Learning incomplete factorization preconditioners for GMRES

Aleix Nieto Juscafresa*, Uppsala University, Sweden
Joint work with Paul Häusner* and Jens Sjölund

*Joint first authors

Numerical Analysis and Scientific Computation with Applications (NASCA26)

June 11, 2026

UPPSALA
UNIVERSITET

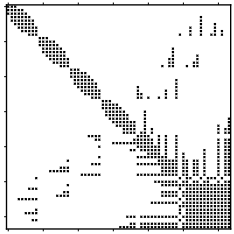
The problem

Solving $Ax = b$

PROBLEM

Solve $\boldsymbol{Ax} = \boldsymbol{b}$ *efficiently*, where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is **large** and **sparse**.

- $\boldsymbol{A}$ is **sparse**: $\text{nnz}(\boldsymbol{A}) \ll n^2$
- Arises in PDE discretizations, optimization problems (Newton, IPMs), Gaussian processes, ...
- Often **many related systems** share sparsity pattern, e.g. parameter studies, optimization iterations, ...



Sparsity pattern

Direct vs. iterative solvers

Direct solvers

Compute the solution via matrix inversion or factorization (LU, QR, Cholesky).

- Accurate and reliable
- **Do not fully exploit sparsity**: factorization introduces **fill-in**, increasing memory and computational cost

Direct vs. iterative solvers

Direct solvers

Compute the solution via matrix inversion or factorization (LU, QR, Cholesky).

- Accurate and reliable
- **Do not fully exploit sparsity**: factorization introduces **fill-in**, increasing memory and computational cost

Iterative solvers (e.g. GMRES)

Refine an initial guess to approximate the solution, efficient when approximations are sufficient.

- Cheap per iteration, low memory
- **Exploit sparsity**: dominated by sparse matrix-vector products; scales with $\mathrm{nnz}(\boldsymbol{A})$

Direct vs. iterative solvers

Direct solvers

Compute the solution via matrix inversion or factorization (LU, QR, Cholesky).

- Accurate and reliable
- **Do not fully exploit sparsity**: factorization introduces **fill-in**, increasing memory and computational cost

Iterative solvers (e.g. GMRES)

Refine an initial guess to approximate the solution, efficient when approximations are sufficient.

- Cheap per iteration, low memory
- **Exploit sparsity**: dominated by sparse matrix-vector products; scales with $\mathrm{nnz}(\boldsymbol{A})$

BOTTLENECK

Iterative solvers scale, but their convergence depends on the **spectrum** of $\boldsymbol{A}$.

GMRES

DEFINITION

For a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{v}$, the k-th **Krylov subspace** $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$ is defined as

$$\operatorname{span}\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{v}\}.$$

[1] Yousef Saad: GMRES: A Generalized Minimal Residual Algorithm for Solving Non-symmetric Linear Systems, 1986.

GMRES

DEFINITION

For a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{v}$, the k-th **Krylov subspace** $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$ is defined as

$$\operatorname{span}\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{v}\}.$$

- GMRES[1] searches for increasingly better approximations within expanding Krylov subspaces
- At step k, it minimizes $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$ over $\boldsymbol{x}_0 + \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$, where $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$

[1] Yousef Saad: GMRES: A Generalized Minimal Residual Algorithm for Solving Non-symmetric Linear Systems, 1986.

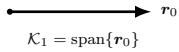
GMRES

DEFINITION

For a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{v}$, the k-th **Krylov subspace** $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$ is defined as

$$\operatorname{span}\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{v}\}.$$

- GMRES[1] searches for increasingly better approximations within expanding Krylov subspaces
- At step k, it minimizes $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$ over $\boldsymbol{x}_0 + \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$, where $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$



[1] Yousef Saad: GMRES: A Generalized Minimal Residual Algorithm for Solving Non-symmetric Linear Systems, 1986.

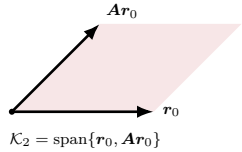
GMRES

DEFINITION

For a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{v}$, the k-th **Krylov subspace** $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$ is defined as

$$\operatorname{span}\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{v}\}.$$

- GMRES[1] searches for increasingly better approximations within expanding Krylov subspaces
- At step k, it minimizes $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$ over $\boldsymbol{x}_0 + \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$, where $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$



[1] Yousef Saad: GMRES: A Generalized Minimal Residual Algorithm for Solving Non-symmetric Linear Systems, 1986.

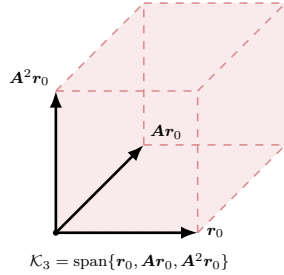
GMRES

DEFINITION

For a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{v}$, the k-th **Krylov subspace** $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$ is defined as

$$\operatorname{span}\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{v}\}.$$

- GMRES[1] searches for increasingly better approximations within expanding Krylov subspaces
- At step k, it minimizes $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$ over $\boldsymbol{x}_0 + \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$, where $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$



[1] Yousef Saad: GMRES: A Generalized Minimal Residual Algorithm for Solving Non-symmetric Linear Systems, 1986.

