1 Background

Here, we are attempting to train an ML agent to output both aggregates and interpolation for an AMG solver given the matrix $\boldsymbol{A}$ for a discretized PDE. We will be primarily be looking at the finite element discretization of the isotropic and anisotropic diffusion problems, i.e. PDEs of the form

$$-\nabla \cdot (\boldsymbol{D} \nabla u) = 0, \tag{1}$$

$$\boldsymbol{D} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & \\ & \varepsilon \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^T, \tag{2}$$

for some arbitrary θ and ε. In the isotropic case, $\theta = 0$ and $\varepsilon = 1$.

In both the isotropic and anisotropic cases, unstructured datasets of the following are generated:

1. A training set of 250 unstructured grids
2. A testing set of 62 unstructured grids

Each of these grid meshes are fairly small, overall containing between around 15 and 400 points. To create a mesh, random points are generated in $[0, 1]^2$, a convex hull constructed, then meshed with gmsh. The training and testing sets are intentionally limited to a small number of grids to reduce training complexity.

2 Generating Aggregates and Interpolation

To compute the aggregates and final interpolation operator, we concurrently train three graph networks to give different outputs:

1. Θ_{agg}, outputting aggregate centers
2. Θ_{soc}, outputting a strength-of-connection matrix for Bellman-Ford
3. Θ_P, outputting a smoothing operator

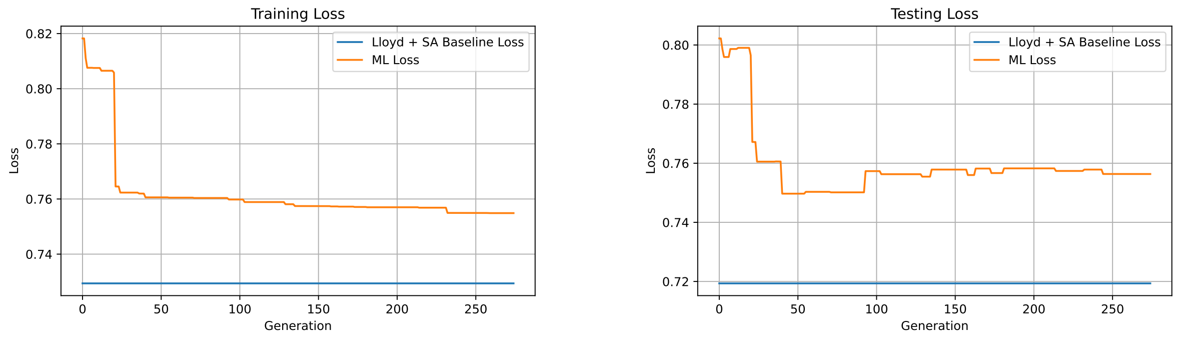
The algorithm for computing the interpolation operator is roughly sketched below:

1. Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be the system we are trying to solve and $\alpha \in (0, 1]$ be some parameter that determines the ratio of aggregates to vertices, i.e. we will be roughly coarsening the graph by $1/\alpha$. Define $k := \lceil \alpha n \rceil$, the number of aggregates we will be outputting.
2. Convolve the graph of $\boldsymbol{A}$ with Θ_{soc} to obtain $\boldsymbol{C}$, a matrix of weights that will be used for Bellman-Ford.
3. Convolve the graph of $\boldsymbol{A}$ concatenated with $\boldsymbol{C}$ using Θ_{agg} to obtain a new set of node values, restarting multiple times and inserting the tenative cluster centers as a node feature. We will use the output node values as a *scoring*. Define the aggregate centers as the indices of the largest k node scores.
4. Run Bellman-Ford on the graph with these aggregate centers to obtain a tentative aggregation operator, Agg.
5. Now, again convolve the graph of $\boldsymbol{A}$ but with Θ_{Interp} (with aggregate information) to obtain the aggregate smoother $\hat{\boldsymbol{P}}$. Form $\boldsymbol{P} := \hat{\boldsymbol{P}}\text{Agg}$.

3 Genetic Training

Training of both networks is done at the same time with a custom-written genetic algorithm that takes advantage of the easily parallelizable fitness calculation. A genetic algorithm is used because it does not require any gradient information, which benefits us because the algorithm used to output the aggregates is not easily differentiable.

