EuroHPC
Joint Undertaking

Data driven Computational Mechanics at EXascale

DCoMEX

Data driven Computational Mechanics at EXascale

Work program topic: EuroHPC-01-2019
Type of action: Research and Innovation Action (RIA)

REPORT ON AI-SOLVE LIBRARY PROTOTYPE

DELIVERABLE D3.2

Version No 1

http://www.dcomex.eu/
This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking ('the JU'), under Grant Agreement No 956201

DOCUMENT SUMMARY INFORMATION

Project Title	**Data driven Computational Mechanics at EXascale**
Project Acronym	DCoMEX
Project No:	956201
Call Identifier:	EuroHPC-01-2019
Project Start Date	01/04/2021
Related work package	WP 3
Related task(s)	Task 3.1, 3.2, 3.3, 3.7
Lead Organisation	NTUA
Submission date	18/02/2024
Re-submission date	
Dissemination Level	PU

Quality Control:

	Who	**Affiliation**	**Date**
Checked by internal reviewer	George Stavroulakis	NTUA	17/02/2024
Checked by WP Leader	Vissarion Papadopoulos	NTUA	17/02/2024
Checked by Project Coordinator	Vissarion Papadopoulos	NTUA	17/02/2024

Document Change History:

Version	**Date**	**Author (s)**	**Affiliation**	**Comment**
1.0	16.02.2024	Ioannis Kalogeris	NTUA	

Contents

1. Description 4
2. Iterative Solvers for large-scale systems 5
2.1 Problem statement 5
2.2 Preconditioned Conjugate gradient method 5
2.3 Block Conjugate Gradient method 6
2.4 Algebraic Multigrid method 7
2.5 Domain Decomposition methods 8
2.5.1 Subdomains and mapping operators 8
2.5.2 Local problem solution 10
2.5.3 Interface problem solution 10
2.5.4 Rigid body modes 11
2.5.5 P-DDM: The Primal Substructuring Method (PSM) 12
2.5.6 D-DDM: The Finite Element Tearing and Interconnecting (FETI) method 12
2.5.7 P-DDM for static analysis with D-DDM based preconditioners: The PFETI method 14
3. AI accelerated Iterative Solvers for parameterized problems 16
3.1 Data-driven solution framework-POD-2G solver 16
3.2 Fusing nonlinear solvers with transformers for accelerating parameterized transient problems 18
4. Contents of the AI-Solve software library 22
4.1 Linear algebra repository 22
4.2 Solvers repository 22
4.3 Machine learning repository 23
4.4 AISolve.core repository 23
4.5 AISolve.MSolve repository 23
5. Numerical Applications 25
5.1 Accelerating crack propagation problems using DDM 25
5.2 Solution to the parameterized Biot problem using POD-2G 26
5.3 Accelerating the nonlinear analysis of water tower under random base excitation 28
6. Summary and future work 32
References 33

1. Description

The rapid advancements in the field of machine learning (ML) has offered researchers new tools to tackle challenging engineering problems, especially in multi-query scenarios. In recent years, reduced-order modeling techniques and neural networks have seen numerous applications in the development of surrogate models, aimed to simplify the solution process for complex engineering problems. In this direction, WP2 of the DCoMEX project was devoted to the development of such surrogate models using feedforward neural networks, convolutional autoencoders and the diffusion maps algorithm, as described in D2.3. Despite their powerful approximation capabilities, however, these surrogate models cannot guarantee convergence to the exact solution of the problem.

On the other hand, advanced iterative solvers from the field of linear algebra can guarantee convergence to the exact solution of linear systems but are unable to attain a uniformly fast convergence for every parameter instance in a parameterized problem. In the effort to take the best of two worlds, the AI-Solve software aims at bridging the gap between machine learning and linear algebra algorithms for accelerating the solution of real-life computational mechanics problems in multi-query scenarios. To achieve this, the ML-based surrogate modeling techniques developed in WP2 of the project are combined with iterative solvers in order to develop novel algorithms capable of handling very demanding problems in an efficient manner.

In particular, the D3.2 “Report on AI-Solve library prototype” reports on the developments in WP3 of the project and it accompanies D3.1, which is the AI-Solve library software. In this deliverable the functionalities of AI-Solve library are presented, followed by an explanation of the library’s content and a set of numerical applications that showcase its capabilities for efficiently solving large-scale linear systems.

This document is structured as follows: Section 2 describes the mathematical theory for the set of algorithms implemented in MSolve for solving large-scale linear systems. Section 3 presents the methodologies developed within the DCoMEX project that utilize machine learning to enhance the previously mentioned linear algebraic solvers for parametrized problems in computational mechanics. Section 4 is devoted to the illustration of the AI-Solve library, the explanation of the code content and its functionalities. Section 5 involves a set of applications that demonstrate the computational merits this library has to offer in the field of computational science and engineering.

2. Iterative Solvers for large-scale systems

2.1 Problem statement

In scientific computing, there is a constant need for solving larger and computationally more demanding problems with increased accuracy and improved numerical performance. This holds particularly true in multi-query scenarios such as optimization, uncertainty quantification, inverse problems and optimal control, where the problems under investigation need to be solved for numerous different parameter instances with high accuracy and efficiency. Therefore, constructing efficient numerical solvers for complex systems described by partial differential equations is crucial for many scientific disciplines.

In the context of the Finite Element Method (FEM), a parameterized partial differential equation can be discretized into a $d \times d$ linear system of equations (or sequence of linear systems for transient problems) of the form:

$$\boldsymbol{K}(\boldsymbol{\theta})\boldsymbol{u}(\boldsymbol{\theta}) = \boldsymbol{f}(\boldsymbol{\theta})$$

where $\boldsymbol{K} \in R^{d\times d}$ is the system matrix, $\boldsymbol{u} \in R^{d}$ is the solution vector containing the unknown nodal values at specific locations of the PDE's domain, $\boldsymbol{f} \in R^{d}$ is the equivalent force vector and $\boldsymbol{\theta} \in R^{n}$ is the vector of parameters that influence the system. Solving such a linear system for a detailed discretization ($d \gg 1$) can be computationally intensive, particularly in multiquery problems that require numerous system evaluations for various instances of parameters $\boldsymbol{\theta}$. Therefore, it becomes evident that efficient numerical solvers for linear systems of equations are of vital importance in the analysis of large scale real-world problems. In the remainder of this section, we revisit the basic ideas behind three of the most efficient methods for solving such systems, namely, the Preconditioned Conjugate Gradient Method (PCG), the block variant of the PCG and the Algebraic Multigrid (AMG) method.

2.2 Preconditioned Conjugate gradient method

As an iterative technique, the conjugate gradient method starts by an initial guess $\boldsymbol{u}^{(0)}$ for the system $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$, with $\boldsymbol{K}$ being a symmetric positive definite matrix, and constructs a sequence of vectors $\{\, \boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \dots \,\}$ that converge to the exact solution $\boldsymbol{K}^{-1}\boldsymbol{f}$ in, at most, d iterations. In practice, however, the algorithm may terminate after $k < d$ iterations, provided that the condition for the residual after the k-th iteration $\|\boldsymbol{r}^{(k)}\| = \|\boldsymbol{K}\boldsymbol{u}^{(k)} - \boldsymbol{f}\| \leq \delta$, with δ being a prescribed accuracy threshold.

It is important to mention that the improvement in the sequence of approximations $\boldsymbol{u}^{(k)}$ is determined by the condition number $c(\boldsymbol{K})$ of the system matrix $\boldsymbol{K}$; the larger $c(\boldsymbol{K})$ is, the slower the improvement. A standard approach to enhance the convergence of the CG method is though preconditioning (PCG), which involves the application of a linear transformation to the system with a matrix $\boldsymbol{T}$, called the preconditioner, in order to reduce the condition number of the problem. Thus, the original system $\boldsymbol{K}\boldsymbol{u} - \boldsymbol{f} = \boldsymbol{0}$ is replaced with $\boldsymbol{T}^{-1}(\boldsymbol{K}\boldsymbol{u} - \boldsymbol{f}) = \boldsymbol{0}$, such that $c(\boldsymbol{T}^{-1}\boldsymbol{K})$ is smaller than $c(\boldsymbol{K})$. The steps of the PCG algorithm are presented in the following algorithm.

Algorithm 1. PCG algorithm

```
1:  Input: $\boldsymbol{K} \in \mathbb{R}^{d\times d}$, rhs $\boldsymbol{f} \in \mathbb{R}^{d}$, preconditioner $\boldsymbol{T} \in \mathbb{R}^{d\times d}$, residual tolerance $\delta$ and an initial approximation $\boldsymbol{u}^{(0)}$
2:  set $k = 0$, initial residual $\boldsymbol{r}^{(0)} = \boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}^{(0)}$
3:  $\boldsymbol{s}_0 = \boldsymbol{T}^{-1}\boldsymbol{r}^{(0)}$
4:  $\boldsymbol{p}_0 = \boldsymbol{s}_0$
5:  while $\|\boldsymbol{r}^{(k)}\| < \delta$ do
6:      $\alpha_k = \frac{(\boldsymbol{r}^{(k)})^T \boldsymbol{s}_k}{\boldsymbol{p}_k^T \boldsymbol{K} \boldsymbol{p}_k}$
7:      $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \alpha_k \boldsymbol{p}_k$
8:      $\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha_k \boldsymbol{K}\boldsymbol{p}_k$
9:      $\boldsymbol{s}_{k+1} = \boldsymbol{T}^{-1}\boldsymbol{r}^{(k+1)}$
10:     $\beta_k = \frac{(\boldsymbol{r}^{(k+1)})^T \boldsymbol{s}_{k+1}}{(\boldsymbol{r}^{(k)})^T \boldsymbol{s}_k}$
11:     $\boldsymbol{p}_{k+1} = \boldsymbol{s}_{k+1} + \beta_k \boldsymbol{p}_k$
12:     $k = k + 1$
13: end while
```

The choice of the precondition T in PCG plays a crucial role in the fast convergence of the algorithm. Some generic choices include the Jacobi (diagonal) preconditioner and the incomplete LU and Choleski factorizations. Moreover, multigrid methods such as the AMG and Domain Decomposition methods that will be elaborated on the next sections, are also very efficient preconditioners to the CG method.

2.3 Block Conjugate Gradient method

The drawback of the CG algorithm is that it cannot exploit the parallel computing capabilities offered by modern systems. The block CG algorithm was specifically developed to remedy this issue.

This algorithm starts with an initial guess $\boldsymbol{x}_0$ for the problem $\boldsymbol{A}\cdot\boldsymbol{x}=\boldsymbol{b}$ and computes the residual $\boldsymbol{r_0}=\boldsymbol{b}-\boldsymbol{A}\cdot\ \boldsymbol{x_0}$. The Krylov subspace of the matrix $\boldsymbol{A}$ with the vector $\boldsymbol{r_0}$ is computed using parallel processing for powers equal to the size of the block. Then, the steps for the block are the same as the PCG method, but the vectors $\boldsymbol{r},\boldsymbol{p},\boldsymbol{x}$ are linear combinations of the Krylov vectors. The weakness of the algorithm is that it tends to exhibit numerical instabilities when the Krylov vectors approach the eigenvectors of A and it also requires 1.5-2 times the steps of the classic PCG algorithm.
However, it offers some significant advantages, namely: (i) its steps can be executed in a parallel fashion and (ii) these steps require negligible computational time, effectively negating the cost of the additional iterations needed compared to CG. The algorithm for a block size of m, for an n x n linear system using the matrix $\boldsymbol{M}$ as a preconditioner is as follows:

```
- Compute the residual: $\boldsymbol{r_0}=\boldsymbol{b}-\boldsymbol{A}\cdot\boldsymbol{x_0}$
- Set: $\boldsymbol{x}=\boldsymbol{x_0}$
- Compute the $n\cdot m$ matrices $\boldsymbol{P}=\boldsymbol{R}=KrylovSubspace(\boldsymbol{A}\cdot\boldsymbol{M^{-1}},\boldsymbol{r})$, that is $(\boldsymbol{A}\cdot\boldsymbol{M^{-1}})^0\cdot\boldsymbol{r}_0$, $(\boldsymbol{A}\cdot\boldsymbol{M^{-1}})^1\cdot\boldsymbol{r}_0$, ....., $(\boldsymbol{A}\cdot\boldsymbol{M^{-1}})^{(m-1)}\cdot\boldsymbol{r}_0$
- Compute and store as $(2m-1)$-sized vectors the inner products: $\boldsymbol{RP}=\boldsymbol{PP}=\boldsymbol{RR}=R_{,i}\cdot\boldsymbol{M^{-1}}\cdot R_{,j}$, for $1\le i,j\le m$ and $i+j-1=1,2,\dots,2m-1$
- Set $rr=RR_1$
- If $rr<threshold$ terminate and accept $\boldsymbol{x_0}$ as the solution, else
- begin first loop
    - Set $\boldsymbol{r_c}=\begin{Bmatrix}1&0\\0&0\\\dots&\dots\end{Bmatrix}$, $\boldsymbol{p_c}=\begin{Bmatrix}0&1\\0&0\\\dots&\dots\end{Bmatrix}$ and $\boldsymbol{x_c}=\begin{Bmatrix}0&0\\0&0\\\dots&\dots\end{Bmatrix}$ that represent $\boldsymbol{r}$, $\boldsymbol{p}$ and $\boldsymbol{x}$ as linear combinations of $\boldsymbol{P}$ and $\boldsymbol{R}$, e.g. $\boldsymbol{r}=\boldsymbol{R}\cdot(\boldsymbol{r_c})_{,\boldsymbol{1}}+\boldsymbol{P}\cdot(\boldsymbol{r_c})_{,\boldsymbol{2}}$.
    - begin second loop for m steps
        - Compute
          $$\boldsymbol{pAp}=\sum_{i=1}^{m}\boldsymbol{RR}_{2i}\,(\boldsymbol{r_c})_i+\sum_{i=1}^{m}\boldsymbol{PP}_{2i}\,(\boldsymbol{p_c})_i+ \sum_{i=1}^{m-1}\sum_{j=i+1}^{m}2\,\boldsymbol{RR}_{i+j}(\boldsymbol{r_c})_i(\boldsymbol{r_c})_j+\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}2\,\boldsymbol{PP}_{i+j}(\boldsymbol{p_c})_i(\boldsymbol{p_c})_j+ \sum_{i=1}^{m}\sum_{j=1}^{m}2\,\boldsymbol{RP}_{i+j}(\boldsymbol{r_c})_i(\boldsymbol{p_c})_j$$
        - If $\boldsymbol{pAp}\le 0$ terminate due to instability, else
        - Compute $\alpha=\frac{rr}{\boldsymbol{pAp}}$
        - Compute $\boldsymbol{x_c}=\boldsymbol{x_c}+\alpha\cdot\boldsymbol{p_c}$
        - Compute $(\boldsymbol{r_c})_{2\dots m}=(\boldsymbol{r_c})_{2\dots m}-a(\boldsymbol{p_c})_{1\dots m-1}$
        - Compute
```

$$rr_2 = \sum_{i=1}^{m} RR_{2i-1}\,(r_c)_i + \sum_{i=1}^{m} PP_{2i-1}\,(p_c)_i +$$
$$\sum_{i=1}^{m-1}\sum_{j=i+1}^{m} 2\,RR_{i+j-1}(r_c)_i(r_c)_j + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} 2\,PP_{i+j-1}(p_c)_i(p_c)_j +$$
$$\sum_{i=1}^{m}\sum_{j=1}^{m} 2\,RP_{i+j-1}(r_c)_i(p_c)_j$$

- Compute $\beta = \frac{rr_2}{rr}$
- Compute $\boldsymbol{p}_c = \boldsymbol{r_c} + \beta \cdot \boldsymbol{p}_c$
- Set $rr = rr_2$
- If $rr < threshold$ proceed to compute $\boldsymbol{x}$ and terminate.