Why GMRES?

- One of the most widely used Krylov subspace methods for **general** linear systems $\boldsymbol{Ax} = \boldsymbol{b}$
- No structure assumed on $\boldsymbol{A} \rightarrow$ works for arbitrary (non-symmetric) matrices

Why GMRES?

- One of the most widely used Krylov subspace methods for **general** linear systems $\boldsymbol{Ax} = \boldsymbol{b}$
- No structure assumed on $\boldsymbol{A} \rightarrow$ works for arbitrary (non-symmetric) matrices

ASSUMPTIONS ON A

- **Invertible** $\left(\boldsymbol{A} \in \mathrm{GL}(n, \mathbb{R})\right)$
- **Sparse**

Preconditioning

GMRES is broadly applicable, but its performance depends heavily on preconditioning.

Preconditioning

IDEA

Transform the original system $\boldsymbol{Ax} = \boldsymbol{b}$ into an **equivalent stystem**, with improved spectral properties

$$\boldsymbol{AP}^{-1}\boldsymbol{z} = \boldsymbol{b}, \qquad \boldsymbol{x} = \boldsymbol{P}^{-1}\boldsymbol{z}.$$

Preconditioning

IDEA

Transform the original system $\boldsymbol{Ax} = \boldsymbol{b}$ into an **equivalent stystem**, with improved spectral properties

$$\boldsymbol{AP}^{-1}\boldsymbol{z} = \boldsymbol{b}, \qquad \boldsymbol{x} = \boldsymbol{P}^{-1}\boldsymbol{z}.$$

- $\boldsymbol{P} = \boldsymbol{I}$: no preconditioning
- $\boldsymbol{P} = \boldsymbol{A}$: direct method
- **Classical choices:** Jacobi, incomplete factorizations (ILU, IC)

Preconditioning

IDEA

Transform the original system $\boldsymbol{Ax} = \boldsymbol{b}$ into an **equivalent stystem**, with improved spectral properties

$$\boldsymbol{AP}^{-1}\boldsymbol{z} = \boldsymbol{b}, \qquad \boldsymbol{x} = \boldsymbol{P}^{-1}\boldsymbol{z}.$$

- $\boldsymbol{P} = \boldsymbol{I}$: no preconditioning
- $\boldsymbol{P} = \boldsymbol{A}$: direct method
- **Classical choices:** Jacobi, incomplete factorizations (ILU, IC)

But: time to compute the preconditioner $\boldsymbol{P}$

vs.

acceleration obtained in the iterative solver

Preconditioning

IDEA

Transform the original system $\boldsymbol{Ax} = \boldsymbol{b}$ into an **equivalent stystem**, with improved spectral properties

$$\boldsymbol{AP}^{-1}\boldsymbol{z} = \boldsymbol{b}, \qquad \boldsymbol{x} = \boldsymbol{P}^{-1}\boldsymbol{z}.$$

- $\boldsymbol{P} = \boldsymbol{I}$: no preconditioning
- $\boldsymbol{P} = \boldsymbol{A}$: direct method
- **Classical choices:** Jacobi, incomplete factorizations (ILU, IC)

But: time to compute the preconditioner $\boldsymbol{P}$
vs.
acceleration obtained in the iterative solver

Classical preconditioners are ***hand-designed heuristics***!

Research question

Research question

- Many real-world problems give rise to **related** linear systems
- Classical preconditioners (ILU, Jacobi) ignore this structure
- Prior **learned** preconditioners assume **SPD** matrices (CG), *general non-symmetric systems remain open*
- A learned model can **amortize** training cost across solves

(A_1, b_1) (A_2, b_2) (A_3, b_3)

Same sparsity pattern, different coefficients

Research question

- Many real-world problems give rise to **related** linear systems
- Classical preconditioners (ILU, Jacobi) ignore this structure
- Prior **learned** preconditioners assume **SPD** matrices (CG), *general non-symmetric systems remain open*
- A learned model can **amortize** training cost across solves

(A_1, b_1) (A_2, b_2) (A_3, b_3)

Same sparsity pattern, different coefficients

Can we learn a preconditioner from data that accelerates GMRES and competes with classical heuristics on related problems?

Our approach

Learning-to-optimize[2,3]

1. Use machine learning to accelerate numerical solvers by exploiting cross-problem similarity.
2. We view preconditioner construction as an amortized optimization problem.

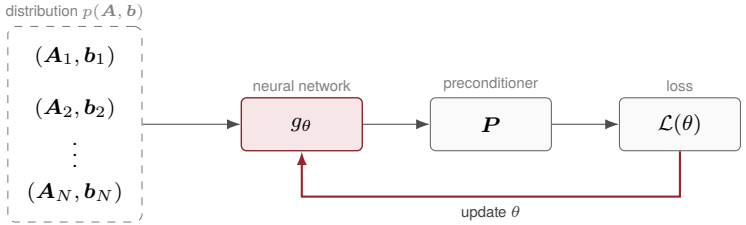
[2] Brandon Amos, Tutorial on amortized optimization, Foundations and Trends in Machine Learning, 2023.

