MOX-Report No. 13/2026

Neural Preconditioning via Krylov Subspace Geometry

Dimola, N.; Coclite, A.; Zunino, P.

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it https://mox.polimi.it

Neural Preconditioning via Krylov Subspace Geometry

Nunzio Dimola[1], Alessandro Coclite[2*], Paolo Zunino[1]

[1]MOX, Dipartimento di Matematica, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milano, 20133, Italy.

[2]Dipartimento di Ingegneria Elettrica e dell'Informazione (DEI), Politecnico di Bari, Via Re David 200, Bari, 70125, Italy.

*Corresponding author(s). E-mail(s): alessandro.coclite@poliba.it;
Contributing authors: nunzio.dimola@polimi.it; paolo.zunino@polimi.it;

We propose a geometry-aware strategy for training neural preconditioners tailored to parametrized linear systems arising from the discretization of mixed-dimensional partial differential equations (PDEs). Such systems are typically ill-conditioned due to embedded lower-dimensional structures and are solved using Krylov subspace methods. Our approach yields an approximation of the inverse operator employing a learning algorithm consisting of a two-stage training framework: an initial static pretraining phase, based on residual minimization, followed by a dynamic fine-tuning phase that incorporates solver convergence dynamics into the training process via a novel loss functional. This dynamic loss is defined by the principal angles between the residuals and the Krylov subspaces. It is evaluated using a differentiable implementation of the Flexible GMRES algorithm, which enables backpropagation through both the Arnoldi process and Givens rotations. The resulting neural preconditioner is explicitly optimized to enhance early-stage convergence and reduce iteration counts across a family of 3D–1D mixed-dimensional problems exhibiting geometric variability in the 1D domain. Numerical experiments show that our solver-aligned approach significantly improves convergence rate, robustness, and generalization.

Keywords: Mixed-Dimensional PDEs; Krylov Subspace Methods; Neural Preconditioning; Differentiable Linear Algebra; Geometry-Aware Optimization.

MSC 2020: 65F08, 65F10, 68T07, 65Y20.

1 Introduction

The efficient numerical solution of parameter-dependent linear systems arising from discretized partial differential equations (PDEs) is a critical computational challenge in scientific computing. This task becomes particularly demanding in multi-query applications, such as uncertainty quantification or control problems, where the underlying PDE must be solved repeatedly for different parameter values. In this context, mixed-dimensional PDEs [1–4] (to cite only a few examples), which describe coupled operators defined on domains of varying dimensions, present additional challenges due to their inherent ill-conditioning, complex geometries, and heterogeneous material properties. To address these large, sparse linear systems, iterative solvers, notably Krylov subspace methods such as the Conjugate Gradient (CG) and Generalized Minimal Residual (GMRES) algorithm, are widely employed [5]. However, the efficiency of these methods strongly depends on the spectral properties and condition number of the system matrices, which often deteriorate with increasing problem size or parameter variations. This fundamental limitation necessitates the extensive use of preconditioning techniques [6], which transform the original system into an equivalent one more amenable to rapid convergence.

In this work, we focus on a family of linear systems arising from the discretizations of parameter-dependent mixed-dimensional PDEs

$$\mathsf{A}^{\mu}\mathsf{u}^{\mu} = \mathsf{b}^{\mu}, \quad \mu \in \mathcal{P},$$

where $\mathcal{P}$ denotes the parameter space encoding geometric variations of embedded lower-dimensional structures within a three-dimensional domain. The central challenge addressed here is the design of effective, parameter-dependent preconditioners that can handle significant geometric variability while ensuring robust solver convergence. Traditional preconditioning strategies, including stationary methods (e.g., Jacobi, Gauss-Seidel), incomplete factorizations, sparse approximate inverses (SPAI) [7], multigrid (MG) [8], and domain decomposition (DD) methods [9], have been developed to improve convergence. For mixed-dimensional problems, physics-based block and algebraic multigrid methods tailored for metric-perturbed coupled problems have proven instrumental for efficient resolution [10–12]. More recently, neural network-based strategies have gained significant attention for their ability to learn effective preconditioners directly from data [13, 14]. These approaches often leverage operator learning frameworks, such as DeepONet [15], to approximate inverse operators or to construct transfer operators, or utilize the spectral bias of neural networks [16] to efficiently address low-frequency error components.

Our prior work [17] introduced a novel neural preconditioner specifically for a class of 3D-1D mixed-dimensional PDEs in which we propose to minimize a residual-based loss functional of the form:

$$\mathcal{L}_{\text{static}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{\mathsf{b}^{\mu} \in \mathcal{B}} \left\| \mathsf{b}^{\mu} - \mathsf{A}^{\mu} \mathcal{N}_{\theta}(\mathsf{b}^{\mu}, \mu) \right\|^{2},$$

where $\mathcal{B}$ is a suitable collection of vectors, and $\mathcal{N}_\theta$ is a nonlinear operator parameterized by the neural network weights θ. This method showcased the potential of unsupervised operator learning via convolutional neural networks (U-Nets [18]) to create preconditioners that can adapt to different shapes of a 1D manifold without the need for retraining and can effectively scale to various mesh resolutions. However, despite its simplicity, this formulation has several limitations, including a lack of interpretability in terms of solver dynamics and poor alignment with performance metrics relevant to iterative methods. To overcome these limitations, we propose an enhanced learning strategy—herein referred to as the *dynamic strategy*—that explicitly incorporates solver convergence dynamics into the training process. The dynamic residual-based loss directly reflects the geometric and convergence properties intrinsic to Krylov subspace methods by employing a differentiable implementation (in the sense of automatic differentiation [19]) of the Flexible GMRES (FGMRES) algorithm [20]. Within this framework, the neural preconditioner is influenced by the residual at each iteration. Specifically, we define a novel dynamic loss functional as

$$\mathcal{L}^{(M)}_{\text{dynamic}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{\mathsf{b}^\mu \in \mathcal{B}} \sum_{i=1}^{M} |s_i(\theta, \mathsf{b}^\mu)| \,,$$

where quantities $s_i(\theta, \mathsf{b}^\mu)$, which are closely linked to the algorithm convergence behavior, offer a geometric characterization of the Krylov subspace generated by the problem $(\mathsf{A}^\mu, \mathsf{b}^\mu)$ and preconditioned by the operator $\mathcal{N}_\theta$. The integer M defines the optimization window, focusing the training on the initial iterations of the solver. Given this construction, we have a structured and solver-aligned loss functional. In this context, our approach bridges numerical linear algebra with deep learning by leveraging modern automatic differentiation tools (e.g., PyTorch). This framework enables the efficient gradient-based optimization of neural preconditioners through backpropagation across the solver's iterative computations. The combined static-dynamic training strategy enables the neural preconditioner to adapt effectively to complex geometry variations characteristic of mixed-dimensional PDEs, resulting in significantly improved solver performance across diverse problem instances.

The remainder of the paper is organized as follows. Section 2 introduces the mixed-dimensional PDE model and its finite element discretization. Section 3 describes the static pretraining strategy. Section 4 presents foundational concepts and geometric results from Krylov subspace theory that support our dynamic optimization. Sections 5 and 6 develop the dynamic loss formulation and detail the training algorithms using differentiable linear algebra. Sections 7 and 8 report numerical results and conclude with final remarks and future directions.

2 Problem Setting

Throughout the manuscript, we adopt a consistent notation scheme to distinguish between different mathematical objects. Calligraphic letters such as $\mathcal{P}$ and $\mathcal{H}$ are used to denote spaces or sets. The notation $|\mathcal{P}|$ refers to the cardinality of the set $\mathcal{P}$. In this context, an exception to this rule is the neural preconditioner $\mathcal{N}_\theta$. Finite-dimensional vectors are represented as v or w, and the same convention applies to matrices, such as A and B. Given a matrix A, its entry located at row i and column j is written as $[\mathsf{A}]_{ij}$. The inner product between two elements is denoted by $\langle \cdot, \cdot \rangle$. We denote vector norms using the Euclidean norm $\|\cdot\|$.

We consider the numerical solution of a family of parametrized mixed-dimensional elliptic problems representative of a broader class of PDEs defined over coupled domains of different dimensions. Specifically, we study a 3D-1D coupled problem set in a three-dimensional domain $\Omega \subset \mathbb{R}^3$ and a one-dimensional manifold $\Lambda \subset \Omega$, thoroughly analyzed in [21] and typically modeling slender structures. The continuous mixed-dimensional formulation reads

$$\begin{aligned} -\nabla \cdot (k_\Omega \nabla u_\Omega) + \sigma_\Omega u_\Omega + 2\pi\epsilon \left(\mathcal{T}_\Lambda u_\Omega - u_\Lambda\right) \delta_\Lambda = 0, &\quad \text{in } \Omega, \\ -\partial_s (k_\Lambda \, \partial_s u_\Lambda) + 2\pi\epsilon \left(u_\Lambda - \mathcal{T}_\Lambda u_\Omega\right) = 0, &\quad \text{on } \Lambda, \end{aligned} \tag{1}$$

subject to appropriate boundary conditions. Here, u_Ω and u_Λ are the unknown fields defined in Ω and Λ, respectively; k_Ω, σ_Ω, and k_Λ are physical coefficients; and $\epsilon > 0$ controls the strength of the coupling between the domains. The operator $\mathcal{T}_\Lambda$ denotes the coupling of Ω to Λ, typically implemented as a cross-sectional average in tubular neighborhoods of Λ. The term δ_Λ indicates that the coupling acts on Ω as a singular source supported on Λ.

We discretize (1) using the Galerkin projection on a (broken) finite element space $V_h = V_h^\Omega \times V_h^\Lambda$ of dimension $N_h = N_h^\Omega + N_h^\Lambda$, being $N_h^\Omega = \dim(V_h^\Omega)$ and $N_h^\Lambda = \dim(V_h^\Lambda)$, which are chosen depending on the characteristics of the problem at hand. Such a discretization leads to a family of linear systems of the form

$$\mathsf{A}^\mu \mathsf{u}^\mu = \mathsf{b}^\mu, \quad \mu \in \mathcal{P},$$

where μ indexes the parameter space $\mathcal{P}$ encoding geometric variations of the embedded structure Λ. The discrete solution vector u^μ and right-hand side b^μ are partitioned into 3D and 1D components

$$\mathsf{u}^\mu = \begin{pmatrix} \mathsf{u}_{\mu,0} \\ \mathsf{u}_{\mu,1} \end{pmatrix}, \quad \mathsf{b}^\mu = \begin{pmatrix} 0 \\ \mathsf{f}_{\mu,1} \end{pmatrix}.$$

The system matrix A^μ has the following block structure

$$\mathsf{A}^\mu = \begin{pmatrix} \mathsf{K}_{\mathsf{h},00} + 2\pi\epsilon \mathsf{M}^\mu_{\mathsf{h},00} & 2\pi\epsilon \mathsf{M}^\mu_{\mathsf{h},01} \\ 2\pi\epsilon \mathsf{M}^\mu_{\mathsf{h},10} & \mathsf{K}^\mu_{\mathsf{h},11} + 2\pi\epsilon \mathsf{M}^\mu_{\mathsf{h},11} \end{pmatrix},$$

where the blocks $\mathsf{K}_{\mathsf{h,ij}}$ correspond to discretizations of the elliptic operators, and $\mathsf{M}^\mu_{\mathsf{h,ij}}$ represent parameter-dependent coupling terms.

