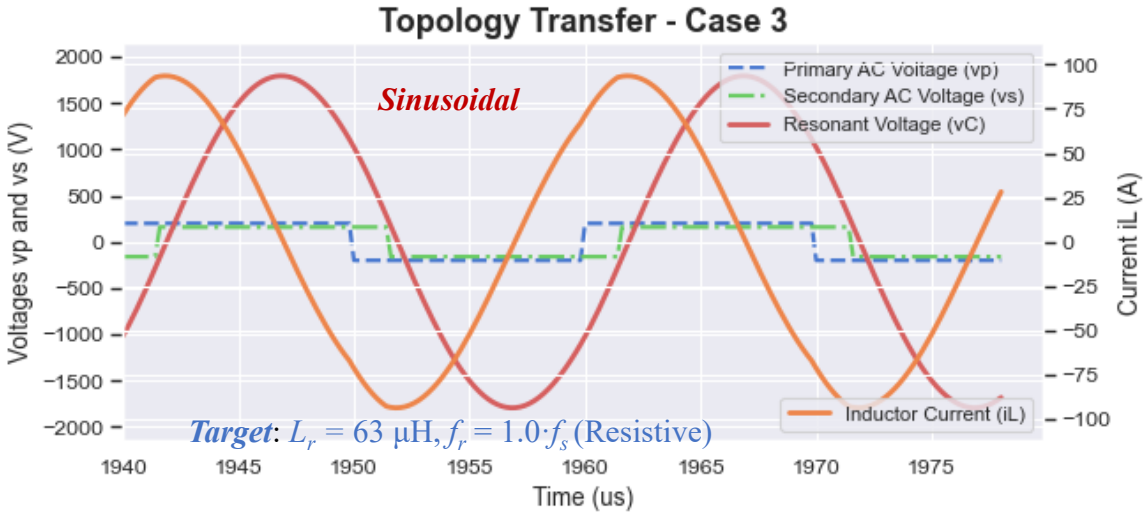


## PANN's Flexibility across Topologies (Code Demo*, "DAB-Topology Transfer.ipynb")



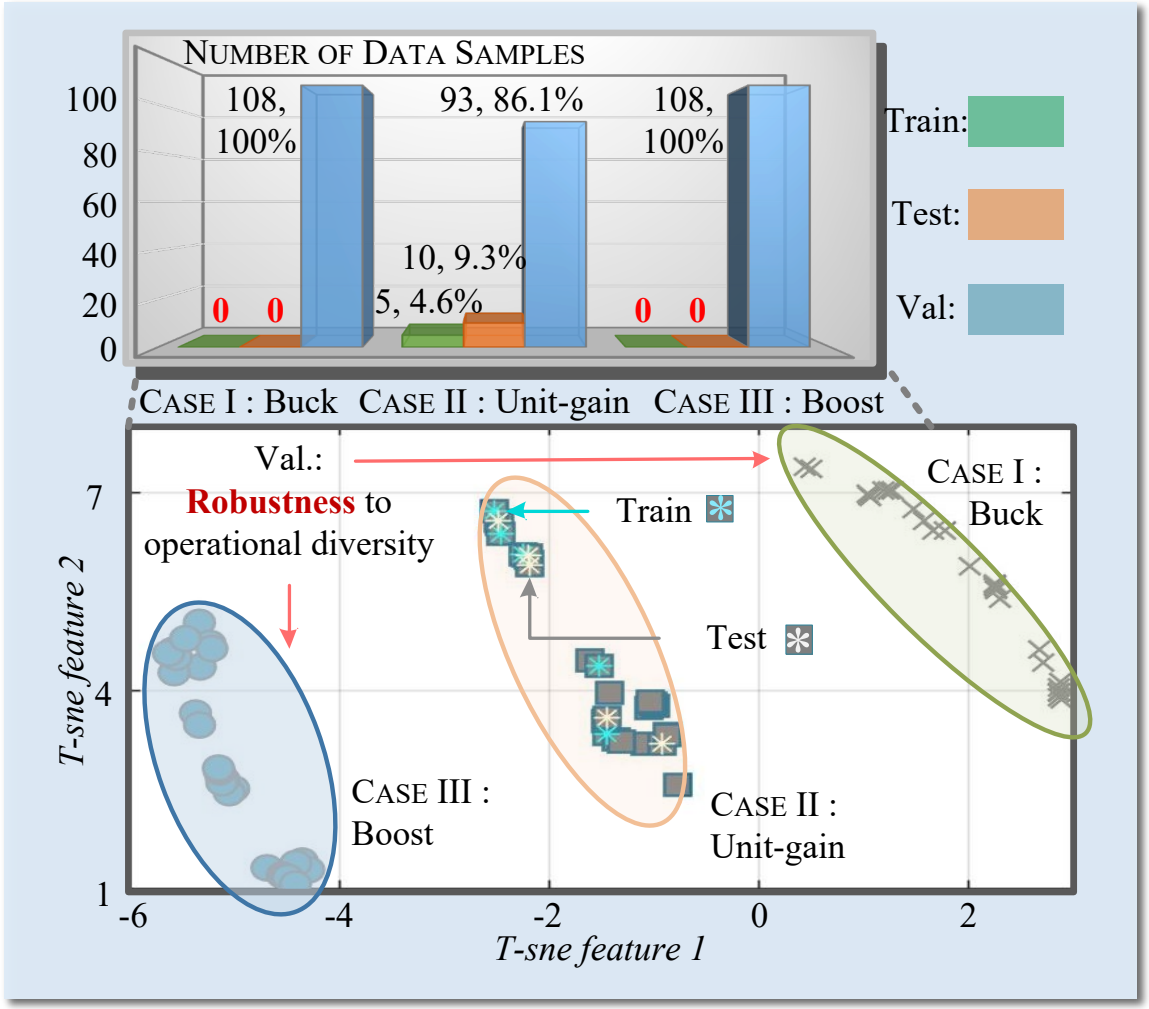



# STATISTICAL STUDY OF PANN'S FLEXIBILITY

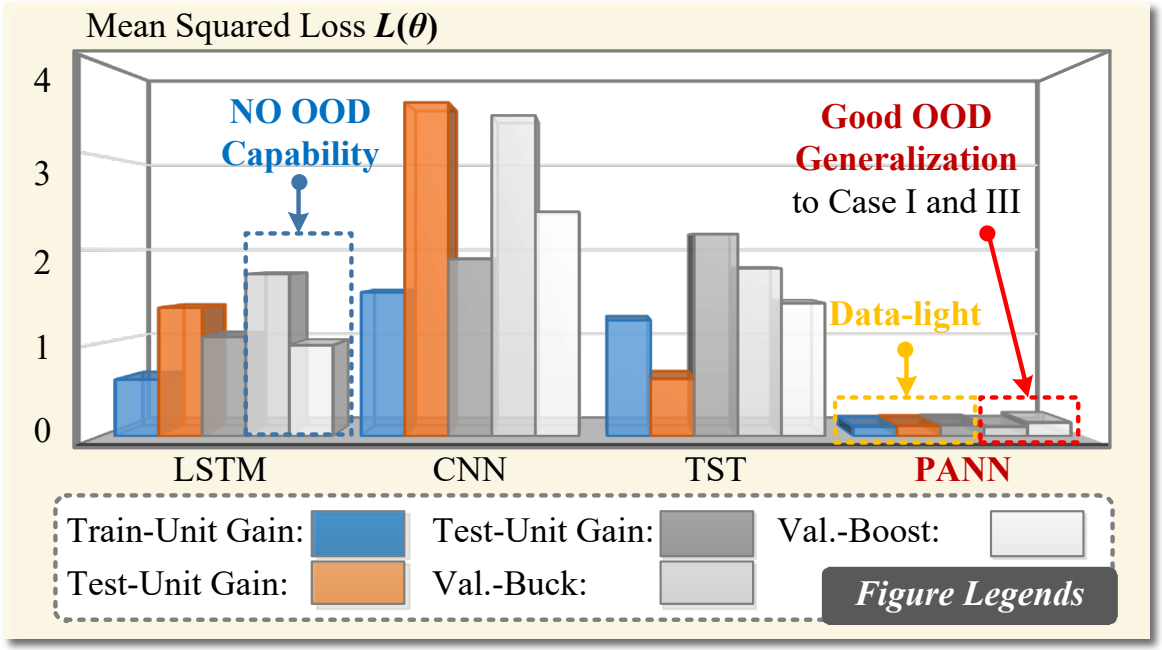