[3] Tianlong Chen et al., Learning to optimize: a primer and a benchmark, JMLR, 2022.

Learning-to-optimize[2,3]

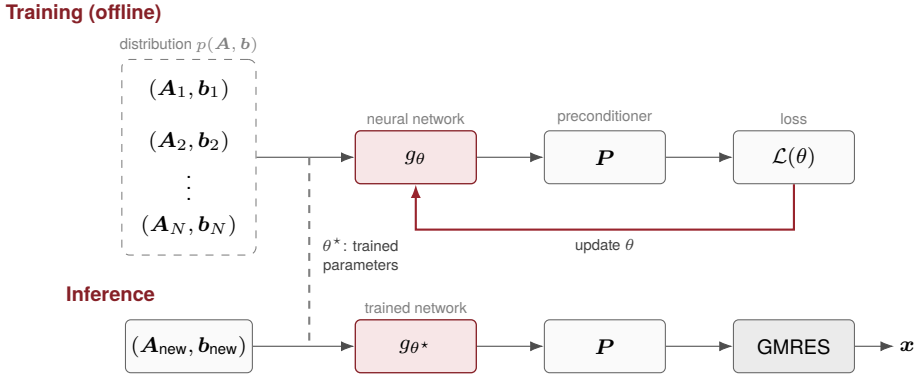
1. Use machine learning to accelerate numerical solvers by exploiting cross-problem similarity.
2. We view preconditioner construction as an amortized optimization problem.

Training (offline)



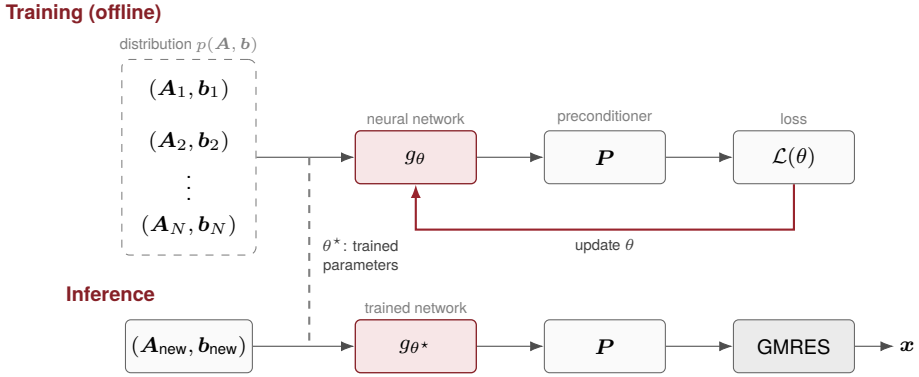
Learning-to-optimize[2,3]

1. Use machine learning to accelerate numerical solvers by exploiting cross-problem similarity.
2. We view preconditioner construction as an amortized optimization problem.



Learning-to-optimize[2,3]

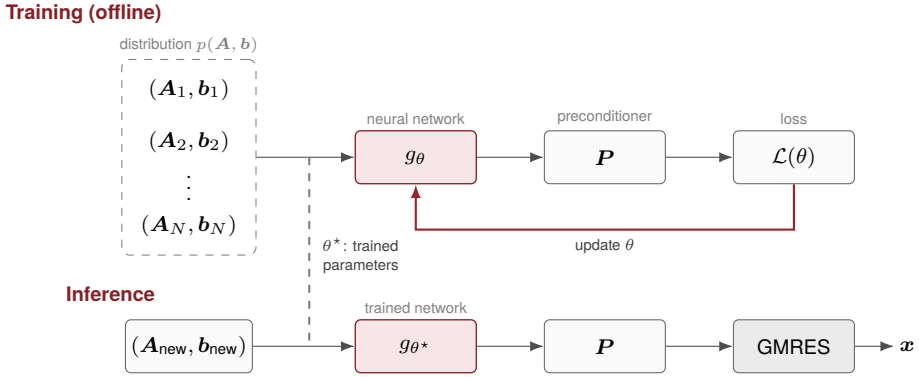
1. Use machine learning to accelerate numerical solvers by exploiting cross-problem similarity.
2. We view preconditioner construction as an amortized optimization problem.



Train **once, offline**, then generate preconditioners for new problems with **low inference cost**.