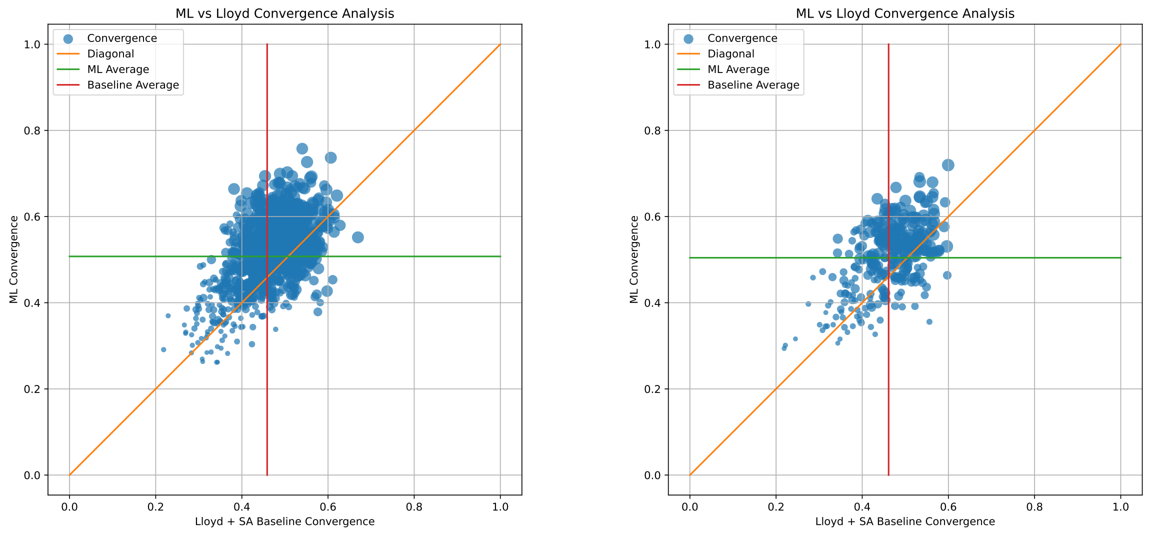
A basic overview of the training algorithm is that the method is seeded with some number of randomly generated networks. A subset of the best performing (most fit) networks are selected and *bred* with one another (crossing weights/traits) and *mutations* inserted (random perturbations to weights) to create another population of networks. This is then repeated for many *generations* until a set of hopefully trained networks is obtained, from which we can pick the best fit as our final network.



(a) Training loss

(b) Testing loss

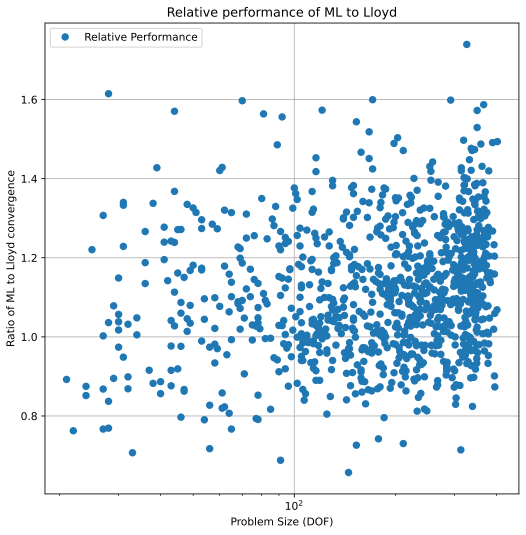
Figure 1: Loss plots per generation for training and testing datasets, respectively.



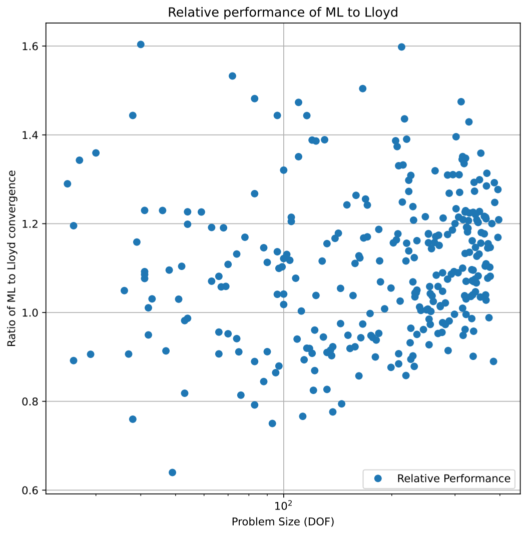
(a) Training convergence

(b) Testing convergence

Figure 2: Convergence data for the ML AMG method vs a baseline Lloyd and Jacobi SA method. Values below the diagonal indicate a better convergence for the ML. Markers are scaled by problem size.

