G-SpNN: GPU-Accelerated Passivity Enforcement for S-Parameter Modeling with Neural Networks

Lijie Zeng[1], Jiatai Sun[1], Xiao Wu[2], Dan Niu[3], Tianshi Wang[4], Yibo Lin[5], Zuochang Ye[6], Zhou Jin[7]

[1]SSSLab, China University of Petroleum-Beijing, China, [2]Huada Empyrean Software Co. Ltd, China,
[3]School of Automation, Southeast University, China, [4]HiSilicon Technologies Co. Ltd,
[5]School of Integrated Circuits, Peking University, Beijing, [6]School of Integrated Circuits, Tsinghua University,
[7]College of Integrated Circuits, Zhejiang University, China

Email: z.jin@zju.edu.cn

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

OUTLINE

1. Background
2. G-SpNN
3. Experiment
4. Conclusion

OUTLINE

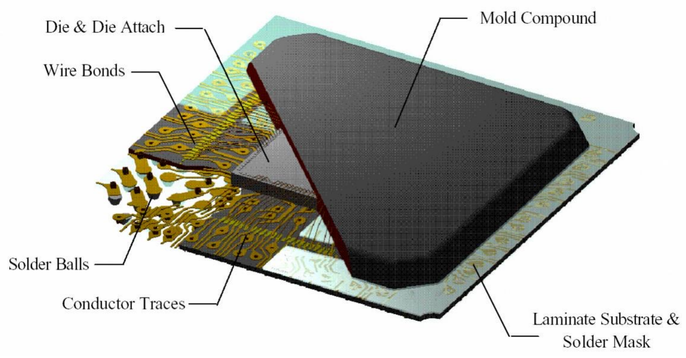
1. Background
2. G-SpNN
3. Experiment
4. Conclusion

Background

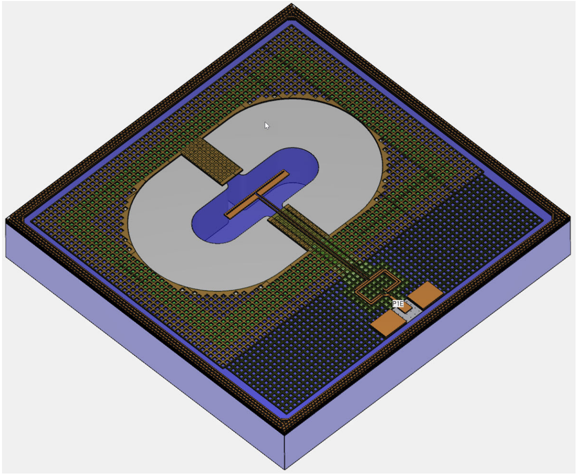
- At high frequencies, **S-parameters** (scattering parameters) are commonly used to describe the performance of microwave and RF devices.

- Running time-domain analyses with them is computationally intensive and often leads to convergence issues.

- **Macromodeling** techniques are applied to simplify these behaviors, enhancing simulation efficiency and stability.



Chip Packaging Simulation



RFIC Simulation

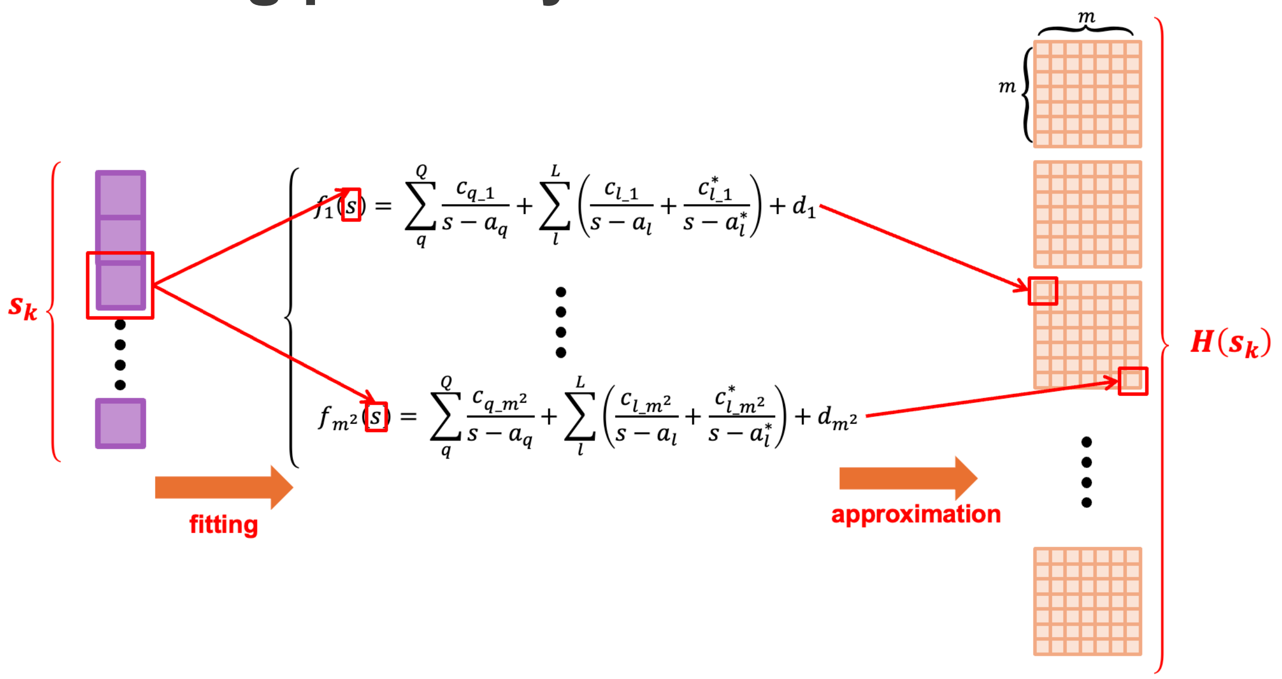
Background

Mainstream methods typically adopt a two-step approach:

① Generate a macromodel without considering passivity constraints

Mainstream method such as: Vector Fitting (VF)[1]

Using the VF method, a set of functions with a given form is fitted to the known frequency data s_k and system response $H(s_k) \in \mathbb{C}^{m\times m}$.



[1] B. Gustavsen and A. Semlyen, "Rational approximation of frequency domain responses by vector fitting," IEEE Transactions on Power Delivery,vol. 14, no. 3, pp. 1052–1061, 1999.