- close second loop

- Compute the actual solution $\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{M}^{-1} \cdot \left[\boldsymbol{R} \cdot (\boldsymbol{x_c})_{,\mathbf{1}} + \boldsymbol{P} \cdot (\boldsymbol{x_c})_{,\mathbf{2}}\right]$
- If number of iterations exceeded the maximun allowed terminate unsuccessfully, else
- Compute $\boldsymbol{r}$ και $\boldsymbol{p}$ as previously (e.g. $\boldsymbol{r} = \boldsymbol{R} \cdot (\boldsymbol{r_c})_{,\mathbf{1}} + \boldsymbol{P} \cdot (\boldsymbol{r_c})_{,\mathbf{2}}$)
- Compute the $n \cdot m$ matrix $\boldsymbol{R} = KrylovSubspace\left(\boldsymbol{A} \cdot \boldsymbol{M}^{-\mathbf{1}}, \boldsymbol{r}\right)$ and the $n \cdot (m+1)$ matrix $\boldsymbol{P} = KrylovSubspace\left(\boldsymbol{A} \cdot \boldsymbol{M}^{-\mathbf{1}}, \boldsymbol{p}\right)$.
- Compute and store in $(2m-1)$-sized vectors the inner products: $\boldsymbol{RR} = \boldsymbol{R}_{,\boldsymbol{i}} \cdot \boldsymbol{M}^{-\mathbf{1}} \cdot \boldsymbol{R}_{,\boldsymbol{j}}$, for $1 \le i, j \le m$ and $i + j - 1 = 1,2, \dots, 2m-1$.
- Compute and store in $(2m+1)$-sized vectors the inner products: $\boldsymbol{PP} = \boldsymbol{P}_{,\boldsymbol{i}} \cdot \boldsymbol{M}^{-\mathbf{1}} \cdot \boldsymbol{P}_{,\boldsymbol{j}}$, for $1 \le i, j \le m+1$ and $i + j - 1 = 1,2, \dots, 2m+1$
- Compute and store in $2m$-sized vectors the inner products: $\boldsymbol{RP} = \boldsymbol{R}_{,\boldsymbol{i}} \cdot \boldsymbol{M}^{-\mathbf{1}} \cdot \boldsymbol{P}_{,\boldsymbol{j}}$, for $1 \le i \le m$, $1 \le j \le m+1$ and $i + j - 1 = 1,2, \dots, 2m$
- close first loop

2.4 Algebraic Multigrid method

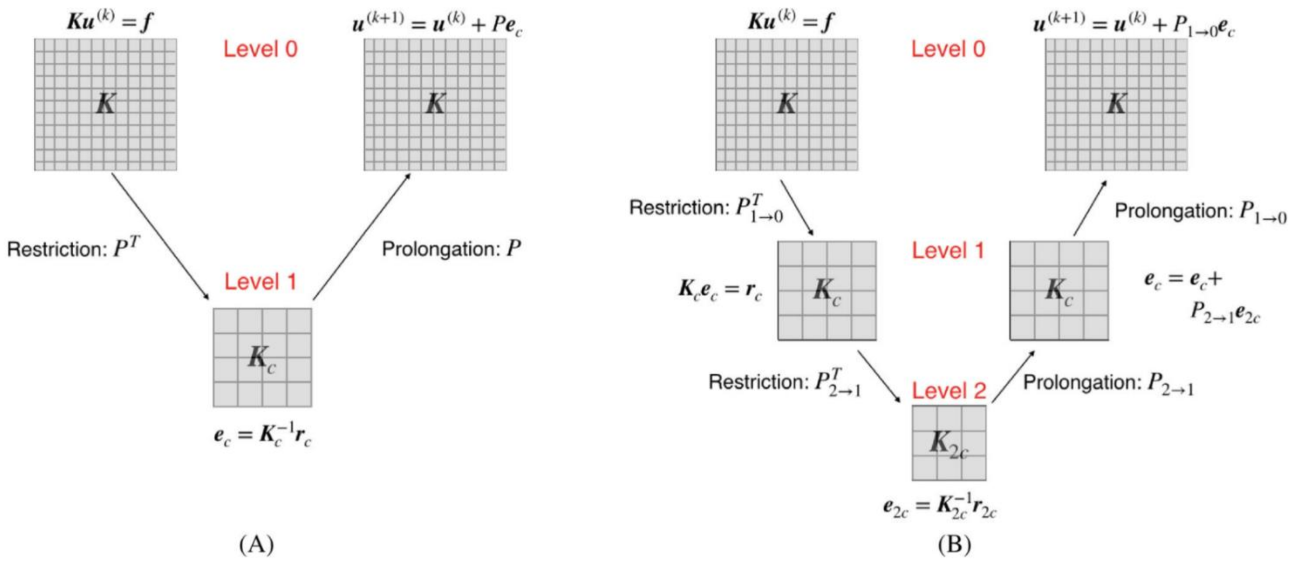
The key idea in AMG algorithms is to employ a hierarchy of progressively coarser approximations to the linear system under consideration in order to accelerate the convergence of classical simple and cheap iterative processes, such as the damped Jacobi or Gauss-Seidel. These methods, commonly referred to as relaxation or smoothing are very efficient in eliminating the high-frequency error modes, but inefficient in resolving low-energy modes. AMG overcomes this problem through the coarse-level correction, as elaborated below.

Let us consider the linear system $\boldsymbol{Ku} = \boldsymbol{f}$, which describes the fine problem and let $\boldsymbol{u}^{(0)}$ be an initial solution to it. The two-level AMG defines a prolongation operator $\boldsymbol{P}$, which is a full-column rank matrix in $R^{d \times d_c}$, $d_c < d$, and a relaxation scheme such as the Gauss-Seidel (GS). Then, the two-level AMG algorithm consists of the steps shown in the following algorithm.

Algorithm 2. Two-level AMG algorithm

```
1:  Input: $\boldsymbol{K} \in \mathbb{R}^{d \times d}$, rhs $\boldsymbol{f} \in \mathbb{R}^{d}$, prolongation operator $\boldsymbol{P} \in \mathbb{R}^{d \times d_c}$, a relaxation scheme denoted as $\mathcal{G}$, residual tolerance
    $\delta$ and an initial approximation $\boldsymbol{u}^{(0)}$
2:  set $k = 0$, initial residual $\boldsymbol{r}^{(0)} = \boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}^{(0)}$
3:  while $\|\boldsymbol{r}^{(k)}\| < \delta$ do
4:      Pre-relaxation: Perform $r_1$ iterations of the relaxation scheme on the current approximation and obtain $\boldsymbol{u}^{(k)}$ as:
    $\boldsymbol{u}^{(k)} \leftarrow \mathcal{G}\left(\boldsymbol{u}^{(k)}; r_1\right)$
5:      Update the residual: $\boldsymbol{r}^{(k)} = \boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}^{(k)}$
6:      Restrict the residual to the coarser level and solve the coarse level system $\boldsymbol{K}_c\boldsymbol{e}_c^{(k)} = \boldsymbol{P}^T\boldsymbol{r}^{(k)}$, where $\boldsymbol{K}_c = \boldsymbol{P}^T\boldsymbol{K}\boldsymbol{P} \in$
    $\mathbb{R}^{d_c \times d_c}$
7:      Prolongate the coarse grid error $\boldsymbol{e}^{(k)} = \boldsymbol{P}\boldsymbol{e}_c^{(k)}$
8:      Correct the fine grid solution: $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \boldsymbol{e}^{(k)}$
9:      Post-relaxation: Perform additional $r_2$ relaxation iterations and obtain $\boldsymbol{u}^{(k+1)} \leftarrow \mathcal{G}\left(\boldsymbol{u}^{(k+1)}; r_2\right)$
10:     $k = k + 1$
11: end while
```

In the above algorithm, lines 4-10 describe what is known as a V-cycle, schematically depicted in Figure 1. The multi-level version of the algorithm can be obtained by recursively applying the two-level algorithm.



$Ku^{(k)} = f$
Level 0
$u^{(k+1)} = u^{(k)} + Pe_c$
K
Restriction: P^T
Prolongation: P
Level 1
K_c
$e_c = K_c^{-1} r_c$
(A)
$Ku^{(k)} = f$
Level 0
$u^{(k+1)} = u^{(k)} + P_{1\to 0} e_c$
Restriction: $P_{1\to 0}^T$
Prolongation: $P_{1\to 0}$
Level 1
$K_c e_c = r_c$
$e_c = e_c + P_{2\to 1} e_{2c}$
Restriction: $P_{2\to 1}^T$
Level 2
Prolongation: $P_{2\to 1}$
K_{2c}
$e_{2c} = K_{2c}^{-1} r_{2c}$
(B)

Figure 1: Multigrid V-cycles in a (A) 2-level and a (B) 3-level setting

We will use the notation $\boldsymbol{u}^{(k+1)} = AMG(\boldsymbol{u}^{(k)}; \boldsymbol{K}, \boldsymbol{f}, r_1, r_2)$ to denote the application of one AMG cycle with r_1 iterations for pre-relaxation and r_2 iterations for post-relaxation.

Even though the AMG algorithm can be used as a standalone iterative solver, yet its true potential lies in its application as a preconditioned in the context of the PCG. The AMG preconditioned PCG algorithm is illustrated below.

Algorithm 3. AMG preconditioned PCG algorithm

```
1: Input: $\boldsymbol{K} \in \mathbb{R}^{d\times d}$, rhs $\boldsymbol{f} \in \mathbb{R}^d$, AMG scheme, residual tolerance $\delta$ and an initial approximation $\boldsymbol{u}^{(0)}$
2: set $k = 0$, initial residual $\boldsymbol{r}^{(0)} = \boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}^{(0)}$
3: $\boldsymbol{s}_0 = AMG(\boldsymbol{u}^{(0)}; \boldsymbol{K}, \boldsymbol{r}^{(0)}, r_1, r_2)$
4: $\boldsymbol{p}_0 = \boldsymbol{s}_0$
5: while $\|\boldsymbol{r}^{(k)}\| < \delta$ do
6:     $\alpha_k = \frac{(\boldsymbol{r}^{(k)})^T \boldsymbol{s}_k}{\boldsymbol{p}_k^T \boldsymbol{K} \boldsymbol{p}_k}$
7:     $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \alpha_k \boldsymbol{p}_k$
8:     $\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha_k \boldsymbol{K}\boldsymbol{p}_k$
9:     $\boldsymbol{s}_{k+1} = AMG(\boldsymbol{u}^{(k+1)}; \boldsymbol{K}, \boldsymbol{r}^{(k+1)}, r_1, r_2)$
10:    $\beta_k = \frac{(\boldsymbol{r}^{(k+1)})^T \boldsymbol{s}_{k+1}}{(\boldsymbol{r}^{(k)})^T \boldsymbol{s}_k}$
11:    $\boldsymbol{p}_{k+1} = \boldsymbol{s}_{k+1} + \beta_k \boldsymbol{p}_k$
12:    $k = k + 1$
13: end while
```

2.5 Domain Decomposition methods

This section presents the basic aspects of domain decomposition methods (DDM) which provide the foundation for the development of the more advanced primal and dual DDM (P-DDM and D-DDM respectively) used for the solution of large-scale systems.

2.5.1 Subdomains and mapping operators

Subdomain mapping operators for DDM can be implemented for mapping either the displacements and applied loads or the Lagrange multipliers of the subdomains. If $\boldsymbol{u}$ and $\boldsymbol{f}$ represent the displacement and applied loads vectors of the global domain and $\boldsymbol{u}^s$and $\boldsymbol{f}^s$ are vectors which refer to the corresponding quantities for every subdomain, the following equations hold:

$$\boldsymbol{u}^s = [u^{(1)} \quad \dots \quad u^{(N_s)}]^T \;, \quad \boldsymbol{f}^s = [f^{(1)} \quad \dots \quad f^{(N_s)}]^T, \quad \boldsymbol{u}^s = \boldsymbol{L}\boldsymbol{u}, \text{ and } \boldsymbol{f} = \boldsymbol{L}^T\boldsymbol{f}^s$$

where N_s is the number of non-overlapping subdomains of the global domain and L is the so-called global to local mapping operator which is a Boolean matrix. These equations, when applied to the interface degrees of freedom become

$$\boldsymbol{u}_{\boldsymbol{b}}^s = \left[u_b^{(1)} \quad \dots \quad u_b^{(N_s)}\right]^T \;, \quad \boldsymbol{f}_{\boldsymbol{b}}^s = \left[f_b^{(1)} \quad \dots \quad f_b^{(N_s)}\right]^T, \quad \boldsymbol{u}_{\boldsymbol{b}}^s = \boldsymbol{L}_{\boldsymbol{b}}\boldsymbol{u}_{\boldsymbol{b}}, \text{ and } \boldsymbol{f}_{\boldsymbol{b}} = \boldsymbol{L}_{\boldsymbol{b}}^T\boldsymbol{f}_{\boldsymbol{b}}^s$$

The traction forces on the interface nodes of the disconnected subdomains are usually expressed as:

$$\boldsymbol{t} = \boldsymbol{f}^s - \boldsymbol{B}^{\boldsymbol{T}}\boldsymbol{\lambda}$$

or

$$\boldsymbol{t}_b = \boldsymbol{f}_b^s - \boldsymbol{B}_b{}^T\boldsymbol{\lambda}$$

when applied to the interface dof, where **λ** is the vector of the Lagrange multipliers and $\boldsymbol{B}$ is the so-called Lagrange mapping operator. The form of the mapping operator depends on the definition of the Lagrange multipliers. In the case of redundant Lagrange multipliers, which are used in the present investigation, $\boldsymbol{B}$ is a signed Boolean matrix.

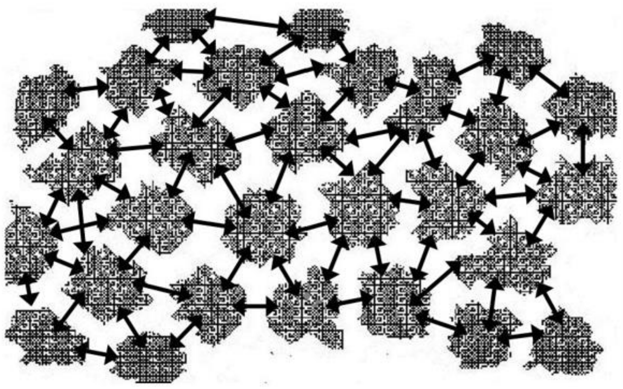


Figure 2: A structural domain, split into subdomains. Arrows show the traction forces between the disconnected subdomains

The Lagrange mapping matrices may also be used to express the displacement compatibility condition at subdomain interfaces as:

$$\boldsymbol{B}\boldsymbol{u}_s = [\boldsymbol{B}^{(1)} \quad \cdots \quad \boldsymbol{B}^{(Ns)}]\begin{bmatrix} \boldsymbol{u}^{(1)} \\ \vdots \\ \boldsymbol{u}^{(Ns)} \end{bmatrix} = \boldsymbol{0}$$

or

$$\boldsymbol{B}_b\boldsymbol{u}_b^s = \left[\boldsymbol{B}_b^{(1)} \quad \cdots \quad \boldsymbol{B}_b^{(Ns)}\right]\begin{bmatrix} \boldsymbol{u}_b^{(1)} \\ \vdots \\ \boldsymbol{u}_b^{(Ns)} \end{bmatrix} = \boldsymbol{0}$$

when applied to the interface dof.