Learning-to-optimize[2,3]

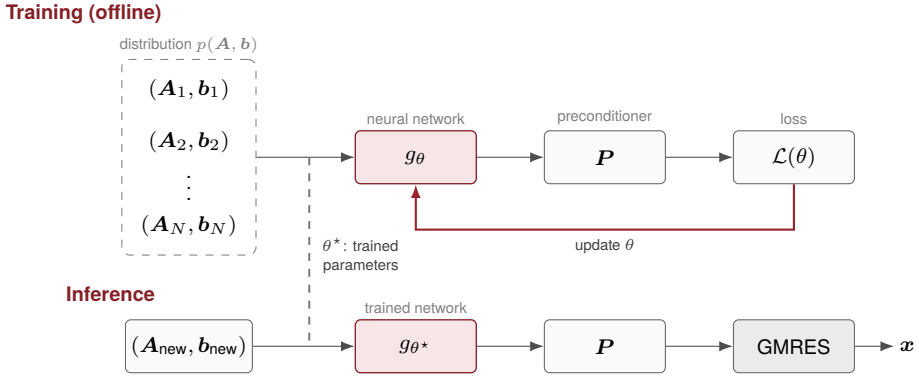
1. Use machine learning to accelerate numerical solvers by exploiting cross-problem similarity.
2. We view preconditioner construction as an amortized optimization problem.



But what is g_θ?

Learning-to-optimize[2,3]

1. Use machine learning to accelerate numerical solvers by exploiting cross-problem similarity.
2. We view preconditioner construction as an amortized optimization problem.



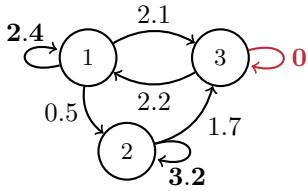
But what is g_θ? $\longrightarrow$ **a graph neural network.**

Graph neural networks and linear algebra

Graph neural networks and linear algebra

- Every matrix $\boldsymbol{A}$ defines a weighted directed graph (**Coates** representation)

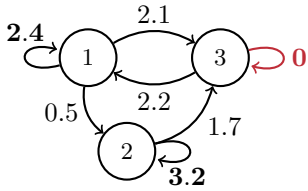
$$\begin{bmatrix} \mathbf{2.4} & 0 & 2.2 \\ 0.5 & \mathbf{3.2} & 0 \\ 2.1 & 1.7 & \mathbf{0} \end{bmatrix}$$



Graph neural networks and linear algebra

- Every matrix $\boldsymbol{A}$ defines a weighted directed graph (**Coates** representation)

$$\begin{bmatrix} \mathbf{2.4} & 0 & 2.2 \\ 0.5 & \mathbf{3.2} & 0 \\ 2.1 & 1.7 & \mathbf{0} \end{bmatrix}$$

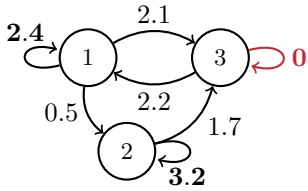


Rows / columns of $\boldsymbol{A}$ $\leftrightarrow$ **nodes**

Graph neural networks and linear algebra

- Every matrix $\boldsymbol{A}$ defines a weighted directed graph (**Coates** representation)

$$\begin{bmatrix} \mathbf{2.4} & 0 & 2.2 \\ 0.5 & \mathbf{3.2} & 0 \\ 2.1 & 1.7 & \mathbf{0} \end{bmatrix}$$



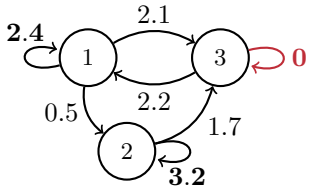
Rows / columns of $\boldsymbol{A}$ $\leftrightarrow$ **nodes**

Non-zero entries $\leftrightarrow$ **edges**

Graph neural networks and linear algebra

- Every matrix $\boldsymbol{A}$ defines a weighted directed graph (**Coates** representation)

$$\begin{bmatrix} \mathbf{2.4} & 0 & 2.2 \\ 0.5 & \mathbf{3.2} & 0 \\ 2.1 & 1.7 & \mathbf{0} \end{bmatrix}$$

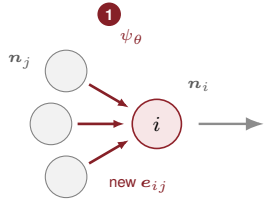


Rows / columns of $\boldsymbol{A}$ $\leftrightarrow$ **nodes**

Non-zero entries $\leftrightarrow$ **edges**

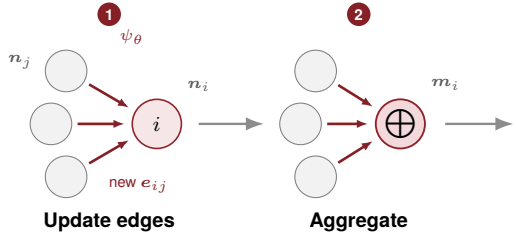
Value of each entry $\leftrightarrow$ **edge feature**

Graph neural networks (GNNs)