Background

② Applying specialized algorithms to restore passivity

Such as: Eigenvalue Perturbation (EPM)[2], Residue Perturbation (RPM)[3], Local Compensation (LC)[4]

Transform the rational function form into a state-space form and apply the EPM/RPM/LC to restore the passivity of model $G(s)$.

$$f(s_k) = \sum_{n=1}^{N_q} \frac{c_n}{s_k - a_n} + d + s_k h \Rightarrow G(s_k) = C(s_k \cdot I - A)^{-1}B + D \Rightarrow G(s_k) + G^H(s_k) \geq 0$$

rational function form

state-space form

passivity condition

[2] S. Grivet-Talocia, "An adaptive sampling technique for passivity characterization and enforcement of large interconnect macromodels," IEEE Transactions on Advanced Packaging, vol. 30, no. 2, pp. 226–237, 2007.

[3] B. Gustavsen, "Fast passivity enforcement for pole-residue models by perturbation of residue matrix eigenvalues," IEEE Transactions on Power Delivery, vol. 23, no. 4, pp. 2278–2285, 2008.

[4] T. Wang and Z. Ye, "Robust passive macro-model generation with local compensation," IEEE Transactions on Microwave Theory and Techniques, vol. 60, no. 8, pp. 2313–2328, 2012.

Background

Two-Step Approach

VF + EPM/RPM/LC

existing issues

compromise model accuracy

the most critical issue

address this issue

Domain-Alternated Optimization (DAO) [5]

the proposal of a new method

[5] Ye Z, Wang T, Li Y. Domain-alternated optimization for passive macromodeling[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2014

Background

DAO introduces two key transformation operators. Based on the transformation using SPF and PFE, the system $G(s_k)$ derived from $W(s_k)$ is guaranteed to preserve passivity, and the original optimization problem can be further converted into an unconstrained optimization problem.

Spectral Factorization (SPF)

$$R = Q^TQ = D + D^T$$
$$\tilde{A} = BR^{-1}C - A,\ \tilde{B} = BQ^{-1},\ \tilde{C} = C^TR^{-1}C$$
$$K\tilde{A} + \tilde{A}^TK - K\tilde{B}\tilde{B}^TK - \tilde{C} = 0$$
$$L = Q^{-T}C - Q^{-T}B^TK$$
$$W(s_k) = L(s_k \cdot I - A)^{-1}B + Q$$

Partial Fractional Expansion (PFE)

$$M = \mathrm{kron}(A^T, I_n) + \mathrm{kron}(I_n, A^T)$$
$$\mathrm{vec}(K) = -M^{-1}\mathrm{vec}(L^TL)$$
$$C = B^TK + Q^TL,\ D = \frac{1}{2}Q^TQ$$

$$G(s_k) = C(s_k \cdot I - A)^{-1}B + D \quad \underset{\text{SPF}}{\overset{\text{PFE}}{\rightleftarrows}} \quad W(s_k) = L(s_k \cdot I - A)^{-1}B + Q$$

$$\begin{cases} \min_{C,D} \sum_k |G(s_k) - H(s_k)| \\ \text{subject to:} \\ G(s_k) \text{ is passive} \end{cases} \Rightarrow \begin{cases} \min_{L,Q} \sum_k |G(s_k) - H(s_k)| \\ \text{subject to:} \\ W(s_k) = L(s_k \cdot I - A)^{-1}B + Q \\ G(s_k) = pfe(W(s_k)) \end{cases}$$

Original Optimization Problem **Unconstrained Optimization Problem**

Background

Three-Step Approach VF + EPM/RPM/LC + Domain-Alternated Optimization

The essence of DAO is to transform the passivity-constrained problem into an unconstrained optimization problem, thereby improving macromodel accuracy while maintaining passivity throughout the process.

Stage	Method	Passivity	Time (s)	Total Error
1	VF	non-passive	544	1.69%
2	LC	passive	253	8.38%
3	DAO	passive	12238	2.63%

case with 21-port system

Background

Three-Step Approach VF + EPM/RPM/LC + Domain-Alternated Optimization

However, as the number of ports in complex integrated circuits continues to grow, the three-step approach increasingly reveals additional issues.

For the previously mentioned 21-port system, the iteration time of DAO is 15 times longer than the total time of the first two steps. For a 64-port system, each iteration consumes an average of 22GB of memory, while for a 138-port system, memory usage exceeds 31.8GB.

Stage	Method	Passivity	Time (s)	Total Error
1	VF	non-passive	544	1.69%
2	LC	passive	253	8.38%
3	DAO	passive	12238	2.63%

case with 21-port system

Background

Three-Step Approach VF + EPM/RPM/LC + Domain-Alternated Optimization

However, as the number of ports in complex integrated circuits continues to grow, the three-step a

the Two Primary Challenges

al issues.

For the previously mentioned 21-port system, the iteration time of DAO is

High Memory Consumption **Slow Convergence**

e first

erage

a 138-port system, memory usage exceeds 31.8GB.

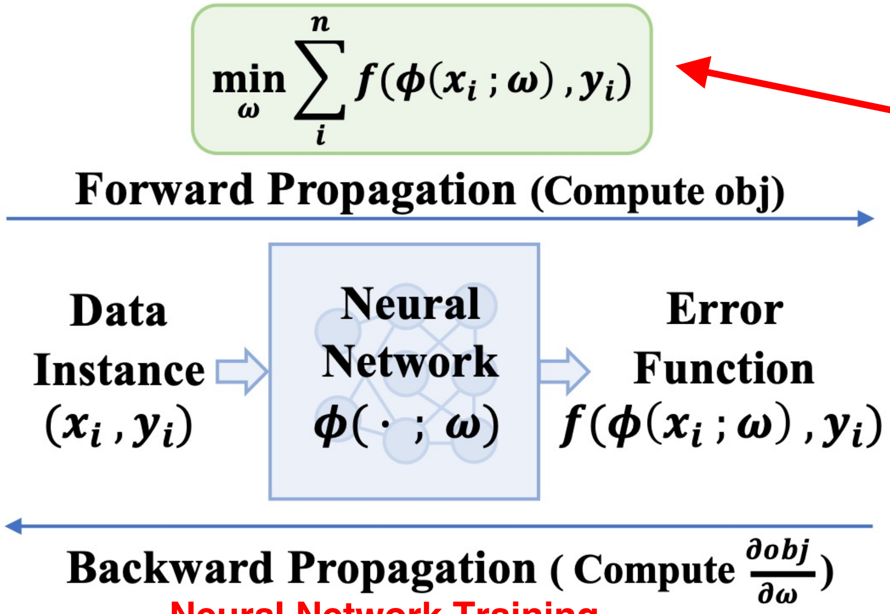
Stage	Method	Passivity	Time (s)	Total Error
1	VF	non-passive	544	1.69%
2	LC	passive	253	8.38%
3	DAO	passive	12238	2.63%

case with 21-port system

OUTLINE

1. Background
2. G-SpNN
3. Experiment
4. Conclusion

G-SpNN | Analogy to Neural Network Training



Neural Network Training

Train a network for weights w

Vector x_i, Label y_i , Predicted Label $\phi(x_i, w)$

VS

$$
\begin{cases}
\boxed{\min_{L,Q} \sum_k |G(s_k) - H(s_k)|} \\
\text{subject to:} \\
W(s_k) = L(s_k \cdot I - A)^{-1}B + Q \\
G(s_k) = pfe\big(W(s_k)\big)
\end{cases}
$$

Unconstrained Optimization Problem

The error function in unconstrained optimization can be viewed as analogous to the prediction error encountered in neural network training.

G-SpNN | Analogy to Neural Network Training

$$\min_{\omega} \sum_{i}^{n} f(\phi(x_i\,;\omega)\,,y_i)$$

Forward Propagation (Compute obj)

Data Instance $(x_i\,,y_i)$ → **Neural Network** $\phi(\cdot\,;\boxed{\omega})$ → **Error Function** $f(\phi(x_i\,;\omega)\,,y_i)$

Backward Propagation (Compute $\frac{\partial obj}{\partial \omega}$)

Neural Network Training

Train a network for weights w

Vector x_i, Label y_i , Predicted Label $\phi(x_i, w)$

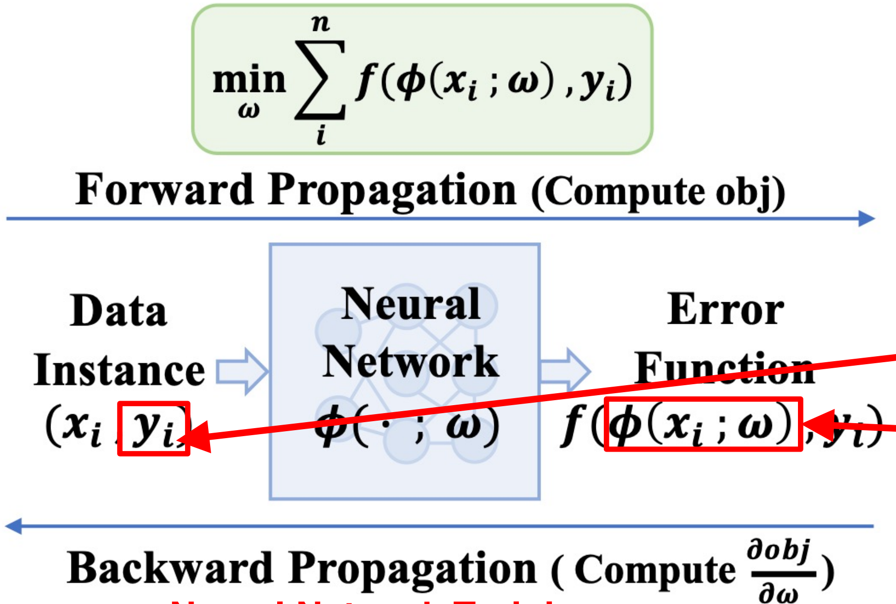
VS

$$\begin{cases} \min_{\boxed{L,Q}} \sum_k |G(s_k) - H(s_k)| \\ \text{subject to:} \\ W(s_k) = L(s_k \cdot I - A)^{-1}B + Q \\ G(s_k) = pfe\big(W(s_k)\big) \end{cases}$$

Unconstrained Optimization Problem

The optimization variables L, Q in unconstrained optimization can be viewed as analogous to the trainable weight encountered in neural network training.

G-SpNN | Analogy to Neural Network Training



Neural Network Training

Train a network for weights w
Vector x_i, Label y_i , Predicted Label $\phi(x_i, w)$

VS

$$\begin{cases} \min_{L,Q} \sum_k |G(s_k) - \boxed{H(s_k)}| \\ \text{subject to:} \\ W(s_k) = L(s_k \cdot I - A)^{-1} B + Q \\ \boxed{G(s_k)} = pfe\big(W(s_k)\big) \end{cases}$$

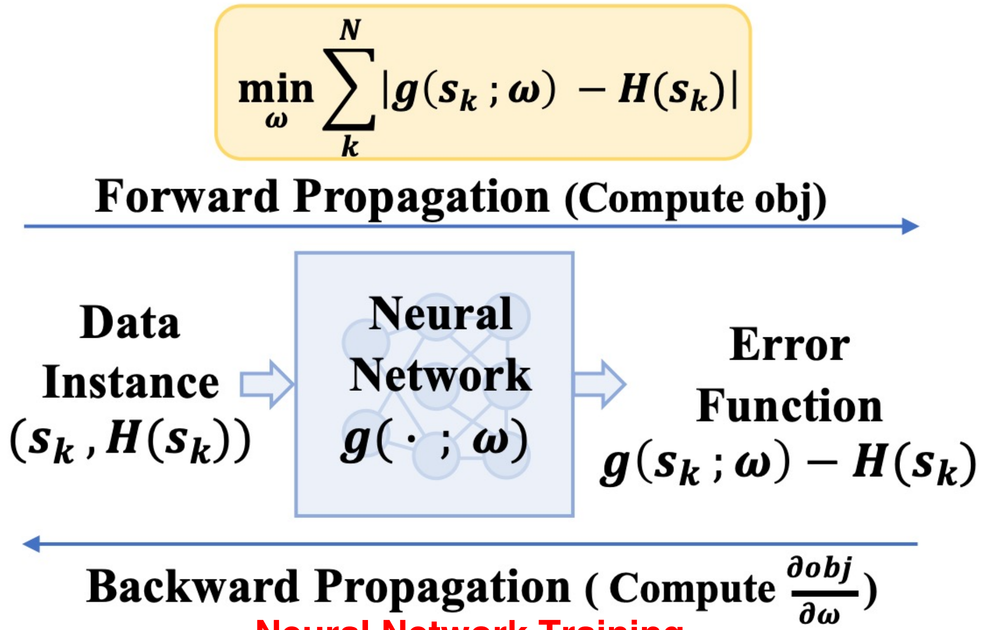
Unconstrained Optimization Problem

$H(s_k)$, $G(s_k)$ in unconstrained optimization can be viewed as analogous to the label and predicted label encountered in neural network training.