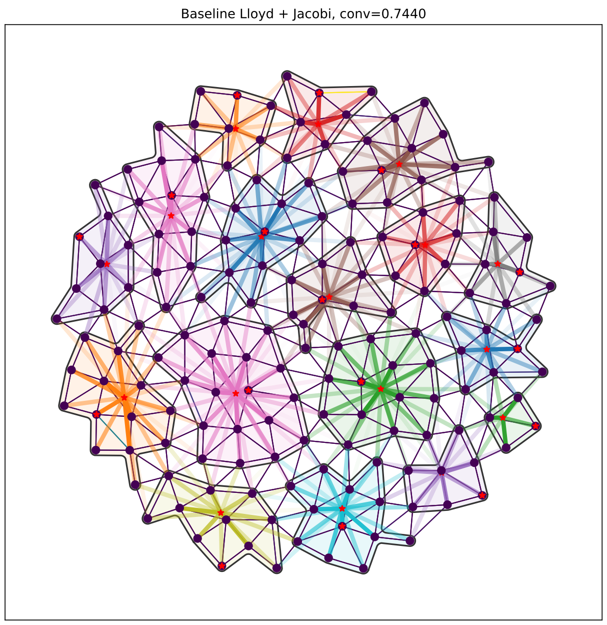


(a) Relative training performance

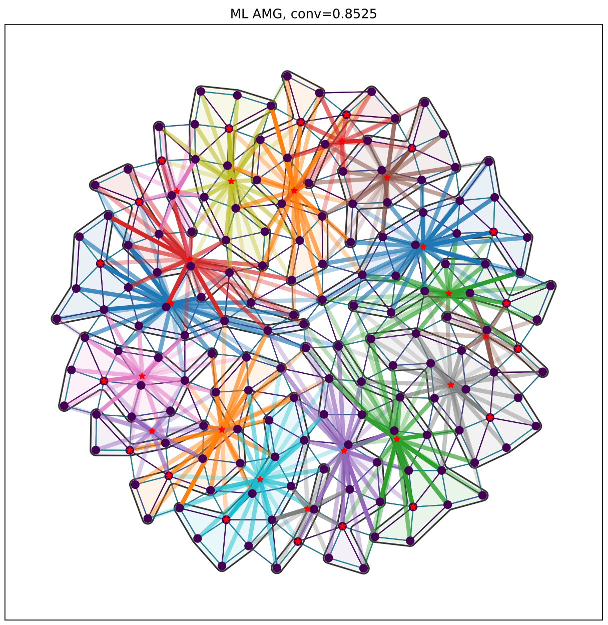


(b) Relative testing performance

Figure 3: Relative performance of the ML to the Lloyd method, plotted against problem size. Relative performance is obtained by dividing the ML convergence by the Lloyd convergence for each problem. Values below 1 indicate better ML performance, while values above 1 indicate better baseline performance.