## Flexibility across Conditions and Modulations

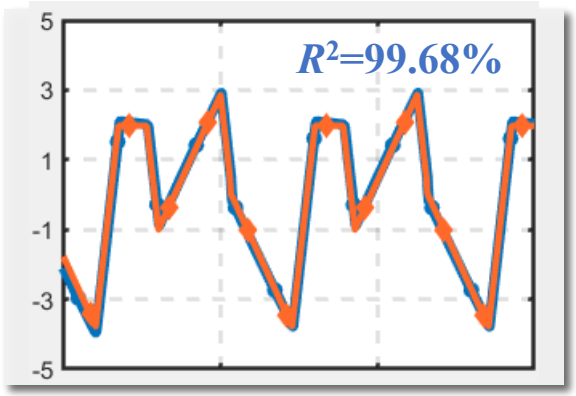


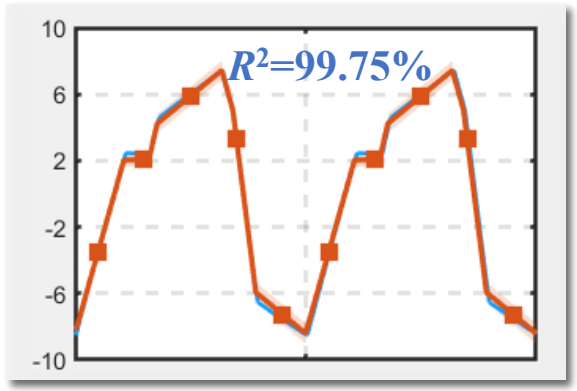

**Data Partition Settings to Validate Flexibility across