The principal computational difficulty lies in solving this system efficiently across multiple instances $\mu \in \mathcal{P}$. In particular, the dominant cost is associated with solving the 3D subproblem

$$\mathsf{A}^\mu_{\mathsf{h},00} \mathsf{u}_{\mu,0} = \mathsf{b}^\mu_{\mathsf{h},0},$$

where

$$\mathsf{A}^{\mu}_{\mathsf{h},00} := \mathsf{K}_{\mathsf{h},00} + 2\pi\epsilon\mathsf{M}^{\mu}_{\mathsf{h},00}, \quad \mathsf{b}^{\mu}_{\mathsf{h},0} := -2\pi\epsilon\mathsf{M}^{\mu}_{\mathsf{h},01}\mathsf{u}_{\mu,1}.$$

Note that the relevant right-hand sides for this reduced system belong to

$$\mathcal{B} := \left\{ -2\pi\epsilon\mathsf{M}^{\mu}_{\mathsf{h},01}\mathsf{x} \;\middle|\; \mu \in \mathcal{P},\ \mathsf{x} \in \mathbb{R}^{N_{\Lambda,h}} \text{ such that } (\mathsf{K}^{\mu}_{\mathsf{h},11} + 2\pi\epsilon\mathsf{M}^{\mu}_{\mathsf{h},11})\mathsf{x} = \mathsf{f}_{\mu,1} \right\}.$$

This set $\mathcal{B}$ characterizes the structure of the right-hand sides that arise during the elimination of the 1D variables from the coupled system and plays a key role in training neural preconditioners.

3 Static Pretraining for Neural Preconditioning

In this section, we introduce the static pretraining phase for neural preconditioners, which serves as the initial step in a two-stage learning process for solving families of parametrized linear systems arising from mixed-dimensional PDEs. The primary goal of this phase is to obtain a good initialization for the neural network parameters θ using an unsupervised learning approach, without requiring knowledge of the true solution vectors. The methodology described here has already been proposed in [17].

From now on, we consider a family of parameter-dependent linear systems of the form

$$\mathsf{A}^{\mu}_{\mathsf{h},00}\mathsf{u}_{\mu,0} = \mathsf{b}^{\mu}_{\mathsf{h},0}, \quad \mu \in \mathcal{P}\,;$$

Given the absence of ambiguity, we drop all subscripts to simplify notation:

$$\mathsf{A}^{\mu}\mathsf{u}^{\mu} = \mathsf{b}^{\mu}, \quad \mu \in \mathcal{P}\,; \tag{2}$$

where $\mathcal{P}$ denotes a compact parameter space encoding geometrical or physical variations of the problem. For each $\mu \in \mathcal{P}$, $\mathsf{A}^{\mu} \in \mathbb{R}^{N\times N}$ is a symmetric positive-definite (SPD) matrix resulting from the discretization of the mixed-dimensional PDEs, and $\mathsf{b}^{\mu} \in \mathbb{R}^{N}$ is the corresponding right-hand side. Notably, although the problem at hand involves an SPD matrix, we do not exploit this property in the solver for two main reasons: *i*) Eq. (2) represents a specific instance of a broader class of operators that may lack symmetry, such as those arising in transport-dominated regimes; *ii*) learned methods like our neural preconditioner generally break symmetry unless such structure is explicitly enforced in the architecture (however, this is a capability for which no efficient implementation is currently available).

The neural preconditioner is defined as a nonlinear operator $\mathcal{N}_\theta : \mathbb{R}^N \times \mathcal{P} \to \mathbb{R}^N$, represented by a neural network with parameters θ. The aim is to train $\mathcal{N}_\theta$ so that it approximately inverts the operator A^{μ} over a subset of right-hand sides relevant to the problem. To this end, we minimize the following residual-based, unsupervised loss functional

$$\mathcal{L}_{\text{static}}(\theta) = \frac{1}{|\mathcal{P}|}\sum_{j=1}^{|\mathcal{P}|}\frac{1}{|\mathcal{K}_{\mu_j}|}\sum_{v\in\mathcal{K}_{\mu_j}}\left\|\mathsf{v} - \mathsf{A}^{\mu_j}\mathcal{N}_\theta(\mathsf{v},\mu_j)\right\|^2,$$

where $\{\mu_j\}_{j=1}^{|\mathcal{P}|} \subset \mathcal{P}$ are sampled parameters, and $\mathcal{K}_{\mu_j} \subset \mathbb{R}^N$ is a training set of normalized right-hand sides associated with parameter μ_j. Each training set $\mathcal{K}_{\mu_j}$ includes both physics-informed and spectrally enriched right-hand sides, and is defined as

$$\mathcal{K}_{\mu_j} = \left\{ \frac{\mathsf{b}^{\mu_\mathsf{j}}}{\|\mathsf{b}^{\mu_\mathsf{j}}\|} \right\} \cup \mathcal{D}_{\mu_j},$$

where $\mathsf{b}_{\mu_\mathsf{j}}$ is a canonical forcing term associated with μ_j, and $\mathcal{D}_{\mu_j}$ is a data augmentation set constructed to enrich the spectral content of the training data.
To construct the set $\mathcal{D}_{\mu_j}$, we sample unrelated random unit vectors $\mathsf{r}_{\mathsf{k,h}}^{\mu_\mathsf{j}} \in \mathbb{S}^{N_h-1} \subset \mathbb{R}^{N_h}$ uniformly from the unit hypersphere. This is achieved in practice by drawing a Gaussian random vector $\mathsf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{N_h})$ and projecting it onto the sphere

$$\mathsf{r}_{\mathsf{k,h}}^{\mu_\mathsf{j}} = \frac{\mathsf{v}}{\|\mathsf{v}\|}.$$

This procedure generates samples that are uniformly distributed on $\mathbb{S}^{N_h-1}$ due to the rotational invariance of the standard multivariate normal distribution; since the distribution of v is isotropic, the normalization $\mathsf{v}/\|\mathsf{v}\|$ yields a uniform distribution over the unit sphere. As such, the resulting augmented set is given by

$$\mathcal{D}_{\mu_j} = \left\{ \mathsf{r}_{\mathsf{0,h}}^{\mu_\mathsf{j}}, \, \mathsf{r}_{\mathsf{1,h}}^{\mu_\mathsf{j}}, \, \dots \right\}.$$

This spectral enrichment ensures that the neural preconditioner learns to handle both low- and high-frequency components of the operator spectrum, which are essential for effective preconditioning in mixed-dimensional settings. The minimization of the empirical risk $\mathcal{L}_{\text{static}}$ is performed using standard gradient-based optimization methods such as Adam. Importantly, this static pretraining phase is fully unsupervised—it does not rely on access to exact solutions u^{μ}, thereby significantly reducing the computational cost associated with the offline training. The resulting neural preconditioner $\mathcal{N}_\theta$ provides a robust initial approximation of the inverse action of A^{μ} on representative right-hand sides. This initialization is then refined during a subsequent solver-integrated fine-tuning stage, where the learned operator is further adapted to enhance the convergence of Krylov subspace methods such as GMRES.

4 Geometric Aspects of Krylov Subspace Methods

This section outlines the geometric foundations of Krylov subspace methods, following the rigorous treatment in [22], with emphasis on the convergence-related quantities that motivate our dynamic loss formulation for neural preconditioning.

We consider the linear system:

$$\mathsf{A}\mathsf{u} = \mathsf{f}, \quad \mathsf{A} \in \mathbb{R}^{n \times n}, \quad \mathsf{f} \in \mathbb{R}^{n},$$

with an initial guess $\mathsf{u_0}$ and residual $\mathsf{r_0} = \mathsf{f} - \mathsf{Au_0}$. Krylov subspace methods generate iterates $\mathsf{u_j} \in \mathsf{u_0}+\mathcal{C}_j$, where the *correction subspace* is defined by,

$$\mathcal{C}_j = \mathcal{K}_j(\mathsf{A}, \mathsf{r_0}) := \mathrm{span}\{\mathsf{r_0}, \mathsf{Ar_0}, \ldots, \mathsf{A^{j-1}r_0}\}.$$

Letting $\mathcal{W}_j := \{\mathsf{x} \,|\, \mathsf{x} = \mathsf{Ay},\ \mathsf{y} \in \mathcal{C}_j\}$ denote the image subspace of the correction subspace, with respect to matrix A, we get that in minimal residual Krylov subspace methods, the residual at iteration j is enforced to be orthogonal to $\mathcal{W}_j$, i.e.,

$$\mathsf{r_j} = \mathsf{f} - \mathsf{Au_j} \perp \mathcal{W}_j\,.$$

The iterate $\mathsf{u_j} = \mathsf{C_j}\nu^*_\mathsf{j}$ is thus obtained by solving the minimization problem:

$$\nu^*_\mathsf{j} = \arg\min_{\nu\in\mathbb{R}^\mathsf{j}} \|\mathsf{r_0} - \mathsf{AC_j}\nu\|\,,$$

where $\mathsf{C_j} \in \mathbb{R}^{n\times j}$ is a column basis matrix for correction space $\mathcal{C}_j$.

As expected, the convergence of Krylov methods is intimately connected to the geometry of residual vectors relative to Krylov subspaces. Indeed, given a nonzero vector x and a subspace $\mathcal{Y}$, define the principal angle between them by

$$\angle(\mathsf{x}, \mathcal{Y}) := \arcsin\left(\frac{\|(I - P_\mathcal{Y})\mathsf{x}\|}{\|\mathsf{x}\|}\right)\,,$$

where $P_\mathcal{Y}$ denotes the orthogonal projector onto $\mathcal{Y}$; then $s_j := \sin(\angle(\mathsf{r_{j-1}}, \mathcal{W}_j))$ denote the sine of the angle between $\mathsf{r_{j-1}}$ and $\mathcal{W}_j$. Given that, by construction, the subspace sequence $\{\mathcal{W}_j\}$ is nested:

$$\{0\} \equiv \mathcal{W}_0 \subset \mathcal{W}_1 \subset \ldots \subset \mathcal{W}_{j-1} \subset \mathcal{W}_j\,.$$

It can be proved that the residual norm satisfies the following recurrence:

$$\|\mathsf{r_j}\| = s_j\|\mathsf{r_{j-1}}\| = s_j s_{j-1}\cdots s_1\|\mathsf{r_0}\|\,, \tag{3}$$

so that, fast convergence *corresponds* to small angle sines $|s_j| \ll 1$ at early iterations.