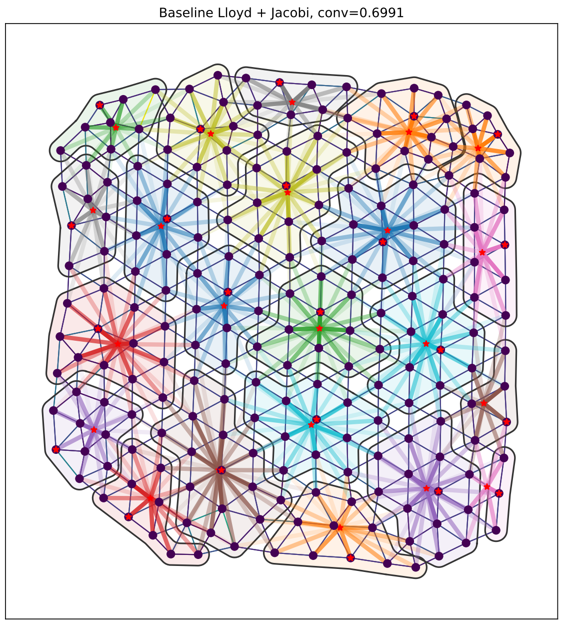


(a) Lloyd + Jacobi

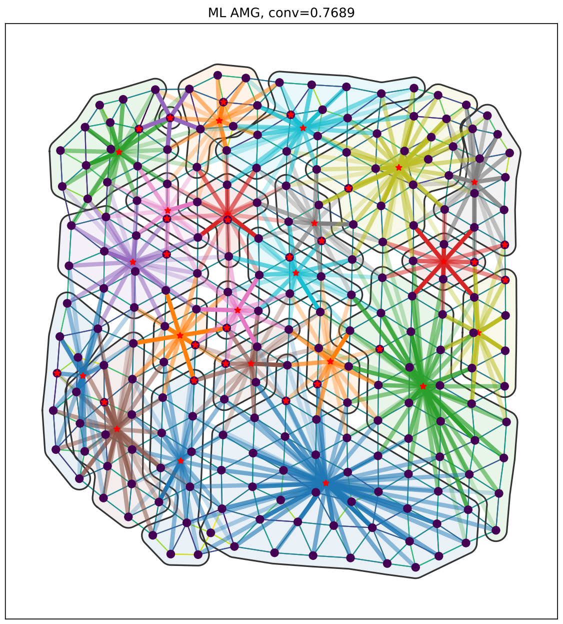


(b) ML

Figure 4: Aggregate and interpolation data for a circular unstructured mesh. This particular mesh has 173 DoF.

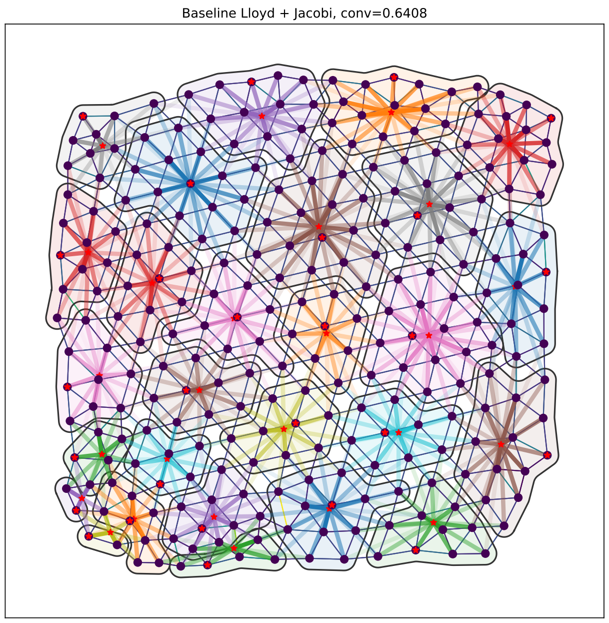


(a) Lloyd + Jacobi

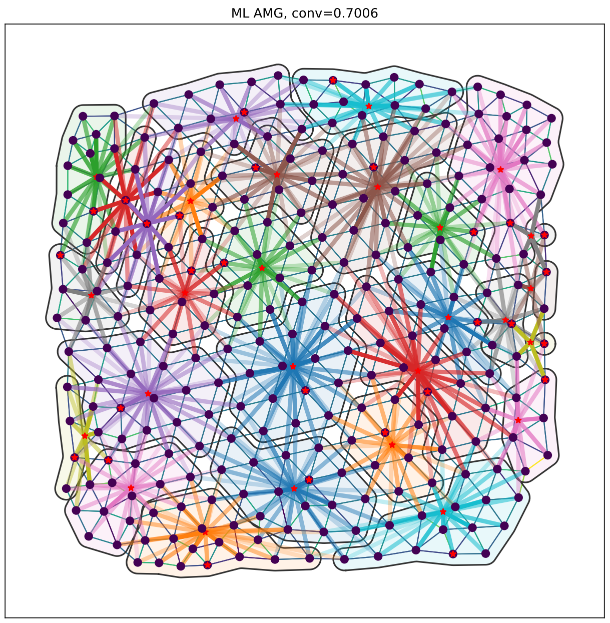


(b) ML

Figure 5: Aggregate and interpolation data for a random unstructured mesh. This particular mesh has 220 DoF.

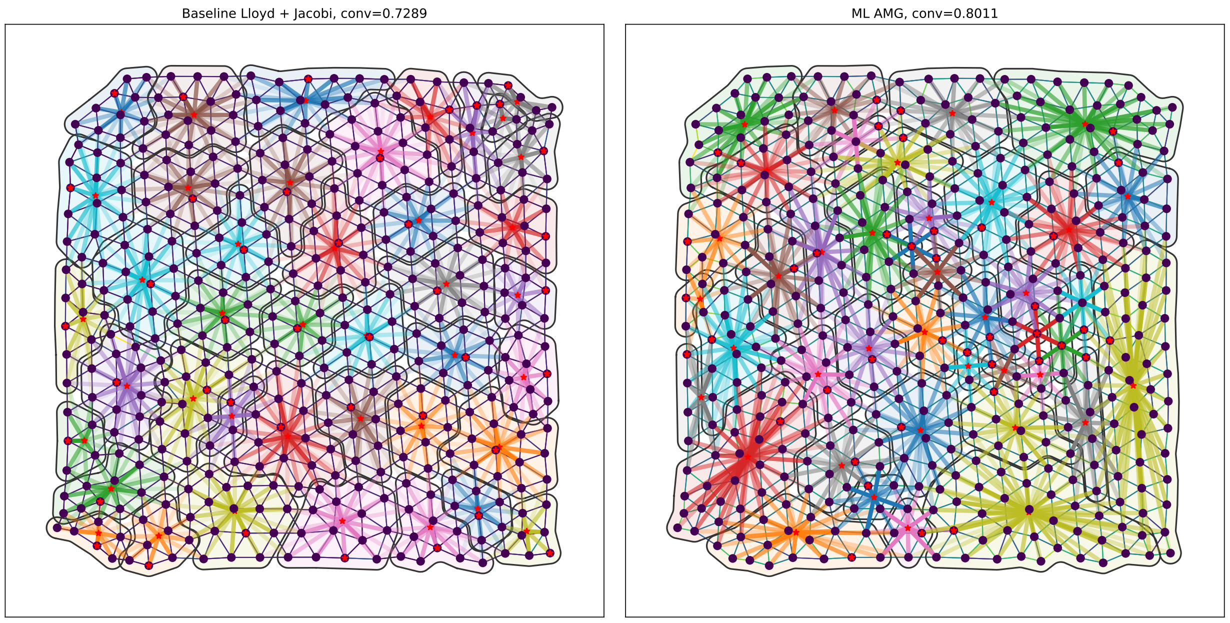


(a) Lloyd + Jacobi



(b) ML

Figure 6: Aggregate and interpolation data for a random unstructured mesh. This particular mesh has 273 DoF.



(a) Lloyd + Jacobi

(b) ML

Figure 7: Aggregate and interpolation data for the largest mesh in the *training set*. This particular mesh has 410 DoF.

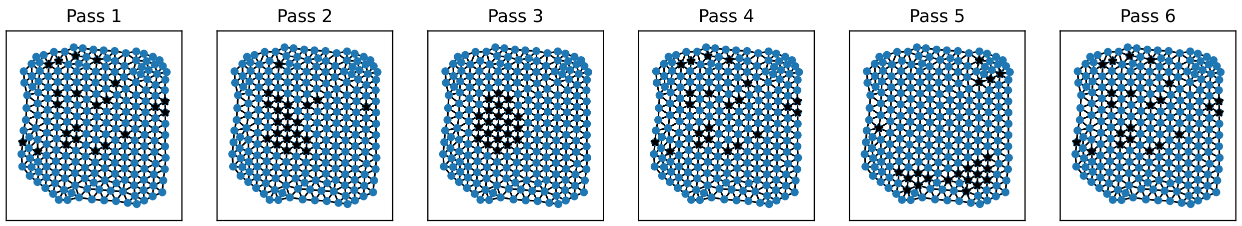


Figure 8: The tentative set of aggregate centers after each convolution "pass". This is the same mesh as in Fig 5.