G-SpNN | Analogy to Neural Network Training

$$
\begin{cases}
\min_{L,Q} \sum_k |G(s_k) - H(s_k)| \\
\text{subject to:} \\
W(s_k) = L(s_k \cdot I - A)^{-1}B + Q \\
G(s_k) = pfe\big(W(s_k)\big)
\end{cases}
$$

Unconstrained Optimization Problem



Neural Network Training

Solve a set of parameters for macromodel with $w = (L, Q)$

The two optimization frameworks share a fundamental similarity ,making it possible to leverage neural network techniques to accelerate macromodeling optimization.

G-SpNN | Loss Function

To make the neural network training process more efficient, the objective function can be reformulated to reduce computational complexity.

The summation of errors over k terms increases computational complexity.

$$\textbf{Error} = \min_{C,D} \sum_{\boxed{k}} |G(s_k) - H(s_k)|$$

Original Objective Function

$$\text{Error\_vec} = |Fy - h|$$

$$y = \begin{bmatrix} \mathrm{vec}(C) \\ \mathrm{vec}(D) \end{bmatrix} \quad \mathrm{F}_i = [\mathrm{kron}([s_i I - A]^{-1}, I_n)\ I_m^2]$$

$$h = \begin{bmatrix} \mathrm{Vec}(Re(H(s_1))) \\ \vdots \\ \mathrm{Vec}(Re(H(s_N))) \\ \mathrm{Vec}(Im(H(s_1))) \\ \vdots \\ \mathrm{Vec}(Im(H(s_N))) \end{bmatrix} \quad F = \begin{bmatrix} Re(F_1) \\ \vdots \\ Re(F_N) \\ Im(F_1) \\ \vdots \\ Im(F_N) \end{bmatrix}$$

$$F = Q_F R_F$$

$$b = Q_F^T h,\ \delta^2 = h^T h - b^T b$$

Intermediate Computation Steps

$$\textbf{loss} = \textbf{Error} = (R_F y - b)^T (R_F y - b) + \delta^2$$

Reformulated Objective Function

G-SpNN | Loss Function

Based on the new definition of the loss function, optimization problem can be further reformulated.

$$\begin{cases} \min_{C,D} \sum_k |G(s_k) - H(s_k)| \\ \text{subject to:} \\ G(s_k) \text{ is passive} \end{cases}$$

Original Optimization Problem

$$\begin{cases} \min_{L,Q} \sum_k |G(s_k) - H(s_k)| \\ \text{subject to:} \\ W(s_k) = L(s_k \cdot I - A)^{-1}B + Q \\ G(s_k) = pfe(W(s_k)) \end{cases}$$

Unconstrained Optimization Problem

$$\begin{cases} \text{variable: } L, Q \\ \min: f(y) = (R_F y - b)^T (R_F y - b) + \delta^2 \\ \text{subject to:} \\ x = \begin{bmatrix} \text{vec}(L) \\ \text{vec}(Q) \end{bmatrix} \\ y = pfe(x) \end{cases}$$

Reformulated Optimization Problem

The reformulated optimization problem can also be analogized to neural network training for solution.

G-SpNN | Further Analysis of the Main Challenges

For the transformed unconstrained optimization problem, it can be solved by computing the gradient vector and Hessian matrix, but this is also the main source of the challenges.

$$\begin{cases} \min_{L,Q} \sum_k |G(s_k) - H(s_k)| \\ \text{subject to:} \\ W(s_k) = L(s_k \cdot I - A)^{-1} B + Q \\ G(s_k) = pfe\big(W(s_k)\big) \end{cases}$$

Unconstrained Optimization Problem

High Time Complexity
High Space Complexity

Hessian Matrix

Gradient Vector

High Memory Consumption

Slow Convergence

Simplified Computational Steps
(involves matrix inversion and dense matrix multiplication)

$$\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y_C}{\partial x_L} & \frac{\partial y_C}{\partial x_Q} \\ \frac{\partial y_D}{\partial x_L} & \frac{\partial y_D}{\partial x_Q} \end{bmatrix} = \begin{bmatrix} -J_{CK} M^{-1} \mathcal{N}_L(L) + \mathcal{K}_Q(Q) & \mathcal{K}_L(L) \\ 0 & \mathcal{N}_Q(Q) \end{bmatrix} \qquad \frac{\partial f}{\partial y} = 2E^T R \qquad \frac{\partial^2 f}{\partial y^2} = 2R^T R \qquad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x}$$

$$\frac{\partial^2 y}{\partial x_k^2} = \frac{\partial^2 y}{\partial x^2}\mathbf{e}_k = \frac{\partial}{\partial x}\left(\frac{\partial y}{\partial x}\right)\mathbf{e}_k = \begin{bmatrix} -J_{CK} M^{-1} \mathcal{N}_L(L_k) + K_Q(Q_k) & K_L(L_k) \\ 0 & \mathcal{N}_Q(Q_k) \end{bmatrix} \qquad \frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right)^T = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\frac{\partial y}{\partial x}\right)^T = \left(\frac{\partial y}{\partial x}\right)^T \frac{\partial^2 f}{\partial y^2}\frac{\partial y}{\partial x} + \frac{\partial^2 y}{\partial x^2}\left(\frac{\partial f}{\partial y}\right)^T$$

G-SpNN | Further Optimization for Memory and Time Consumption

As previously mentioned, the full computation of the Hessian matrix incurs significant memory and time overhead; therefore, we adopt the LBFGS method to approximate the inverse of the Hessian matrix.