The key computational tool in Krylov methods is the Arnoldi process, which constructs an orthonormal basis $\mathsf{V_j}$ of $\mathcal{K}_j(\mathsf{A}, \mathsf{r_0})$ that satisfies the matrix relation

$$\mathsf{AV_j} = \mathsf{V_{j+1}}\widetilde{\mathsf{H}}_\mathsf{j}\,,$$

with $\widetilde{\mathsf{H}}_\mathsf{j} \in \mathbb{R}^{(j+1)\times j}$ an upper Hessenberg matrix. Geometric information is hidden in the Hessenberg matrix, so that, given an orthogonal matrix $\mathsf{Q_j} \in \mathbb{R}^{j+1,j+1}$ such that

$$\mathsf{Q_j}\widetilde{\mathsf{H}}_\mathsf{j} = \begin{bmatrix}\mathsf{R_j}\\ 0\end{bmatrix}.$$

We have that

$$s_m = \left| \frac{[\mathsf{Q_j}]_{j+1,1}}{[\mathsf{Q_{j-1}}]_{j,1}} \right| .$$

If one constructs the orthogonal transformation as a sequence of Givens rotations $\mathsf{Q_j} = \mathsf{G_j G_{j-1}} \cdots \mathsf{G_1}$, where each $\mathsf{G_k}$ annihilates subdiagonal elements of $\widetilde{\mathsf{H}}_\mathsf{j}$, we can recover s_j in $\mathsf{G_j}$ skew-symmetric part.

Given all stated above, consider a θ-parametrized, possibly non-linear, preconditioner $\mathcal{N}_\theta : \mathbb{R}^n \to \mathbb{R}^n$ and the associated flexible variant GMRES, so that the associated Krylov subspace generated in the iterative process results

$$\mathcal{K}_j^{\text{FGMRES}} = \text{span}\{\mathsf{r_0}, \mathsf{A} \circ \mathcal{N}_\theta(\mathsf{r_0}), \dots, (\mathsf{A} \circ \mathcal{N}_\theta)^{j-1}(\mathsf{r_0})\} ,$$

and the Arnoldi relation generalizes to

$$\mathsf{AZ_j} = \mathsf{V_{j+1}} \widetilde{\mathsf{H}}_\mathsf{j} ,$$

where $\mathsf{Z_j} = [\mathcal{N}_\theta(\mathsf{v_0}), \dots, \mathcal{N}_\theta(\mathsf{v_{j-1}})]$, with $\{\mathsf{v_l}\}_{\mathsf{l=1}}^{\mathsf{j}}$ the orthonormal basis vectors generated during the iterations. As in the classical case, the sine angles s_j can still be computed from $\widetilde{\mathsf{H}}_\mathsf{j}$ via Givens rotations, but, given the introduction of $\mathcal{N}_\theta$ in the nested subspace construction process, they become differentiable functions of the operator parameters θ, thus we write $s_{j,\theta}$.

These geometric insights serve as the foundation for the dynamic loss functional at the heart of our neural preconditioning framework. The idea is to search for a set of optimal parameters $\theta^\star$ that discourages excessively large values of $|s_{j,\theta}|$. In doing so, the dynamic loss effectively acts to reduce the degree of misalignment between the residual vectors and the Krylov search directions. This alignment is crucial, as it allows the optimization process to become aware of —and adapt to— the solver's behavior, ultimately promoting convergence patterns that are consistent with those observed in FGMRES.

5 Dynamic Loss Functional Formulation

We now formalize the geometric intuition developed in the previous section into a differentiable optimization framework suitable for training μ-parametrized nonlinear preconditioners via gradient-based methods. Our approach is centered on the construction of a loss functional that quantifies and penalizes the misalignment between the residual vector and the Krylov subspace generated during the solution process, to accelerate convergence. Such a misalignment is measured using the sine of the angle between the residual and the Krylov subspace at each iteration. This metric is embedded directly into the training procedure and is evaluated dynamically during the execution of the Flexible GMRES (FGMRES) algorithm, wherein each Krylov basis vector is modified by a nonlinear, parameterized preconditioner $\mathcal{N}_\theta : \mathbb{R}^N \times \mathcal{P} \to \mathbb{R}^N$. For the parameter instance, μ, consider M step of the generalised Arnoldi procedure,

$$\mathsf{A}^\mu \mathsf{Z}_\mathsf{M}^\mu = \mathsf{V}_{\mathsf{M+1}}^\mu \widetilde{\mathsf{H}}_\mathsf{M}^\mu ,$$

with

$$\mathsf{Z}^{\mu}_{\mathsf{M}} = [\mathcal{N}_\theta(\mathsf{v}_0, \mu), \, \ldots, \mathcal{N}_\theta(\mathsf{v}_\mathsf{j}, \mu), \, \ldots, \mathcal{N}_\theta(\mathsf{v}_{\mathsf{M}-1}, \mu)].$$

Each preconditioned vector $\mathsf{A}^\mu \mathcal{N}_\theta(\mathsf{v}_j, \mu)$ undergoes a Gram–Schmidt orthogonalization process, preserving the nested structure of the Krylov subspaces.

At iteration M, we define $\{s^\mu_1, \ldots, s^\mu_M\}$ the sequence of sines of the angles between residuals and Krylov subspaces generated up to that point. This sequence is a by-product of the sequence of M Givens rotations, $\mathsf{G}_\mathsf{M}\mathsf{G}_{\mathsf{M}-1} \ldots \mathsf{G}_2\mathsf{G}_1$ that orthogonalize $\widetilde{\mathsf{H}}^\mu_\mathsf{M}$. We encapsulate this process of sine extraction from the Givens rotations orthogonalizing sequence with the shorthand notation $\mathbf{AG}_M$.

Since the basis vectors depend on θ via the action of $\mathcal{N}_\theta$, the entire sequence $\{s^\mu_j\}_{j=1}^M \equiv \{s^\mu_{j,\theta}\}_{j=1}^M$ becomes differentiable with respect to the model parameters. The goal is to find a parameter vector $\theta^\star$ such that

$$\prod_{j=1}^{M} |s^\mu_{j,\theta^\star}| \ll 1 \,,$$

for all $\mu \in \mathcal{P}$, indicating rapid convergence of the residual norm over the first M iterations for the whole problem family.

The recursive nature of the Arnoldi-Givens process induces a computational graph where each variable depends on the previous ones. This structure forms a Directed Acyclic Graph (DAG), in which nodes represent intermediate quantities (e.g., inner products, basis vectors, rotation angles), and edges encode standard algebraic operations, such as those involved in performing orthogonalization or applying Givens rotations. The parameter dependence enters the graph via the preconditioner $\mathcal{N}_\theta$, which affects every step of the $\mathbf{AG}_M$ iterative procedure. In Fig. 1, we visualize this computational structure, emphasizing the role of θ in shaping the algorithm's trajectory.

This computational structure enables seamless application of reverse-mode automatic differentiation. In particular, frameworks such as PyTorch automatically construct and traverse the computational graph, allowing efficient evaluation of the gradient [23]

$$\delta\theta \mapsto \left\{\partial s^\mu_{j,\theta}/\partial\theta\right\}_{j=1}^{M} \,,$$

which can be used to train the neural preconditioner by minimizing a suitably defined loss functional. The complete learning procedure is outlined in Algorithm 1. The average $\langle\cdot\rangle_\mu$ denotes the empirical mean over a batch of parameterized linear systems $(\mathsf{A}^\mu, \mathsf{b}^\mu)\, \mu \in \mathcal{P}_{\text{train}} \subset \mathcal{P}$ or $\mu \in \mathcal{P}_{\text{test}} \subset \mathcal{P}$, depending on the context,

$$\langle \mathsf{x}^\mu \rangle_\mu := \frac{1}{|\mathcal{P}|} \sum_{\mu \in \mathcal{P}} \mathsf{x}^\mu \,.$$

This formulation allows the learned preconditioner to generalize across the entire problem class. The dynamic loss functional is thus defined as

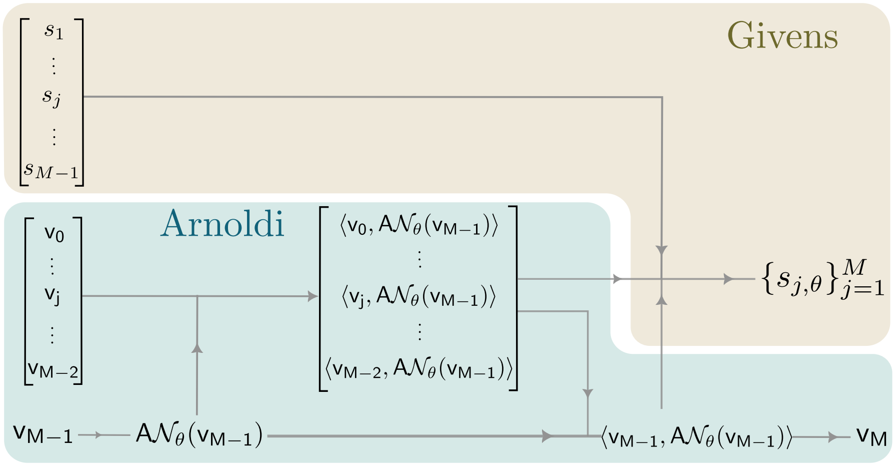
Givens
s_1
s_j
s_{M-1}
Arnoldi
v_0
v_j
$\mathsf{v}_{\mathsf{M}-2}$
$\langle \mathsf{v}_0, \mathsf{A}\mathcal{N}_\theta(\mathsf{v}_{\mathsf{M}-1})\rangle$
$\langle \mathsf{v}_\mathsf{j}, \mathsf{A}\mathcal{N}_\theta(\mathsf{v}_{\mathsf{M}-1})\rangle$
$\langle \mathsf{v}_{\mathsf{M}-2}, \mathsf{A}\mathcal{N}_\theta(\mathsf{v}_{\mathsf{M}-1})\rangle$
$\{s_{j,\theta}\}_{j=1}^M$
$\mathsf{v}_{\mathsf{M}-1}$
$\mathsf{A}\mathcal{N}_\theta(\mathsf{v}_{\mathsf{M}-1})$
$\langle \mathsf{v}_{\mathsf{M}-1}, \mathsf{A}\mathcal{N}_\theta(\mathsf{v}_{\mathsf{M}-1})\rangle$
v_M

Fig. 1 Computational graph corresponding to the M-th iteration of the Arnoldi-Givens process ($\mathbf{AG}_M$). Dependencies on the learnable parameters θ are introduced through the nonlinear preconditioner $\mathcal{N}_\theta$. For brevity, we suppress the dependence on parameters μ and θ when superfluous.