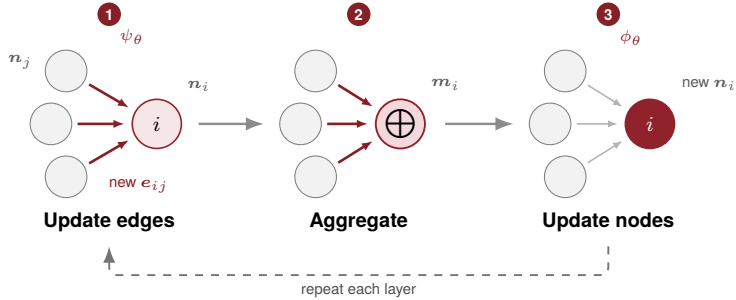


Update edges

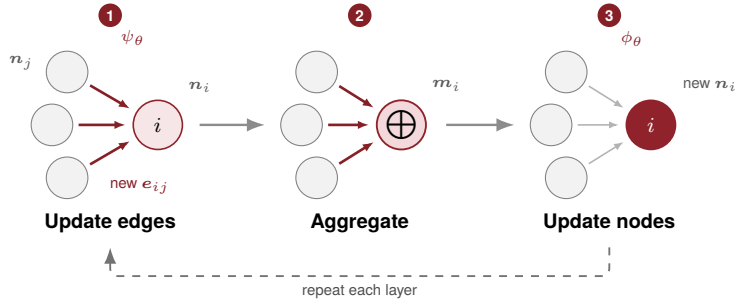
Graph neural networks (GNNs)



Graph neural networks (GNNs)

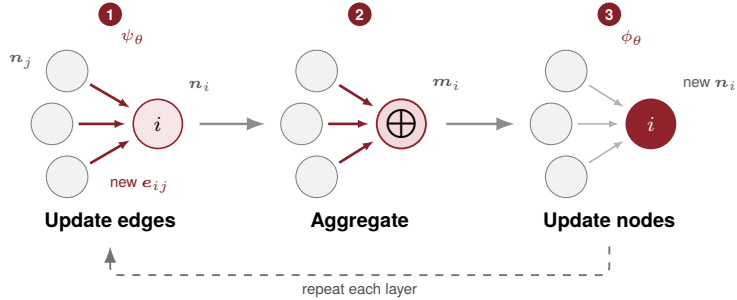


Graph neural networks (GNNs)



A GNN repeats **message passing** each layer, updating *both* edge and node features.
Node features $\boldsymbol{n}_i$ on nodes, edge features $\boldsymbol{e}_{ij}$ on edges (= matrix entries $\boldsymbol{A}_{ij}$).

Graph neural networks (GNNs)



A GNN repeats **message passing** each layer, updating *both* edge and node features.
Node features n_i on nodes, edge features e_{ij} on edges (= matrix entries A_{ij}).

The $\bigoplus$ is **permutation-invariant** — therefore independent of node ordering.

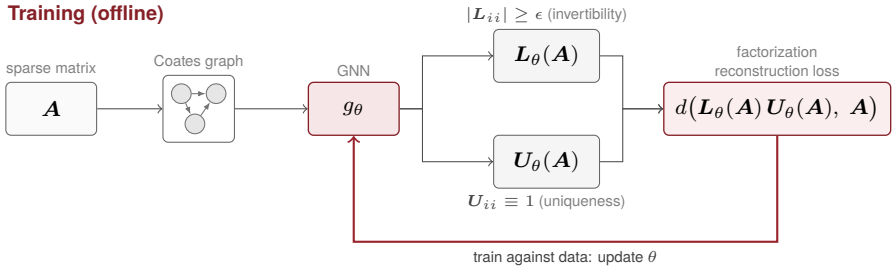
Learning factorization preconditioners

Learned preconditioners

Goal: learn a parameterized mapping g_θ that takes a matrix $\boldsymbol{A}$ and predicts a suitable preconditioner.

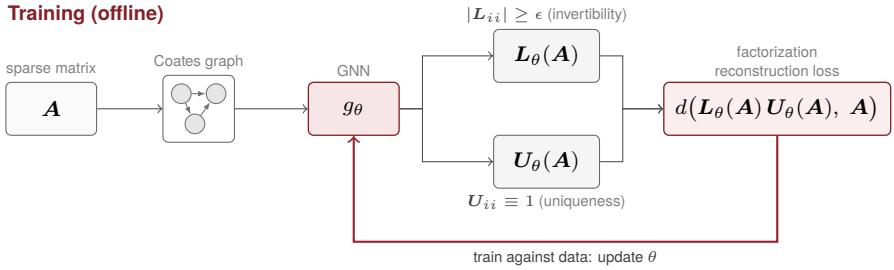
Learned preconditioners

Goal: learn a parameterized mapping g_θ that takes a matrix $\boldsymbol{A}$ and predicts a suitable preconditioner.



Learned preconditioners

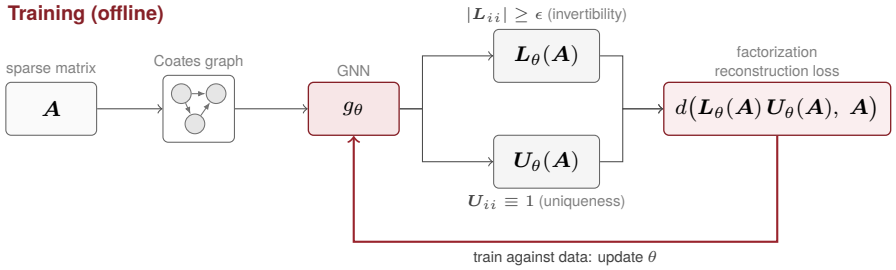
Goal: learn a parameterized mapping g_θ that takes a matrix $\boldsymbol{A}$ and predicts a suitable preconditioner.