The update formula in LBFGS method

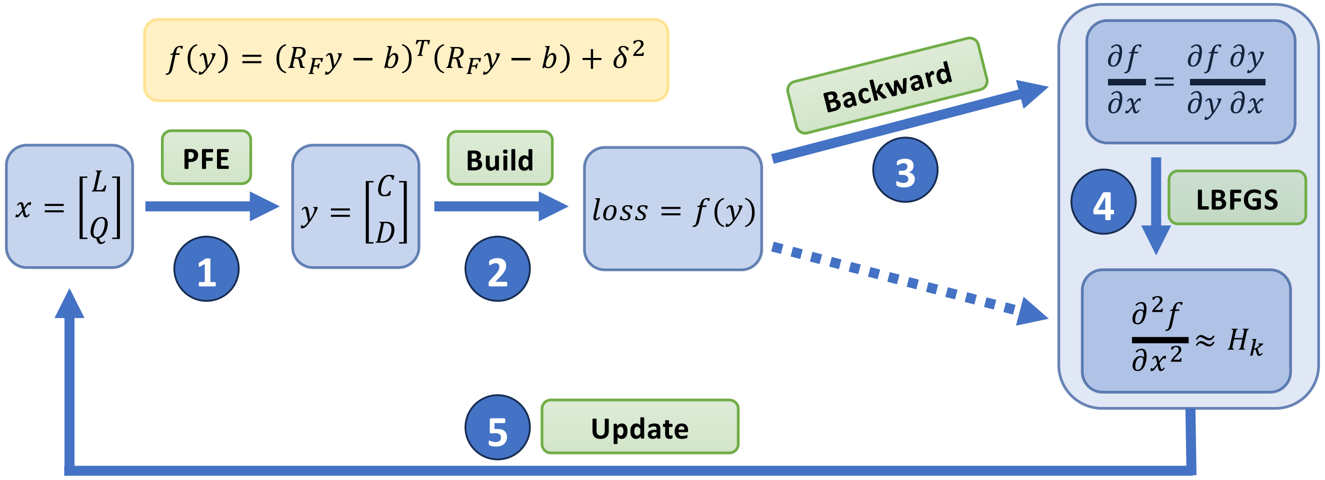
$$H_{k+1} = H_k - \frac{H_k \Delta g_k \Delta g_k^T H_k}{\Delta g_k^T \Delta x_k} + \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T \Delta g_k}$$

Where:

1) H_k is the Hessian inverse approximation at iteration k
2) $\Delta g_k = \left(\frac{\partial f}{\partial x}\right)_{k+1} - \left(\frac{\partial f}{\partial x}\right)_k$ is the change in the gradient
3) $\Delta x_k = x_{k+1} - x_k$ is the change in the parameter

The essence of LBFGS is to efficiently approximate the inverse of the Hessian matrix by retaining information from the most recent iterations, thereby accelerating the convergence of large-scale optimization problems.

G-SpNN | Further Optimization for Memory and Time Consumption



Computational Graph with LBFGS Method

Step 1: $\boldsymbol{x}$ undergoes the $\boldsymbol{PFE}$ operation to generate parameters $\boldsymbol{y}$ (the passive system $\boldsymbol{G}(\boldsymbol{s})$).

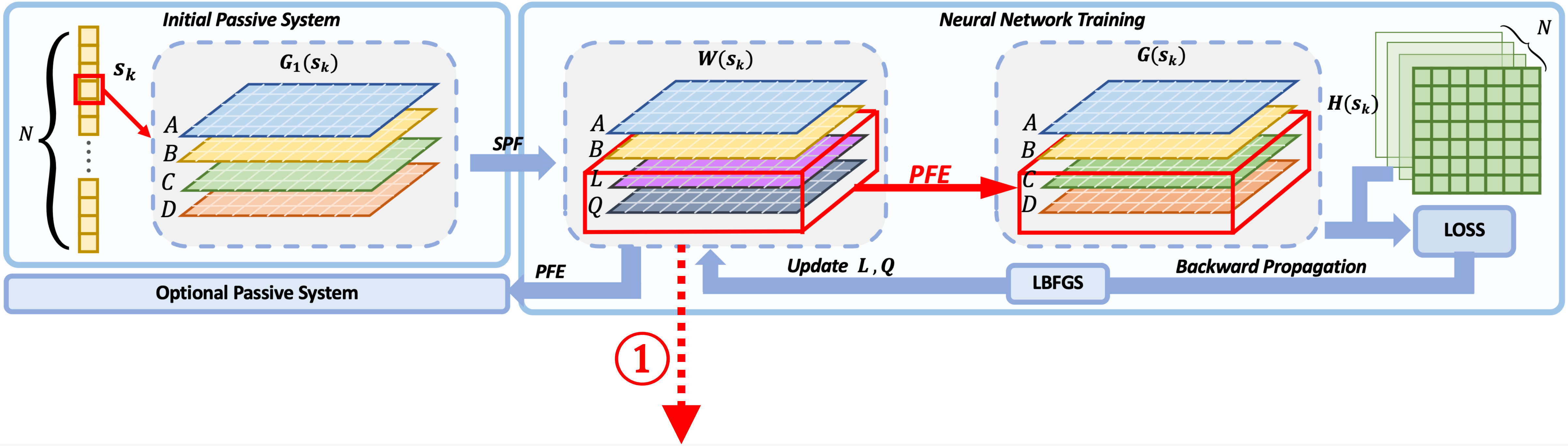
Step 2: Construct the loss function.

Step 3: Use automatic differentiation to perform backpropagation via the chain rule and compute the first-order derivative of the loss function with respect to the network weights $\boldsymbol{x}$.

Step 4: Use the LBFGS method to approximate the inverse of the Hessian matrix.

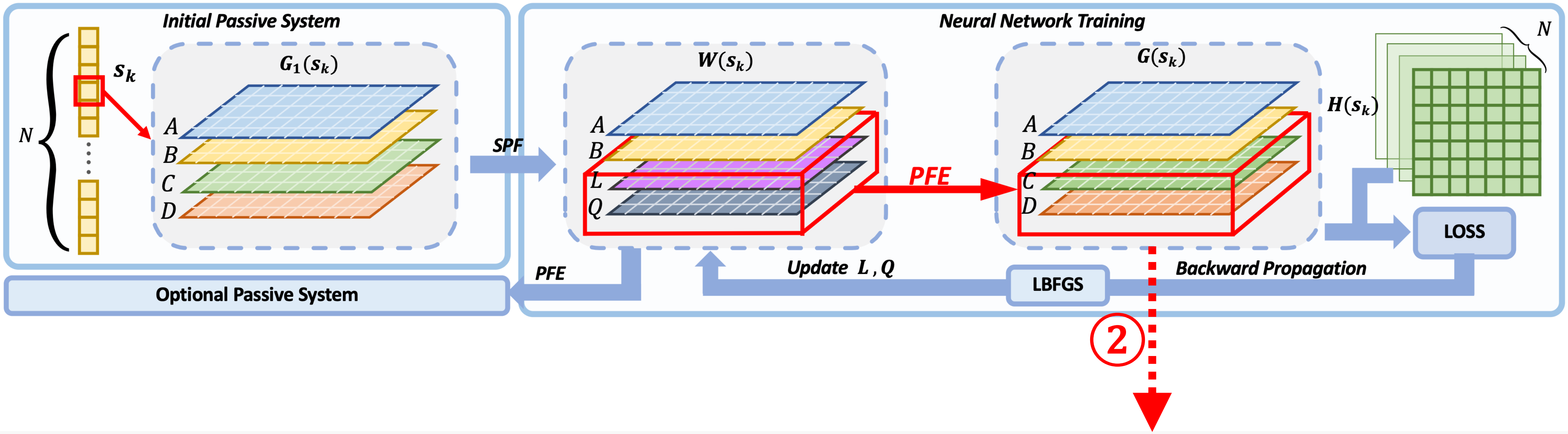
Step 5: Update the parameters L and Q of the initial passive system.

G-SpNN | The Overall Framework



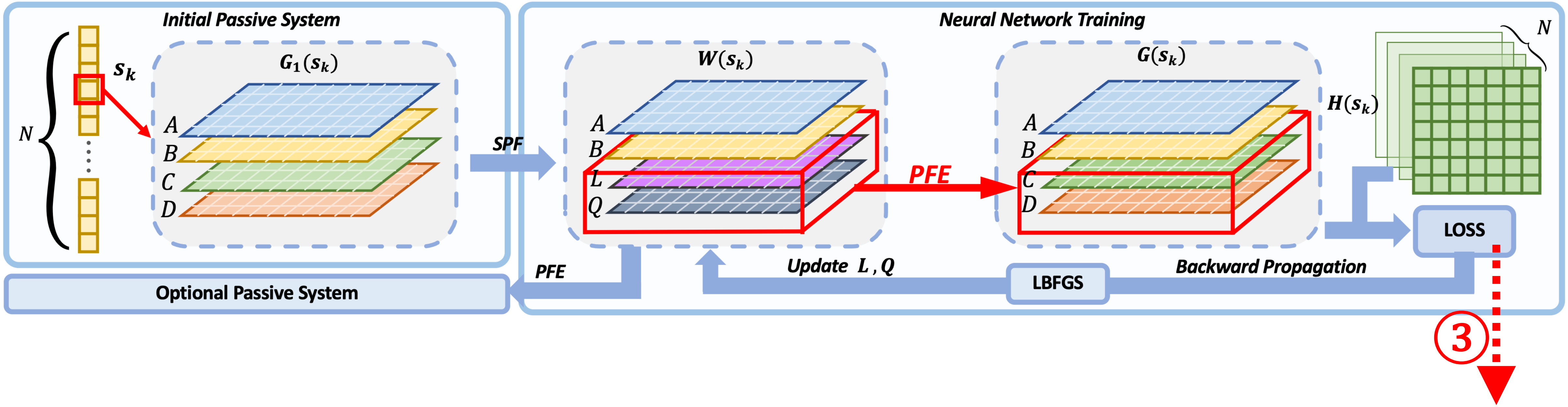
Starting with a given passive system $G_1(sk)$, an unconstrained system $W(sk)$ is first derived by SPF transformation and represented as a layer in the neural network.

G-SpNN | The Overall Framework



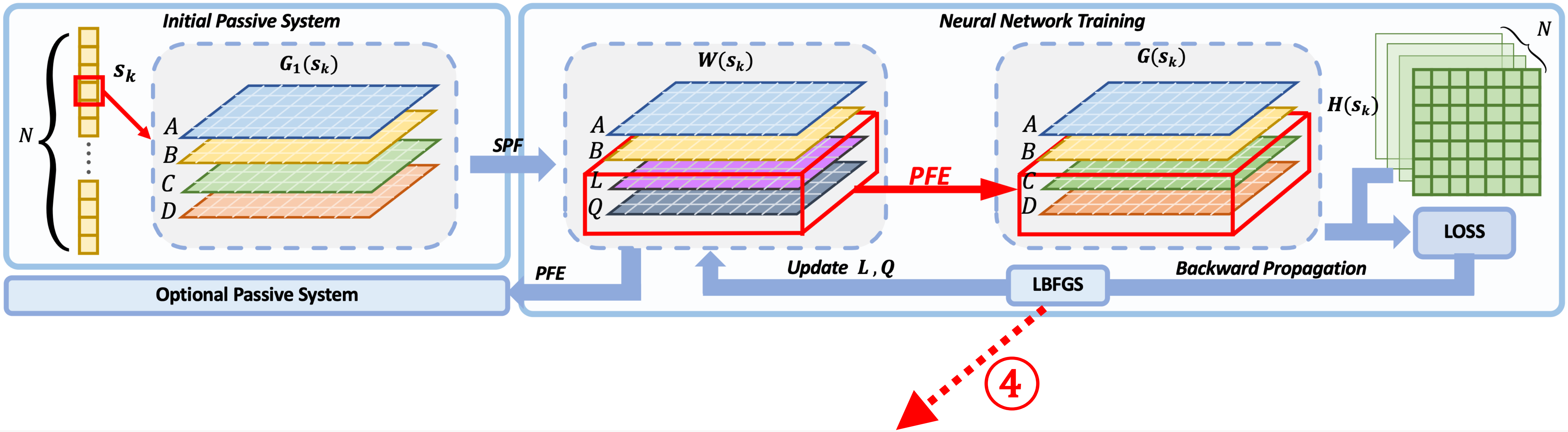
Next, the PFE transformation is applied to generate a new network layer $G(sk)$, corresponding to a passive system. This reformulates the problem as a neural network training task.

G-SpNN | The Overall Framework



During forward propagation, the system, together with the tabulated data $H(sk)$, is used to **compute the loss value**.

G-SpNN | The Overall Framework



For efficient training, the LBFGS method is futher incorporated with backpropagation to **compute gradients and update the network parameters**. Once the iteration stopping criteria are satisfied, the optimized passive system is obtained.

OUTLINE

1. Background
2. G-SpNN
3. Experiment
4. Conclusion

Experiment | Experimental Setup

Experimental Environment

- We implement and test the proposed G-SpNN on an i7-14700KF @5.6GHz CPU with 32GB of memory, and a GeForce RTX 4070 SUPER GPU with 12GB of VRAM.
- G-SpNN is implemented based on PyTorch and compared against the framework DAO, which is implemented in MATLAB and is open-sourced on GitHub.

Touchstone files information

Num	Case	Ports	n	m	N
1	Telluride	11	56	11	258
2	test_5	30	199	30	2000
3	sp125_uniform_2	64	342	64	400
4	CKDIST_TUNEDBUF	64	338	64	2000
5	pll_testcase	138	727	138	300

$H^{m\times m}(s_k)$ $(1 \le k \le N)$ $A^{n\times n}$, $B^{n\times m}$, $C^{m\times n}$, and $D^{m\times m}$

Fitting results of VF and LC

Num	VF			LC		
	SS Error	Time(s)	Passivity	SS Error	Time(s)	Passivity
1	1.52e-4	0.8906	non-passive	5.91e-1	0.1875	passive
2	1.72e-7	17.78	non-passive	6.56e-5	7.64	passive
3	6.29e-4	25.59	non-passive	6.56e-4	6.906	passive
4	4.27e-3	96.89	non-passive	5.16e-3	17.21	passive
5	5.49e-4	75.23	non-passive	5.70e-4	91.79	passive

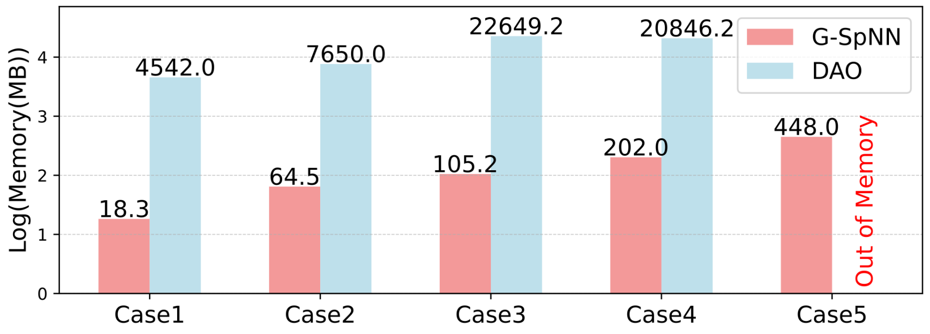
[1]https://www.github.com/yezuochang/pmm

Experiment | Convergence Speed and Memory Usage

Comparison of G-SpNN and DAO. The "–" indicates memory overrun during execution.

Num	Initial Loss	DAO					G-SpNN				
		Time (s)	#Iteration	Final Loss	SS Error	Passivity	Time (s)	#Iteration	Final Loss	SS Error	Passivity
1	6.17e12	17.47	93	4.8656	2.26e-3	passive	94.19	232	4.19e-2	2.46e-4	passive
2	6.50e8	104.65	7	3.86e-2	2.65e-5	passive	97.93	328	3.89e-2	2.65e-5	passive
3	17.36	2116.48	4	17.21	6.51e-4	passive	300.46	803	17.15	6.47e-4	passive
4	2.31e3	3923.14	9	2.24e3	4.97e-3	passive	176.56	469	2.14e3	4.69e-3	passive
5	78.32	–	–	–	–	–	403.35	456	77.62	5.58e-4	passive
Average		1540.44	28	565.53	1.98e-3		214.49	458	446.97	1.23e-3	

➢ **G-SpNN achieves an average speedup of 7.63× compared to DAO.**



➢ **DAO's average memory consumption is 171.3x that of G-SpNN.**

➢ **Keeping the memory usage almost constant with an increasing number of ports.**

Experiment | Convergence Speed and Memory Usage

Comparison of G-SpNN and DAO. The "–" indicates memory overrun during execution.

Num	Initial Loss	DAO					G-SpNN				
		Time (s)	#Iteration	Final Loss	SS Error	Passivity	Time (s)	#Iteration	Final Loss	SS Error	Passivity
1	6.17e12	17.47	93	4.8656	2.26e-3	passive	94.19	232	4.19e-2	2.46e-4	passive
2	6.50e8	104.65	7	3.86e-2	2.65e-5	passive	97.93	328	3.89e-2	2.65e-5	passive
3	17.36	2116.48	4	17.21	6.51e-4	passive	300.46	803	17.15	6.47e-4	passive
4	2.31e3	3923.14	9	2.24e3	4.97e-3	passive	176.56	469	2.14e3	4.69e-3	passive
5	78.32	–	–	–	–	–	403.35	456	77.62	5.58e-4	passive
Average		1540.44	28	565.53	1.98e-3		214.49	458	446.97	1.23e-3	

More Detailed Explanation

For Case 1, although the runtime of DAO appears shorter, the comparison of the final loss and steady-state error shows that DAO actually experiences **pseudo-convergence** and does not reach the optimal solution.

Experiment | Convergence Speed and Memory Usage

Comparison of G-SpNN and DAO. The "–" indicates memory overrun during execution.

Num	Initial Loss	DAO					G-SpNN				
		Time (s)	#Iteration	Final Loss	SS Error	Passivity	Time (s)	#Iteration	Final Loss	SS Error	Passivity
1	6.17e12	17.47	93	4.8656	2.26e-3	passive	94.19	232	4.19e-2	2.46e-4	passive
2	6.50e8	104.65	7	3.86e-2	2.65e-5	passive	97.93	328	3.89e-2	2.65e-5	passive
3	17.36	2116.48	4	17.21	6.51e-4	passive	300.46	803	17.15	6.47e-4	passive
4	2.31e3	3923.14	9	2.24e3	4.97e-3	passive	176.56	469	2.14e3	4.69e-3	passive
5	78.32	–	–	–	–	–	403.35	456	77.62	5.58e-4	passive
Average		1540.44	28	565.53	1.98e-3		214.49	458	446.97	1.23e-3	

More Detailed Explanation

- For Case 3 and Case 4, it should be noted that the DAO method is **forcibly terminated** during the iteration process due to **memory overflow** and does not reach the predefined convergence criterion.
- For Case 5, DAO experiences a **memory overflow** during the first iteration and could not complete the iteration.

Experiment | Convergence Speed and Memory Usage

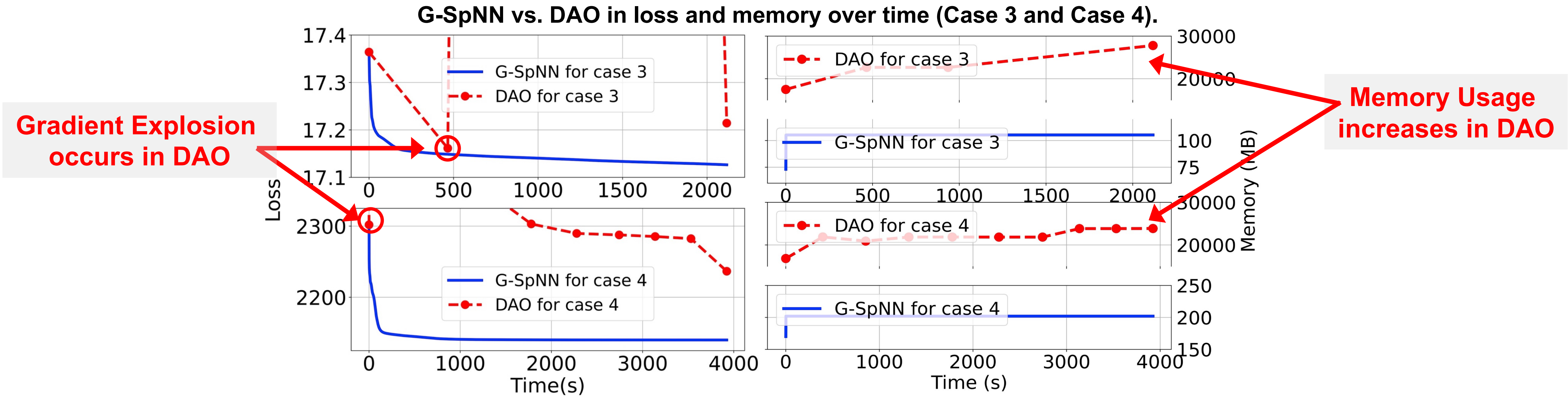
Comparison of G-SpNN and DAO. The "–" indicates memory overrun during execution.

Num	Initial Loss	DAO					G-SpNN				
		Time (s)	#Iteration	Final Loss	SS Error	Passivity	Time (s)	#Iteration	Final Loss	SS Error	Passivity
1	6.17e12	17.47	93	4.8656	2.26e-3	passive	94.19	232	4.19e-2	2.46e-4	passive
2	6.50e8	104.65	7	3.86e-2	2.65e-5	passive	97.93	328	3.89e-2	2.65e-5	passive
3	17.36	2116.48	4	17.21	6.51e-4	passive	300.46	803	17.15	6.47e-4	passive
4	2.31e3	3923.14	9	2.24e3	4.97e-3	passive	176.56	469	2.14e3	4.69e-3	passive
5	78.32	–	–	–	–	–	403.35	456	77.62	5.58e-4	passive
Average		1540.44	28	565.53	1.98e-3		214.49	458	446.97	1.23e-3	

More Detailed Explanation

Due to the high time and space complexity of the DAO method, we limit **the number of poles** in the VF method for cases 3-5 to ensure computational feasibility, which leads to higher SS Error and limits the reduction in loss.

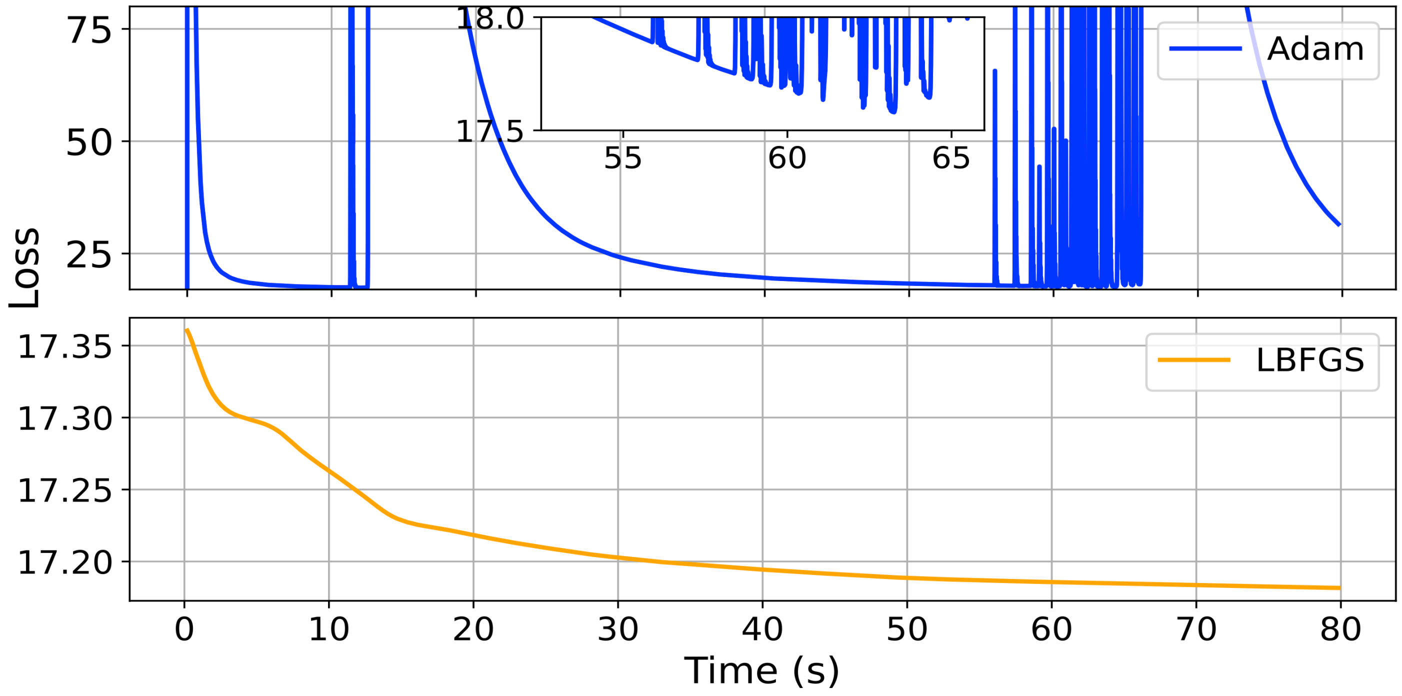
Experiment | Convergence Speed and Memory Usage



- ➢ Using Case 3 and Case 4 as examples, figure shows the **loss and memory usage** variations during iterations.
- ➢ G-SpNN has a **smoother convergence** process with better performance, achieving **lower loss** compared to DAO.

Experiment | Compare with Adam

Loss over time for G-SpNN with Adam and with LBFGS (Case 3).



The LBFGS method enables G-SpNN to progress steadily toward convergence, owing to second-order information guiding more effective update directions.

OUTLINE

1. Background
2. G-SpNN
3. Experiment
4. Conclusion

Conclusion

- Casting the passive macromodeling problem to **neural network training**, thus leveraging **GPU acceleration**.
- Using the LBFGS method to efficiently **approximate the Hessian inverse matrix**, efficiently decrease the memory cost and time overhead. Keeping the memory usage almost constant with an increasing number of ports.
- Experimental results show that G-SpNN not only converges more stably and quickly than DAO, with an average speedup of **7.63×**, its memory usage can be **reduced by two orders of magnitude** in test cases.

Thanks for Listening
Any Questions?

Email: z.jin@zju.edu.cn