Algorithm 1 Training a Neural Preconditioner via Krylov Subspace Alignment

```
 1: Select linear systems $(\mathsf{A}^\mu, \mathsf{b}^\mu)$, for $\mu \in \mathcal{P}_{\text{train}}$
 2: Initialize the neural operator $\mathcal{N}_\theta$ and the optimizer
 3: for each training epoch do
 4:     Compute the initial residual $\mathsf{r}_0^\mu$                                   ⎫
 5:     for $j = 1$ to $M$ do                                                           ⎪
 6:         Construct $\mathsf{A}^\mu \mathsf{Z}_\mathsf{j}^\mu = \mathsf{V}_{\mathsf{j}+1}^\mu \widetilde{\mathsf{H}}_\mathsf{j}^\mu$          ⎬ $\mathbf{AG}_M(\mathsf{A}^\mu, \mathsf{r}_0^\mu, \mathcal{N}_\theta)$.
 7:         Orthogonalize $\widetilde{\mathsf{H}}_\mathsf{j}^\mu$ via Givens rotations; compute $s_{j,\theta}^\mu$   ⎭
 8:     end for
 9:     Evaluate the loss $\mathcal{L}(\theta) = \sum_{j=1}^M \left\langle |s_{j,\theta}^\mu| \right\rangle_\mu$
10:     Compute $\nabla_\theta \mathcal{L}$ using backpropagation
11:     Update $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$
12: end for
```

$$\mathcal{L}_{\text{dynamic}(M)}(\theta) := \frac{1}{M} \sum_{j=1}^{M} \left\langle |s_{j,\theta}^\mu| \right\rangle_{\mu \in \mathcal{P}_{\text{train}}},$$

where the sine series $\{s_{j,\theta}^\mu\}$ is evaluated *dynamically* in the training process, i.e. whenever parameter is updated, $\theta \leftarrow (\theta - \eta \nabla_\theta \mathcal{L})$ (line 11 in Algorithm 5), the Arnoldi-Givens algorithm is called to obtain the new series

$$\mathbf{AG}_M : (\mathsf{A}^\mu, \mathsf{r}_0^\mu, \mathcal{N}_{(\theta - \eta \nabla_\theta \mathcal{L})}) \longmapsto \{s_{j,(\theta-\eta\nabla_\theta\mathcal{L})}^\mu\}_{j=1}^M.$$

As $\mathcal{L}_{\text{dynamic}(M)} \approx \mathcal{O}(M)$, it provides a robust surrogate for minimizing residual misalignment over the first M iterations. Moreover, the inequality,

$$\prod_{j=1}^{M} \langle |s_j^{\mu}| \rangle_{\mu} \leq \left(\mathcal{L}_{\text{dynamic}(M)}(\theta) \right)^{M} ,$$

relates the geometric mean of the sine values to the average loss, offering a theoretical guarantee that minimizing $\mathcal{L}_{\text{dynamic}(M)}$ leads to improved convergence behavior.

The parameter M plays a central role in balancing optimization depth and computational efficiency. While larger values of M provide a more accurate reflection of long-term solver dynamics, they also result in deeper computational graphs and increased memory usage. In practice, moderate values of M (e.g., $5 \leq M \leq 10$) are sufficient to achieve meaningful learning without incurring prohibitive computational costs. It is important to note that, unlike matrix inversion, preconditioning does not require a unique operator. The space of valid preconditioners is large and diverse, allowing the learning process to exploit non-uniqueness to improve convergence. In the limiting case $M = 1$, the loss effectively trains $\mathcal{N}_\theta$ to approximate the inverse of A^{μ}, which is typically insufficient for practical tolerances. Larger values of M enable iterative refinement and allow the network to learn effective strategies for accelerating convergence across a wide range of systems.

6 Training Strategy for the Neural Preconditioner

The training of the nonlinear preconditioner $\mathcal{N}_\theta$ follows a two-phase procedure designed to balance computational efficiency with solver-aware optimization. First, we perform an unsupervised pretraining stage based on a static loss functional that avoids unrolling the Krylov solver. Then, the network undergoes fine-tuning using a dynamic loss that incorporates geometric information derived from the iterative process. This two-phase strategy reflects a trade-off between generalization and specialization. The static phase enables the network to learn a coarse but broadly applicable approximation of the inverse operator, leveraging only residual information without requiring ground-truth solutions. Data augmentation-based randomized perturbations enrich the spectral content of the training set, promoting robustness across parameter variations. On the other hand, the subsequent dynamic phase injects solver-specific information into the training loop by minimizing the absolute value of sine angles $s_{j,\theta}$ between residuals and Krylov subspaces. These angles govern convergence behavior in GMRES and its flexible variant, and their minimization aligns the learned preconditioner with the subspace geometry of the iterative process. This fine-tuning step optimizes the early-stage convergence rate and reduces iteration counts in practice. By decoupling representation learning from solver dynamics, this hybrid approach ensures both structural adaptability and performance-driven refinement. The static loss captures coarse inverse structure, while the dynamic loss promotes alignment with Krylov geometry, resulting in a preconditioner that is both generalizable and solver-aware.

6.1 Neural Architecture: A Multi-Level U-Net Design

We briefly recall here the definition of the architecture chosen for $\mathcal{N}_\theta$, previously discussed in [17]. To define the hypothesis space $\mathcal{H}$, we adopt a neural operator $\mathcal{N}_\theta$ instantiated via a U-Net architecture of depth L, denoted by U_L. This architecture is well-suited for mixed-dimensional PDEs due to its ability to capture multiscale spatial features. Then, we set $\mathcal{H} = \{U_L(\cdot\,;\theta)\}_{\theta\in\mathbb{R}^t;\, L\in\mathbb{N}^+}$, so that the neural preconditioner is defined as $\mathcal{N}_\theta \equiv U_L(\cdot\,;\theta)$.

The U-Net architecture comprises an encoder,

$$\Phi = \Phi_0 \circ \Phi_1 \circ \cdots \circ \Phi_{L-1} : \mathbb{R}^{c_{\text{in}}\times n} \to \mathbb{R}^{c_b \times m}\,,$$

and a decoder,

$$\Psi = \Psi_{L-1} \circ \cdots \circ \Psi_0 : \mathbb{R}^{c_b\times m} \to \mathbb{R}^{c_{\text{out}}\times n}.$$

Given an input tensor $\mathbf{X} = [\mathbf{X}_1 \mid \ldots \mid \mathbf{X}_{c_{\text{in}}}]$ with $\mathbf{X}_i \in \mathbb{R}^{n_1\times\cdots\times n_d}$, the recursive structure of the U-Net is defined as

$$\begin{aligned}
&U_0(\mathbf{X}) := \Psi_0 \circ \Phi_0(\mathbf{X}),\\
&U_j(\mathbf{X}) := \Psi_j\left([U_{j-1}\circ\Phi_j(\mathbf{X}) \mid \Phi_j(\mathbf{X})]\right), \quad j>0\,,
\end{aligned}$$

where $[\,\cdot \mid \cdot\,]$ denotes channel-wise concatenation (skip connection), enhancing both gradient propagation and spatial information retention.

Each convolutional block Φ_j and Ψ_j applies c_{out} filters $\mathbf{k}_{i,j}$ to the c_{in} input channels,

$$\mathbf{Y}_i = \sum_{j=1}^{c_{\text{in}}} \mathbf{k}_{i,j} * \mathbf{X}_j, \qquad i = 1,\ldots,c_{\text{out}}\,,$$

with a discrete convolution defined by

$$(\mathbf{k}_{i,j} * \mathbf{X}_j)(\mathbf{p}) = \sum_{\mathbf{q}\in\mathbb{Z}^d} \mathbf{k}_{i,j}(\mathbf{q})\,\mathbf{X}_j(\mathbf{p}-\mathbf{q}).$$

To comply with convolutional formats, the raw inputs $\{\mathbf{x}_i\}_{i=1}^{c_{\text{in}}} \subset \mathbb{R}^{N_h}$ are reshaped into a tensor

$$\mathbf{x}_1,\ldots,\mathbf{x}_{c_{\text{in}}} \mapsto \mathbf{X} \in \mathbb{R}^{c_{\text{in}}\times n_1\times\cdots\times n_d}, \qquad N_h = n_1\cdot\cdots\cdot n_d.$$

This choice limits the architecture to structured tensor-product meshes; generalization to unstructured domains may require mesh-aware models [24].

The specific architecture used in our experiments is detailed in Table 1. It includes downsampling (via max pooling) and upsampling (via transposed convolution) layers.

Layer (type)	Input size	Output size	Input channels	Output channels	# Parameters
Conv.	21^3	21^3	2	32	13,840
Conv. + Max Pooling	21^3	10^3	32	64	55,296
Conv. + Max Pooling	10^3	5^3	64	128	221,184
Conv.	5^3	5^3	128	128	442,368
Transposed Conv.	5^3	10^3	128	64	286,784
Transposed Conv.	10^3	21^3	64	32	71,912

Table 1 Architecture of the three-level U-Net U_3. The total number of trainable parameters is approximately 10^6.

6.2 Phase 1: Static Pretraining via Residual Loss

In the static pretraining phase, we minimize a residual-based loss functional that does not require unrolling the Krylov solver

$$\mathcal{L}_{\text{static}}(\theta) = \frac{1}{|\mathcal{P}_{\text{train}}|} \sum_{j=1}^{|\mathcal{P}_{\text{train}}|} \frac{1}{|\mathcal{K}_{\mu_j}|} \sum_{\mathsf{v}\in\mathcal{K}_{\mu_j}} \left\|\mathsf{v} - \alpha^{-1}\mathsf{A}^{\mu_j}\mathcal{N}_\theta(\mathsf{v},\mu_j)\right\|^2 ,$$

Here, α is a normalization factor based on the average RMS of the matrix diagonals

$$\alpha := \left(\frac{1}{|\mathcal{P}_{\text{train}}|} \sum_{j=1}^{|\mathcal{P}_{\text{train}}|} \frac{1}{\sqrt{N_h}} \|\text{diag}(\mathsf{A}^{\mu_j})\|_2\right)^{-1} .$$

Inputs consist of the right-hand side vector and the discrete parameter field d^{μ_j}, reshaped to match the U-Net structure. The training configuration is summarized in Table 2. We will refer to the neural preconditioner resulting from minimization of $\mathcal{L}_{\text{static}}$ as $\mathcal{N}_\theta^{\text{st}}$.

Training	1D Graphs	Samples	Validation	1D Graphs	Samples
Size	480	2400	Size	120	600
Mini-Batch Size		5	Epochs		250
Learning Rate		10^{-3}	Scheduler		Decaying

Table 2 Pretraining dataset and parameters. Data augmentation increases the number of samples by a factor of five.

6.3 Phase 2: Dynamic Fine-Tuning via Krylov Geometry

To integrate solver-specific information, we fine-tune the pre-trained model using the dynamic loss functional aligned with the geometry of Krylov subspaces

$$\mathcal{L}_{\text{dynamic}(M)}(\theta) = \frac{1}{M}\sum_{j=1}^{M} \left\langle |s_{j,\theta}^{\mu}| \right\rangle_{\mu\in\mathcal{P}_{\text{train}}} , \tag{4}$$

where the sequence $\{s_{j,\theta}^{\mu}\}$ of the angles sine between residuals and Krylov subspaces comes from the on-the-fly application of $\mathbf{AG}_M$ algorithm and the average $\langle\cdot\rangle_\mu$ is computed over mini-batches of systems $(\mathsf{A}^\mu, \mathsf{b}^\mu)$. We use $M = 10$ to prioritize early-stage convergence. The fine-tuning configuration is reported in Table 3. We will refer to the neural preconditioner resulting from minimization of $\mathcal{L}_{\text{dynamic}}$ as $\mathcal{N}_\theta^{\text{dy}}$.

Training		**Validation**	
1D Graphs	100	1D Graphs	20
Mini-Batch Size	20	Epochs	150
Learning Rate	10^{-3}	M (**AG** iterations)	10

Table 3 Fine-tuning dataset and parameters. No data augmentation is used in this phase.

7 Results and Discussion

We present a comprehensive analysis of the proposed neural preconditioning strategy. We begin by evaluating the impact of the static pretraining phase on residual reduction. Then, we assess the fine-tuning stage and demonstrate how the Krylov-geometry-aware loss improves early-stage convergence. Finally, we compare the FGMRES iteration counts with and without preconditioning across a variety of test parameter instances.

7.1 Pre-Training Step: Static Loss

We first examine the performance of the neural preconditioner trained with the static loss functional $\mathcal{L}_{\text{static}}$, as introduced in [17]. The static-training stage was conducted over 100 epochs on a dataset enriched with high-frequency perturbations to promote spectral diversity by adopting an AMD Instinct MI210 GPU with 64 GB of VRAM. The registered average time per epoch was 7.2 s/epoch, resulting in a total training time of ~ 12 min.

As illustrated in Figure 2 (Left column), both training and validation losses consistently decrease up to approximately 60 epochs, after which a mild overfitting trend emerges. The validation loss stabilizes around a value of 0.4, which is sufficient to produce an effective preconditioner for the considered 3D–1D problem. Further analysis highlights the improvement in problem conditioning induced by the learned preconditioner. Before training, the relative residual exhibits slow decay, with sine values close to 0.9, indicating near-orthogonality of the residual and Krylov image subspaces—an unfavorable condition for convergence (Figure 2, Middle column). After 100 optimization steps, the average sine value is reduced to approximately 0.6, resulting in a significantly enhanced convergence profile (Figure 2, Right column). As a result, the solver reaches the target tolerance of 10^{-6} in roughly 26 iterations, compared to 147 iterations in the unpreconditioned case.

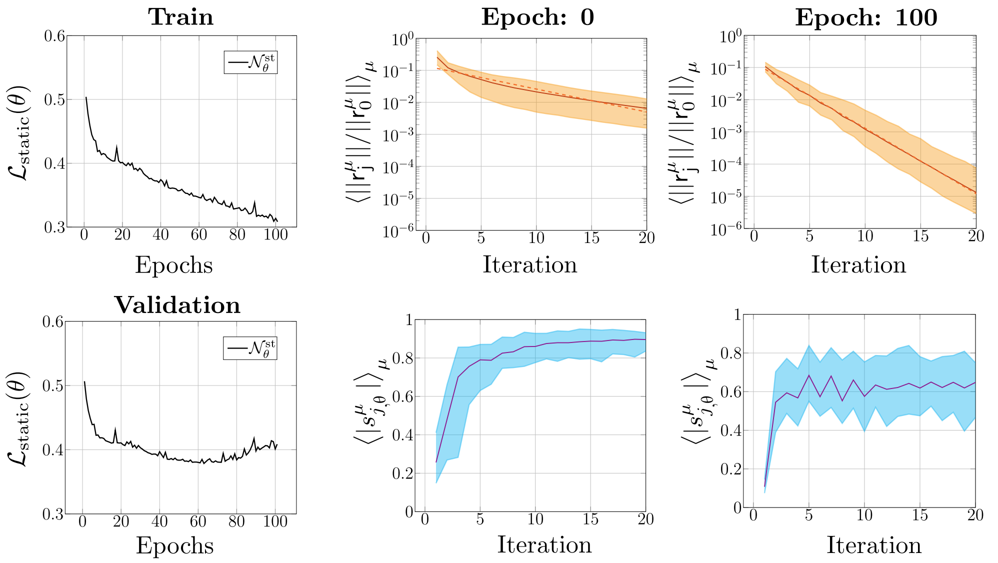

Fig. 2 **Left column:** Training (top) and validation (bottom) errors obtained during training of $\mathcal{N}_\theta^{\text{st}}$ preconditioner with residual-based static loss. Training is stopped at the onset of data overfitting. **Middle columns:** Relative residual $\|\mathsf{r}_\mathsf{j}\|/\|\mathsf{r}_0\|$ and averaged absolute subspaces angles sine $\langle|s_{j,\theta}^\mu|\rangle_\mu$ for iteration $j \in [1, 20]$ at the onset of training (epoch 0). **Right column:** Relative residual $\|\mathsf{r}_\mathsf{j}\|/\|\mathsf{r}_0\|$ and averaged absolute subspace angles sine $\langle|s_{j,\theta}^\mu|\rangle_\mu$ for iteration $j \in [1, 20]$ at the end of training step 1 (epoch 100).

7.2 Fine-tuning via Krylov Geometry

The second phase involves fine-tuning the network using the dynamic loss $\mathcal{L}_{\text{dynamic}}$, which incorporates geometric information extracted from the Krylov solver. As shown in Figure 3, both training and validation losses decrease and stabilize around a value of approximately 0.3, indicating consistent learning behavior across the parameter space and successful integration of subspace alignment objectives.

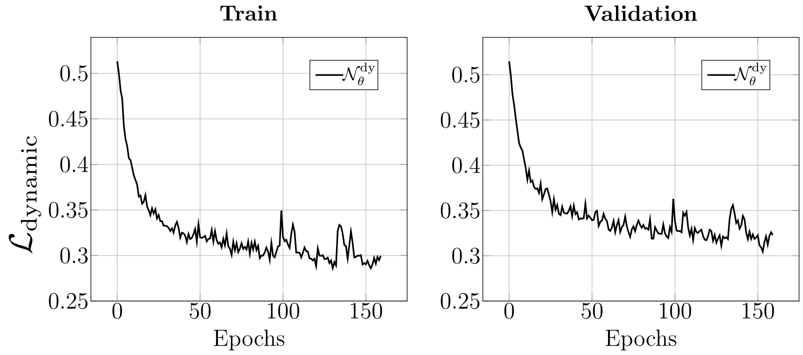

Fig. 3 Training and validation loss $\mathcal{L}_{\text{dynamic}} = \frac{1}{M}\sum_{j=1}^{M}\langle|s_{j,\theta}^\mu|\rangle_\mu$ during fine-tuning stage for, respectively, $\mu \in \mathcal{P}_{\text{train}}$ and $\mu \in \mathcal{P}_{\text{validation}}$.

The dynamic training stage comprises 160 epochs, with an average runtime of 2.5 min/epoch, producing a total training time of approximately 400 min. The observed computational cost is primarily due to the unrolling of the **AG** algorithm required for the dynamic loss evaluation. This constitutes the key distinction from the static training step, where reliance solely on residual evaluations results in a lower computational burden but limited control over the convergence rate.

7.2.1 Evolution of Principal Angles

To gain further insight, we monitor the evolution of the principal angles between the residual vector and the generated Krylov subspace basis vectors, as encoded by the sine quantities $\{|s^{\mu}_{j,\theta}|\}$. In the ideal case of rapid convergence, these angles should decrease rapidly toward zero, indicating that new Krylov directions contribute substantial progress toward the solution subspace. Under the first-stage training, based on the static residual loss $\mathcal{L}_{\text{static}}$, the learned preconditioner produces a global improvement of the system conditioning, reflected in a near-uniform reduction of the angles $\{|s^{\mu}_{j,\theta}|\}$. This behavior suggests that the static training implicitly regularizes the spectrum, albeit without explicit control over the subspace geometry. In contrast, the second-stage fine-tuning, based on the dynamic loss $\mathcal{L}_{\text{dynamic}}$, directly penalizes large angles within the first $M = 10$ Arnoldi iterations.

Empirically, during dynamic loss minimization, the averaged principal angles $\langle |s^{\mu}_{j,\theta}| \rangle_{\mu}$ over the first M iterations follow an approximately linear trend in the iteration index j,

$$\left\langle |s^{\mu}_{j,\theta}| \right\rangle_{\mu} \approx aj + b, \quad \text{for } j = 1, \dots, M\,.$$

While the slope a remains nearly constant across training epochs (on the order of 10^{-3}), the intercept b decreases progressively during optimization, from an initial value of approximately $b \approx 0.48$ down to $b \approx 0.30$ after fine-tuning. Results are summarized in Table 4 and Figure 4.

This decrease reflects improved initial alignment of the residual with the Krylov subspace within the first M iterations. Assuming that a is negligible, we may estimate

Epoch	$\frac{1}{M}\sum_{j=1}^{M}\langle \lvert s^{\mu}_{j,\theta}\rvert \rangle_{\mu}$	$\left(\frac{1}{M}\sum_{j=1}^{M}\langle \lvert s^{\mu}_{j,\theta}\rvert \rangle_{\mu}\right)^{M}$	Δ_{+}	Δ_{-}	a	b
0	0.592766	5.36×10^{-3}	0.246203	-0.518429	0.009823	0.489628
10	0.399244	1.03×10^{-4}	0.346335	-0.119267	0.006651	0.362663
20	0.373864	5.30×10^{-5}	0.360928	-0.148762	0.004453	0.349371
40	0.342721	2.20×10^{-5}	0.402198	-0.127696	0.005862	0.310482
60	0.335581	1.80×10^{-5}	0.352728	-0.122402	0.004203	0.312462
80	0.328283	1.50×10^{-5}	0.382090	-0.114524	0.006195	0.294211
150	0.317659	1.00×10^{-5}	0.414758	-0.111201	0.003036	0.300960

Table 4 Evolution of principal angles over training epochs. Linear fit of the form $\langle |s^{\mu}_{j,\theta}| \rangle_{\mu} = aj + b$ is applied to sine angles across inner iterations $j = 1, \dots, M$, with $M = 10$.

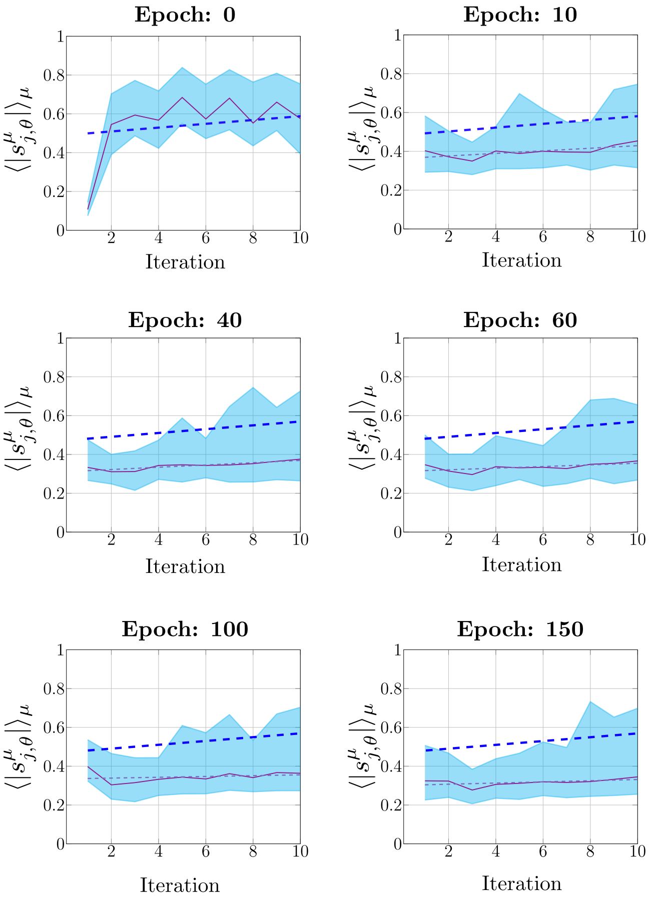

Fig. 4 Averaged sine angles $\langle |s^{\mu}_{j,\theta}|\rangle_{\mu}$ for $j = 1, \ldots, M$, with $\mu \in \mathcal{P}_{\text{validation}}$ (solid purple line). The shaded region denotes the envelope of $|s^{\mu}_{j,\theta}|$ curves across the parameter space. The thick dashed blue line indicates the linear fit of $\langle |s^{\mu}_{j,\theta}|\rangle_{\mu}$ at epoch 0, while purple dotted lines show the linear fits at subsequent epochs. Lower sine values reflect improved alignment of residuals with Krylov subspace directions.

the product of sines governing the convergence factor via

$$\prod_{j=1}^{M} \left\langle |s^{\mu}_{j,\theta}| \right\rangle_{\mu} \approx \left(\frac{1}{M} \sum_{j=1}^{M} \left\langle |s^{\mu}_{j,\theta}| \right\rangle_{\mu} \right)^{M}, \tag{5}$$

which approximates the asymptotic bound for the Krylov Subspace method convergence. By looking at the estimate above, we can observe a substantial drop (two orders of magnitude) of its value, from 5.3×10^{-3} to 1.0×10^{-5}, experienced during the dynamic training stage. Given the tight relation between principal angles s_j and the convergence rate of the FGMRES algorithm, we expect the same amount of reduction in the value of the relative residual.

7.2.2 Residual Decay Characterization

The progressive alignment between the principal angles and the Krylov subspace basis, discussed in Section 7.2.1, has a direct impact on the convergence profile of the iterative solver. Indeed, the reduction of $|s_j|$ induces a contraction of the residual norm, consistent with the geometric recurrence relation detailed in (3).

The evolution of the mean residuals can be tracked by a least-squares fitting with an exponential model

$$\left\langle \frac{\|\mathsf{r}_j^\mu\|}{\|\mathsf{r}_0^\mu\|} \right\rangle_\mu = e^\beta e^{\alpha j}, \quad j = 1, \dots, M,$$

where α encodes the decay rate and β controls the residual offset. The fitting is performed over $M = 10$ iterations, in line with the optimization horizon used in the dynamic loss functional introduced in (4). At the onset of training (epoch 0, mean sine ≈ 0.59), when the network $\mathcal{N}_\theta$ has only undergone static pretraining via the residual-based loss $\mathcal{L}_{\text{static}}$ (cf. Section 6.2), we observe a relative slow decay with parameters $\beta \approx -1.85$ and $\alpha \approx -0.4$ which leads to an average relative residual of $\sim 10^{-3}$ after 10 iterations. As fine tuning progresses, the minimization of the dynamic loss $\mathcal{L}_{\text{dynamic}(M)}$ actively promotes alignment of the residual with the Krylov subspace; thus α decreases significantly, from $\alpha \approx -0.4$ at epoch 0 to $\alpha \approx -1.13$ at epoch 150. The reduction in α reflects an acceleration in the asymptotic convergence rate.

Epoch	$\langle \|\mathsf{r}_M^\mu\|/\|\mathsf{r}_0^\mu\| \rangle_\mu$	Δ_+	Δ_-	α	β
0	1.20×10^{-3}	0.003509	-0.000849	-0.489807	-1.851620
10	1.05×10^{-4}	0.000415	-0.000079	-0.919889	-0.123565
20	5.50×10^{-5}	0.000332	-0.000041	-0.975239	-0.161445
40	2.70×10^{-5}	0.000257	-0.000021	-1.049013	-0.225478
60	2.30×10^{-5}	0.000252	-0.000018	-1.077473	-0.136004
80	1.80×10^{-5}	0.000298	-0.000015	-1.090334	-0.240775
100	2.90×10^{-5}	0.000274	-0.000024	-1.057390	-0.052897
150	1.30×10^{-5}	0.000209	-0.000010	-1.134431	-0.095151

Table 5 Evolution of relative residual at iteration M. Exponential fit of the form $\langle \|\mathsf{r}_j^\mu\|/\|\mathsf{r}_0^\mu\| \rangle_\mu = e^\beta e^{\alpha j}$ is applied to relative residuals for inner iterations $j = 1, \dots, M$.

For $M = 10$, the mean relative residual (averaged over the validation batch) decreases from 1.12×10^{-3} at epoch 0 to 1.3×10^{-5} at the final epoch. This represents a two-order-of-magnitude improvement in the reduction of the residual norm achieved through the fine-tuning phase.

As expected, these results perfectly align with the estimates in Eq. 5 reported in Table 4. All results are summarized in Table 5 and Figure 5, which report the fitted parameters and associated statistics.

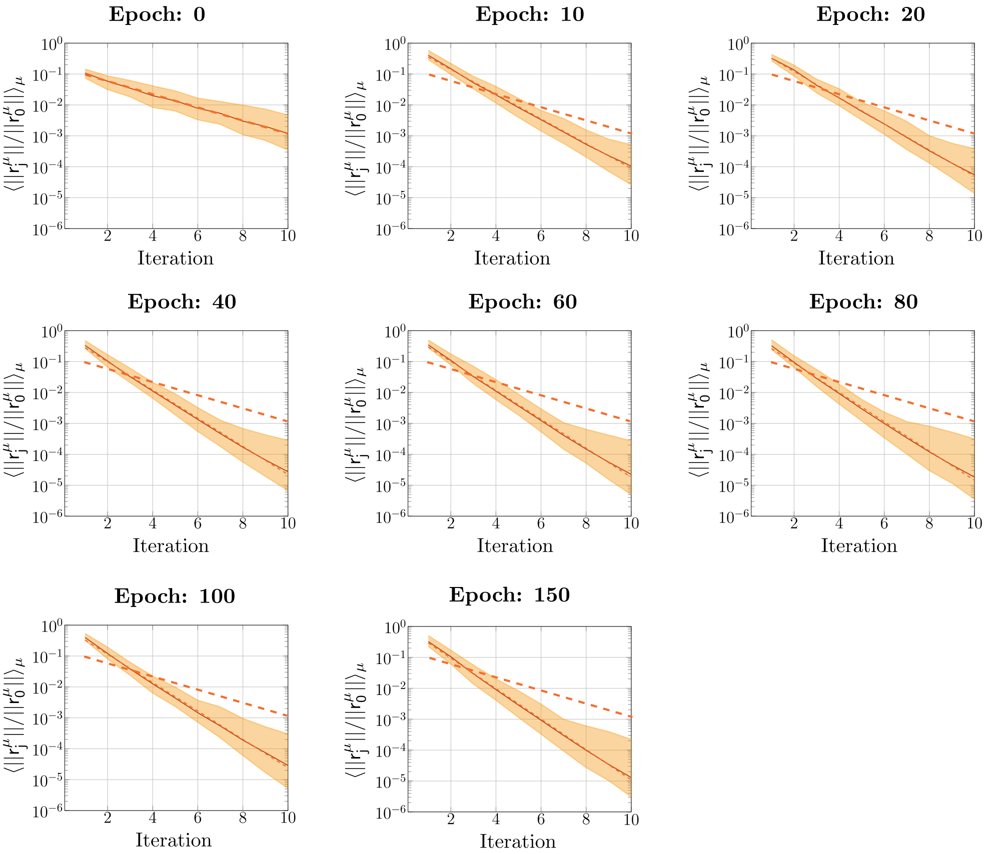

Fig. 5 Averaged relative residuals $\langle ||\mathsf{r}_j^\mu||/||\mathsf{r}_0^\mu|| \rangle_\mu$ for $j = 1, \ldots, M$, with $\mu \in \mathcal{P}_{\text{validation}}$ (solid red line). The shaded region denotes the envelope of $||\mathsf{r}_j^\mu||/||\mathsf{r}_0^\mu||$ curves across the parameter space. The thick dashed red line indicates the linear fit of $\langle ||\mathsf{r}_j^\mu||/||\mathsf{r}_0^\mu|| \rangle_\mu$ at epoch 0, while red dotted lines show the exponential fits at subsequent epochs.

7.3 Localized Control over Krylov Dynamics

An important observation is that the dynamic loss structure defined by $\mathcal{L}_{\text{dynamic}(M)}$ induces a localized improvement in Krylov geometry: by restricting the supervision window to $M = 10$ Arnoldi iterations, the neural preconditioner $\mathcal{N}_\theta^{\text{dy}}$ *specializes* in optimizing early-stage convergence—where computational savings are most impactful. Beyond the supervised horizon ($j > M$), the principal angles $|s_{j,\theta}^\mu|$ tend to increase, reflecting the absence of penalization in later iterations.

Crucially, this behavior is not a limitation but a controllable feature: by tuning the value of M, one can trade depth for early accuracy, thereby aligning the optimization objective with specific solver performance targets. This targeted control provides a flexible and interpretable framework for tailoring the action of learned preconditioners to varying computational demands.

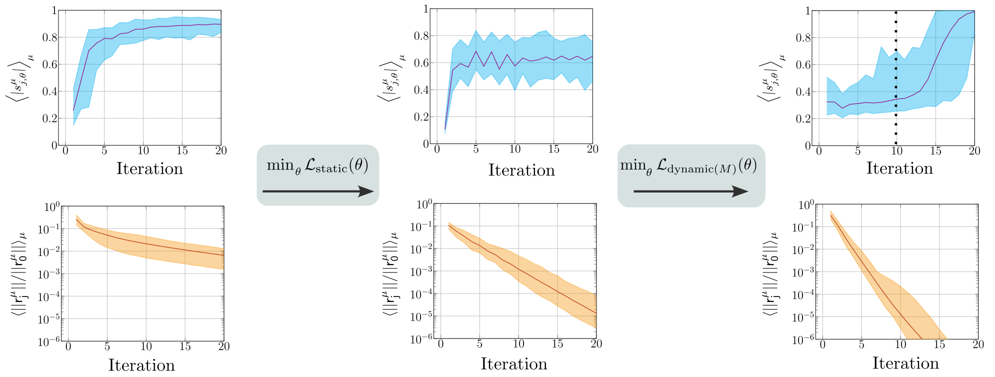

Fig. 6 Comparison of mean principal angles $\langle |s^{\mu}_{j,\theta}| \rangle_{\mu}$ and relative residuals $\langle ||\mathsf{r}^{\mu}_{\mathsf{j}}|| / ||\mathsf{r}^{\mu}_{0}|| \rangle_{\mu}$ for $j = 1, \ldots, 2M$, after static training via $\mathcal{L}_{\mathrm{static}}(\theta)$ (**left**) and after dynamic fine-tuning via $\mathcal{L}_{\mathrm{dynamic}(M)}(\theta)$ with $M = 10$ (**right**).

7.4 Iteration Count Analysis

We now evaluate the performance of the dynamically fine-tuned neural preconditioner $\mathcal{N}^{\mathrm{dy}}_{\theta}$ obtained via the two-stage training protocol outlined in Section 6. The benchmark involves a parametric family of coupled 3D–1D problems, where the 1D domain represents vascular networks modeled as randomly generated graphs of different geometric complexity. This setting introduces significant variability in the system matrices, making it a robust test case for generalization across operator structures. The objective of the preconditioner is to improve both spectral and geometric properties of the system matrix to accelerate Krylov subspace convergence. To assess this, we measure the average number of FGMRES iterations required to reduce the relative residual norm below 10^{-6} across a test set of 100 previously unseen graph configurations. Without preconditioning, the solver requires, on average 147.5 iterations, with values ranging from 100 to 245. With the statically trained preconditioner, the count drops sharply to 26.71, with a narrower range of 23 to 31 iterations. Following dynamic fine-tuning, the average count further decreases to 12.86, with values ranging from 11 to 16.

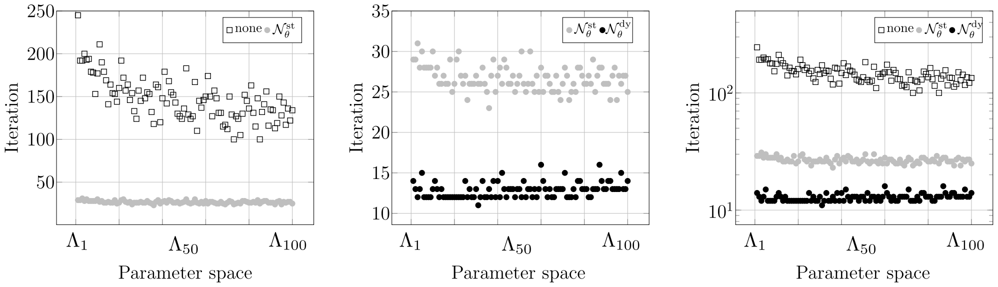

Fig. 7 FGMRES iteration counts across 100 test 1D graph instances Λ_i. **Left:** unpreconditioned vs. statically trained $\mathcal{N}^{\mathrm{st}}_{\theta}$. **Center:** static vs. dynamically fine-tuned $\mathcal{N}^{\mathrm{dy}}_{\theta}$. **Right:** global overview across all configurations.

These results show that the neural preconditioner $\mathcal{N}_\theta$ reduces the average iteration count by more than an order of magnitude across the parametric problem family $(\mathsf{A}^\mu, \mathsf{b}^\mu)$. The additional gain from dynamic tuning confirms that the Krylov subspace generation process itself is being effectively manipulated by the neural preconditioner.

7.5 Performance Comparisons and Execution Time

Having assessed the positive impact of the dynamic fine-tuning strategy on the neural-preconditioned FGMRES solver, in this section, we present a comparison between $\mathcal{N}_\theta^{\text{dy}}$ and well-established preconditioners, namely Jacobi, Incomplete LU factorization (ILU), and algebraic multigrid (AMG). The `PETSc` library implementation [25] was employed for all standard preconditioners, with the following configurations: the Jacobi method applies point-wise diagonal scaling; the ILU preconditioner employs an incomplete LU factorization with no fill-in (default `PETSc` setting); the AMG preconditioner relies on the `BoomerAMG` solver with three multigrid levels. Performances are recorded on a computational mesh consisting of $\sim 10^4$ degrees of freedom, with computations carried out on a Lenovo Legion 9 16 workstation equipped with an Intel Core i9-14900HX processor, 64 GB of RAM, and an NVIDIA GeForce RTX 4090 GPU with 16 GB of GDDR6 memory.

In the following experiments, the GPU is used exclusively for the inference of the neural-network preconditioner, while the classical preconditioners (Jacobi, ILU, AMG) rely on their standard CPU PETSc implementations (single MPI rank). This choice reflects the fact that PETSc's classical preconditioners are highly optimized for CPU execution, whereas neural networks are naturally suited for GPU-based inference; each method therefore operates in its most appropriate computational environment. We note that GPU-accelerated variants of ILU and AMG generally rely on different algorithmic formulations and software packages (e.g., AmgX, [26], Kokkos [27], Hypre-GPU [28]), which do not directly correspond to the standard PETSc implementations used in our study. For this reason, we restrict the classical preconditioners to their widely adopted CPU versions. Moreover, the neural preconditioner does not exploit batch parallelism or throughput-oriented acceleration on the GPU: each preconditioning call involves a single forward pass, without amortizing computation across multiple inferences. This setup ensures that the comparison focuses on algorithmic performance rather than hardware-driven optimizations.

As reported in the previous section and in Table 6, the dynamically fine-tuned neural preconditioner, $\mathcal{N}_\theta^{\text{dy}}$, achieves a mean iteration count of 12.87 with stable performance across the parameter space ($\Delta^+ = +3.24$, $\Delta^- = -1.86$). This result demonstrates performance comparable to standard `PETSc` preconditioners, with only AMG(3) performing better, requiring 9.65 iterations on average.

In terms of execution time, $\mathcal{N}_\theta^{\text{dy}}$ is the fastest method, with an average execution time of 17.12 ms; this corresponds to a ×3 speed-up compared to AMG(3), which achieves fewer iterations but requires a more expensive setup phase. The ILU preconditioner follows, with an average execution time of 17.95 ms. ILU variants with fill-in levels up to three were tested, with the no-fill configuration (the one considered in the table) yielding the lowest execution time among them. The reported analysis

Precond. type.	**Mean Iter.**	Δ^+	Δ^-	**Mean Time**	Δ_t^+	Δ_t^-	**Speed-Up w.r.t. AMG(3)**
$\mathcal{N}_\theta^{\mathrm{st}}$	26.71	+4.29	-3.71	35.15	+5.12	-4.82	1.47
$\mathcal{N}_\theta^{\mathrm{dy}}$	12.86	+3.14	-1.86	17.12	+5.72	-2.73	3.02
Jacobi	115.06	+78.94	-40.06	59.10	+31.39	-19.02	0.88
ILU	24.73	+8.27	-5.73	17.95	+9.66	-3.42	2.89
AMG(3)	9.56	+2.44	-1.56	51.73	+5.53	-7.56	1.0

Table 6 Mean FGMRES iterations and execution time (in milliseconds [ms]) for different preconditioners. $\Delta^\pm$ denotes the deviation in iteration count from the mean value to the worst (best) case within the test set, while $\Delta_t^\pm$ indicate the corresponding deviations in execution time. The execution-time speed-up relative to the AMG(3) preconditioner is also reported for comparison.

shows that $\mathcal{N}_\theta^{\mathrm{dy}}$ represents a valuable preconditioning strategy due to its fast execution time and the absence of online setup requirements. Of course, an offline training phase is required; however, it needs to be performed only once. Thereafter, the trained neural preconditioner can be applied to any parametric scenario within the considered parameter distribution. Importantly, the method does not rely on previously computed problem solution data. It only requires access to system matrices and right-hand-side vectors, which makes the generation of the training set straightforward and computationally efficient.

8 Conclusion and Perspectives

This work introduces a novel training strategy for neural preconditioners that leverages the geometric structure of Krylov subspace methods, specifically the Generalized Minimal Residual (GMRES) algorithm. The core contribution lies in the formulation of a dynamic fine-tuning phase, wherein the loss functional directly optimizes the subspace angles $s_{i,\theta}$ that govern the convergence behavior of iterative solvers. By embedding this geometric insight into the training process, we offer a principled and solver-aligned approach to performance enhancement.

A central advantage of our method is that it preserves the strengths of unsupervised learning, such as straightforward data generation and independence from ground truth solutions, while simultaneously introducing a transparent and solver-integrated performance metric. The use of a differentiable formulation of the Flexible GMRES algorithm enables efficient gradient-based optimization through backpropagation across the solver's iterative process. The dynamic residual-based loss we propose captures the alignment between the residual vector and the Krylov subspace at each iteration, quantified by the sine of their principal angle, thereby offering a geometrically meaningful proxy for convergence rate. The effectiveness of this solver-aware training scheme is validated by substantial reductions in iteration counts for parameter-dependent linear systems derived from mixed-dimensional partial differential equations (PDEs)—a class of problems often characterized by ill-conditioning and geometric complexity. Building upon our earlier work on unsupervised neural preconditioners for mixed-dimensional models, we demonstrate that the combined static-dynamic training strategy significantly enhances the preconditioner's adaptability to heterogeneous problem instances. In our numerical tests, the average number of FGMRES iterations dropped from 147.5 (unpreconditioned case) to 26.71 after static training, and further to 12.86 following dynamic fine-tuning. This outcome attests to the ability of the neural preconditioner to influence the subspace geometry and accelerate solver convergence directly. Comparison with established preconditioning strategies demonstrates the effectiveness of the proposed approach, which yields the fastest FGMRES execution time among the tested methods.

Despite these encouraging results, several important directions remain for future research. Extending the proposed approach to alternative iterative solvers and different classes of preconditioners represents a natural next step. Investigating hybrid training strategies that blend static and dynamic loss formulations, or integrate additional physics-informed components, could yield further improvements in robustness and generalization. A particularly relevant challenge is to assess the preconditioner's performance on out-of-distribution samples, including unseen geometries and parametric regimes, to better understand its extrapolative capabilities. From a theoretical standpoint, formalizing the role of spectral data augmentation and characterizing the learning dynamics of Krylov-aware neural networks are crucial for building a more rigorous foundation for learning-based preconditioning. Moreover, current limitations related to the use of convolutional neural networks on structured grids suggest the need for architectural generalizations. Mesh-informed neural networks or graph-based models could provide the necessary flexibility to extend applicability to unstructured meshes and more general computational domains. Finally, scaling up the approach to large-scale problems remains a critical avenue. Optimizing multi-resolution network design, exploiting GPU-parallel computations, and integrating ensemble methods can substantially improve throughput in multi-query settings such as parameter studies, control, or uncertainty quantification.

In conclusion, this study establishes the feasibility and effectiveness of neural preconditioning guided by Krylov subspace geometry as a powerful tool for accelerating iterative solvers in complex PDE applications. It provides a promising foundation for advancing the integration of deep learning and numerical linear algebra in scientific computing.

Acknowledgments

PZ acknowledges the support of the MUR PRIN 2022 grant No. 2022WKWZA8 *Immersed methods for multiscale and multiphysics problems* (IMMEDIATE) part of the Next Generation EU program Mission 4, Comp. 2, CUP D53D23006010006. The present research is part of the activities of the *Dipartimento di Eccellenza* 2023-2027, Department of Mathematics, Politecnico di Milano. AC acknowledges the support of the MUR PRIN 2022 project *Evolution problems involving interacting scales* project code 2022M9BKBC, Grant No. CUP D53D23005880006. All authors are members of the Gruppo Nazionale per il Calcolo Scientifico (GNCS), Istituto Nazionale di Alta Matematica (INdAM).

Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] D'Angelo, C., Quarteroni, A.: On the coupling of 1d and 3d diffusion-reaction equations: Application to tissue perfusion problems. Mathematical Models and Methods in Applied Sciences **18**(08), 1481–1504 (2008)

[2] Boon, W.M., Nordbotten, J.M., Vatne, J.E.: Functional analysis and exterior calculus on mixed-dimensional geometries. Ann. Mat. Pura Appl. **200**(2), 757–789 (2021)

[3] Kuchta, M., Laurino, F., Mardal, K.-A., Zunino, P.: Analysis and approximation of mixed-dimensional pdes on 3d-1d domains coupled with lagrange multipliers. SIAM Journal on Numerical Analysis **59**(1), 558–582 (2021)

[4] Heltai, L., Zunino, P.: Reduced lagrange multiplier approach for non-matching coupling of mixed-dimensional domains. Mathematical Models and Methods in Applied Sciences **33**(12), 2425–2462 (2023)

[5] Saad, Y.: Iterative Methods for Sparse Linear Systems. Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2003)

[6] Mardal, K.-A., Winther, R.: Preconditioning discretizations of systems of partial differential equations. Numerical Linear Algebra with Applications **18**(1), 1–40 (2011)

[7] Chen, K.: Matrix Preconditioning Techniques and Applications vol. 19. Cambridge University Press, Cambridge, UK (2005)

[8] Trottenberg, U., Oosterlee, C.W., Schuller, A.: Multigrid. Elsevier, Amsterdam, The Netherlands (2000)

[9] Quarteroni, A., Valli, A.: Domain Decomposition Methods for Partial Differential Equations. Oxford University Press, Oxford, UK (1999)

[10] Kuchta, M., Nordaas, M., Verschaeve, J.C.G., Mortensen, M., Mardal, K.-A.: Preconditioners for saddle point systems with trace constraints coupling 2d and 1d domains. SIAM Journal on Scientific Computing **38**(6), 962–987 (2016)

[11] Budiša, A., Hu, X., Kuchta, M., Mardal, K.-A., Zikatanov, L.: Algebraic multigrid methods for metric-perturbed coupled problems. SIAM Journal on Scientific Computing **46**(3), 1461–1486 (2024)

[12] Firmbach, M., Steinbrecher, I., Popp, A., Mayr, M.: An approximate block factorization preconditioner for mixed-dimensional beam-solid interaction. Computer Methods in Applied Mechanics and Engineering **431**, 117256 (2024)

[13] Azulay, Y., Treister, E.: Multigrid-augmented deep learning preconditioners for the helmholtz equation. SIAM Journal on Scientific Computing **45**(3), 127–151 (2023)

[14] Kopaničáková, A., Karniadakis, G.E.: Deeponet based preconditioning strategies for solving parametric linear systems of equations. SIAM Journal on Scientific Computing **47**(1), 151–181 (2025)

[15] Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.: Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nature Machine Intelligence **3**, 218–229 (2021)

[16] Xu, Z.-Q.J., Zhang, Y., Luo, T.: Overview frequency principle/spectral bias in deep learning. Communications on Applied Mathematics and Computation, 1–38 (2024)

[17] Dimola, N., Franco, N.R., Zunino, P.: Numerical Solution of Mixed-Dimensional PDEs Using a Neural Preconditioner (2025)

[18] Williams, C., Falck, F., Deligiannidis, G., Holmes, C., Doucet, A., Syed, S.: A unified framework for u-net design and analysis. arXiv **stat.ML**(2305.19638) (2024)

[19] Gunes Baydin, A., Pearlmutter, B.A., Andreyevich Radul, A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. arXiv e-prints, 1502 (2015)

[20] Saad, Y.: A flexible inner-outer preconditioned gmres algorithm. SIAM Journal on Scientific Computing **14**(2), 461–469 (1993)

[21] Laurino, F., Zunino, P.: Derivation and analysis of coupled pdes on manifolds with high dimensionality gap arising from topological model reduction. ESAIM: Mathematical Modelling and Numerical Analysis **53**(6), 2047–2080 (2019)

[22] Eiermann, M., Ernst, O.G.: Geometric aspects of the theory of krylov subspace methods. Acta Numerica **10**, 251–312 (2001)

[23] Paszke, A.: Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703 (2019)

[24] Franco, N.R., Manzoni, A., Zunino, P.: Mesh-informed neural networks for operator learning in finite element spaces. Journal of Scientific Computing **97**(35) (2023)

[25] Balay, S., Abhyankar, S., Adams, M.F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E., Dalcin, L., Dener, A., et al.: Petsc/tao users manual (rev. 3.20). Technical report, Argonne National Laboratory (ANL), Argonne, IL (United States) (2023)

[26] Naumov, M., Arsaev, M., Castonguay, P., Cohen, J., Demouth, J., Eaton, J., Layton, S., Markovskiy, N., Reguly, I., Sakharnykh, N., *et al.*: Amgx: A library for gpu accelerated algebraic multigrid and preconditioned iterative methods. SIAM Journal on Scientific Computing **37**(5), 602–626 (2015)

[27] Rajamanickam, S., Acer, S., Berger-Vergiat, L., Dang, V., Ellingwood, N., Harvey, E., Kelley, B., Trott, C.R., Wilke, J., Yamazaki, I.: Kokkos kernels: Performance portable sparse/dense linear algebra and graph kernels. arXiv preprint arXiv:2103.11991 (2021)

[28] Falgout, R., Cleary, A., Jones, J., Chow, E., Henson, V., Baldwin, C., Brown, P., Vassilevski, P., Yang, U.: Hypre: high performance preconditioners. Users Manual. Version **1**(0) (2010)

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

12/2026 Corbetta A; Logan K.M.; Ferro M; Zuccolo L; Perola M.; Ganna A.; Di Angelantionio E.;Ieva F.
Longitudinal patterns of statin adherence and factors associated with decline in over one million individuals in Finland and Italy

11/2026 Cicalese, G.; Ciaramella, G.; Mazzieri, I.; Gander, M. J.
Optimized Schwarz Waveform Relaxation for the Damped Wave Equation

10/2026 Dimola, N.; Franco, N. R.; Zunino, P.
Numerical Solution of Mixed-Dimensional PDEs Using a Neural Preconditioner

09/2026 Manzoni, V.; Ieva, F.;Larranaga, A.C.; Vetrano, D.L.; Gregorio, C.
Hidden multistate models to study multimorbidity trajectories

08/2026 Micheletti, S.
Newmark time marching as a preconditioned iteration for large SPD linear systems

Micheletti, S.
Newmark time marching as a preconditioned iteration for large SPD linear systems

07/2026 Corti, M.; Ahern, A.; Goriely, A.; Kuhl, E.; Antonietti, P.F.
A whole-brain model of amyloid beta accumulation and cerebral hypoperfusion in Alzheimer's disease

06/2026 Corti, M.; Gómez, S.
On the compact discontinuous Galerkin method for polytopal meshes

05/2026 Ranno, A.; Ballarin, F.; Lespagnol, F.; Zunino, P.; Perotto, S.
A fictitious domain formulation based on hierarchical model reduction applied to drug-eluting stents

04/2026 Ragni, A.; Cavinato, L.; Ieva, F.
Penalized Likelihood Optimization for Adaptive Neighborhood Clustering in Time-to-Event Data with Group-Level Heterogeneity