- **One GNN, two factors:** edges with $i > j$ form $\boldsymbol{L}$, edges with $i < j$ form $\boldsymbol{U}$
- **Invertibility:** non-zero diagonals make $\boldsymbol{L}, \boldsymbol{U}$ invertible $\Rightarrow \boldsymbol{P} = \boldsymbol{L}\boldsymbol{U}$ is a valid preconditioner
- **Why factorize?** Triangular $\boldsymbol{L}, \boldsymbol{U} \rightarrow$ ***efficient sparse triangular solves***; forward/back-substitution

Learned preconditioners

Goal: learn a parameterized mapping g_θ that takes a matrix $\boldsymbol{A}$ and predicts a suitable preconditioner.



The sparsity of $\boldsymbol{L}$ and $\boldsymbol{U}$ is inherited from $\boldsymbol{A}$ via the GNN architecture[4]

$$\begin{cases} \boldsymbol{L}_{ij} = 0 & \text{if } \boldsymbol{A}_{ij} = 0, \\ \boldsymbol{U}_{ij} = 0 & \text{if } \boldsymbol{A}_{ij} = 0. \end{cases}$$

[4] P. Häusner et al., Neural incomplete factorization: learning preconditioners for CG, TMLR, 2024.

Loss functions

We design loss functions such that $\boldsymbol{P} = \boldsymbol{LU} \approx \boldsymbol{A}$ and analyze their effect on the singular values of the preconditioned system $\boldsymbol{AP}^{-1}$:

$\mathcal{L}_{\max}(\boldsymbol{P}, \boldsymbol{A}) = \|\boldsymbol{P} - \boldsymbol{A}\|_F$

Reduces large singular values.

+ Easy to compute
– Classical preconditioners are good at it

$\mathcal{L}_{\min}(\boldsymbol{P}, \boldsymbol{A}) = \|\boldsymbol{PA}^{-1} - \boldsymbol{I}\|_F$

Encourages larger small singular values.

+ Small singular values affect convergence
– Requires computations with the inverse

Loss functions

We design loss functions such that $\boldsymbol{P} = \boldsymbol{LU} \approx \boldsymbol{A}$ and analyze their effect on the singular values of the preconditioned system $\boldsymbol{AP}^{-1}$:

$\mathcal{L}_{\max}(\boldsymbol{P}, \boldsymbol{A}) = \|\boldsymbol{P} - \boldsymbol{A}\|_F$

Reduces large singular values.

+ Easy to compute
– Classical preconditioners are good at it

$\mathcal{L}_{\min}(\boldsymbol{P}, \boldsymbol{A}) = \|\boldsymbol{P}\boldsymbol{A}^{-1} - \boldsymbol{I}\|_F$

Encourages larger small singular values.

+ Small singular values affect convergence
– Requires computations with the inverse

Our proposal: combine both losses $\longrightarrow$ $\mathcal{L}_{\text{com}} \approx \mathcal{L}_{\max} + \alpha\, \mathcal{L}_{\min}$.

We set $\alpha = 1/7$, chosen on the validation set to balance the two terms; the combined loss $\mathcal{L}_{\text{com}}$ can be unstable to train (sensitive to α).

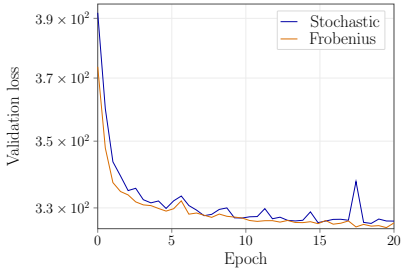
Numerical efficiency

PROBLEM
The Frobenius norm scales computationally poorly.

Unbiased approximation using Hutchinson's trace estimator:

$$\|\boldsymbol{M}\|_F^2 \approx \|\boldsymbol{M}\boldsymbol{w}\|_2^2, \qquad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}).$$

Intuition: estimate a Frobenius norm from a single matrix-vector product, without ever forming $\boldsymbol{M}$.



Efficient model training and inference:

- Training needs only matrix-vector products (or precomputed solves)
- Linear inference-time complexity in the number of non-zero elements

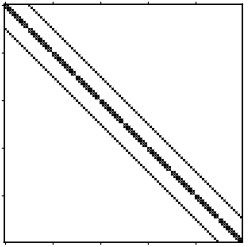
Data

Synthetic dataset inspired by the **Poisson equation**:

- FEM discretization, $n = 2\,500$ ($\approx 50\,000$–$150\,000$ non-zeros)
- Perturb *only* the existing non-zeros with Gaussian noise X_{ij} — the sparsity pattern is preserved:

$$a'_{ij} = \begin{cases} a_{ij} + X_{ij} & \text{if } a_{ij} \neq 0, \\ 0 & \text{if } a_{ij} = 0. \end{cases}$$

Non-symmetry is introduced through coefficient perturbations.