Conditions**

## Statistics of Condition Transferability




## Statistics of Modulation Transferability

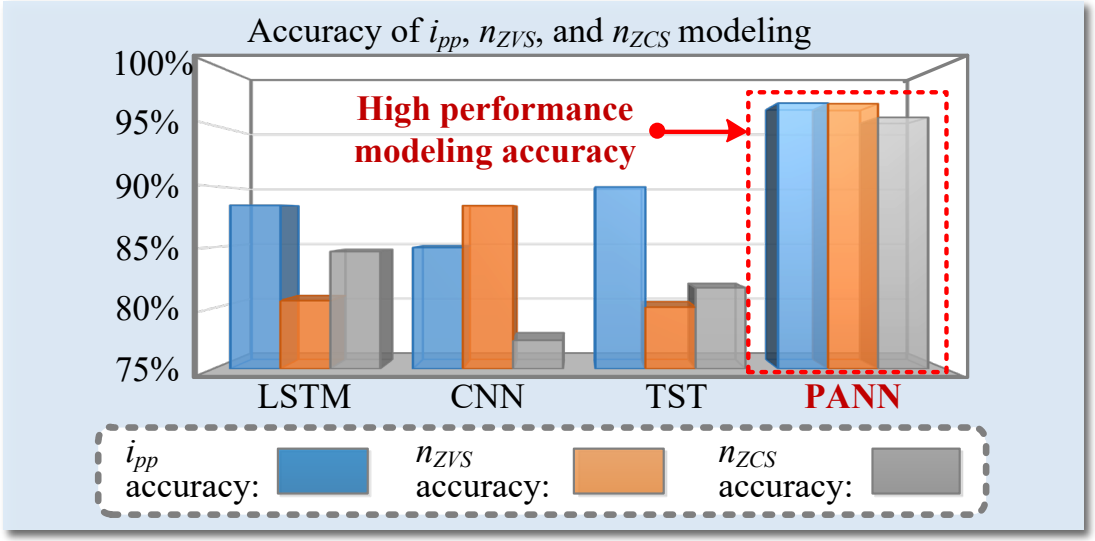


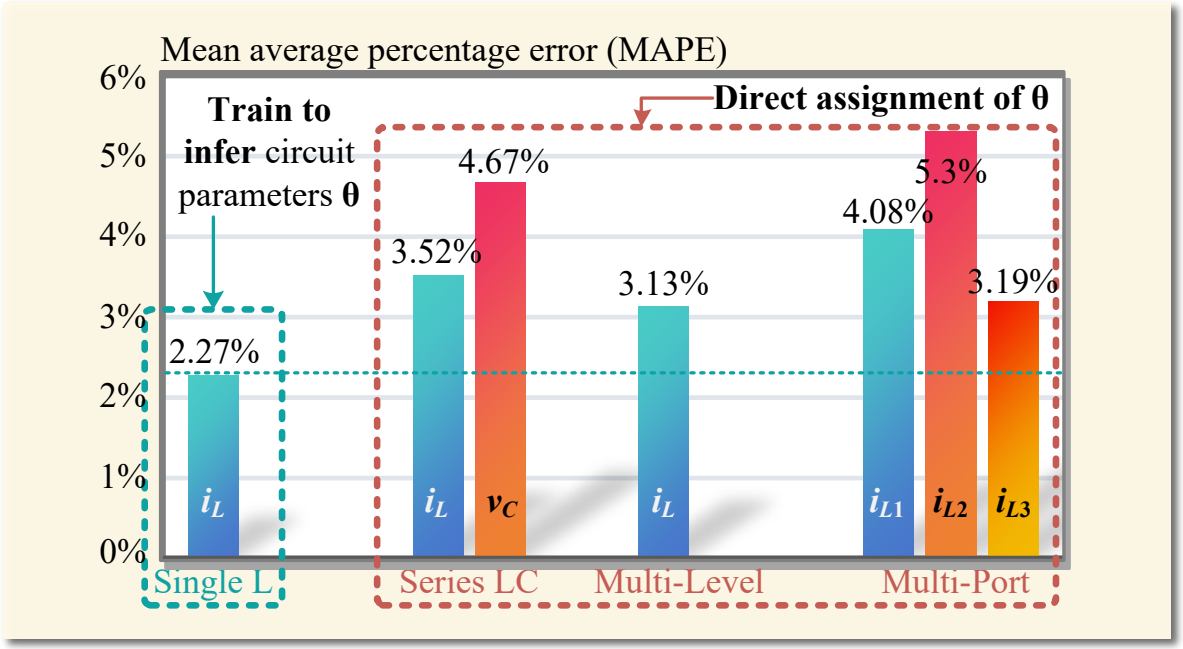
# STATISTICAL STUDY OF PANN'S FLEXIBILITY



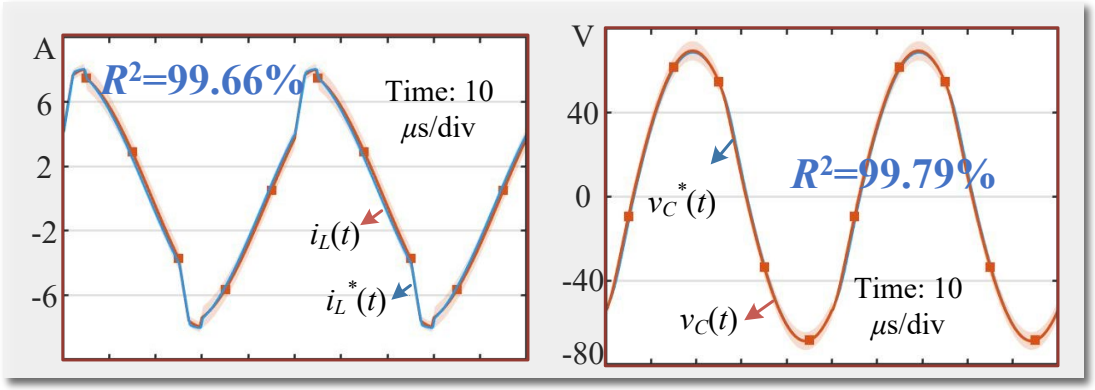
## Flexibility across Performance Metrics and Topologies



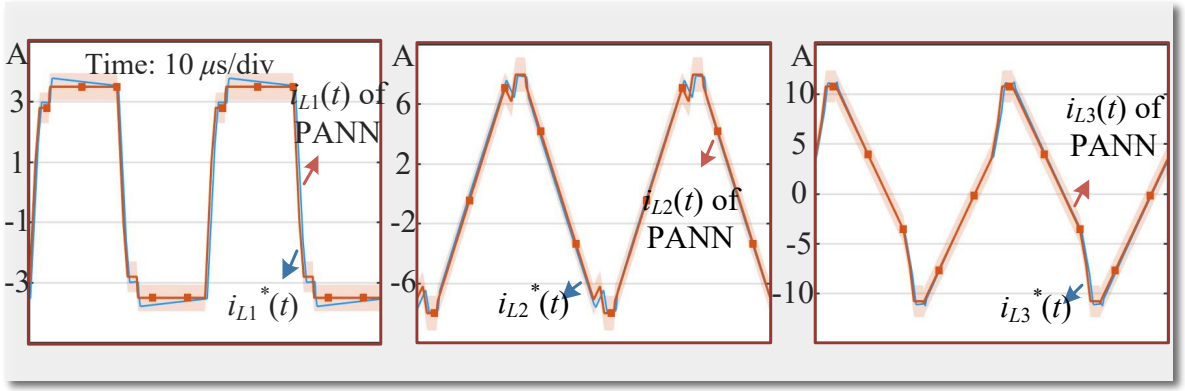

**Statistics of Performance Transferability**




**Statistics of Topology Transferability**




**Waveforms of Topology Transferability - LC**




**Waveforms of Topology Transferability - TAB**

# KEY TAKEAWAYS OF PANN'S FLEXIBILITY



## Flexibility I. Operating Conditions

- Simply update operating params. in **expert system** to change input signals
- **Training-free**

## Flexibility II. Modulation Strategy

- Switching behaviors in **expert system** should be updated accordingly
- **Training-free**

***Takeaways***

***PANN's Flexibility***

## Flexibility III. Performance Metrics

- **Numerically** gauge metrics based on **input** signals from expert system and **state** waveforms from PANN
- Metric adapter is **case-specific**, rely on topology, modulation, etc.
- **Training-free**

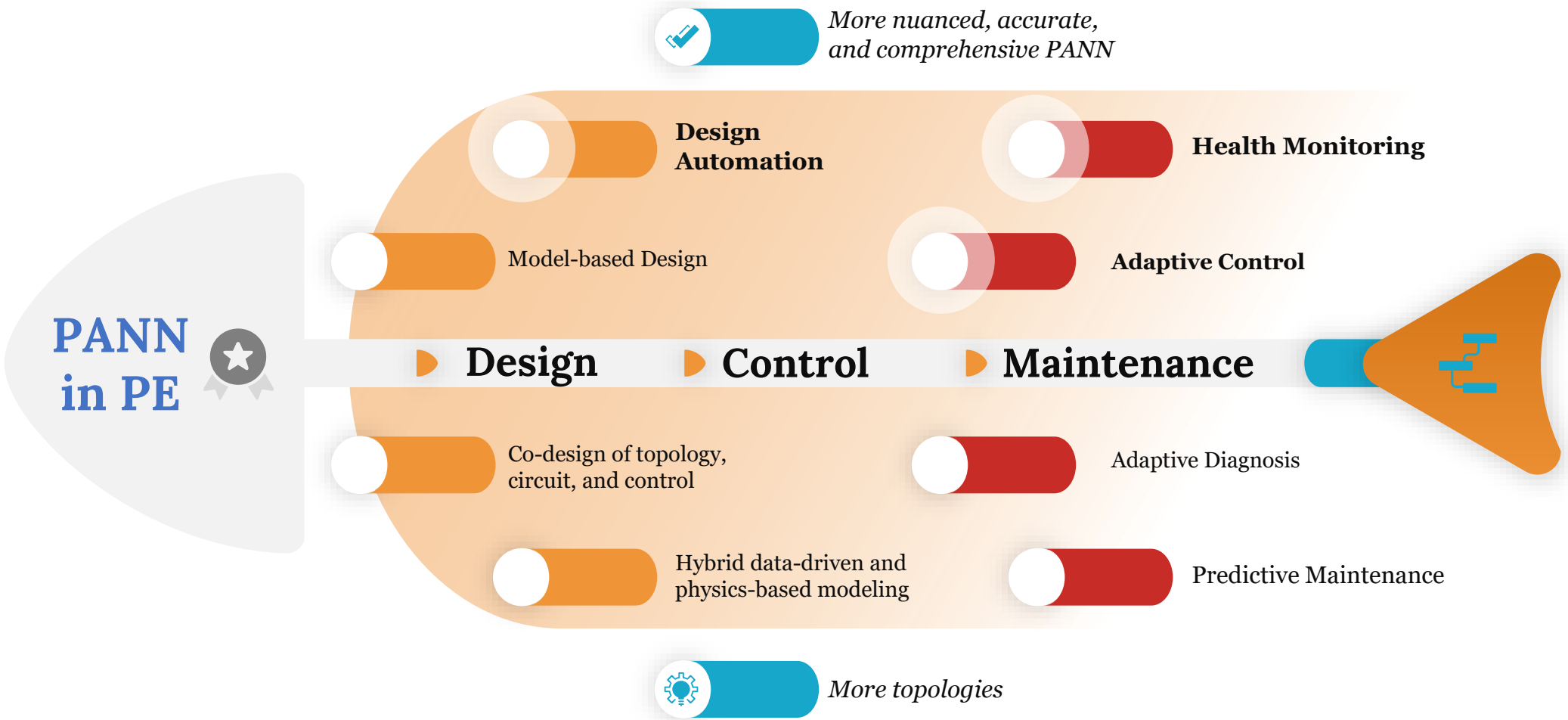
## Flexibility IV. Circuit Parameters & Topology

- Circuit θ is adjustable if there is: **1) training data; 2) priori knowledge of θ** (direct value assignment)
- Topology transfer: add/del. **neural links** of PANN

# NEXT GENERATION OF AI FOR POWER ELECTRONICS

# Future Directions of PANN in Power Electronics





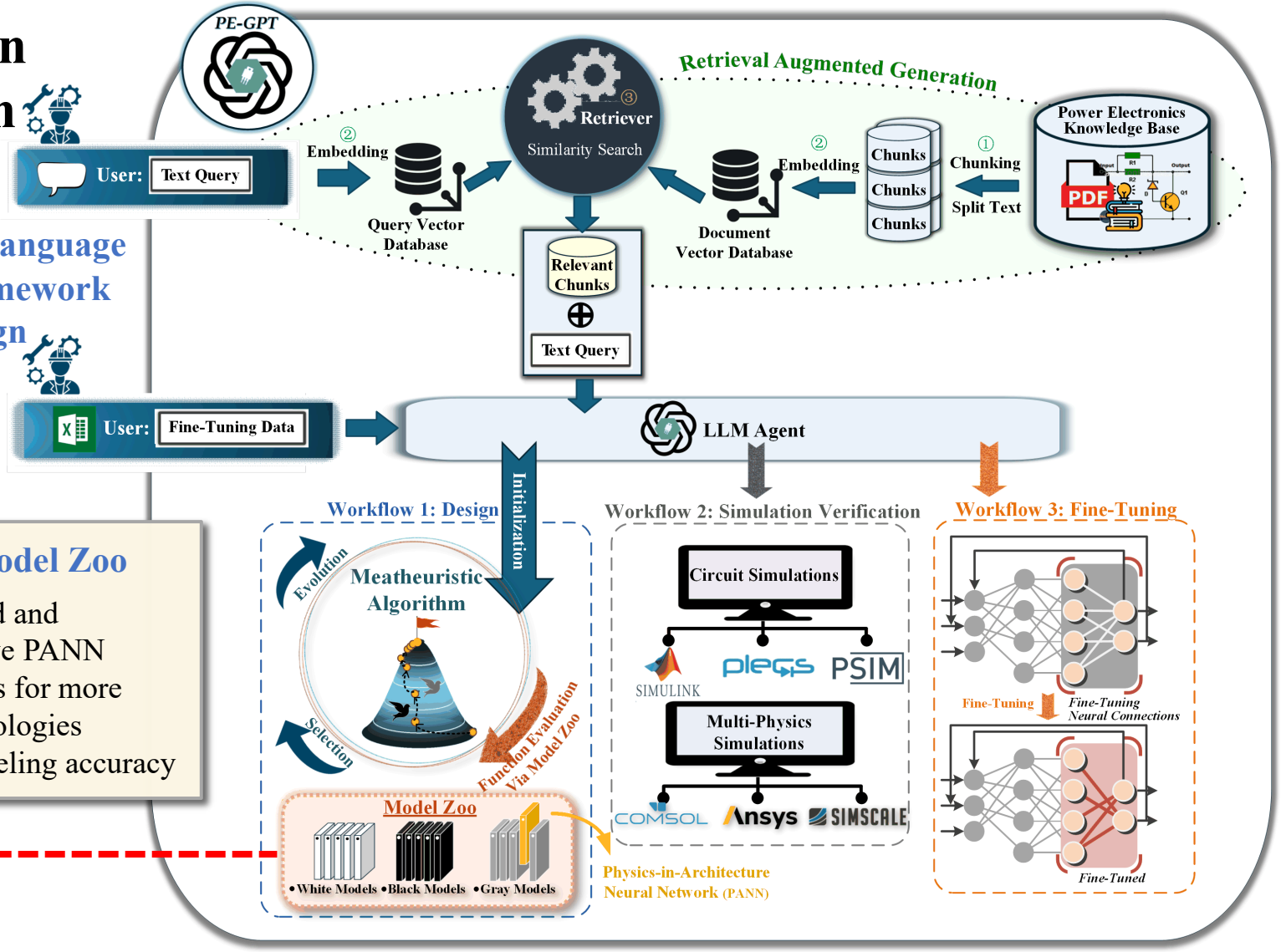

**PANN in the Life Cycle of Power Converters**

# Future Directions of PANN in Power Electronics



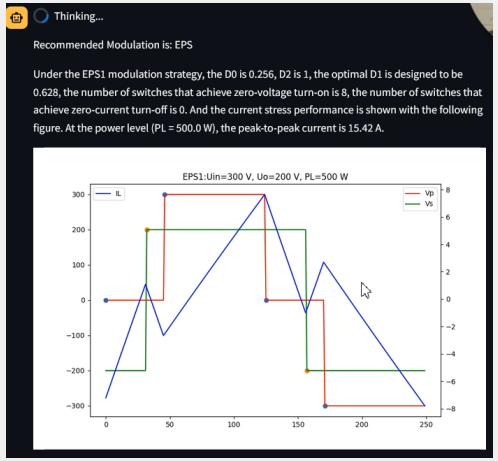
## I. PE Design Automation

**PE-GPT**: Large language model-based framework for PE design




### Enhance Model Zoo

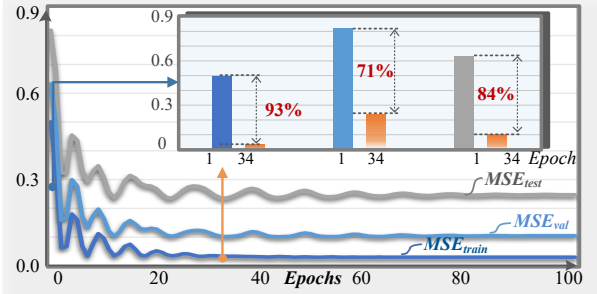
- More nuanced and comprehensive PANN
- PANN models for more converter topologies
- Improve modeling accuracy




*Modeling and optimization*

**Extended Phase Shift (EPS) Modulation Insights:** The EPS strategy is characterized by its ability to enhance zero-voltage switching (ZVS) across a broader range of operating conditions. In this strategy, the duty cycles are as follows: D0 is set to 0.256, indicating the initial phase shift, and D2 is maximized at 1, ensuring full utilization of the phase shift capability. The optimal duty cycle for D1 is determined to be 0.628, which is a critical parameter for balancing power flow and minimizing switching losses. This configuration allows for eight switches to achieve ZVS, significantly reducing switching losses and enhancing efficiency. However, it does not facilitate zero-current switching (ZCS), which could further optimize performance by reducing current stress. At a power level of 500 W, the EPS strategy achieves a peak-to-peak current of 15.42 A, indicating a moderate level of current stress.

*Optimal design insights*

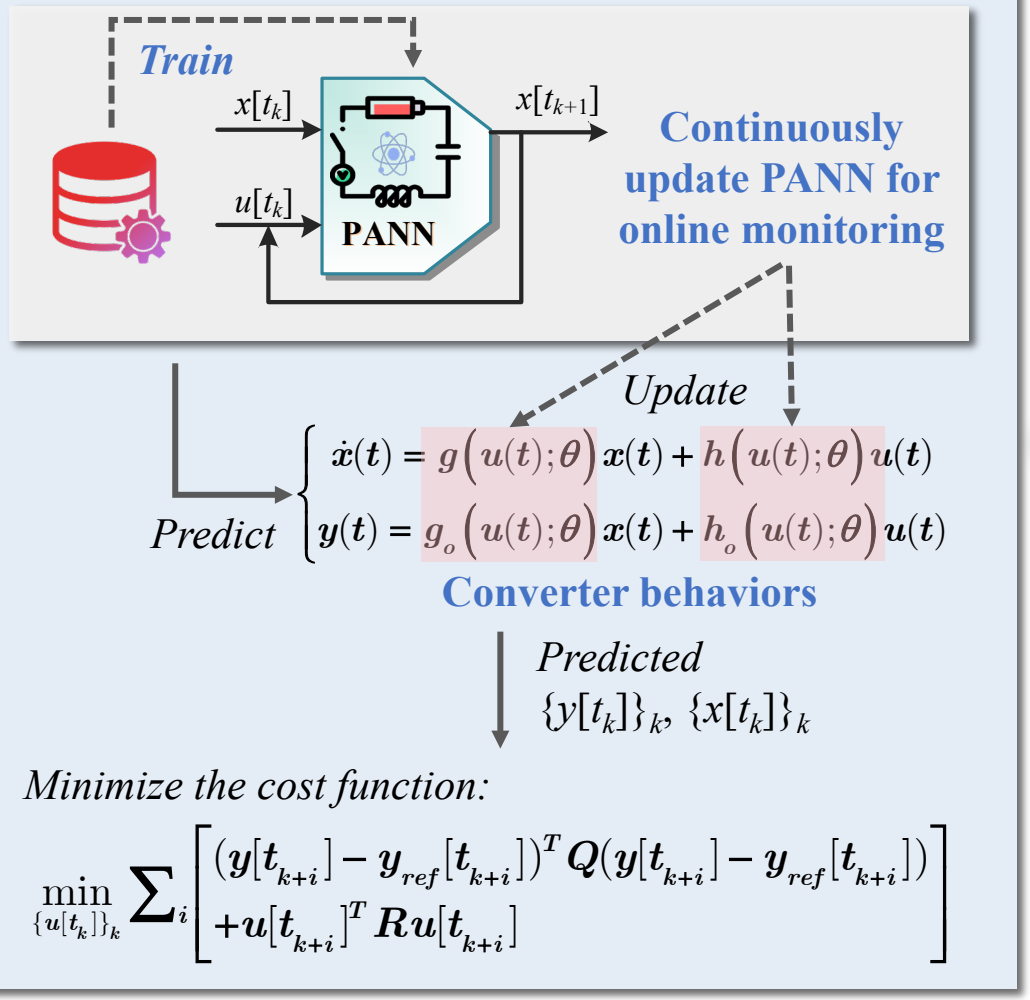


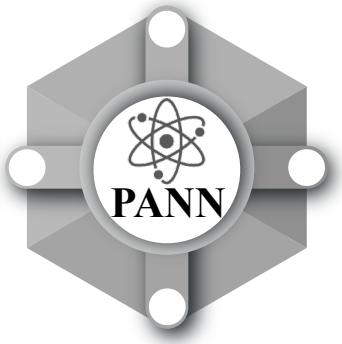

*Training of PANN*

**Source**: F. Lin et al., "PE-GPT: A New Paradigm for Power Electronics Design," in *IEEE Trans. on Ind. Electron.*

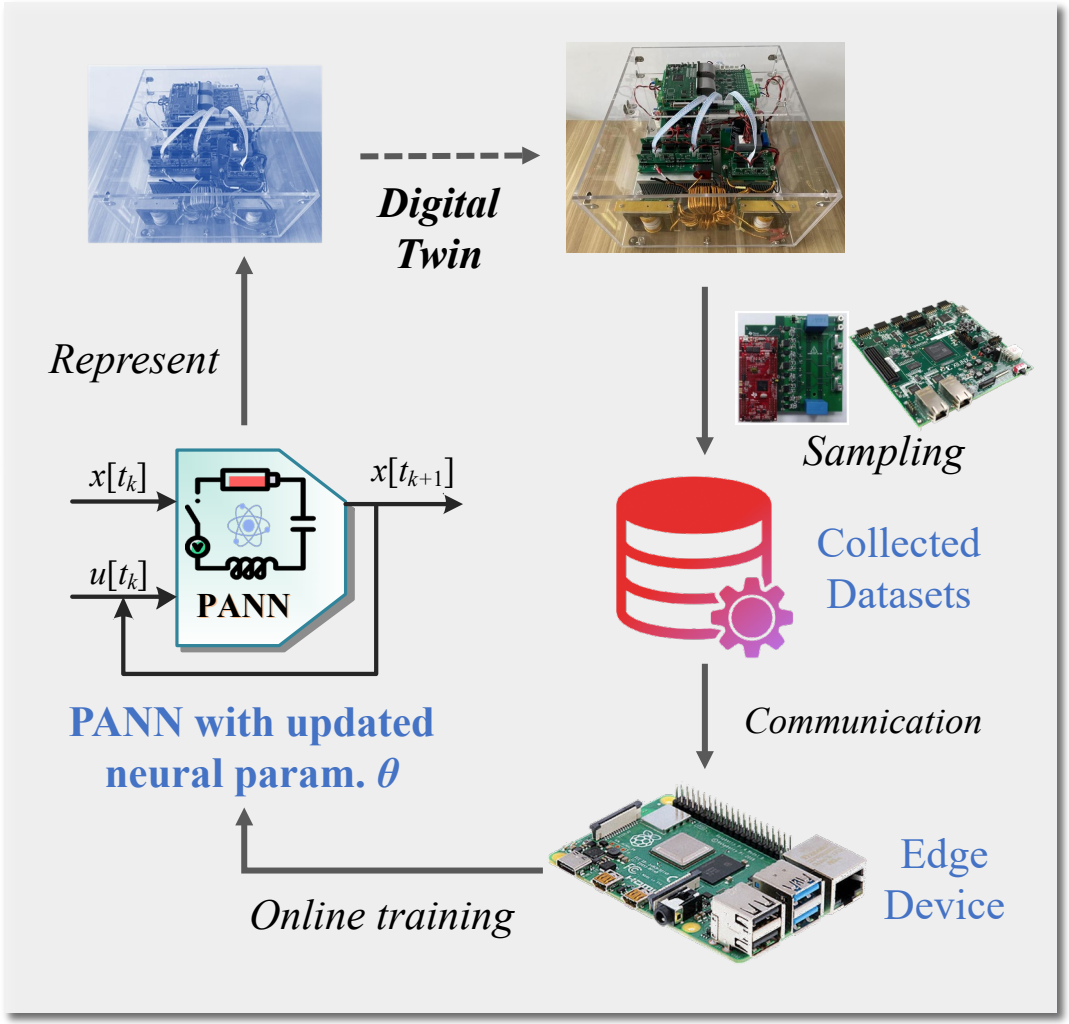
**Code**: https://github.com/XinzeLee/PE-GPT

## II. Adaptive Control



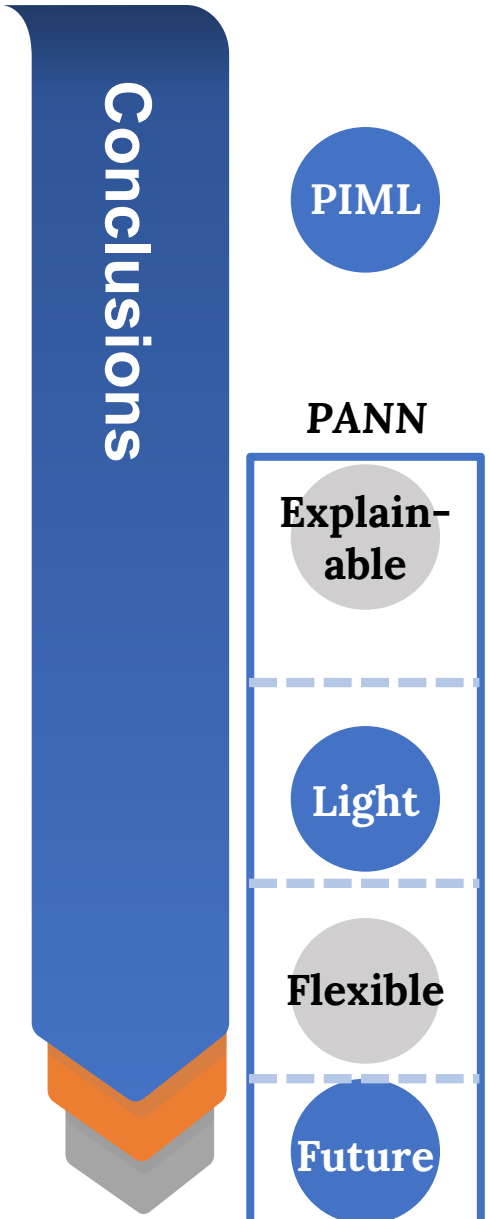




## III. Health Monitoring



# CONCLUSIONS






- AI algorithms applied in PE require explainable, light, and flexible solutions

- **PIML introduces physics expressed as PDEs into ML models to steer learning towards identifying physically consistent solutions**
- PIMLs can solve PDEs in forward ways and identify parameters in inverse settings
- **PIMLs are effective for ill-posed and mesh-free forward problems to solve PDEs and inverse problems to identify physical parameters.**
- Three types of PIMLs: Physics-in-loss, physics-in-architecture, physics-in-initialization

- **Physically crafted recurrent networks to integrate PDEs discretized by numerical methods**
- PANN features time-domain modeling for power electronics, where state variables are recurrently predicted leveraging the previous predictions of states
- PANN is explainable in PE: Neural parameters of PANN are converter parameters, and switching behaviors, commutation loops, operating modes and so on are embodied

- Training only requires few data samples, reducing data requirements by 3 orders of magnitude
- PANN is lightweight (several kB), deployable on edge devices
- PANN training equals to parameter identification, and learning rates are impactful hyperparameters

- Flexible across operating conditions, modulation strategies, performance metrics, circuit parameters, and topological variants
- Training-free transfer (use knowledge to replace data)

- Apply PANN in design automation, adaptive control, and health monitoring for power converters
- More nuanced, accurate, and comprehensive PANN to incorporate more topologies

# Next Generation of AI for Power Electronics:

The ultimate frontier of AI for power electronics is:

**physics-informed AI.**

Xinze Li

xinzel@uark.edu

UNIVERSITY OF ARKANSAS


**2024 Nobel Prize - Physics**

**Awarded to AI Pioneers**

**Open Source**

https://github.com/XinzeLee