Specific attention has to be paid when using these mapping operators in preconditioning steps of dual DDM. In the case of global to local mapping operator with respect to homogeneous problems, the global to local mapping operator in the preconditioning step can be written as:

$$\boldsymbol{L}_p = \boldsymbol{L}(\boldsymbol{M}^s)^{-1}$$

or

$$\boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}} = \boldsymbol{L}_b(\boldsymbol{M}_b^s)^{-1}$$

Moreover, for cases of splitting displacements or forces to heterogeneous subdomains inside a preconditioning step, we get:

$$\boldsymbol{u}_{b_{estimate}} = \boldsymbol{L}_{p_b}^{T} \boldsymbol{u}_b^s$$
$$\boldsymbol{f}_{b_{estimate}}^{s} = \boldsymbol{L}_{p_b} \boldsymbol{f}_b$$

In the past, a number of modified versions of the Lagrange mapping operator that incorporate scaling effects have been used in preconditioning steps of the dual DDM. In the case of redundant Lagrange multipliers and homogeneous problems, the Lagrange mapping operator in the preconditioning step can be written as:

$$\boldsymbol{B}_p = \boldsymbol{B}(\boldsymbol{M}^s)^{-1}$$

or

$$\boldsymbol{B}_{p_b} = \boldsymbol{B}_b(\boldsymbol{M}_b^s)^{-1}$$

when applied to the interface dof, where $\boldsymbol{M}^s$ and M_b^s are diagonal matrices with diagonal entries the multiplicity of the corresponding dof, which correspond to the number of subdomains that this dof belongs to.

2.5.2 Local problem solution

Domain decomposition methods require the repeated solution of many local subdomain problems corresponding to the dof of the subdomains. These local subdomain problems are typically solved using a direct method since their size is very small compared to the size of the global problem. In the case of D-DDM, local subdomain problems are of the form:

$$\boldsymbol{K}^{(s)} \boldsymbol{u}^{(s)} = f^{(s)} - \boldsymbol{B}^{(s)^T} \boldsymbol{\lambda}$$

where $\boldsymbol{K}^{(s)}$is the stiffness matrix of each subdomain.

In the case of P-DDM, similar local subdomain problems require repeated solution. These problems are of the form:

$$\boldsymbol{S}^{(s)} \boldsymbol{u}_b^{(s)} = \hat{\boldsymbol{f}}_b^{(s)} - \boldsymbol{B}_b^{(s)^T} \boldsymbol{\lambda}$$

where $\boldsymbol{S}^{(s)}$ is the Schur complement matrix and $\hat{\boldsymbol{f}}_b{}^{(s)}$ is the condensed force vector, as shown later. The main difference of these local problems is the fact that they either refer to all of the subdomain dof or to the interface ones. In the latter case, the equations which are related to internal dof of the subdomains are eliminated first. In order to obtain the corresponding relations, the local subdomain problem is re-arranged so that it can be written in the form:

$$\begin{bmatrix} \boldsymbol{K}_{bb}^{(s)} & \boldsymbol{K}_{bi}^{(s)} \\ \boldsymbol{K}_{ib}^{(s)} & \boldsymbol{K}_{ii}^{(s)} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_b^{(s)} \\ \boldsymbol{u}_i^{(s)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_b^{(s)} \\ \boldsymbol{f}_i^{(s)} \end{bmatrix} - \begin{bmatrix} \boldsymbol{B}_b{}^T \\ 0 \end{bmatrix} \boldsymbol{\lambda}$$

with subscripts b and i denoting the restriction of the matrices to interface (boundary) and internal d.o.f., respectively. With this re-arrangement, the following matrices and vectors are defined:

$$\boldsymbol{S}^{(s)} = \boldsymbol{K}_{bb}^{(s)} - \boldsymbol{K}_{bi}^{(s)} \left(\boldsymbol{K}_{ii}^{(s)}\right)^{-1} \boldsymbol{K}_{ib}^{(s)}$$
$$\hat{\boldsymbol{f}}_b^{(s)} = \boldsymbol{f}_b^{(s)} - \boldsymbol{K}_{bi}^{(s)} \left(\boldsymbol{K}_{ii}^{(s)}\right)^{-1} \boldsymbol{f}_i^{(s)}$$

In the case of implicit dynamics, matrices $\boldsymbol{K}^{(s)}$, $\boldsymbol{K}_{bb}^{(s)}$, $\boldsymbol{K}_{ii}^{(s)}$ and $\boldsymbol{S}^{(s)}$ are coefficient matrices of the integrated dynamic equilibrium equations and as such, they are always positive definite. This means that the corresponding matrices of the subdomains have no null space and the structure they refer to (adequately constrained or not) have no rigid body modes.

2.5.3 Interface problem solution

Domain decomposition methods require the solution of an interface problem in the form of $\boldsymbol{Ax} = \boldsymbol{b}$**,** where $\boldsymbol{A}, \boldsymbol{x}$and $\boldsymbol{b}$ are the left-hand side matrix, the solution vector and the right-hand side vector, respectively. Usually, the left-hand side matrix is symmetric and positive definite or semi-definite. Furthermore, the above equation is typically solved iteratively with the standard PCG method. The use of the PCG method also requires the definition of a positive definite preconditioning matrix $\widetilde{\boldsymbol{A}}^{-1}$, as an approximation of the inverse or generalized inverse of $\boldsymbol{A}$.

In the particular case of a semi-definite left-hand side matrix $\boldsymbol{A}$ (i.e. for an unconstrained subdomain of a structural mechanics problem), a solution of the interface problem exists under the condition:

$$\boldsymbol{b} \in range(\boldsymbol{A}) \Leftrightarrow null(\boldsymbol{A})^T\, \boldsymbol{b} = \boldsymbol{0}$$

When the above condition holds, the interface problem has infinite solutions of the form:

$$\boldsymbol{x} = \widehat{\boldsymbol{x}} + null(\boldsymbol{A})\, \boldsymbol{a},\, \boldsymbol{a} \in \mathbb{R}^d$$

where $\widetilde{\boldsymbol{x}}$ is a particular solution of the local interface and a is any vector with dimension equal to the dimension d of the null-space of **A**. In the case of a semi-definite matrix **A**, the PCG succeeds by computing one of the above infinite solutions of the equation.

2.5.4 Rigid body modes

By splitting the displacements $\boldsymbol{u}^{(s)}$ of a subdomain into $\boldsymbol{u}_r^{(s)}$ and $\boldsymbol{u}_d^{(s)}$, the following stands:

$$\boldsymbol{u}^{(s)} = \boldsymbol{u}_r{}^{(s)} + \boldsymbol{u}_d{}^{(s)}$$

Displacements $\boldsymbol{u}_r^{(s)}$ are caused by the rigid body modes of the subdomain while displacements $\boldsymbol{u}_d^{(s)}$ are due to the stiffness $\boldsymbol{K}^{(s)}$. To exclude these displacements, a set of artificial constraints is imposed so that displacements $\boldsymbol{u}_r^{(s)}$ are equal to **0**. In practice, this can be accomplished by magnifying the corresponding diagonals of the stiffness matrix by orders of magnitude, constituting these dof practically rigid. The stiffness matrix that occurs from this procedure now represents a statically determined subdomain thus being positive definite and invertible, with its inverse being equal to the generalized inverse $\boldsymbol{K}^{(s)^+}$, and displacements $\boldsymbol{u}_d^{(s)}$ being equal to:

$$\boldsymbol{u}_d{}^{(s)} = \boldsymbol{K}^{(s)^+}\left(\boldsymbol{f}^{(s)} - \boldsymbol{B}^{(s)^T}\boldsymbol{\lambda}_b\right)$$

while displacements $\boldsymbol{u}_r^{(s)}$ are equal to:

$$\boldsymbol{u}_r{}^{(s)} = \boldsymbol{R}^{(s)}\boldsymbol{a}^{(s)}$$

where $\boldsymbol{R}^{(s)}$ is a matrix consisting of the rigid body modes of the subdomain and is equal to the null space of the stiffness matrix $\boldsymbol{K}^{(s)}$ and $\boldsymbol{a}^{(s)}$ is a vector representing the contribution of each rigid body mode at the displacement vector. This implies that the loads $\boldsymbol{f}^{(s)} - \boldsymbol{B}^{(s)^T}\boldsymbol{\lambda}_b$ are self-equilibrated which means that:

$$\boldsymbol{R}^{(s)^T}\left(\boldsymbol{f}^{(s)} - \boldsymbol{B}^{(s)^T}\boldsymbol{\lambda}_b\right) = 0$$

By combining the above equations we get:

$$\boldsymbol{u}^{(s)} = \boldsymbol{K}^{(s)^+}\left(\boldsymbol{f}^{(s)} - \boldsymbol{B}^{(s)^T}\boldsymbol{\lambda}_b\right) + \boldsymbol{R}^{(s)}\boldsymbol{a}^{(s)}$$

The previous two equations can be written in block form as:

$$\boldsymbol{R}^{s^T}\left(\boldsymbol{f}^s - \boldsymbol{B}^{s^T}\boldsymbol{\lambda}_b\right) = \boldsymbol{0}$$
$$\boldsymbol{u}^s = \boldsymbol{K}^{s^+}(\boldsymbol{f}^s - \boldsymbol{B}^T\boldsymbol{\lambda}) + \boldsymbol{R}^s\boldsymbol{a}$$

where

$$\boldsymbol{K}^{s^+} = \begin{bmatrix} K^{(1)^+} & . & . \\ . & \ddots & . \\ . & . & K^{(Ns)^+} \end{bmatrix}, \boldsymbol{R}^s = \begin{bmatrix} R^{(1)} & . & . \\ . & \ddots & . \\ . & . & R^{(Ns)} \end{bmatrix}, \boldsymbol{a} = \begin{bmatrix} a^{(1)} \\ \vdots \\ a^{(Ns)} \end{bmatrix}$$

Regarding the interface equations, a corresponding block form is similarly derived:

$$\boldsymbol{R}_b^{s^T}\left(\hat{\boldsymbol{f}}_b^s - \boldsymbol{B}_b^T\boldsymbol{\lambda}\right) = 0$$

$$\boldsymbol{u}_b^s = \boldsymbol{S}^{s^+}\left(\hat{\boldsymbol{f}}_b^s - \boldsymbol{B}_b^T\boldsymbol{\lambda}\right) + \boldsymbol{R}_b^s\boldsymbol{a}$$

with

$$\boldsymbol{S}^{s^+} = \begin{bmatrix} S^{(1)^+} & . & . \\ . & \ddots & . \\ . & . & S^{(Ns)^+} \end{bmatrix}, \hat{\boldsymbol{f}}_b^s = \begin{bmatrix} \hat{f}_b^{\,(1)} \\ \vdots \\ \hat{f}_b^{\,(Ns)} \end{bmatrix}$$

and $\boldsymbol{R}_b^s$ denoting the restriction of $\boldsymbol{R}^s$ to the interface dof.

2.5.5 P-DDM: The Primal Substructuring Method (PSM)

The basic DDM is the primal substructuring method, abbreviated in the following as PSM. In the context of this DDM, the internal dof of the subdomains are eliminated first. The PSM interface displacement problem is thus obtained:

$$\widehat{\boldsymbol{S}}\boldsymbol{u}_b = \hat{\boldsymbol{f}}_b$$

where

$$\widehat{\boldsymbol{S}} = \boldsymbol{L}_b^T\boldsymbol{S}^s\boldsymbol{L}_b,\ \hat{\boldsymbol{f}}_b = \boldsymbol{L}_b^T\hat{\boldsymbol{f}}_b^s$$

$$\hat{\boldsymbol{f}}_b^s = \left[\hat{f}_b^{(1)} \quad \dots \quad \hat{f}_b^{(Ns)}\right]^T$$

$$\boldsymbol{S}^s = \begin{bmatrix} S^{(1)} & . & . \\ . & \ddots & . \\ . & . & S^{(Ns)} \end{bmatrix}$$

The solution of the above linear system is usually performed with the PCG method, since the left-hand side matrix is symmetric and positive definite or semi-definite.

In the past, several strategies have been proven efficient for preconditioning the underlying iterative solver for these types of problems with a common choice being the preconditioner:

$$\widetilde{\boldsymbol{A}}^{-1} = \boldsymbol{L}_{p_b}^T\boldsymbol{S}^{s^+}\boldsymbol{L}_{p_b}$$

which is used in the so-called Neumann–Neumann PSM. More precisely, the preconditioner $\widetilde{\boldsymbol{A}}^{-1}$ is implemented as follows:

$$\widetilde{\boldsymbol{A}}^{-1} = \boldsymbol{L}_{p_b}^T\boldsymbol{N}_b^s\boldsymbol{K}^{s^+}\boldsymbol{N}_b^{s^T}\boldsymbol{L}_{p_b}$$

where $\boldsymbol{N}_b^s$ is a Boolean matrix which extracts the interface dof from subdomain dof vectors, as:

$$\boldsymbol{u}_b^s = \boldsymbol{N}_b^s\boldsymbol{u}^s, \quad \boldsymbol{f}_b^s = \boldsymbol{N}_b^s\boldsymbol{f}^s$$

2.5.6 D-DDM: The Finite Element Tearing and Interconnecting (FETI) method

The FETI method, is a dual DDM that has been implemented for a number of problems in computational mechanics. Since its introduction, it has attracted a lot of attention and is considered as a fast domain decomposition algorithm suitable for both serial and parallel computing environments. While its predecessor, the PSM, performs iterations in order to compute the interface displacement vector $\boldsymbol{u}_b$ of the structure, the FETI method iterates on the Lagrange multiplier vector $\boldsymbol{\lambda}$. The Lagrange multipliers, which represent the interaction forces between the subdomains, are dual with respect to the interface displacements and this explains the name dual substructuring method, in comparison to PSM. In the context of the FETI method, the nodal force vector $\boldsymbol{f}$ of the structure is first split to the subdomains:

$$\boldsymbol{f}^s = \boldsymbol{L}_p\boldsymbol{f}$$

and the following system of equations is obtained:

$$\begin{bmatrix} \boldsymbol{F}_I & -\boldsymbol{G} \\ -\boldsymbol{G}^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \boldsymbol{d} \\ -\boldsymbol{e} \end{bmatrix}$$

where

$$\boldsymbol{F_I} = \boldsymbol{B}\boldsymbol{K}^{s^+}\boldsymbol{B}^T, \boldsymbol{G} = \boldsymbol{B}\boldsymbol{R}^s, \boldsymbol{d} = \boldsymbol{B}\boldsymbol{K}^{s^+}\boldsymbol{f}^s, \boldsymbol{e} = \boldsymbol{R}^{s^T}\boldsymbol{f}^s$$

Then, the following projector is introduced:

$$\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{Q}\boldsymbol{G}\left(\boldsymbol{G}^T\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{G}^T$$

where, for homogeneous problems, operator $\boldsymbol{P}$ is usually implemented with $\boldsymbol{Q} = \boldsymbol{I}$. However, for heterogeneous problems matrix $\boldsymbol{Q}$ might be set otherwise as described at the end of this section.
Computations involving projector $\boldsymbol{P}$ require the solution of linear problems of the form:

$$\left(\boldsymbol{G}^T\boldsymbol{Q}\boldsymbol{G}\right)\boldsymbol{x} = \boldsymbol{b}$$

which constitutes the so-called ''coarse-grid'' problem of the FETI method. This name is explained by the fact that $\boldsymbol{G}^T\boldsymbol{Q}\boldsymbol{G}$ is a sparse matrix, with the typical sparsity pattern of a finite element stiffness matrix, if one considers each subdomain as a finite element node having the same number of dof as the number of its zero energy modes. This coarse problem ensures the exchange of information between remote subdomains of the structure at each iteration of the underlying iterative solver used for the interface problem, thus guaranteeing fast convergence.
Premultiplying the first of the two matrix equations with $\left(\boldsymbol{G}^T\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{G}^T\boldsymbol{Q}$, it follows that for a given Lagrange multiplier vector $\boldsymbol{\lambda}$, the vector of the zero energy mode amplitudes $\boldsymbol{a}$ is equal to:

$$\boldsymbol{a} = -\left(\boldsymbol{G}^T\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{G}^T\boldsymbol{Q}(\boldsymbol{d} - \boldsymbol{F_I}\boldsymbol{\lambda})$$