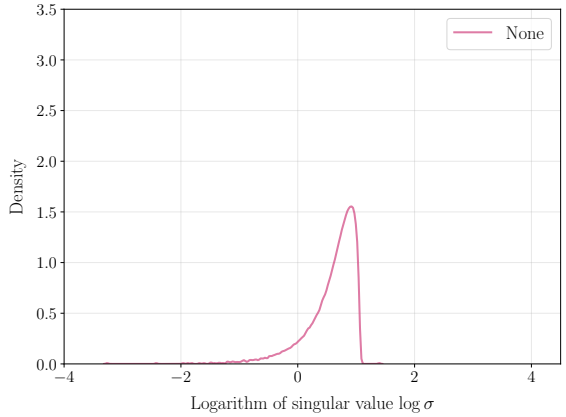
Sparsity pattern across the dataset

Results

Singular value distributions

Small synthetic dataset inspired by Poisson PDE discretization.

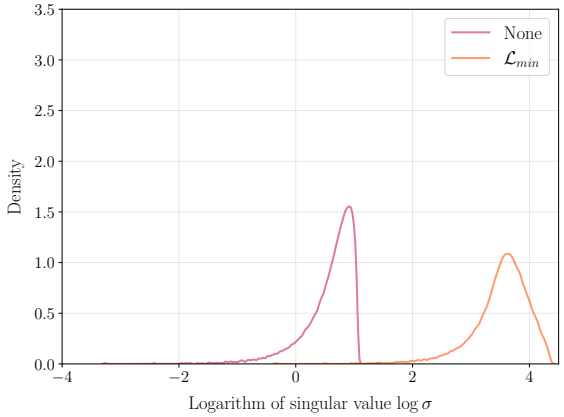
Goal: cluster the singular values and keep them away from zero — better conditioning → fewer iterations



Singular value distributions

Small synthetic dataset inspired by Poisson PDE discretization.

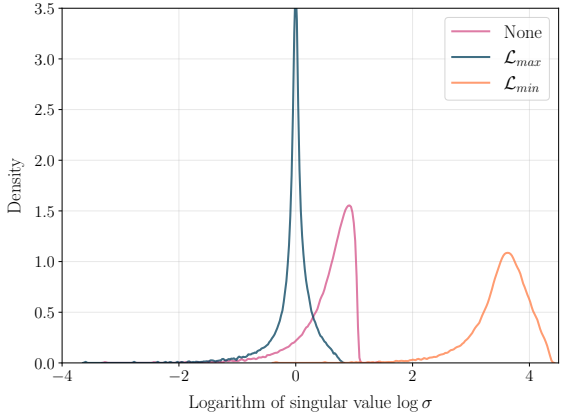
Goal: cluster the singular values and keep them away from zero — better conditioning → fewer iterations



Singular value distributions

Small synthetic dataset inspired by Poisson PDE discretization.

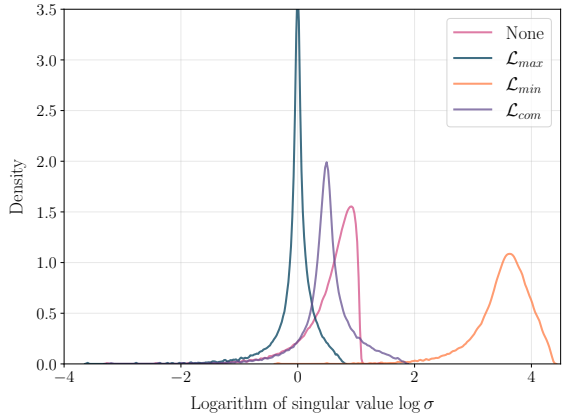
Goal: cluster the singular values and keep them away from zero — better conditioning → fewer iterations



Singular value distributions

Small synthetic dataset inspired by Poisson PDE discretization.

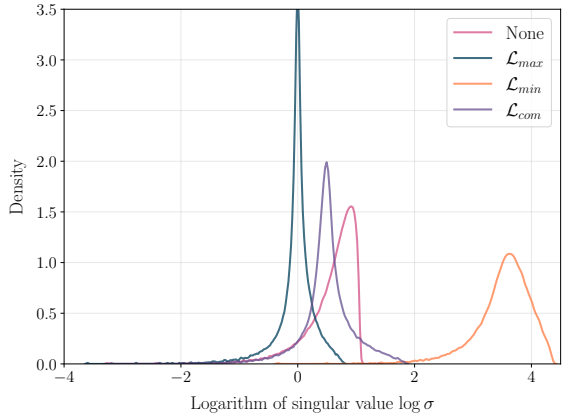
Goal: cluster the singular values and keep them away from zero — better conditioning → fewer iterations



Singular value distributions

Small synthetic dataset inspired by Poisson PDE discretization.

Goal: cluster the singular values and keep them away from zero — better conditioning → fewer iterations



Learned losses reshape the distribution of singular values in different ways.