Furthermore, it follows that the jump $\varDelta\boldsymbol{u_b} = \boldsymbol{B}\boldsymbol{u}^s$ of the displacement field at subdomain interfaces is equal to:

$$\varDelta\boldsymbol{u_b} = \boldsymbol{B}\boldsymbol{u_s} = \boldsymbol{d} - \boldsymbol{F_I}\boldsymbol{\lambda} + \boldsymbol{G}\boldsymbol{a} = \boldsymbol{P}^T(\boldsymbol{d} - \boldsymbol{F_I}\boldsymbol{\lambda})$$

Thus, the linear system is equivalent to the following system where the unknown vectors $\boldsymbol{\lambda}$ and $\boldsymbol{a}$ are decoupled:

$$\boldsymbol{P}^T\boldsymbol{F_I}\boldsymbol{\lambda} = \boldsymbol{P}^T\boldsymbol{d}$$

$$\boldsymbol{G}^T\boldsymbol{a} = \boldsymbol{e}$$

In order to solve the above equations for the Lagrange multiplier vector λ, the latter is being split as follows:

$$\boldsymbol{\lambda} = \boldsymbol{\lambda_0} + \boldsymbol{P}\bar{\boldsymbol{\lambda}}$$

where vector $\boldsymbol{\lambda_0}$ is chosen equal to:

$$\boldsymbol{\lambda_0} = \boldsymbol{Q}\boldsymbol{G}\left(\boldsymbol{G}^T\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{e}$$

Therefore, we end up with the following interface problem:

$$\left(\boldsymbol{P}^T\boldsymbol{F_I}\boldsymbol{P}\right)\bar{\boldsymbol{\lambda}} = \boldsymbol{P}^T(\boldsymbol{d} - \boldsymbol{F_I}\boldsymbol{\lambda_0})$$

In order to calculate the total displacement field $\boldsymbol{u}$, the following steps are followed:

- The Lagrange multiplier vector $\bar{\boldsymbol{\lambda}}$ is computed by solving the interface problem of the previous equation.
- The Lagrange multiplier vector $\boldsymbol{\lambda}$ is evaluated from $\boldsymbol{\lambda} = \boldsymbol{\lambda_0} + \boldsymbol{P}\bar{\boldsymbol{\lambda}}$.
- The amplitudes a of the subdomain rigid body modes are computed from $\boldsymbol{a} = -(\boldsymbol{G}^T\boldsymbol{Q}\boldsymbol{G})^{-1}\boldsymbol{G}^T\boldsymbol{Q}(\boldsymbol{d} - \boldsymbol{F_I}\boldsymbol{\lambda})$.
- Subdomain displacement fields u^s are computed from $\boldsymbol{u}^s = \boldsymbol{K}^{s^+}(\boldsymbol{f}^s - \boldsymbol{B}^T\boldsymbol{\lambda}) + \boldsymbol{R}^s\boldsymbol{a}$.
- The total displacement field u of the structure is finally given by $\boldsymbol{u} = \boldsymbol{L}_p^T\boldsymbol{u}^s$

The two most widely used preconditioners for the FETI method are:

$$\widetilde{\boldsymbol{F}}_{\boldsymbol{I}}^{\boldsymbol{D}^{-1}} = \boldsymbol{B}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{S}^{s}\boldsymbol{B}_{\boldsymbol{p}_{\boldsymbol{b}}}^{\boldsymbol{T}}$$

$$\widetilde{\boldsymbol{F}}_{\boldsymbol{I}}^{\boldsymbol{L}^{-1}} = \boldsymbol{B}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{K}_{\boldsymbol{bb}}^{s}\boldsymbol{B}_{\boldsymbol{p}_{\boldsymbol{b}}}^{\boldsymbol{T}}$$

namely, the Dirichlet and the lumped preconditioners. The Dirichlet preconditioner is typically used in fourth-order problems. Moreover, in second-order problems, the lumped preconditioner is usually more efficient in terms of the total solution time. In some second-order problems however, namely in highly heterogeneous structures and in problems where subdomains of bad aspect ratio are generated, the Dirichlet preconditioner may outperform the lumped one. Variant forms of the Dirichlet preconditioner using approximate expressions for $\boldsymbol{K}_{ii}^{s}$ of the Schur complement $\boldsymbol{S}^{s}$may also be used. Accordingly, these preconditioners can be used as values for matrix $\boldsymbol{Q}$ of projector $\boldsymbol{P}$ in case of heterogeneous problems.

2.5.7 P-DDM for static analysis with D-DDM based preconditioners: The PFETI method

This section introduces a new category of preconditioners for the PSM originally proposed by Fragakis and Papadrakakis [1]. An iterative solver is applied for the solution of the interface problem in order to compute the interface displacement vector $\boldsymbol{u}_{b}$, given an interface force vector $\widehat{\boldsymbol{f}}_{b}$. A good preconditioner for the PSM must treat the k-th residual $\boldsymbol{r}^{\boldsymbol{k}} = \widehat{\boldsymbol{f}}_{\boldsymbol{b}} - \widehat{\boldsymbol{S}}\boldsymbol{u}_{\boldsymbol{b}}^{\boldsymbol{k}}$ as applied forces on the interface nodes of the structure and return in $\boldsymbol{z}^{\boldsymbol{k}} = \widetilde{\boldsymbol{A}}^{-1}\boldsymbol{r}^{\boldsymbol{k}}$ a good estimate of the interface displacements of the structure for the applied forces $\boldsymbol{r}^{\boldsymbol{k}}$. For instance, if the PSM preconditioning step is performed with any solver, like for example the FETI method, the iterative solver will immediately converge in the first iteration. The PFETI method consists of using as preconditioner of the PSM, a crude approximation of the FETI solution and as such, the first estimate for the interface displacements of the structure obtained from the first iteration of the FETI method is chosen.
For example, consider the FETI algorithm with an applied forces vector $\boldsymbol{f}$equal to:

$$\boldsymbol{f} = \boldsymbol{N}_{\boldsymbol{b}}^{\boldsymbol{T}}\boldsymbol{r}^{\boldsymbol{k}}$$

Since all forces in the load vector are applied on the interface nodes of the structure, we have $\boldsymbol{f}_{\boldsymbol{b}} = \boldsymbol{r}^{\boldsymbol{k}}$ and $\boldsymbol{f}_{\boldsymbol{i}}^{\boldsymbol{s}} = \boldsymbol{0}$. Furthermore, the interface forces $\boldsymbol{f}_{\boldsymbol{b}}$ may be split to the subdomains with the equation:

$$\boldsymbol{f}_{\boldsymbol{b}}^{\boldsymbol{s}} = \boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{f}_{\boldsymbol{b}} = \boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{r}^{\boldsymbol{k}}$$

Then, it follows that $\widehat{\boldsymbol{f}}_{\boldsymbol{b}}^{\boldsymbol{s}} = \boldsymbol{f}_{\boldsymbol{b}}^{\boldsymbol{s}} = \boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{r}^{\boldsymbol{k}}$. The components of e and d thus become $\boldsymbol{e} = \boldsymbol{R}^{\boldsymbol{s}^{\boldsymbol{T}}}\boldsymbol{f}^{\boldsymbol{s}} = \boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{s}^{\boldsymbol{T}}}\boldsymbol{r}^{\boldsymbol{k}}$ and $d = \boldsymbol{B}\boldsymbol{K}^{\boldsymbol{s}^{+}}\boldsymbol{f}^{\boldsymbol{s}} = \boldsymbol{B}_{\boldsymbol{b}}\boldsymbol{S}^{\boldsymbol{s}^{+}}\boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{r}^{\boldsymbol{k}}$. Moreover, with respect to interface values, matrix $\boldsymbol{F}_{\boldsymbol{I}}$ may be written as $\boldsymbol{F}_{\boldsymbol{I}} = \boldsymbol{B}\boldsymbol{K}^{\boldsymbol{s}^{+}}\boldsymbol{B}^{\boldsymbol{T}} = \boldsymbol{B}_{\boldsymbol{b}}\boldsymbol{S}^{\boldsymbol{s}^{+}}\boldsymbol{B}_{\boldsymbol{b}}{}^{\boldsymbol{T}}$. Futhermore, the initial Lagrange multiplier vector $\boldsymbol{\lambda}_{\boldsymbol{0}}$ is equal to:

$$\boldsymbol{\lambda}_{\boldsymbol{0}} = \boldsymbol{Q}\boldsymbol{G}\left(\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{s}^{\boldsymbol{T}}}\boldsymbol{r}^{\boldsymbol{k}}$$

The initial zero energy mode amplitude $\boldsymbol{a}_{\boldsymbol{0}}$ is equal to:

$$\boldsymbol{a}_{\boldsymbol{0}} = -\left(\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}(\boldsymbol{d} - \boldsymbol{F}_{\boldsymbol{I}}\boldsymbol{\lambda}_{\boldsymbol{0}}) = -\left(\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{B}_{\boldsymbol{b}}\boldsymbol{S}^{\boldsymbol{s}^{+}}\left(\boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{r}^{\boldsymbol{k}} - \boldsymbol{B}_{\boldsymbol{b}}^{\boldsymbol{T}}\boldsymbol{\lambda}_{\boldsymbol{0}}\right)$$

and the interface displacements $\boldsymbol{u}_{b_0}$ estimated from the initialization of the FETI method are:

$$\begin{aligned}
\boldsymbol{u}_{\boldsymbol{b}_{\boldsymbol{0}}} &= \boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}}^{\boldsymbol{T}}\boldsymbol{u}_{\boldsymbol{b}}^{\boldsymbol{s}} \\
&= \boldsymbol{L}_{\boldsymbol{pb}}^{\boldsymbol{T}}\left(\boldsymbol{S}^{\boldsymbol{s}^{+}}\left(\widehat{\boldsymbol{f}}_{\boldsymbol{b}}^{\boldsymbol{s}} - \boldsymbol{B}_{\boldsymbol{b}}^{\boldsymbol{T}}\boldsymbol{\lambda}_{\boldsymbol{0}}\right) + \boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{s}}\boldsymbol{a}_{\boldsymbol{0}}\right) \\
&= \boldsymbol{L}_{\boldsymbol{pb}}^{\boldsymbol{T}}\left(\boldsymbol{S}^{\boldsymbol{s}^{+}}\left(\boldsymbol{r}^{\boldsymbol{k}} - \boldsymbol{B}_{\boldsymbol{b}}^{\boldsymbol{T}}\boldsymbol{\lambda}_{\boldsymbol{0}}\right) - \boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{s}}\left(\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{B}_{\boldsymbol{b}}\boldsymbol{S}^{\boldsymbol{s}^{+}}\left(\boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{r}^{\boldsymbol{k}} - \boldsymbol{B}_{\boldsymbol{b}}^{\boldsymbol{T}}\boldsymbol{\lambda}_{\boldsymbol{0}}\right)\right) \\
&= \boldsymbol{L}_{\boldsymbol{pb}}^{\boldsymbol{T}}\left(\boldsymbol{I} - \boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{s}}\left(\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{B}_{\boldsymbol{b}}\right)\boldsymbol{S}^{\boldsymbol{s}^{+}}\left(\boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{r}^{\boldsymbol{k}} - \boldsymbol{B}_{\boldsymbol{b}}^{\boldsymbol{T}}\boldsymbol{\lambda}_{\boldsymbol{0}}\right) \\
&= \boldsymbol{L}_{\boldsymbol{pb}}^{\boldsymbol{T}}\left(\boldsymbol{I} - \boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{s}}\left(\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{B}_{\boldsymbol{b}}\right)\boldsymbol{S}^{\boldsymbol{s}^{+}}\left(\boldsymbol{I} - \boldsymbol{B}_{\boldsymbol{b}}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{G}\left(\boldsymbol{G}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{G}\right)^{-1}\boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{s}^{\boldsymbol{T}}}\right)\boldsymbol{L}_{\boldsymbol{p}_{\boldsymbol{b}}}\boldsymbol{r}^{\boldsymbol{k}}
\end{aligned}$$

From the above equation, the PSM preconditioner is deduced:

$$\widetilde{\boldsymbol{A}}^{-1} = \boldsymbol{L}_{p_b}^{T} \boldsymbol{H}_b^{T} \boldsymbol{S}^{s^+} \boldsymbol{H}_b \boldsymbol{L}_{p_b}$$

where

$$\boldsymbol{H}_b = \boldsymbol{I} - \boldsymbol{B}_b^{T} \boldsymbol{Q}\boldsymbol{G}\left(\boldsymbol{G}^{T}\boldsymbol{Q}\boldsymbol{G}\right)^{-1} {\boldsymbol{R}_b^{s}}^{T}$$

3. AI accelerated Iterative Solvers for parameterized problems

The applications of the DCoMEX project, as described in Work Package 7, involve the modelling of tumor growth during immunotherapy treatment and the design of high-performance composite materials. Both these applications are described by a complex set of PDEs and are parametrized either by random variables (forward/inverse UQ analysis) and/or design variables (optimization). The solution to these problems requires massive number of problem simulations for various instances of the problem parameters. The computational cost of this task is computationally unaffordable, even with the use of the advanced iterative algorithms mentioned in the previous section. To overcome this problem, a set of algorithms has been developed so far that combine advanced linear algebraic solvers with ML algorithms to accelerate the solutions to these complex problems.

3.1 Data-driven solution framework-POD-2G solver

The POD-2G algorithm has been developed in the context of the DCoMEX project in the effort to accelerate the solution process of large-scale parameterized systems [2]. It consists of the following three steps:

Step 1: Development of a surrogate model using convolutional neural networks and autoencoders

In this step, a small yet sufficient number of parameter instances $\boldsymbol{\theta}_i$ for $i = 1, \dots, N$, is generated and it is fed as input to the governing PDE of the problem. Then, using a discretization scheme, the corresponding high-fidelity solutions $\boldsymbol{u}_i \in R^d$ for $i = 1, \dots, N$, are obtained. For a detailed numerical model d is very large number and this poses significant challenges when trying to establish a direct mapping from the problem's parametric space to its solution space. To tackle this, we will employ the tools developed in WP2 of the project, namely the CAE-based surrogate model illustrated in figures 3.

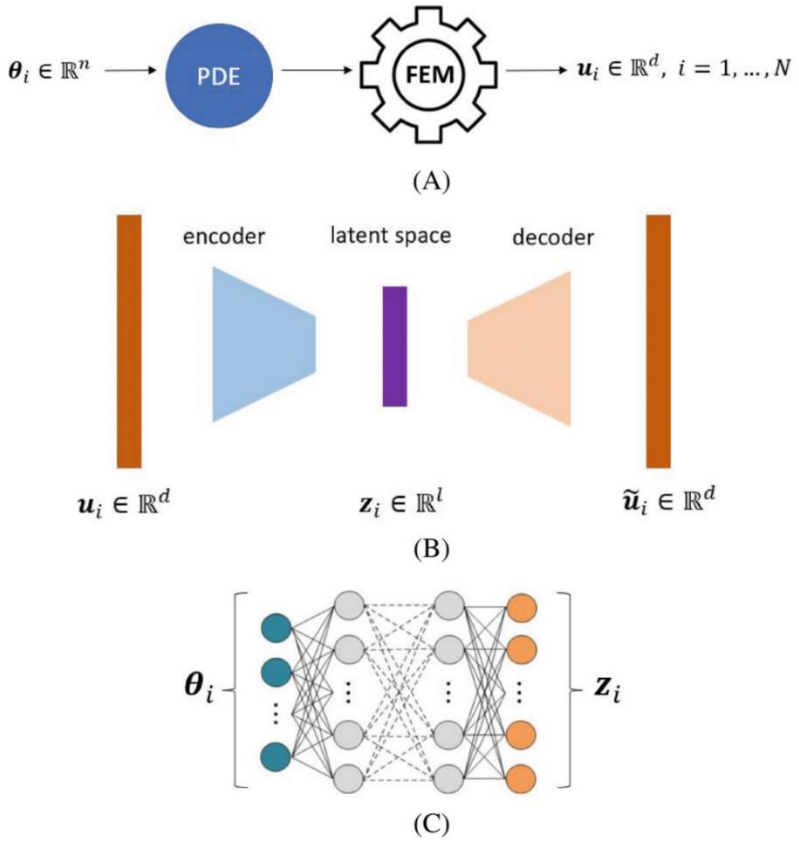

$\boldsymbol{\theta}_i \in \mathbb{R}^n$
PDE
FEM
$\boldsymbol{u}_i \in \mathbb{R}^d,\ i = 1, \dots, N$
(A)
encoder
latent space
decoder
$\boldsymbol{u}_i \in \mathbb{R}^d$
$\boldsymbol{z}_i \in \mathbb{R}^l$
$\tilde{\boldsymbol{u}}_i \in \mathbb{R}^d$
(B)
$\boldsymbol{\theta}_i$
$\boldsymbol{z}_i$
(C)

Figure 3: A schematic representation of the surrogate's construction phase: (A) Generation of training samples by solving the high-fidelity numerical model for different parameter instances, (B) Learning of a low-dimensional representation for the data set and a corresponding reconstruction map using a CAE, (C) Learning a mapping from the parameter space to the encoded space using a feedforward neural network.

According to this figure, the CAE is trained over the initial high dimensional data set of $\{\boldsymbol{u}_i\}$, in order to learn a low dimensional representation denoted with $\{\boldsymbol{z}_i\}$, with $\boldsymbol{z}_i \in R^l, l \ll d$ and then learn how to minimize the objective function

$$Loss_{CAE} = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{u}_i - \widetilde{\boldsymbol{u}}_i\|$$

with $\widetilde{\boldsymbol{u}}_i$ being the reconstructed input. Next, the FFNN is used to establish a nonlinear mapping form the parametric space of $\boldsymbol{\theta} \in R^n$ to the latent space $\boldsymbol{z} \in R^l$, by minimizing the loss function

$$Loss_{FFNN} = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{z}_i - \tilde{\boldsymbol{z}}_i\|$$

with $\tilde{\boldsymbol{z}}_i = FFNN(\boldsymbol{\theta}_i)$ being the network's output.

Based on this surrogate modeling scheme, for a new parameter instance $\boldsymbol{\theta}_j$, the system's solution can be obtained as:

$$\boldsymbol{u}_j = decoder\left(FFNN(\boldsymbol{\theta}_j)\right) := F^{sur}(\boldsymbol{\theta}_j)$$

as shown in figure 4.

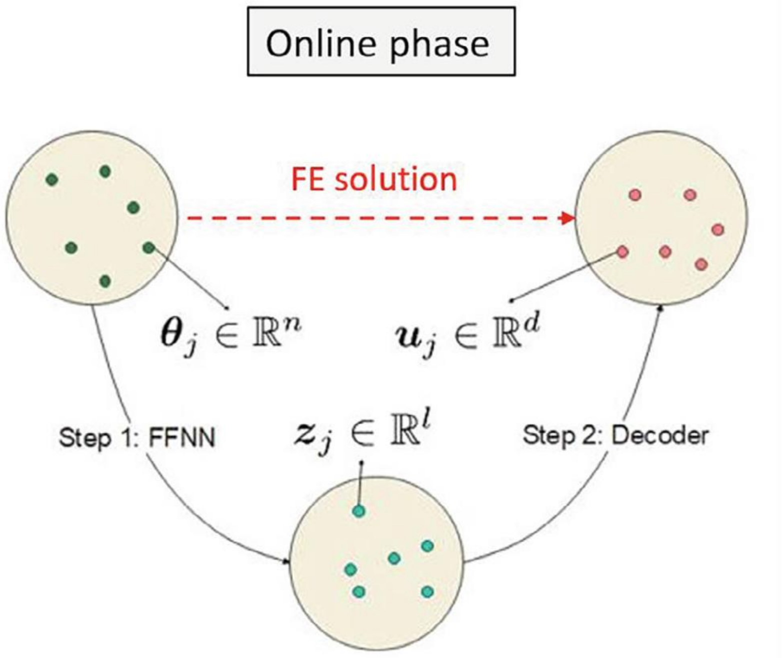

Online phase
FE solution
$\boldsymbol{\theta}_j \in \mathbb{R}^n$
$\boldsymbol{u}_j \in \mathbb{R}^d$
Step 1: FFNN
$\boldsymbol{z}_j \in \mathbb{R}^l$
Step 2: Decoder

Figure 4: A schematic representation of the surrogate online (prediction) phase

Step 2: Development of a multigrid-inspired POD solver, POD-2G

The solution vectors $\boldsymbol{u}_i$ that have already been collected in the previous step, are utilized once more to obtain a set of reduced basis vectors by applying the Proper Orthogonal Decomposition (POD) method on the solution matrix $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_N]$. By denoting with $\boldsymbol{\Phi}_r \in R^{d\times r}$ the matrix of the r most energetic modes of POD, then the high-dimensional system solutions $\boldsymbol{u}$ can be approximated as $\boldsymbol{u} = \boldsymbol{\Phi}_r \boldsymbol{u}_r$, with $\boldsymbol{u}_r \in R^r$ being the unknown coefficients of the projection on the truncated POD basis. Next, we exploit the similarity between the two-level AMG and the POD method, under the identification of $\boldsymbol{\Phi}_r$ as the prolongation operator and $\boldsymbol{\Phi}_r^T$ the corresponding restriction. By appropriately modifying the AMG algorithm, we obtain the POD-2G algorithm, illustrated below.

Algorithm 4. POD-2G algorithm

```
1: Input: $\boldsymbol{K} \in \mathbb{R}^{d\times d}$, rhs $\boldsymbol{f} \in \mathbb{R}^{d}$, prolongation operator $\boldsymbol{\Phi} \in \mathbb{R}^{d\times r}$, a relaxation scheme denoted as $\mathcal{G}$, residual tolerance $\delta$ and an initial approximation $\boldsymbol{u}^{(0)}$
2: set $k = 0$, initial residual $\boldsymbol{r}^{(0)} = \boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}^{(0)}$
3: while $\|\boldsymbol{r}^{(k)}\| < \delta$ do
4:     Pre-relaxation: Perform $r_1$ iterations of the relaxation scheme on the current approximation and obtain $\boldsymbol{u}^{(k)}$ as: $\boldsymbol{u}^{(k)} \leftarrow \mathcal{G}\left(\boldsymbol{u}^{(k)}; r_1\right)$
5:     Update the residual: $\boldsymbol{r}^{(k)} = \boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}^{(k)}$
6:     Restrict the residual to the coarser level and solve the coarse level system $\boldsymbol{K}_c\boldsymbol{e}_c^{(k)} = \boldsymbol{\Phi}^T\boldsymbol{r}^{(k)}$, where $\boldsymbol{K}_c = \boldsymbol{\Phi}^T\boldsymbol{K}\boldsymbol{\Phi} \in \mathbb{R}^{d_c\times d_c}$
7:     Prolongate the coarse grid error $\boldsymbol{e}^{(k)} = \boldsymbol{\Phi}\boldsymbol{e}_c^{(k)}$
8:     Correct the fine grid solution: $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \boldsymbol{e}^{(k)}$
9:     Post-relaxation: Perform additional $r_2$ relaxation iterations and obtain $\boldsymbol{u}^{(k+1)} \leftarrow \mathcal{G}\left(\boldsymbol{u}^{(k+1)}; r_2\right)$
10:    $k = k + 1$
11: end while
```

Step 3: Proposed data-driven framework

The final step is to combine the surrogate model of step 1 and POD-2G solver of step 2 into a single solution pipeline. In particular, for each new parameter instance $\boldsymbol{\theta}$, a highly accurate approximation of the system's solution is obtained as $F^{sur}(\boldsymbol{\theta})$. If we denote with $\boldsymbol{u}^\star$ the exact solution, then the surrogate's error will be $e^{sur} = \|\boldsymbol{u}^\star - F^{sur}(\boldsymbol{\theta})\|$, which, despite one's best efforts will now be equal to zero. At this point, we will utilize the *POD-2G* iterative scheme to refine the surrogate's prediction to any desired level of accuracy. This approach is expected to drastically reduce the computational cost of solving the system due to good initial prediction $F^{sur}(\boldsymbol{\theta})$ and the fact that POD-2G requires fewer iterations to converge than other conventional solvers. This procedure is summarized in the following algorithm:

Algorithm 5. Proposed data-driven solution framework

```
1: Step 1: Generation of the initial data set
2: Generate $N$ instances of the system parameters $\{\boldsymbol{\theta}_i\}_{i=1}^N$ and solve the detailed FE model to obtain the corresponding system responses $\{\boldsymbol{u}_i\}_{i=1}^N$.

3: Step 2: Construction of the data-driven solution framework
4: Step 2a: Utilize the data set $\{\boldsymbol{u}_i\}_{i=1}^N$ to train a CAE in order to learn a reduced representation $\boldsymbol{z} \in \mathbb{R}^l$ for $\boldsymbol{u} \in \mathbb{R}^d$, as well as a reconstruction map.
5: Step 2b: Utilize the data set of $\{\boldsymbol{\theta}_i\}_{i=1}^N - \{\boldsymbol{z}_i\}_{i=1}^N$ to train a FFNN in order to learn a mapping between the system parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ and the reduced representation $\boldsymbol{z} \in \mathbb{R}^l$.
6: Step 2c: Construct the matrix $\boldsymbol{U} = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_N]$ of solution vectors and perform POD by computing the eigenvalues and eigenvectors of the correlation matrix $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{U}^T$, or equivalently, the matrix $\boldsymbol{U}^T\boldsymbol{U}$. From the matrix of eigenvectors $\boldsymbol{\Phi}$, form the reduced basis $\boldsymbol{\Phi}_r$ by retaining only the $r$ most important eigenvectors.

7: Step 3: Application of the framework to additional model simulations
8: Step 3a: For a new parameter instance $\boldsymbol{\theta}$ use the FFNN to obtain the corresponding reduced representation $\boldsymbol{z}$.
9: Step 3b: Use the CAE's decoder to map $\boldsymbol{z}$ to the approximate solution vector $\boldsymbol{u}$.
10: Step 3c: Perform additional iterations of the POD-2G algorithm with $\boldsymbol{u}$ as the initial solution $\boldsymbol{u}^{(0)}$ until a prescribed accuracy threshold has been reached.
```

3.2 Fusing nonlinear solvers with transformers for accelerating parameterized transient problems

A second methodology that has been developed so far, is aimed at accelerating the solution to nonlinear transient problems using state-of-the-art machine learning tools. In particular, the proposed approach harnesses the power of cutting-edge Temporal Fusion Transformers (TFTs) to accelerate the solution of such problems in multi-query scenarios. At each time step of the transient problem, TFT models, renowned for their time series forecasting capabilities, are combined with dimensionality reduction techniques to efficiently generate initial solutions for nonlinear solvers.

Specifically, during the training phase, a reduced set of high-fidelity system solutions is obtained by solving the system of differential equations governing the problem for different parameter instances. Then, dimensionality reduction is applied to create a reduced latent space to simplify the representation of the complex system solutions. Subsequently, TFT models are trained for one-step-ahead forecasting in the latent space, utilizing information from previous states to make accurate predictions about future states. The TFTs' predictions are fed back to the system as initial guesses at each time step of the solution algorithm and are then guided towards the exact solutions that satisfy equilibrium using Newton-Raphson (NR) iterations. The basic premise of the proposed idea is that having accurate initial predictions will significantly decrease the number of the costly NR-iterations needed in nonlinear dynamic problems, effectively reducing the solution time.

The proposed algorithm consists of the following steps:

Step 1: Training phase

A training data set, denoted as $\boldsymbol{U}_{train} = \{\boldsymbol{U}_i\}_{i=1}^{N_{train}}$, with $\boldsymbol{U}_i = \left[\boldsymbol{u}_{i,t_0}, \dots, \boldsymbol{u}_{i,t_{N_T}}\right] \in R^{d \times N_T}$, N_T being the total number of time steps, is initially generated by performing N_{train} simulations of the high-fidelity model for different parameter instances $\{\theta_i\}_{i=1}^{N_{train}}$ (see figure 5).

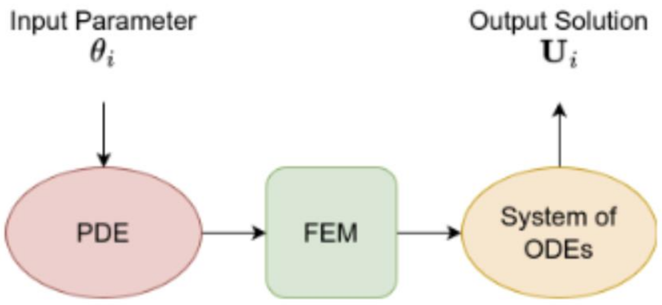
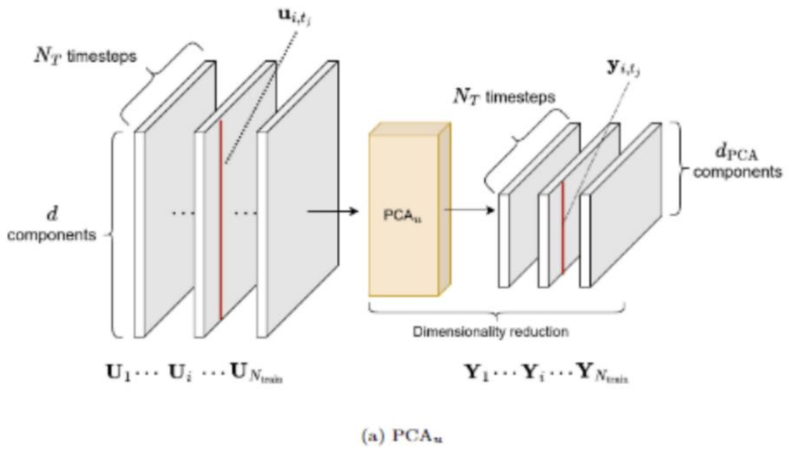
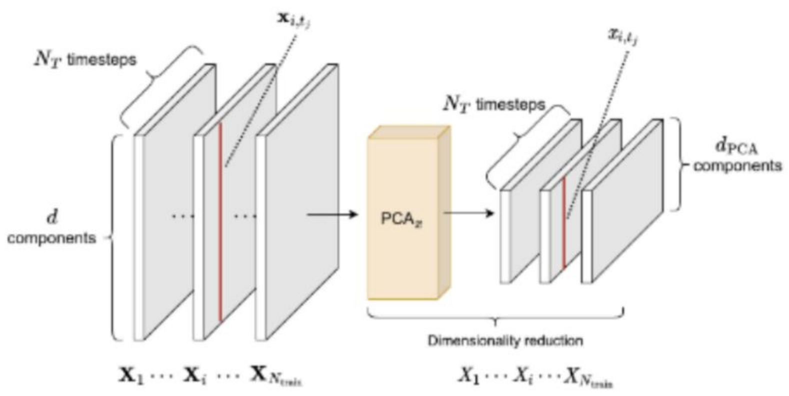


Figure 5: Generation of training data set

Since TFTs are rather resource-intensive networks, it is pivotal that the number of original space dimensions is reduced. In this regard, a dimensionality reduction algorithm is employed, particularly Principal Component Analysis (PCA), since it was proven to be sufficient and performant. We utilize two instances of the PCA algorithm: first, $PCA_u: R^d \rightarrow R^{d_{PCA}}$ is fitted to the previously generated dataset, as shown in Fig. 6, mapping the original-space solutions to their latent-space projections denoted as $\boldsymbol{y} = PCA_u(\boldsymbol{u})$. Similarly, a second PCA, namely $PCA_x: R^d \rightarrow R^{d_{PCA}}$, is fitted to the parameters $\mathbf{X}_{train}$ associated with solutions $\boldsymbol{U}_{train}$, where $\mathbf{X}_{train} = \{\mathbf{X}_\mathrm{i}\}_{i=1}^{N_{train}}$, with $\boldsymbol{X}_i = \left[\mathbf{x}_{i,t_0}, \dots, \mathbf{x}_{i,t_{N_T}}\right] \in R^{d \times N_T}$ are the external loading values in the problem. Then, $\boldsymbol{x} = PCA_\mathrm{x}(\mathbf{x})$ transforms the external loading vectors to their reduced space projections.



$\mathbf{u}_{i,t_j}$
N_T timesteps
d components
PCA$_u$
$\mathbf{y}_{i,t_j}$
N_T timesteps
d_{PCA} components
Dimensionality reduction
$\mathbf{U}_1 \cdots \mathbf{U}_i \cdots \mathbf{U}_{N_{\text{train}}}$
$\mathbf{Y}_1 \cdots \mathbf{Y}_i \cdots \mathbf{Y}_{N_{\text{train}}}$
(a) PCA$_u$



$\mathbf{x}_{i,t_j}$
N_T timesteps
d components
PCA$_x$
x_{i,t_j}
N_T timesteps
d_{PCA} components
Dimensionality reduction
$\mathbf{X}_1 \cdots \mathbf{X}_i \cdots \mathbf{X}_{N_{\text{train}}}$
$X_1 \cdots X_i \cdots X_{N_{\text{train}}}$

Figure 6: Dimensionality reduction with separate PCA fits for the displacement and excitation time-series $\boldsymbol{U_i}$ and $\mathbf{X}_i$, respectively.

Subsequently, a separate, scalar TFT^j model is trained on each latent-space component for $j = 1, \dots, d_{PCA}$ resulting in a total of d_{PCA} networks with identical architecture. At timestep t, each network TFT^j has access to a past k-length window $(y_{t-k:t}^{(j)}, x_{t-k:t}^{(j)})$, containing previous latent-space solutions and their corresponding parameters, and directly tries to predict only the next scalar term of the time series $y_{t+1}^{(j)}$, as shown in figure 7.

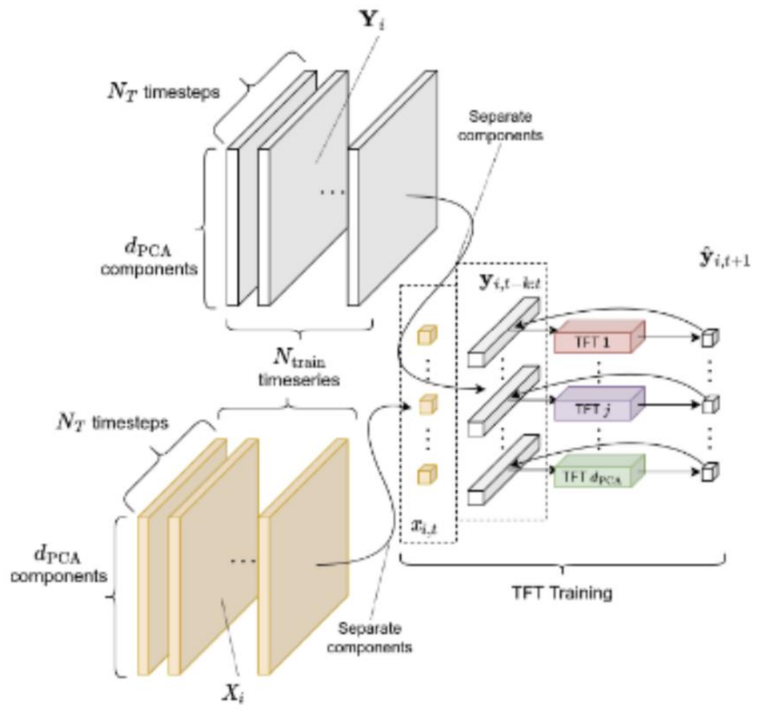


Figure 7: Proposed architecture during training.

Step 2: Prediction phase

As illustrated in figure 8, the proposed inference scheme consists of four phases. Initially, two separate low-dimensional projections are applied to fixed time windows containing instances of previous solutions and their associated parameters, respectively. Subsequently, scalar-output TFTs are employed to forecast future values in each reduced dimension. Following the predictions, the results are restored to the original solution space and iteratively refined until a predefined error threshold is met. Finally, each refined solution is reused for the subsequent prediction.

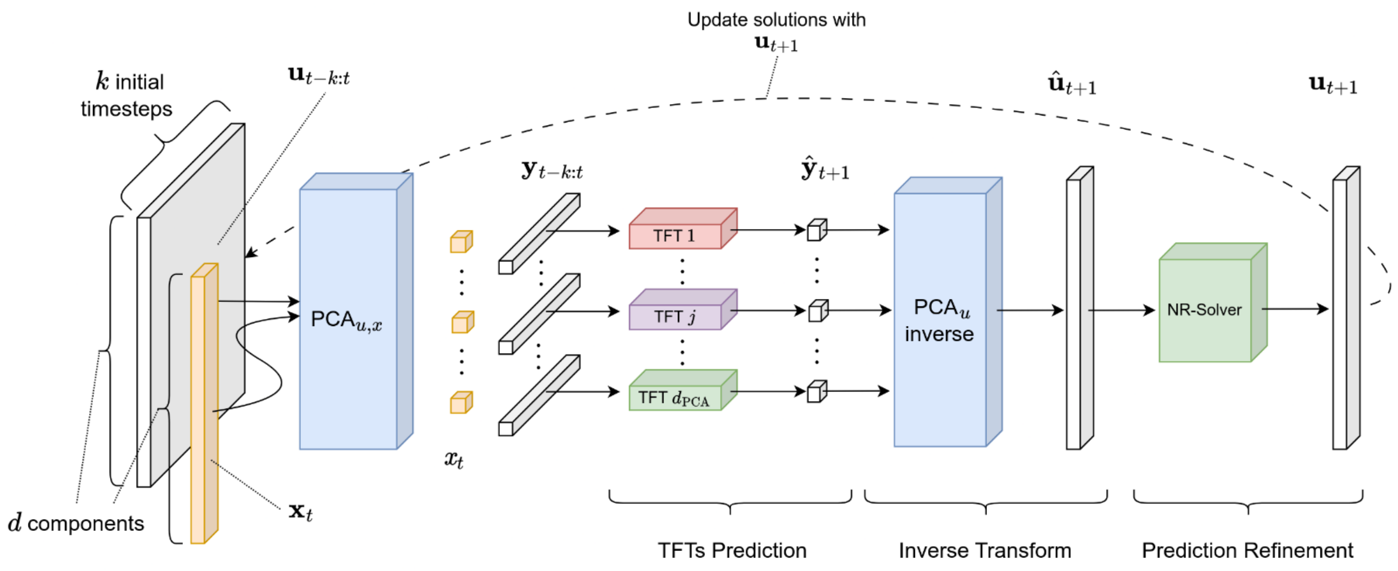


Figure 8: Proposed architecture for inference.

4. Contents of the AI-Solve software library

This section details the content and functionalities of the AI-Solve library that was developed within the project. All project related software implementations can be found in https://github.com/mgroupntua, and are publicly available.

4.1 Linear algebra repository

The linear algebra public repository (https://github.com/mgroupntua/LinearAlgebra) includes implementations and wrappers for linear operations in C# and in the context of the DCoMEX project the following algorithms have been developed:

- The Gauss-Seidel iterative algorithm: https://github.com/mgroupntua/LinearAlgebra/blob/develop/src/MGroup.LinearAlgebra/Iterative/GaussSeidel/GaussSeidelAlgorithm.cs
- The conjugate gradient method: https://github.com/mgroupntua/LinearAlgebra/blob/develop/src/MGroup.LinearAlgebra/Iterative/ConjugateGradient/CGAlgorithm.cs
- The preconditioned conjugate gradient method: https://github.com/mgroupntua/LinearAlgebra/blob/develop/src/MGroup.LinearAlgebra/Iterative/PreconditionedConjugateGradient/PcgAlgorithm.cs
- The block preconditioned conjugate gradient method: https://github.com/mgroupntua/LinearAlgebra/blob/develop/src/MGroup.LinearAlgebra/Iterative/PreconditionedConjugateGradient/BlockPcgAlgorithm.cs
- The generalized minimal residual method (GMRES): https://github.com/mgroupntua/LinearAlgebra/blob/develop/src/MGroup.LinearAlgebra/Iterative/GeneralizedMinimalResidual/GmresAlgorithm.cs
- The algebraic multigrid method: https://github.com/mgroupntua/LinearAlgebra/blob/develop/src/MGroup.LinearAlgebra/Iterative/AlgebraicMultiGrid/AlgebraicMultiGrid.cs
- The Proper Orthogonal Decomposition algorithm: https://github.com/mgroupntua/LinearAlgebra/blob/develop/src/MGroup.LinearAlgebra/Iterative/AlgebraicMultiGrid/PodAmg/ProperOrthogonalDecomposition.cs
- The POD-2G method described in section 3.1 to be used as a standalone iterative solver: https://github.com/mgroupntua/LinearAlgebra/blob/develop/src/MGroup.LinearAlgebra/Iterative/AlgebraicMultiGrid/PodAmg/PodAmgAlgorithm.cs
- The POD-2G method described in section 3.1 to be used as a preconditioner in the context of CG: https://github.com/mgroupntua/LinearAlgebra/blob/develop/src/MGroup.LinearAlgebra/Iterative/AlgebraicMultiGrid/PodAmg/PodAmgPreconditioner.cs

The verify the correct implementation of these algorithms several numerical tests are included in this repository that can be found in https://github.com/mgroupntua/LinearAlgebra/tree/develop/tests/MGroup.LinearAlgebra.Tests

4.2 Solvers repository

The Solvers public repository (https://github.com/mgroupntua/Solvers) utilizes the linear algebra algorithms to solve linear systems of equations arising from engineering problems. The Solvers repository acts as a mediator between the LinearAlgebra repo and MSolve modules, making the proper connections and associations between the discretization processes of the PDEs to be solved and the actual linear algebra objects that are used for solving the emerging linear systems. Moreover, the Solvers repo has all necessary information, in order to compose problem-specific preconditioners and solution strategies.

- PCG solver: https://github.com/mgroupntua/Solvers/blob/develop/src/MGroup.Solvers/Iterative/PcgSolver.cs
- Block PCG solver: https://github.com/mgroupntua/Solvers/blob/develop/src/MGroup.Solvers/Iterative/BlockPcgSolver.cs
- GMRES solver: https://github.com/mgroupntua/Solvers/blob/develop/src/MGroup.Solvers/Iterative/GmresSolver.cs

- POD-2G solver: https://github.com/mgroupntua/Solvers/blob/develop/src/MGroup.Solvers.MachineLearning/PodAmg/AmgAISolver.cs

The family of DDM algorithms is currently at a seperate repository (https://github.com/SerafeimBakalakos/MSolveOne/tree/paper/xfem_pfetidp_mpi) and will be integrated in the main code in the following months. All DDM algorithms described in sections 2.5.5-2.5.7 can be found in:

- P-DDM (or PSM): https://github.com/SerafeimBakalakos/MSolveOne/blob/paper/xfem_pfetidp_mpi/src/MGroup.Solvers.DDM/PSM/PsmSolver.cs
- Feti-DP : https://github.com/SerafeimBakalakos/MSolveOne/blob/paper/xfem_pfetidp_mpi/src/MGroup.Solvers.DDM/FetiDP/FetiDPSolver.cs
- P-Feti-DP: https://github.com/SerafeimBakalakos/MSolveOne/blob/paper/xfem_pfetidp_mpi/src/MGroup.Solvers.DDM/PFetiDP/PFetiDPSolver.cs

4.3 Machine learning repository

The MachineLearning public repository (https://github.com/mgroupntua/MachineLearning) contains the different types of neural networks that were implemented and used for the project's purposes. These include:

- Feedforward Neural Networks: https://github.com/mgroupntua/MachineLearning/blob/develop/src/MGroup.MachineLearning.TensorFlow/NeuralNetworks/FeedForwardNeuralNetwork.cs
- Convolutional Neural Networks: https://github.com/mgroupntua/MachineLearning/blob/develop/src/MGroup.MachineLearning.TensorFlow/NeuralNetworks/ConvolutionalNeuralNetwork.cs
- Autoencoders: https://github.com/mgroupntua/MachineLearning/blob/develop/src/MGroup.MachineLearning.TensorFlow/NeuralNetworks/Autoencoder.cs
- Convolutional Autoencoders: https://github.com/mgroupntua/MachineLearning/blob/develop/src/MGroup.MachineLearning.TensorFlow/NeuralNetworks/ConvolutionalAutoencoder.cs

Based on these building blocks, the customized surrogate modeling technique mentioned in D2.3 that combines FFNN and CAEs to provide predictions for complex systems (step 1 of the data-driven solution framework in Section 3.1) can be found in https://github.com/mgroupntua/MachineLearning/blob/develop/src/MGroup.Constitutive.Structural.MachineLearning/Surrogates/CaeFffnSurrogate.cs

In addition, dedicated surrogate modeling techniques were developed to accelerate the DCoMEX-Mat applications in WP 7. These include neural networks that emulate the constitutive behavior of composite materials and can be found in: https://github.com/mgroupntua/MachineLearning/tree/develop/src/MGroup.Constitutive.Structural.MachineLearning/Continuum

4.4 AISolve.core repository

AISOLVE.core (https://github.com/mgroupntua/AISolve.Core/tree/develop/src) is a collection of interfaces and classes for the workflow definition of AISolve. AISolve has been designed in a fashion allowing for the interconnection of heterogeneous software modules that will tackle the solution of the various phases of the AISolve workflow.

4.5 AISolve.MSolve repository

AISolve.MSolve repository (https://github.com/mgroupntua/AISolve.MSolve) is the MSolve implementation of AISolve.Core interfaces and workflow. Specifically, this repository implements all the interfaces defined in the

AISolve.Core repository and connects it with the appropriate MSolve modules for the solution of engineering problems with the aid of AISolve.

5. Numerical Applications

5.1 Accelerating crack propagation problems using DDM

The extended finite element method (XFEM) is one of the most popular methods to simulate fracture phenomena. However, this approach requires the solution of large-scale systems that are often ill-conditioned and thus require specialized iterative solving techniques. In [3] several variants of Domain Decomposition Methods, which were implemented in the context of DCoMEX and are part of the AI-Solve library, were examined and their computational merits were showcased.

The problem under investigation involves a crack propagating in a beam supported at three points and loaded at a fourth point, as illustrated in figure 9.

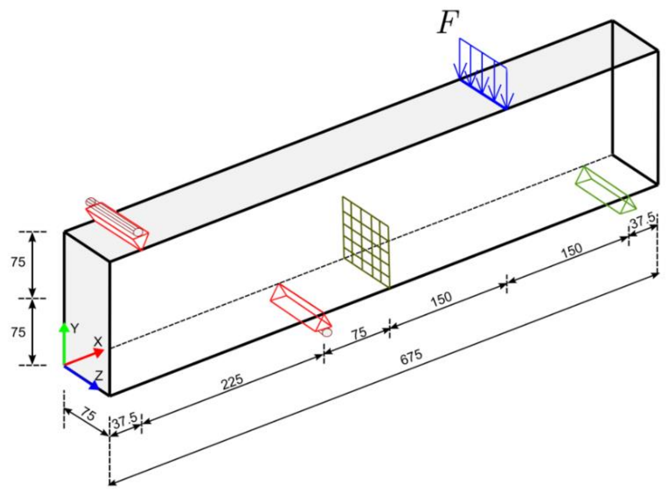


Figure 9: 4-point bending beam test case. Geometry, boundary conditions and initial configuration of the crack surface (dimensions in mm)

The material properties are $E = 3 \cdot 10^7 \frac{N}{mm^2}, v = 0.3$ and the applied load is $F = 1000\ N$. The dimensions of the beam, the placement of supports and load and the initial configuration of the crack surface are given in the figure. It is assumed that the crack propagates in a quasi-static manner with a constant increment, until it reaches the boundary of the domain and collapse occurs after 13 propagation steps. The whole analysis is repeated for various mesh densities, with each mesh consisting of 8-node hexahedral elements. The initial and final number of dof for each mesh are listed in the table below:

Mesh	dof at first step	dof at last step
45 x 10 x 5	9,492	9,816
90 x 20 x 10	64,130	65,384
135 x 30 x 15	204,336	206,880
180 x 40 x 20	470,820	475,419
225 x 50 x 25	903,812	910,832
270 x 60 x 30	1,537,228	1,548,295
315 x 70 x 35	2,431,872	2,444,940

Table XX: Number of dof per mesh. The difference in dof between the first and last step of the crack propagation analysis is due to the mesh enrichment required by XFEM to capture the crack's advancing front

To test the performance of the DDM methods, the following cases were considered: Dirichlet preconditioned FETI-DP (abbreviated as FETI-DP-D), lumped-preconditioned FET-DP (abbreviated as FETI-DP-L), and the P-FETI-DP. The improved versions of the these methods using re-initializations techniques for crack propagation problems are denoted as FETI-DP-D-I, FETI-DP-L-I and P-FETI-DP-I. The speedup each of these methods achieved is compared to supernodal sparse Cholesky factorization direct solver [4]. The results of these analyses are presented in figure 10.

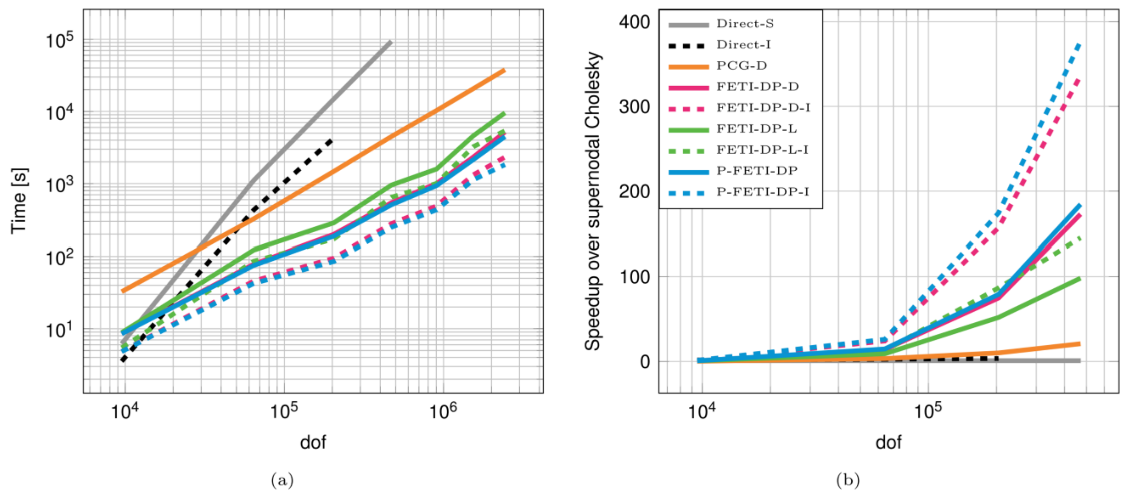


Figure 10: Performance comparison of the DDM solvers for the 4-point bending beam problem: (a) Computing time (in seconds), (b) speedup of the solvers relative to the supernodal Cholesky solver.

As evidenced by the results in figure 10, all DDM methods exhibit superior computational performance in comparison with the direct solver and the PCG method. Also, the P-FETI-DP and its improved version P-FETI-DP-I were the most efficient solution techniques from the family of DDM methods.

5.2 Solution to the parameterized Biot problem using POD-2G

The data-driven solution framework that described in section 3.1 and developed in the context of the DCoMEX problem was employed to solve a parameterized version of the Biot problem (deformable porous medium) based on the $u - p$ formulation [2]. The problem under investigation involves a 3D solid cube under prescribed displacement and pressure boundary conditions, as shown in figure 11.

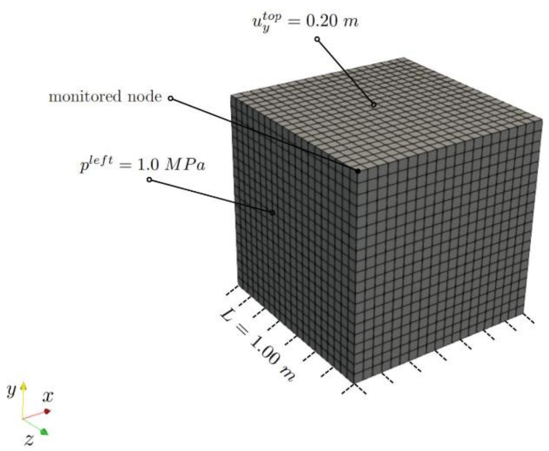
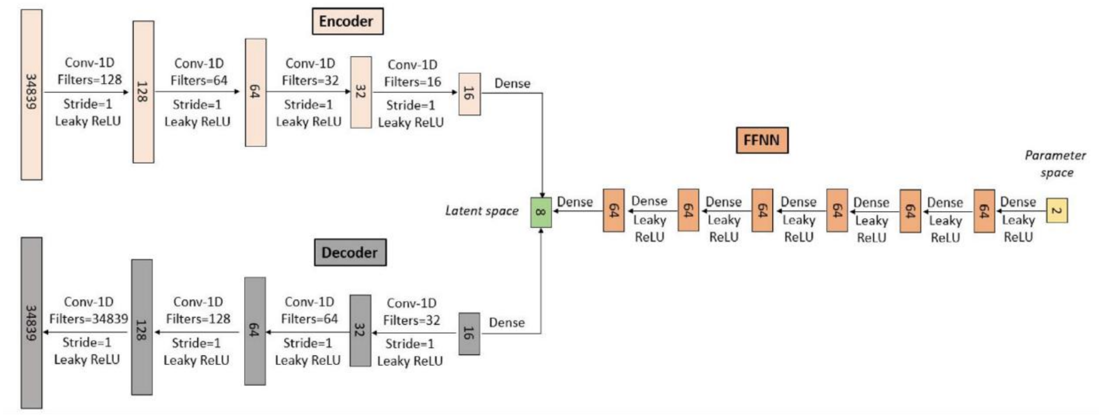


Figure 11: Geometry, boundary conditions and FE discretization of the Biot problem

We assumed for this problem that the Lame coefficients μ and λ are random variables following the distributions given in table 1. As a first step, the Latin Hybercube sampling method was utilized to generate $N_{train} = 300$ parameter samples $\{\mu_i, \lambda_i\}_{i=1}^{N_{train}}$. The surrogate's architecture is presented in Figure 12. The CAE is trained for 100 epochs with a batch size of 10 and a learning rate of 10-3, while the FFNN is trained for 5000 epochs with a batch size of 20 and a learning rate of 10-4. The average normalized *l*2 norm error of the surrogate model in the test data set is 0.68%.

Parameter	Distribution	Mean	Standard deviation
$\mu\ (MPa)$	Lognormal	0.30	0.09
$\lambda\ (MPa)$	Lognormal	1.70	0.51

Table 1: Random parameters of the Biot problem



Encoder
34839
Conv-1D Filters=128 Stride=1 Leaky ReLU
128
Conv-1D Filters=64 Stride=1 Leaky ReLU
64
Conv-1D Filters=32 Stride=1 Leaky ReLU
32
Conv-1D Filters=16 Stride=1 Leaky ReLU
16
Dense
FFNN
Parameter space
Latent space
8
Dense
64
Dense Leaky ReLU
2
Decoder
34839
Conv-1D Filters=34839 Stride=1 Leaky ReLU
Conv-1D Filters=128 Stride=1 Leaky ReLU
Conv-1D Filters=64 Stride=1 Leaky ReLU
Conv-1D Filters=32 Stride=1 Leaky ReLU

Figure 12: Surrogate model architecture

Subsequently, a number of N_{test} parameter vectors $\{\{\mu_i,\lambda_i\}_{i=1}^{N_{test}}$were generated according to their distribution and the corresponding problems were solved with the proposed POD-based solver and different Ruge-Stüben AMG solvers, with the number of grids ranging from 2 to 6. The size of the system of equations at the coarsest level for each of these solvers is presented in Table 2. For this example, eight eigenvectors were retained in the POD expansion, as these were sufficient for capturing 99.99% of the dataset's variance.

	System size
Initial problem	34839×34839
AMG-2G	8625×8625
AMG-3G	1421×1421
AMG-4G	229×229
AMG-5G	47×47
AMG-6G	9×9
POD-2G	8×8

Table 2: Size of the problem at the coarsest grid for the different solvers

The mean value of the CPU time and the number of cycles required for convergence to the desired value of tolerance are displayed in Figure 13 and Table 3. The results are very promising in terms of computational cost. For instance, for $\varepsilon = 10^{-5}$ and $\boldsymbol{u}^{(0)} = \mathbf{0}$, a reduction of computational cost of ×7.32 is achieved when comparing the proposed solver with the 3-grid AMG solver. Furthermore, obtaining an accurate initial solution $\boldsymbol{u}^{(0)}$ is again a very important component of the proposed framework. Specifically, by considering $\boldsymbol{u}^{(0)} = \boldsymbol{u}_{sur}$ instead of $\boldsymbol{u}^{(0)} = \mathbf{0}$ for $\varepsilon = 10^{-5}$, an additional decrease in CPU time of ×4.31 can be achieved.

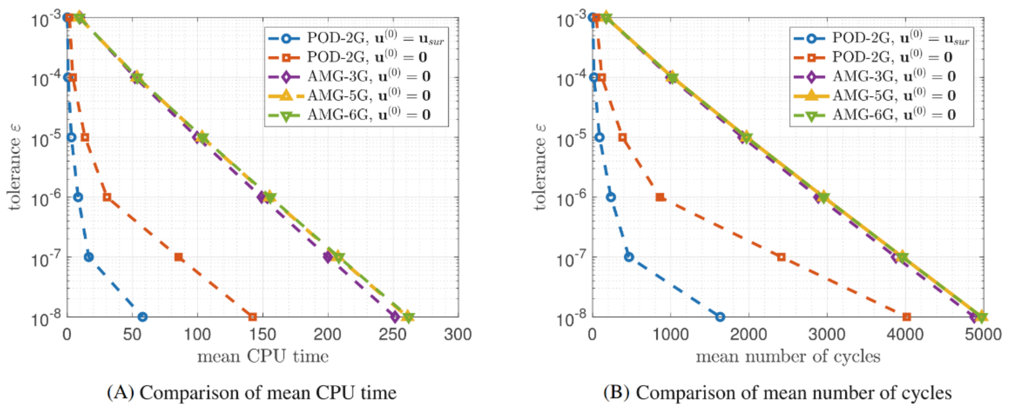
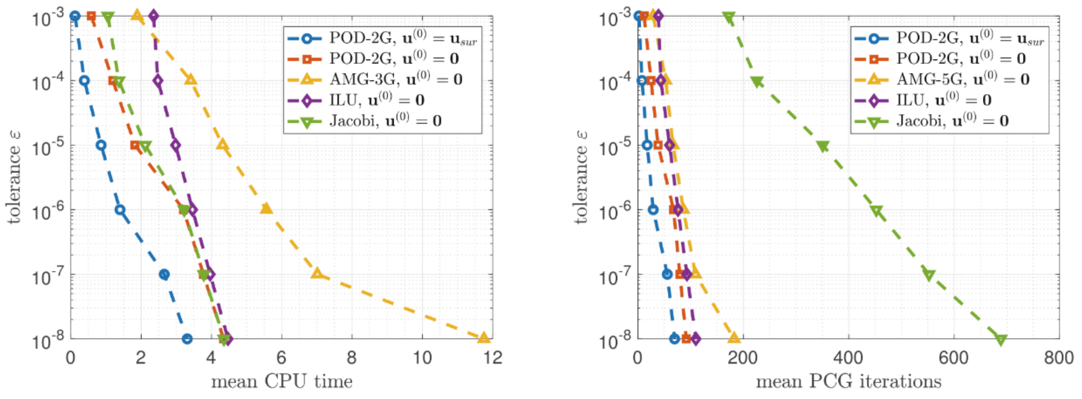


Figure 13: Comparison of mean CPU time and mean number of cycles over 500 analyses for different multigrid solvers

	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-5}$	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-8}$
AMG-3G ($\boldsymbol{u}^{(0)} = \boldsymbol{0}$)	×1.00	×1.00	×1.00	×1.00	×1.00
AMG-5G ($\boldsymbol{u}^{(0)} = \boldsymbol{0}$)	×0.97	×0.96	×0.96	×0.96	×0.96
AMG-6G ($\boldsymbol{u}^{(0)} = \boldsymbol{0}$)	×0.97	×0.96	×0.96	×0.96	×0.96
POD-2G ($\boldsymbol{u}^{(0)} = \boldsymbol{0}$)	×12.31	×7.32	×4.89	×2.34	×1.77
POD-2G ($\boldsymbol{u}^{(0)} = \boldsymbol{u}_{sur}$)	×76.89	×31.54	×17.90	×12.12	×4.35

Table 3: Computational speedup of solvers compared to AMG-3G

The convergence behavior of the proposed method when used as a preconditioner in the context of the PCG method is presented in Figure 14. Again, the results delivered by the proposed methodology showed its superior performance not only over AMG preconditioners but also over ILU and Jacobi preconditioners. In this case, for $\varepsilon = 10^{-5}$ and $\boldsymbol{u}^{(0)} = \boldsymbol{0}$, a reduction of computational cost of ×2.37 is observed between the proposed method and the 3-grid AMG, of ×1.63 with the ILU and of ×1.16 with the Jacobi. Last but not least, the initial solution delivered by the surrogate model, $\boldsymbol{u}^{(0)} = \boldsymbol{u}_{sur}$, managed to further reduce the computational time by ×2.12 when compared to POD-2G with $\boldsymbol{u}^{(0)} = \boldsymbol{0}$ (See Table 4).



(A) Comparison of mean CPU time (B) Comparison of mean PCG iterations

Figure 14: Comparison of mean CPU time and mean number of PCG iterations over 500 analyses for different preconditioners

	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-5}$	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-8}$
AMG-3G ($\boldsymbol{u}^{(0)} = \boldsymbol{0}$)	×1.00	×1.00	×1.00	×1.00	×1.00
ILU ($\boldsymbol{u}^{(0)} = \boldsymbol{0}$)	×1.38	×1.45	×1.61	×1.77	×2.63
Jacobi ($\boldsymbol{u}^{(0)} = \boldsymbol{0}$)	×2.50	×2.04	×1.73	×1.85	×2.70
POD-2G ($\boldsymbol{u}^{(0)} = \boldsymbol{0}$)	×2.86	×2.37	×1.74	×1.86	×2.71
POD-2G ($\boldsymbol{u}^{(0)} = \boldsymbol{u}_{sur}$)	×8.88	×5.02	×3.98	×2.64	×3.55

Table 4: Computational speedup of different preconditioners compared to the AMG-3G preconditioner

5.3 Accelerating the nonlinear analysis of water tower under random base excitation

The data-driven solution framework developed in Section 3.2 of the deliverable was employed in [5] to reduce the cost for nonlinear transient analysis of a water tower under random base excitation. In this example, the water tower shown in figure 15 is initially subjected to a series of monochromatic seismic ground accelerations of the form $\boldsymbol{p}_{ext} = -\boldsymbol{M} \cdot \boldsymbol{1}_d \cdot A \cdot sin(\theta \cdot t)$, where $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{1}_d$ is the excitation's directivity vector, A is an amplitude modifier and θ is the angular frequency of the excitation, which is considered a system parameter ranging from 0 to 0.5 . The structure's model consists of 4104 tetrahedral elements and 1767 nodes corresponding to d = 5241 degrees of freedom.

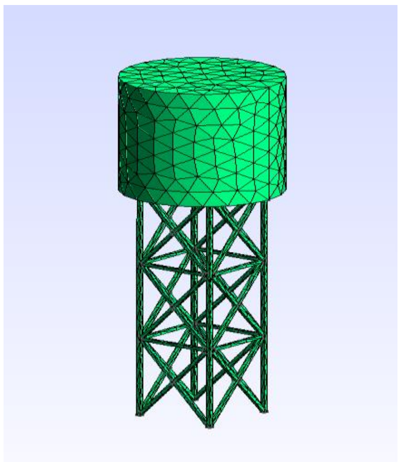


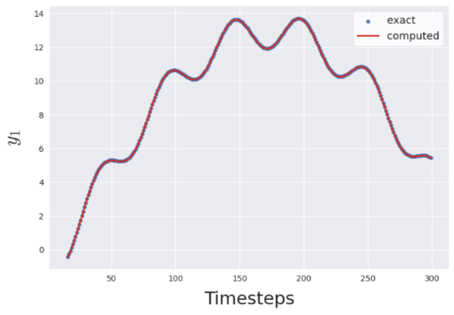
Figure 15: Water tower geometry and finite element discretization

Initially, the response of the structure is calculated for N_{train} separate angular frequency values considering zero initial conditions for each performed analysis. The second step of the proposed framework is to start by conducting a PCA analysis and evaluate how the captured variance changes as the number of principal components increases. After careful observation, we opted to choose the first $d_{PCA} = 9$ components as they effectively retained 99.98% of the original variance for both the solution and the related force vector space.

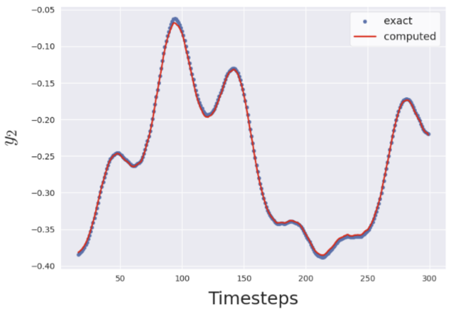
Following that, we proceeded to train nine distinct scalar-valued TFTs for a total of 2000 epochs. Each TFT corresponded to one of the latent space dimensions that were derived. The models were trained to generate one-step-ahead forecasts, where the value k of the length of the look-back window was set to 16. Their performance on the training dataset is summarized in Table 5. Figure 16 illustrates the performance of the TFTs' prediction on the temporal evolution of the 9 PCA components over time for the sample θ in the training data set with the median test percentage error.

Performance	% training error
Overall % loss	0.2813
Best sample	0.1804
Worst sample	0.7579

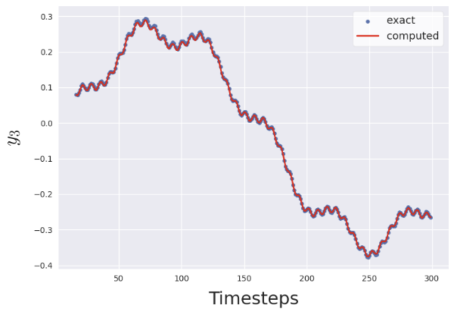
Table 5: Percentage training error



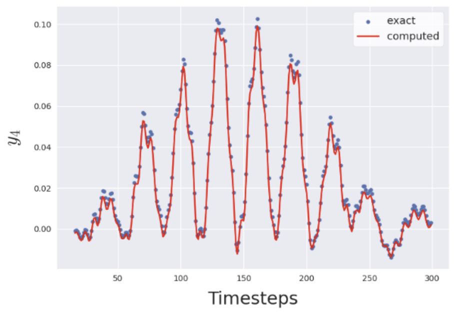
(a) TFT_1



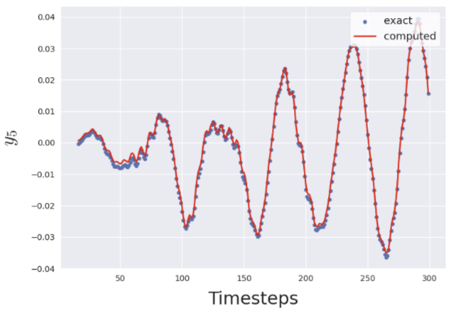
(b) TFT_2



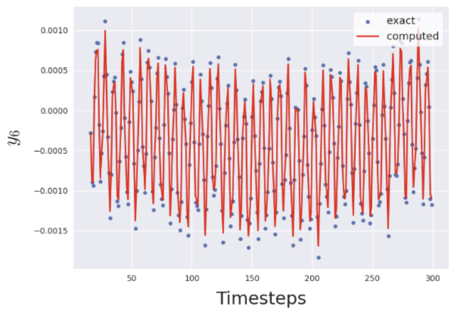
(c) TFT_3



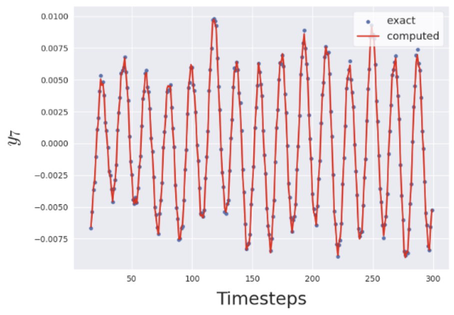
(d) TFT_4



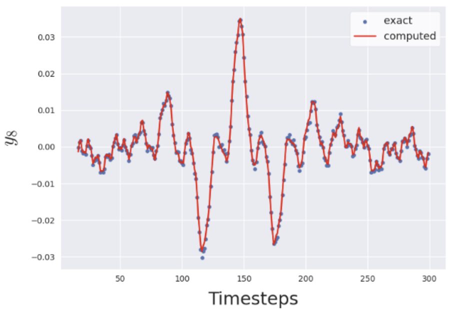
(e) TFT_5



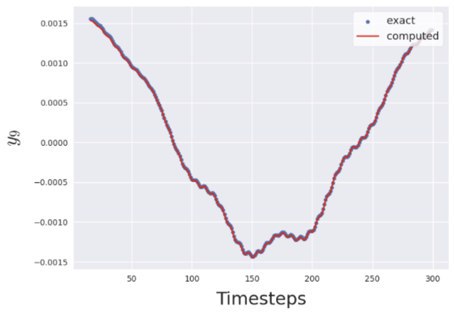
(f) TFT_6



(g) TFT_7



(h) TFT_8



(i) TFT_9

Figure 16: Performance of the TFTs' prediction on the temporal evolution of the 9 PCA components over time for the sample θ in the training data set with the median test percentage error.

The next step was to generate artificial ground motions that resemble actual earthquakes using the Spectral Representation method [6]. Then, to test the performance of the previously trained surrogate model on this more challenging scenario, we conducted 1200 simulations of the FE model for different ground motions using the surrogate's predictions as input to the FE nonlinear solver. We observed a noticeable reduction in the average values of NR iterations needed, as shown in table 6, even though the surrogate wasn't trained on this type of excitations.

Iterations without surrogate	Iteration with surrogate
1	1.000
2	1.121
3	1.374
4	1.513
5	2.419
6	3.471
7	3.857
8	2.2
9	3.5

Table 6: Extrapolation Performance on the earthquake time series: The right column illustrates the average iteration count for our solver to converge, utilizing surrogate's predictions as initial solutions. These averages are calculated across time series where the number of iterations needed prior to employing our approach corresponds to the values in the left column, highlighting our approach's effectiveness in handling multiple-frequency time series.

In addition, figure 17 shows the average number of iterations per time step for 1200 simulations with and without the use of the surrogate. In this test case, the average number of iterations per simulation without the use of the surrogate was 1246.88 and the total number of iterations was 1496256 for the 1200 simulations. However, using the surrogate's predictions as initial solutions, we managed to drop these numbers to 576,13 and 691356, respectively, as shown in figure 18. These results provide a strong indication that the TFT models are able to learn the system's dynamics in the latent space and can extrapolate beyond the training data set.

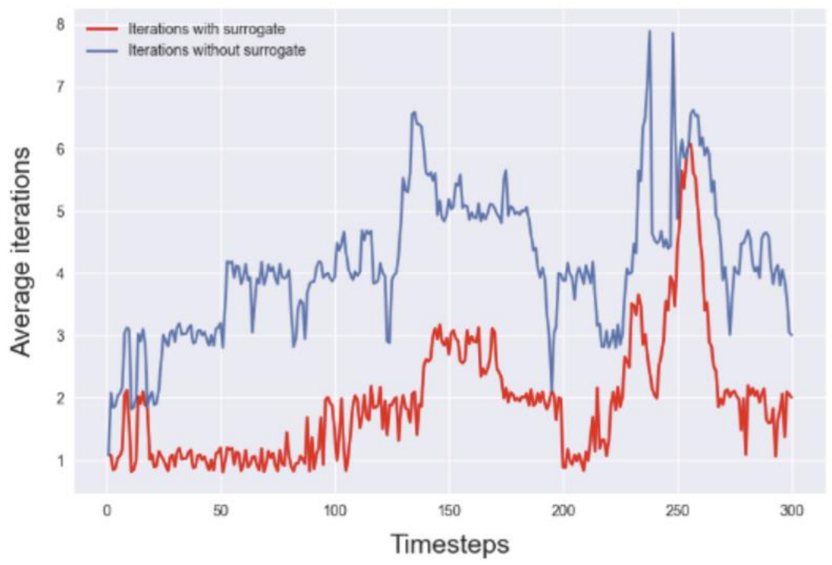


Figure 17: Per timestep comparison on the earthquake timeseries

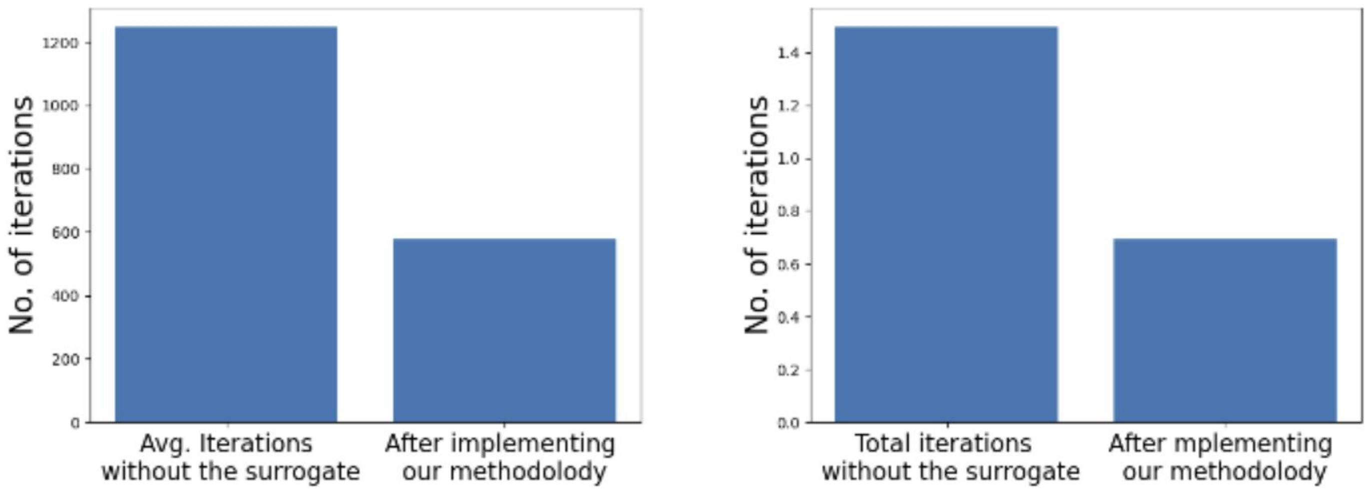


Figure 18: Extrapolation Results on the earthquake time series: This figure compares convergence metrics before and after applying our methodology, showcasing the contrast in average iterations for individual time series on the left and the total iterations for the entire dataset on the right

6. Summary and future work

In this report, the software implementations regarding the AI-Solve library were presented, in accordance with WP3 of the proposal. These included (i) a set of state-of-the-art algorithms for solving large scale linear systems, along with parallelizable versions (e.g. block PCG, DDM), (ii) two novel algorithms that combine the aforementioned solvers with machine learning to tackle parameterized problems and (iii) a set of numerical applications that demonstrate the capabilities of the AI-Solve library for handling challenging problems from the field of computational mechanics.

Further improvement of the software and extension of its capabilities is an ongoing process that will last until the project end and after. These, among others include integration of a family of domain decomposition methods, further optimization of its performance with (i) scalable sparse computations and (ii) communication optimization techniques and utilization of the AI-Solve software to accelerate the applications described in WP7 (DCoMEX-BIO and DCoMEX-MAT)

References

1. Y. Fragakis and M. Papadrakakis, "*The mosaic of high performance domain decomposition methods for structural mechanics: Formulation, interrelation and numerical efficiency of primal and dual methods*," Computer Methods in Applied Mechanics and Engineering, 2003.

2. S. Nikolopoulos, I.Kalogeris, G. Stavroulakis, V. Papadopoulos, "*AI-Enhanced iterative solvers for accelerating the solution of large-scale parametrized systems*", International Journal of Numerical Methods in Engineering, 2023

3. S. Bakalakos, M. Georgioudakis, M. Papadrakakis, "*Domain decomposition methods for 3D crack propagation problems using XFEM",* Computer Methods in Applied Mechanics and Engineering, 2022.

4. Y. Chen, T.A. Davis, W.W. Hager, S. Rajamanickam, "*Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate"*, ACM Trans. Math. Software, 2008.

5. L. Papadopoulos, K. Atzarakis, G. Sotiropoulos, I. Kalogeris, V. Papadopoulos, "Fusing nonlinear solvers with transformers for accelerating the solution of parametric transient problems", Computer Methods in Applied Mechanics and Engineering, 2024 (under review)

6. M. Shinozuka, G. Deodatis, "Simulation of Stochastic Process by Spectral Representation", Applied Mechanics Reviews, 1991