GMRES convergence performance

Performance in terms of GMRES iterations and time:

	Method	$\sigma_{\min} \uparrow$	$\sigma_{\max} \downarrow$	$\kappa \downarrow$	$\|\boldsymbol{P} - \boldsymbol{A}\|_F \downarrow$	$\|\boldsymbol{P}\boldsymbol{A}^{-1} - \boldsymbol{I}\|_F \downarrow$	Time ↓	Iterations ↓
Precond.	No preconditioner	0.0014	20.70	31 152.94	255.26	1 577.65	30.85	1 153
	Jacobi	0.0003	5.16	31 166.13	205.83	6 319.45	30.12	1 152
	ILU(0)	0.0006	30.32	120 740.90	138.31	3 688.45	**3.33**	**413**
	Learned IC	0.0006	7.58	27 405.57	143.23	3 719.47	12.84	692
Loss	$\mathcal{L}_{\max}$	0.0007	**5.00**	**16 139.46**	**88.76**	3 261.23	3.67	437
	$\mathcal{L}_{\min}$	**1.1375**	20 318.88	37 030.72	287.62	**50.05**	24.41	1 054
	$\mathcal{L}_{\text{com}}$	0.0017	52.92	66 691.71	197.82	1 240.60	3.42	418

GMRES convergence performance

Performance in terms of GMRES iterations and time:

	Method	$\sigma_{\min} \uparrow$	$\sigma_{\max} \downarrow$	$\kappa \downarrow$	$\|\boldsymbol{P} - \boldsymbol{A}\|_F \downarrow$	$\|\boldsymbol{P}\boldsymbol{A}^{-1} - \boldsymbol{I}\|_F \downarrow$	Time ↓	Iterations ↓
Precond.	No preconditioner	0.0014	20.70	31 152.94	255.26	1 577.65	30.85	1 153
	Jacobi	0.0003	5.16	31 166.13	205.83	6 319.45	30.12	1 152
	ILU(0)	0.0006	30.32	120 740.90	138.31	3 688.45	**3.33**	**413**
	Learned IC	0.0006	7.58	27 405.57	143.23	3 719.47	12.84	692
Loss	$\mathcal{L}_{\max}$	0.0007	**5.00**	**16 139.46**	**88.76**	3 261.23	3.67	437
	$\mathcal{L}_{\min}$	**1.1375**	20 318.88	37 030.72	287.62	**50.05**	24.41	1 054
	$\mathcal{L}_{\text{com}}$	0.0017	52.92	66 691.71	197.82	1 240.60	3.42	418

Our loss functions are working as expected, pushing the singular values in the intended direction.

GMRES convergence performance

Performance in terms of GMRES iterations and time:

	Method	$\sigma_{\min} \uparrow$	$\sigma_{\max} \downarrow$	$\kappa \downarrow$	$\|\boldsymbol{P} - \boldsymbol{A}\|_F \downarrow$	$\|\boldsymbol{P}\boldsymbol{A}^{-1} - \boldsymbol{I}\|_F \downarrow$	Time ↓	Iterations ↓
Precond.	No preconditioner	0.0014	20.70	31 152.94	255.26	1 577.65	30.85	1 153
	Jacobi	0.0003	5.16	31 166.13	205.83	6 319.45	30.12	1 152
	ILU(0)	0.0006	30.32	120 740.90	138.31	3 688.45	**3.33**	**413**
	Learned IC	0.0006	7.58	27 405.57	143.23	3 719.47	12.84	692
Loss	$\mathcal{L}_{\max}$	0.0007	**5.00**	**16 139.46**	**88.76**	3 261.23	3.67	437
	$\mathcal{L}_{\min}$	**1.1375**	20 318.88	37 030.72	287.62	**50.05**	24.41	1 054
	$\mathcal{L}_{\text{com}}$	0.0017	52.92	66 691.71	197.82	1 240.60	**3.42**	**418**

The learned preconditioners achieve the best singular values clustering and condition number, while already matching classical methods (ILU(0)) on GMRES iterations and runtime.

Conclusion

Conclusion

- A **GNN architecture** that outputs a **sparse, non-singular** incomplete LU factorization
- Studied how different **loss functions** shape the **"spectrum"** of the preconditioned system $\boldsymbol{AP}^{-1}$
- Learned preconditioners nearly match GMRES convergence with ILU(0) on related systems

Future work

UPPSALA UNIVERSITET

Future work

- Take the right-hand side $\boldsymbol{b}$ into account during training
- Apply to new problem domains such as Newton's method, IPMs, GPs
- Learn additional heuristics, such as the sparsity pattern of the preconditioner directly
- Combine data-driven and classical algorithms[5]

The development of data-driven preconditioners has just begun!

[5] Vladislav Trifonov et al., Learning from linear algebra: a graph neural network approach to preconditioner design for conjugate gradient solvers, arXiv, 2024.

Thank you for your attention!

Aleix Nieto Juscafresa, Uppsala University

E-mail: aleix.nieto.juscafresa@it.uu.se

Supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation