MOX-Report No. 23/2025

MAGNET: an open-source library for mesh agglomeration by Graph Neural Networks

Antonietti, P. F.; Caldana, M.; Mazzieri, I.; Re Fraschini, A.

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

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

MAGNET: an open-source library for mesh agglomeration by Graph Neural Networks

Paola F. Antonietti[a], Matteo Caldana[a,*], Ilario Mazzieri[a], Andrea Re Fraschini[a]

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

April 30, 2025

Abstract

We introduce `MAGNET`, an open-source Python library designed for mesh agglomeration in both two- and three-dimensions, based on employing Graph Neural Networks (GNN). `MAGNET` serves as a comprehensive solution for training a variety of GNN models, integrating deep learning and other advanced algorithms such as METIS and k-means to facilitate mesh agglomeration and quality metric computation. The library's introduction is outlined through its code structure and primary features. The GNN framework adopts a graph bisection methodology that capitalizes on connectivity and geometric mesh information via SAGE convolutional layers, in line with the methodology proposed in [1, 2]. Additionally, the proposed `MAGNET` library incorporates reinforcement learning to enhance the accuracy and robustness of the model initially suggested in [1, 2] for predicting coarse partitions within a multilevel framework. A detailed tutorial is provided to guide the user through the process of mesh agglomeration and the training of a GNN bisection model. We present several examples of mesh agglomeration conducted by `MAGNET`, demonstrating the library's applicability across various scenarios. Furthermore, the performance of the newly introduced models is contrasted with that of METIS and k-means, illustrating that the proposed GNN models are competitive regarding partition quality and computational efficiency. Finally, we exhibit the versatility of `MAGNET`'s interface through its integration with `lymph`, an open-source library implementing discontinuous Galerkin methods on polytopal grids for the numerical discretization of multiphysics differential problems.

Keywords: agglomeration, polytopal meshes, graph neural networks, open-source library.
MSC subject classification: 65N22, 65N30, 65N50, 68T07

1 Introduction

Many problems arising in the numerical solution of Partial Differential Equations (PDEs) involve domains characterized by highly complex shapes, heterogeneous media, moving geometries, fractures, inclusions, and/or immersed boundaries that can make the mesh generation process challenging. The ever-increasing need to compute the numerical solution with enough accuracy and at reasonable computational costs has favored in the last decade the development of polytopal Finite Element Methods, i.e., Finite Element Methods that can support general polygonal and polyhedral (polytopal, for short) meshes. These methods offer a natural and flexible way to facilitate the process of mesh generation and allow to describe the computational domain in detail while employing a lower number of degrees of freedom when compared to classical FEM. Examples of polytopal FEMs include the Virtual Element Method (VEM) [3–6], the mimetic

*Corresponding author: `matteo.caldana@polimi.it`

finite difference method [7–9], the Polytopal Discontinuous Galerkin (PolyDG) method [10–13], the Hybrid High-Order method (HHO) [14–16], and the hybridizable discontinuous Galerkin method [17–19]. One of the many advantages of polytopal methods is that they can incorporate naturally mesh agglomeration. In the context of the numerical solution of PDEs, mesh agglomeration finds many essential applications. First, it can be used to construct coarser grids for the underlying differential problems that employ fewer mesh elements, being an accurate geometric description of complicated geometrical features (e.g., complicated boundaries, inclusions, microstructures, interior barriers, layers). Second, it can be employed in an adaptive mesh framework to coarsen the mesh in regions where the error is already under control. Finally, it is one of the main pillars in the construction of multilevel/multigrid iterative solvers to generate a hierarchy of grids to be employed to accelerate the solution of the resulting algebraic problems [20–22].

Performing automatic and suitable quality (polytopal) mesh agglomeration is an open field of research. During mesh agglomeration, it is crucial to preserve the quality of the original mesh, as quality deterioration could negatively affect the numerical method in terms of stability and accuracy. However, there are no well-established effective strategies for this task yet, especially when the underlying computational domain involves heterogeneous materials, inclusions, microstructures, or interior barriers that should be preserved to make the coarse mesh consistent with the different problems under investigation.

Most state-of-the-art classical methods for graph partitioning use a multilevel approach [23–25]: the graph is initially recursively coarsened by collapsing together groups of adjacent nodes, obtaining a sequence of progressively smaller graphs until a target size is reached; then, the coarsest graph is partitioned cheaply and uncoarsened by successively projecting the partition onto the finer graphs. Since we are interpolating back to a larger graph, a refining step is necessary after each projection to preserve the quality of the initial partition. The main advantage of the multilevel approach is being able to efficiently tackle very large graphs by running the partitioning algorithm only on the smallest graph. Examples of available tools for multilevel graph partitioning are METIS [23] and SCOTCH [24]. Other graph partitioning methods include spectral methods [26], which are based on eigenvalue decomposition of the graph Laplacian matrix. These methods produce high-quality partitions, but can be prohibitively expensive on large graphs.

In the last years, much effort has been directed towards the application of Machine Learning (ML) techniques to enhance and accelerate numerical methods. ML algorithms have the advantage of automatically extracting information from large datasets, making them ideal for situations where establishing *a-priori* criteria for the task is not feasible due to the number of possible configurations, which is the case for mesh agglomeration. Among ML techniques, Graph Neural Networks (GNNs) have gathered great interest due to their ability to handle at the same time both mesh connectivity and geometric/physical information about the underlying cells. GNNs are also flexible, easily incorporating additional properties of the mesh simply by adding new input features during the training process.

In this work, we introduce `MAGNET` (Mesh Agglomeration by Graph Neural Network), an open-source Python library, released under the GNU Lesser General Public License, version 3 (LGPLv3). `MAGNET` provides a simple framework for mesh agglomeration in both two- and three-dimensions, it allows to experiment with different neural network architectures, training them, and comparing their performance to state-of-the-art methods like METIS [23] and k-means on standard quality metrics. `MAGNET` can also be easily integrated with other software, in particular, it already interfaces with `lymph` [27], an open-source Matlab library for PolyDG methods: this allows an assessment of the agglomerated meshes by solving differential problems on it. Our approach extends the results presented in [2, 28] and consists of re-framing the problem of mesh agglomeration as a graph partitioning problem by exploiting the dual mesh. A key innovation of

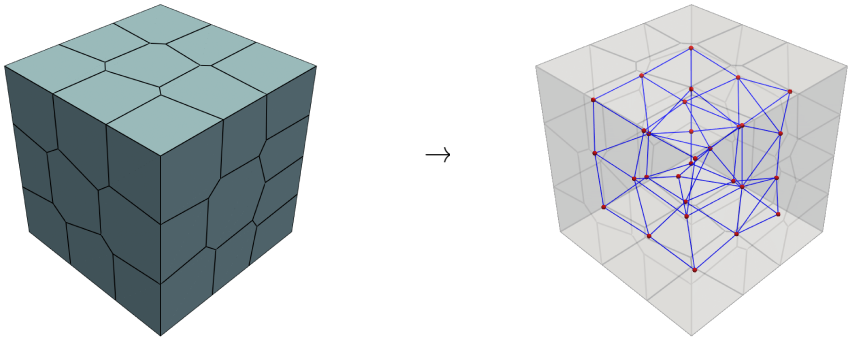


Figure 1: Example of a mesh obtained from the centroidal Voronoi tessellation of a cube (*left*) and a representation of its computational graph $\mathcal{G}$: in red the nodes $\mathcal{V}$, in blue the edges $\mathcal{E}$ (*right*).

the present work is the adoption of the reinforcement learning (RL) approach introduced in [29] to train the model, offering a novel paradigm to adaptively optimize graph-based partitioning in heterogeneous domains. One of the main advantages of GNN-based algorithms is the possibility to incorporate the physical properties of the computational domain during mesh agglomeration and thus taking into account heterogeneities of the computational physical domain, embedded microstructures, and/or interior barriers, and preserve them during the agglomeration process. We remark that our library is independent of both the PDE and the polytopal method under investigation, as it solely relies on the underlying grid agglomeration procedure, making it adaptable in principle for integration into any polytopal code.

The rest of the paper is structured as follows. In Section 2, we frame the problem of mesh agglomeration as a graph partitioning problem. In particular, we describe the GNN approaches used in `MAGNET`, including a novel reinforcement learning approach to mesh agglomeration. In Section 3 we describe the main features of the library and its code structure. In Section 4 we provide a brief user guide, proceeding step-by-step in the agglomeration of a mesh, and showcase some agglomerated examples, comparing the performance of the GNNs approach with state-of-the-art methods. Finally, in Section 5 we illustrate the interface with `lymph`, solving two-dimensional test cases on polygonal meshes agglomerated by `MAGNET`.

The library is available at the GitHub repository `lymphlib/magnet` together with the complete documentation and tutorials.

2 Mesh Agglomeration

In this section, we frame the mesh agglomeration problem as a graph partitioning problem. We also briefly review the state-of-the-art models available in the literature and included in `MAGNET`. In particular, we introduce the GNN agglomeration algorithm, and we show how reinforcement learning can be used to better improve and fine-tune our models.

2.1 Mesh Agglomeration and a Graph Partitioning Problem

Let us consider an open and bounded computational domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$. We consider a discretization $\mathcal{T}_h$ that approximates Ω composed of disjoint open polytopal elements P. Mesh agglomeration consists of merging some elements P of the mesh $\mathcal{T}_h$ to obtain a coarser (connected) element. Most state-of-the-art methods reframe mesh agglomeration as a graph partitioning problem. Namely, they consider the dual mesh, an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where each node $v \in \mathcal{V}$ corresponds to an element P of the mesh, and two nodes v_i, v_j are connected by an edge $e = (v_i, v_j) \in \mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ if the corresponding elements are adjacent, i.e., they share an

edge in two-dimensions or a face in three-dimensions, see Figure 1. Mesh agglomeration is then equivalent to partitioning the dual graph.

Given a connected subgraph $\mathcal{S} = (\mathcal{V}_S, \mathcal{E}_S)$, that is a connected graph where $\mathcal{V}_S \subset \mathcal{V}$ and $\mathcal{E}_S = \{(v_i, v_j) \in \mathcal{E} : v_i, v_j \in \mathcal{V}_S\}$, we define its cut and volume respectively as:

$$\text{cut}(\mathcal{S}) = |(v_i, v_j) \in \mathcal{E} : v_i \in \mathcal{V}_S, v_j \in \mathcal{V} \setminus \mathcal{V}_S|, \quad \text{vol}(\mathcal{S}) = \sum_{v_i \in \mathcal{V}_S} \deg(v_i),$$

where $\deg(v) = |(v, v_i) \in \mathcal{E}, \forall\, v_i \in \mathcal{V}|$ is the degree of node v, and $|\cdot|$ indicates the cardinality of the set. The definition of cut can be extended to a generic partition $(\mathcal{S}_1, \dots, \mathcal{S}_M)$, where $M \in \mathbb{N}$ is the total number of subsets in the partition:

$$\text{cut}(\mathcal{S}_1, \dots, \mathcal{S}_M) = \sum_{i=1}^{M} \text{cut}(\mathcal{S}_i).$$

Typically, when partitioning a graph, we want to minimize the cut while keeping each set balanced, i.e., the subgraphs should have approximately equal size. In the context of mesh agglomeration, this corresponds to requiring that the interface between different agglomerated elements is small and that they have similar sizes (areas or volumes). So, we consider the normalized cut instead, which penalizes partitions including subgraphs with very different volumes:

$$\text{NC}(\mathcal{S}_1, \dots, \mathcal{S}_M) = \sum_{i=1}^{M} \frac{\text{cut}(\mathcal{S}_i)}{\text{vol}(\mathcal{S}_i)}. \tag{1}$$

Depending on the application, other notions of volume may be considered, e.g., the number of nodes. The graph partitioning problem can then be formulated as follows: Find $M > 1$ disjoint subgraphs $(\mathcal{S}_1, \dots, \mathcal{S}_M)$, $\cup_{i=1}^{M} \mathcal{S}_i = \mathcal{V}$, $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset\ \forall i \neq j$ minimizing the total normalized cut $\text{NC}(\mathcal{S}_1, \dots, \mathcal{S}_M)$. The problem of finding a balanced partition that minimizes the cut is NP-complete [30]. As such, current solution algorithms rely on heuristics to find approximate solutions; however, these algorithms tend to be sequential and not easily parallelizable (like Kernigham-Lin (KL) [31] and Fiduccia–Mattheyses (FM) [32]), so they cannot fully exploit modern hardware. On the other hand, deep learning techniques, like graph neural networks, can run efficiently on GPUs, are parallelizable [33], and very flexible in their application; this has motivated research on their employment as graph partitioners [29, 34].

2.2 Graph Neural Network for Mesh Agglomeration

In this section, we introduce the GNN approach to mesh agglomeration; further details can be found in [2, 28].

The mesh agglomeration algorithm we employ consists of recursively applying a single GNN bisection model to the graph extracted from the mesh. Namely, the graph is partitioned by performing a node classification task: taking as input a graph $\mathcal{G}$ and a node feature tensor $\mathbf{X} \in \mathbb{R}^{N \times F}$, where N is the number of nodes and F is the number of features per node, the GNN model returns a probability tensor $\mathbf{Y} \in \mathbb{R}^{N \times 2}$, where Y_{ij} is the probability assigned by the GNN that node v_i belongs to the partition $\mathcal{S}_j$. Since the output of the GNN is a probability distribution, the loss function used during the training process is the expected normalized cut, which takes into account the uncertainty in our predictions on node classification:

$$\sum_{k=1,2} \frac{\sum_{i=1}^{N} \sum_{(v_i,v_j)\in\mathcal{E}} Y_{ik}(1 - Y_{jk})}{\sum_{i=1}^{N} Y_{ik} \deg(v_i)} = \sum_{i,j=1}^{N} \left((\mathbf{Y} \oslash \mathbf{Y}^T \mathbf{D})(1 - \mathbf{Y})^T \odot \mathbf{A} \right)_{ij},$$

where $\mathbf{D}$ is the vector of node degrees $D_i = \deg(v_i)$, $\mathbf{A}$ is the adjacency matrix of the graph, $\oslash$ and $\odot$ denote the element-wise division and multiplication respectively, and the sum is over

all the elements of the resulting matrix (for a more detailed explanation of this framework, we refer to [35]). To bisect a graph, a single forward pass of the GNN model is computed, and nodes are assigned to one of the two classes based on which probability is highest in the output probability matrix $\mathbf{Y}$. This approach could be easily generalized to any number of partitions instead of just two, but this would require a differently trained GNN, with several outputs equal to the number of classes, for each desired one; by instead bisecting the graph recursively, we can obtain any number of partitions using only one model. Moreover, GNNs have the unique advantage of being able to process the connectivity together with geometric data that has been embedded into the graph as node features.

2.2.1 The SAGE-Base Model

The standard GNN model implemented in `MAGNET`, that we will call SAGE-Base in the following, employs four SAGE convolutional layers (sample and aggregate [36]) followed by four linear, or dense, layers; the hyperbolic tangent is used after each layer to keep the output in the range $[-1, 1]$ and facilitate learning. A final softmax layer ensures that the GNN output is a probability distribution over the two classes for each node. We use as node features the coordinates of the centroids of each mesh element and the measure of each element, so that we have a total of four node features (three in the two-dimensional case). Before feeding them to the GNN, an additional normalization step, consisting of normalizing coordinates to zero mean and unit variance and rotating the mesh so that the widest direction is aligned with the x-axis, is performed to reduce variability in input. Since graphs are dimensionless objects, the approach is the same for polygonal and polyhedral meshes: only the number of input features and the normalization procedure have to be adapted between the two. The two-dimensional version of this model has been trained on a set of 800 meshes comprising structured quadrilaterals, structured triangles, random Delaunay triangulations, and random Voronoi tessellations in equal measure. The training has been performed using the Adam optimizer [37] for 300 epochs with learning rate $\gamma = 10^{-5}$, weight decay parameter $\lambda = 10^{-5}$ and batch size 4. The three-dimensional version has a greater number of parameters to account for the increased variability of the three-dimensional case, doubling the size of SAGE and linear layers from 64 and 32 to 128 and 64 units, respectively. The training dataset comprised 400 tetrahedral meshes, of which 100 were of the unit cube and 300 were of randomly generated portions of it (see Figure 4g); other changes include using a learning rate of $\lambda = 10^{-4}$ and lengthening training to 400 epochs. In both cases, meshes have been randomly rotated during training to artificially augment the dataset.

2.2.2 The SAGE-Heterogeneous Model

The SAGE-Base model can be easily extended to include additional node features; in particular, it is possible to perform agglomeration of heterogeneous meshes by including a parameter taking values in the interval $[0, 1]$ that describes the heterogeneity of the mesh, which we will refer to as the physical group. By updating the loss function to include a penalty term for agglomerating elements with very different physical parameters, the GNN can learn to partition it while respecting the heterogeneity of the physical parameters. We choose such a term in this way

$$\frac{a}{|\mathcal{V}|} \sum_{i=1}^{N} \sum_{j=1}^{2} \left(\mathbf{P} \odot \mathbf{Y}\right)_{ij},$$

where $\mathbf{P} \in \mathbb{R}^{N \times 2}$ is a matrix that contains the physical parameter p and its complementary $1 - p$ on each row, $\odot$ denotes element-wise multiplication, $|\mathcal{V}|$ is the number of nodes in the graph and a is a suitable coefficient chosen by performing a hyperparameter search; the term is normalized with respect to graph size to make it scale invariant. While this model works only when there are at most two different physical parameters, this approach could be extended to

an arbitrary number of different physical groups by adding as node features a one-hot encoding of the various heterogeneous parts; however, this would require a different GNN for each total number of physical groups. The obtained model is referred to as SAGE-Heterogeneous in the code. The bisection algorithm is the same since the output matrix $\mathbf{Y}$ has the same meaning. The SAGE-Heterogeneous model included in `MAGNET` has the same architecture as the SAGE-Base model and was trained using mostly the same hyperparameters. The two-dimensional version has been trained for 200 epochs on a dataset of 600 meshes of the unit square, of which 200 are homogeneous, 200 are divided into two parts along a line, and 200 have 6-12 circular inclusions. The three-dimensional version has been trained on a set of 400 meshes of the unit cube divided into one to four physically heterogeneous parts each for 150 epochs.

2.3 Reinforcement Learning

In this section, we aim to introduce the main tools of reinforcement learning and show how to use them to train the GNN model, following the work of [29]. For a complete overview of RL, we refer the interested reader to [38].

Reinforcement learning considers the sequential decision process of an agent (or actor) aiming to optimize its interaction with the environment. At every time step $t \in \mathbb{N}$, the agent observes a state $s_t \in \mathbb{R}^{d_S}$ and takes accordingly a feasable action $a_t \in \mathcal{A} \subset \mathbb{R}^{d_A}$ according to a stochastic policy $\pi_t(a|s_t)$. The environment returns to the agent a new state s_{t+1} and a scalar reward r_{t+1}. The decision process is assumed to be Markov, that is, the policy only depends on the current state s_t and not the previous ones. The objective of the agent is to maximize the cumulative discounted reward:

$$R_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+1+k},$$

where $\gamma \in (0, 1]$ is the discount factor, which describes how far into the future rewards are taken into account by the agent. The value

$$V^\pi(s) = \mathbb{E}_\pi[R_t|s_t = s]$$

is the expected return for starting in state s and following policy π, which is a measure of how "good" state s_t is. In deep reinforcement learning, neural networks are employed to approximate a suitable policy that maximizes R_t. Among a very wide variety of available algorithms, we employ the so-called synchronous advantage actor-critic (A2C) policy gradient approach [39]. A2C is a model-free reinforcement learning algorithm that employs a neural network to approximate the value function V, together with π. Namely, the neural network is formed by two heads, an actor and a critic. Given a state s_t, the actor generates a probability distribution over the actions (that is, a discrete vector of probabilities if $\mathcal{A}$ is finite or parameters of a distribution otherwise). The critic branch of the network is then used to estimate the value $V(s_t)$ of the state s_t. The value is then used to compute the advantage $(R_t - v(s_t))$, which is then used to reinforce the action taken by the agent. In the A2C framework, actor and critic parameters are updated simultaneously, leading to a generally more stable training procedure. The neural network parameters $\boldsymbol{\theta}$ are updated in the direction of the gradient of the policy log-probabilities

$$\mathbb{E}_{\pi_\theta}[\nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t)(R_t - v(s_t))],$$

where the advantage is used instead of directly R_t to stabilize training. An additional hyperparameter $\alpha \in (0, 1]$ dictates how fast the critic learns with respect to the actor. Since RL algorithms heavily depend on implementation-level details [40], we refrain from describing how the gradient updates are computed in practice and direct the reader to the original paper [39] for a complete overview.

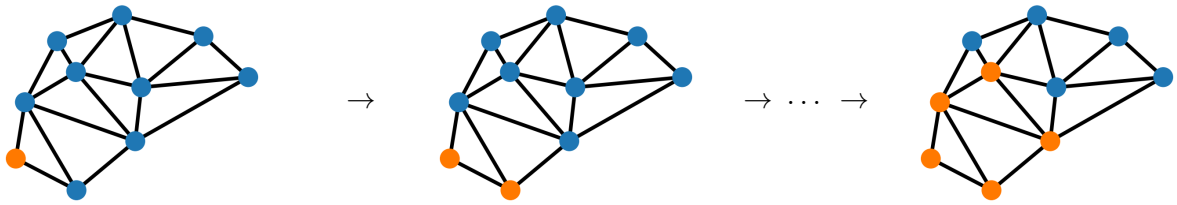


Figure 2: Representation of how the reinforcement learning agent bisects the graph. First, all nodes are in subset $\mathcal{S}_1$ (blue) except for one node with minimum degree (the orange one). At each step, the agent picks a blue node and flips it to an orange one until the two sets have the same number of nodes.

2.3.1 The Reinforcement Learning Partitioner

The reinforcement learning environment for mesh agglomeration that we propose is an extension of the one presented in [29] for graph partitioning to the specific application of mesh agglomeration. The objective of the actor is to partition the graph extracted from the mesh into two subgraphs $\mathcal{S}_1, \mathcal{S}_2$ while minimizing the normalized cut Eq. (1). The state corresponds to the partition of the graph, which is represented in the node features tensor $\mathbf{X} \in \mathbb{R}^{N \times 2}$ by a one-hot encoding: $X_i = [1, 0]$ if $v_i \in \mathcal{S}_1$, while $X_i = [0, 1]$ if $v_i \in \mathcal{S}_2$. Each action, chosen according to the policy of the actor, corresponds to moving one node from $\mathcal{S}_1$ to $\mathcal{S}_2$ or vice versa, modifying the underlying feature tensor. The actor's policy is thus a probability tensor over all the nodes of the graph. We employ as a reward for each action its corresponding decrease in normalized cut, since this is the quantity we want to minimize. For simplicity, initially, all nodes are in subgraph $\mathcal{S}_1$, except one node which is chosen among the ones with minimum degree so that the cut is as low as possible, and the actor can only move nodes from $\mathcal{S}_1$ to $\mathcal{S}_2$. This procedure is illustrated in Figure 2.

It is possible to further customize the learning process by including additional rewards or penalties, e.g., for having non-connected subgraphs. To improve the quality of the partition, we include among the features the centroid coordinates of each cell, so that the actor may also exploit geometric data to inform its policy. The length of the episode is chosen as $|\mathcal{V}|/2$, that is the actor moves nodes from $\mathcal{S}_1$ to $\mathcal{S}_2$ until they have the same cardinality; in this way, at the end of the episode we naturally obtain a balanced partition. Like for the SAGE-Base model, a partition is obtained by recursively applying this bisection model to the mesh, except now a forward pass of the GNN is needed for each action in the episode instead of only once per bisection. We note that the computational cost could be reduced by selecting multiple actions from the same policy, effectively reducing the number of forward passes needed to perform the task.

The model is implemented in `MAGNET` using four convolutional SAGE layers and two linear layers that are common to both actor and critic; the critic branch is then formed by two further linear layers with a final softmax layer to create a probability distribution over the graph nodes; to enforce the constraint that chosen nodes can no longer be moved back, we set to minus infinity the input to the softmax layer corresponding to nodes that have already been flipped, so that the probability that they will be chosen is zero. The critic branch uses an Attentional Aggregation layer [41] followed by two linear layers that taper off in dimension, finally returning a singular scalar corresponding to the value estimate. All layers, except for the last ones, are interlaced with a hyperbolic tangent activation function. The pre-trained RL Partitioner model bundled with `MAGNET` has been trained on a dataset of 1000 meshes (with the same composition as the one of the two-dimensional SAGE-Base model) for one epoch, updating the model parameters at the end of each episode using the Adam optimizer with a learning rate of 10^{-3}. The discount factor γ is chosen to be 0.9, since we are not interested in a greedy approach but rather the

long-term result of the episode, while α is set to 0.1 to let the actor learn faster than the critic.

2.3.2 The Reinforcement Learning Refiner

As we explained in Section 2.4, most state-of-the-art methods use a multilevel approach, which requires a refinement step after each successive uncoarsening of the graph. This process can be tackled by using an actor-critic agent similar to the one presented for graph partitioning, with the difference that nodes can now be moved back and forth between the two subgraphs instead of only progressively growing one of the two from zero, and that the starting state is the projected coarse partition. Assuming that the bisection algorithm has already achieved a good quality partition on the coarser problem, only a few nodes will need to be moved during the refinement process, and these nodes will be close to the interface between the two subgraphs. Consequently, we choose a short episode length equal to the cut itself $\text{cut}(\mathcal{S}_1, \mathcal{S}_2)$, and instead of running the model on the entire graph we consider only the k-hop subgraph around the cut (that is the subgraph obtained by taking all the nodes with distance at most k from a given center), with $k = 2, 3, 4$. This enhancement significantly reduces the computational cost. The loss of the global context means that two node features indicating the current partition volume balance need to be added to the feature tensor as a trade-off. As before, the reduction in the normalized cut will be the actor's reward. Since the initial coarse bisection is likely to already have very balanced volumes, a decrease in cut tends to be more significant and disproportionately preferred by the agent than an increase in partition balance. To counterbalance this effect, we add a penalty term for imbalanced volumes:

$$b\frac{(\text{Vol}(\mathcal{S}_1) - \text{Vol}(\mathcal{S}_2))^2}{\text{Vol}(\mathcal{V})},$$

where $b > 0$ is a hyperparameter that weights the imbalance penalty. Enforcing a stricter balance requirement is crucial since small imbalances can compound over many uncoarsening and refinement steps, leading to agglomerated elements with very different sizes. In the RL refiner model implemented in `MAGNET`, actor and critic have one SAGE and one dense layer each, sharing two additional common SAGE layers; the hyperbolic tangent is used as an activation function, and the actor branch terminates with a softmax layer as usual. The refiner architecture can afford to be very light due to the limited task it needs to perform and lower variability in starting configurations. The model has been trained on a dataset of around 5000 meshes generated starting from a kernel of 1000 meshes (of the same type as the ones used for the two-dimensional SAGE-Base model) by recursively coarsening them and adding the coarsened versions to the dataset. The SAGE-Base model of Section 2.2.1 has been used as a coarse partitioner to generate the initial cut. The network parameters are updated every few steps (in this case 8) instead of only at the end of each episode because, by starting at a configuration close to the optimum, almost all episodes will lead to a negative reward and the network will struggle to learn; more frequent parameters updates give the agent more granular information on the value of its decisions, alleviating the issue. The imbalance penalty used was $b = 0.35$; otherwise, the same hyperparameters of the RL Partitioner model were used.

2.4 Other Agglomeration Strategies Available in `MAGNET`

In addition to the GNN strategies described in the previous Section 2.2 and Section 2.3, `MAGNET` also includes both METIS and k-means as possible agglomeration approaches. METIS [23] is a widely used library for computing k-way partitions of large irregular graphs based on a multilevel procedure. It uses simple and efficient algorithms (like greedy approaches) for partitioning the coarsest graph, while the refinement is performed based on local heuristics like KL and FM, while respecting balancing constraints. To agglomerate a mesh, METIS is run with the requirement of creating connected subgraphs; also, we use as node weights the volumes of the corresponding

cells so that the balance constraint in the partitioning algorithm is imposed in terms of geometric size, which is more relevant to our application than simple node cardinality. K-means [42] is a quantization algorithm that clusters a set of points in $\mathbb{R}^N$ into k sets by minimizing the squared Euclidean distances of each point to the centroid of its group, i.e. by minimizing the within-group variance. While it cannot be applied directly to graphs because they are non-Euclidean data, k-means can be used with node embedding techniques to map each node onto a feature space, producing an effective clustering of the nodes. Indeed, instead of applying k-means to the mesh graph, we use it to directly cluster the cells by using as features the centroid coordinates of the mesh elements, producing fairly rounded agglomerated elements of similar size. In this way, we are neglecting the mesh connectivity, but this is usually not an issue because for meshes with similarly sized elements, centroid coordinates are strongly correlated with two cells being adjacent.

3 `MAGNET`: Library Overview

In this section, we describe the tools and design principles of `MAGNET`. The main design goal of `MAGNET` is to provide a single flexible framework for comparing the performance of different methods for mesh agglomeration, paying particular attention to emerging Machine Learning approaches. `MAGNET` is able to agglomerate two-dimensional and three-dimensional meshes, including ones obtained from mesh generation of heterogeneous domains, for which agglomeration should preserve the underlying distribution of physical parameters. Functionalities of the package include: basic input-output of meshes, GNN training, generation of meshes for testing/training, agglomeration by various methods, computation of quality metrics for agglomerated meshes, and visualization.

3.1 Code Structure

`MAGNET` is organized in the following sub-modules:

- `mesh`, `cell`: defines the mesh and cell data structures;
- `aggmodels`: contains the agglomeration algorithms, in particular, the recursive bisection algorithms and GNN architectures definition;
- `io`: functions needed to load and write meshes together with the graph extraction process, plus additional post-processing functions for visualization.
- `generate`: utility functions for generating meshes for training GNNs and testing.

The package additionally includes the `agglomerate.py` script, which can be called from the command line to agglomerate a single mesh, and is also used for communication with the `lymph` library (see Section 5). Finally, the `models` folder contains some pre-trained GNN models according to the specifications detailed in the previous sections.

3.1.1 The Agglomeration Models

The core of `MAGNET` is the `aggmodels` Sub-module. It contains the `AgglomerationModel` abstract class, from which all the implemented models inherit, containing the definition of the recursive bisection algorithms. In particular, here, the SAGE-Base, SAGE-Heterogeneous, RL Partitioner, and RL Refiner Neural network architectures and their training algorithms are defined. METIS and k-means are also included as state-of-the-art methods to which compare them to. The GNNs are all implemented using Pytorch Geometric [43] and defaulting to a GPU backend (if possible) to speed up computations.

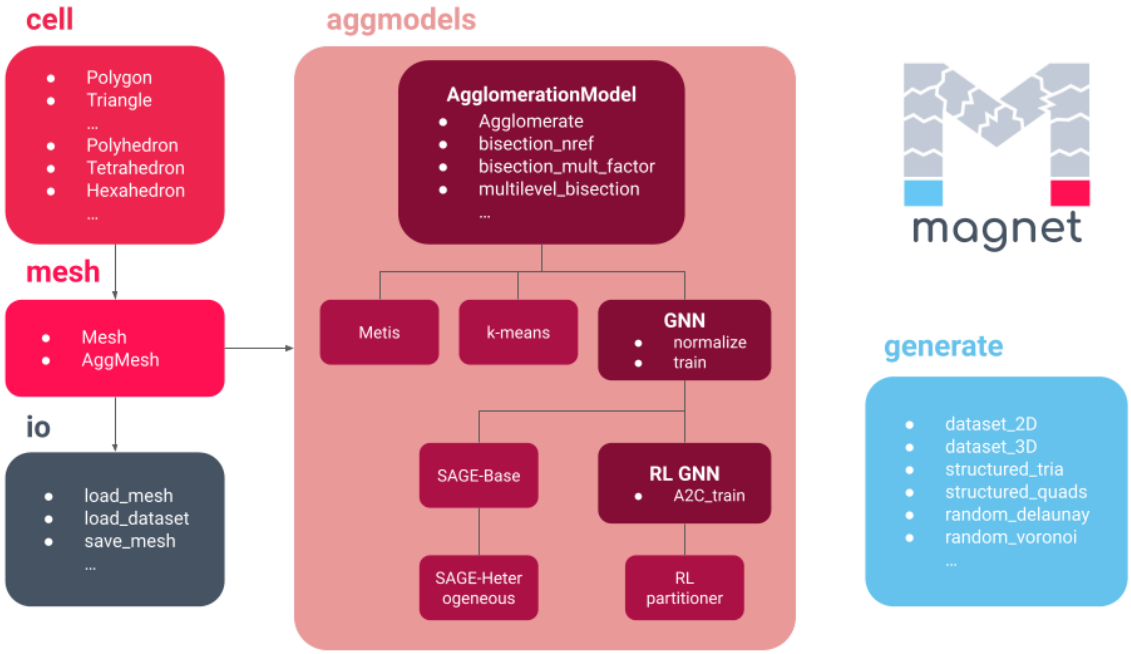


Figure 3: Code structure of `MAGNET`, highlighting the agglomeration models class hierarchy. Arrows denote import relations.

3.1.2 The IO Sub-module

This module contains functions for loading meshes and datasets, extracting their graph, and saving agglomerated meshes to a file. The package relies on `meshio` [44] for input, so most of its wide assortment of supported formats can be agglomerated; polygonal, tetrahedral, hexahedral, and pyramidal cells are currently supported. As far as output is concerned, `meshio` is once again used in the two-dimensional case for flexibility in output format; on the other hand, in the three-dimensional case, we rely on the `vtk` library [45], since `meshio` has some limitations when dealing with polyhedra. The graph extraction process consists of the computation of the adjacency matrix describing the mesh, together with areas/volumes and centroid coordinates for each mesh element. The adjacency matrix is computed starting from the mesh connectivity data using Algorithm 1. The algorithm achieves linear complexity w.r.t. the number of cells in the average case by the use of a hash map.

3.1.3 The Generate Sub-module

The `generate` Sub-module enables the generation of simple meshes of the unit square (including structured triangles, structured quadrilaterals, random Delaunay triangulations, random Voronoi tessellations, random circular holes and inclusions) and of the unit cube or portions of it. Some examples of meshes that can be generated through this module are shown in Figure 4. For mesh generation, `MAGNET` relies on the open source software `Gmsh` [46]. The main purpose of this module is to supply a convenient way of randomly generating large datasets through the `dataset_2D` and `dataset_3D` functions for training Neural Networks and testing different agglomeration approaches.

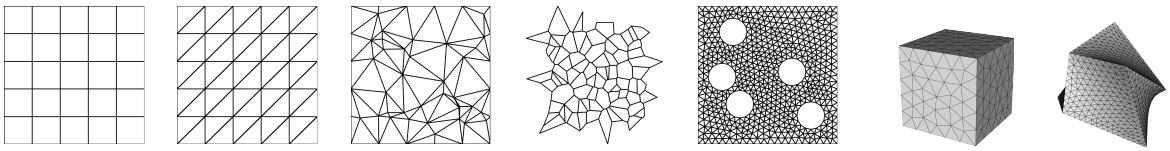
3.2 Agglomeration Algorithms

At the core of `MAGNET` is the interplay between agglomeration models and agglomeration modes. Agglomeration models, like SAGE-Base or METIS, are subclasses of the `AgglomerationModel` class and define how the mesh is bisected and what data is needed to perform it. Agglomeration modes,

Algorithm 1 Adjacency matrix computation algorithm

Input: `cells`: List of N cells, where each cell has a list of faces.
Output: **A**: Adjacency matrix of the mesh in sparse format.

```
face_to_cell ← {}                          ▷ Initialize to empty dictionary, with default element list
for cell_id ← 0 to N − 1 do
    for face_id in cells[cells_id].faces do
        face_to_cell[face_id].append(cell_id)
    end for
end for
A_ij ← 0, i, j = 1, ..., N                 ▷ Initialize adjacency matrix as sparse empty matrix
for i ← 0 to N − 1 do                      ▷ Loop over the id of the cell
    for face_id in cells[cells_id].faces do
        for j in face_to_cell[face_id] do
            if j ≠ i then                  ▷ Exclude the cell itself
                A_ij ← 1
            end if
        end for
    end for
end for
return A
```



(a) Quads (b) Triangles (c) Delaunay (d) Voronoi (e) Holes (f) Cube (g) Portion

Figure 4: Examples of the meshes that can be generated using the `generate` Sub-module of `MAGNET`.

on the other hand, dictate how the bisection is applied in practice and act on an algorithmic level. `MAGNET` provides a few different agglomeration modes:

- *direct k-way*: the mesh graph is partitioned in k parts in one shot. This is not available for the GNNs methods, as they rely on recursive bisection.
- *number of refinements*: the mesh is bisected recursively $N_{ref} \in \mathbb{N}$ times, leading to a total of $2^{N_{ref}}$ agglomerated elements.
- *target size*: the mesh is bisected recursively until all elements have a diameter that is smaller than the given input quantity.
- *multiplicative factor*: similar to the previous one, but the target dimension is computed as a fraction of the diameter of the entire mesh, expressed by the input parameter `mult_factor`.
- *segregated*: exclusively for heterogeneous meshes; the heterogeneous parts of the mesh are separated and their graphs partitioned independently, relying on the *multiplicative factor* algorithm; the node classification is then used to perform the actual agglomeration of the cells in one shot.
- *coarsen*: allows to agglomerate (according to one of the previous modes) only part of the mesh by passing the indices of the cells corresponding to it.

Algorithm 2 Sequential *multiplicative factor* algorithm

Input: $\mathcal{G}$, `mult_factor`: graph to be partitioned, multiplicative factor
Output: Elements classified based on their agglomerated element

$\hat{h} \leftarrow \text{diam}(\mathcal{G}) * \texttt{mult\_factor}$ ▷ $\text{diam}(\mathcal{G})$ is the largest pair distance of vertexes of τ_h.
`parts` $\leftarrow [\mathcal{G}]$
`output` $\leftarrow []$
while `parts` **is not** $\emptyset$ **do**
    `new_parts` $\leftarrow []$
    for $\mathcal{S}$ **in** `parts` **do**
        if $\text{diam}(\mathcal{S}) > \hat{h}$ **then**
            $\mathcal{S}_1, \mathcal{S}_2 \leftarrow \text{BISECT}(\mathcal{S})$ ▷ If the element is too "large", bisect it
            `new_parts` $\leftarrow$ `new_parts` $+ [\mathcal{S}_1, \mathcal{S}_2]$
        else
            `output.append`$(\mathcal{S})$ ▷ Otherwise, insert it in the final output
        end if
    end for
    `parts` $\leftarrow$ `new_parts`
end while
return `output`

- *multilevel recursive bisection*: the mesh graph is initially coarsened using a greedy heavy edge matching algorithm [25], then bisected; the coarse partition is successively projected unto the finer graph and refined (using, for example, the RL Refiner of Section 2.3.2); this process is recursively repeated to obtain the desired partition. See `_absaggmodels.py` line 382 for details.

While methods like METIS and k-means generally perform better when partitioning the mesh directly in the desired number of parts with *k-way* mode instead of applying them recursively, it is still possible to do so using one of the other modes. This is important because recursive bisection algorithms naturally generate a hierarchy of nested coarser grids that can be used in multigrid methods to accelerate numerical solvers. We note that for really small `mult_factor`, the necessary number of bisections and corresponding recursive calls in the stack grows rapidly; this can quickly use up a lot of memory, so the implementation is sequential rather than recursive. A schematic of the *multiplicative factor* bisection algorithm is reported in Algorithm 2. We remark that some agglomeration approaches, especially k-means and SAGE-Base, tend to agglomerate parts of the mesh that are not connected, in particular when holes are present or the underlying geometry is very complex; as such, at the end of the graph partitioning process an additional check on the connectedness of each subgraph is performed, and if more than one connected component is found they are separated (an efficient algorithm for this task is described in [47]). Finally, since we always frame mesh agglomeration as a graph partitioning problem, the actual geometric merging of the cells can be performed later after the nodes have been classified; this modularity allows for great flexibility in the classification process while sharing the same implementation of the merging step across all agglomeration models.

3.3 Extensibility

The agglomeration model class hierarchy has been designed with extensibility, making adding new bisection strategies easy. When defining a new agglomeration model inheriting from the abstract base class, only two new methods strictly need to be defined: the `get_graph` method that specifies the graph data needed for bisection, and the `bisect_graph` method that performs bisection on it; GNN models will additionally need an initialization method defining their archi-

tecture and a `forward` method. Of course, further customization is always possible by overriding existing methods or implementing new ones.

3.4 Heterogeneous Mesh Agglomeration

Traditional mesh agglomeration strategies, like METIS, cannot deal with meshes describing heterogeneous domains, i.e. meshes that have regions with different physical parameters. The only way to do this using these methods is to agglomerate different regions independently and then merge them; however, the merging process can be rather expensive. Also, this strategy is not well suited for domains with micro-structures, where independent agglomeration of the structures leads to poorer quality agglomerated elements. `MAGNET` can agglomerate heterogeneous meshes in two different ways:

1. By using a GNN (SAGE-Heterogeneous) that takes as input an additional node feature, the physical group, that describes the heterogeneity of the mesh, by using the strategy illustrated in Section 2.2.2.
2. By first separating the heterogeneous parts of the domain and partitioning each corresponding graph separately, using *segregated* mode, described in Section 3.2.

The first approach is less expensive, but can only handle a certain class of physical groups and could wrongly agglomerate elements with different physical parameters. The second approach always separates correctly physical groups and can exploit any of the supported agglomeration strategies, but is more expensive, especially in the case of many small inclusions in the domain.

3.5 Quality Metrics

The notion of mesh quality for triangular, quadrilateral, and tetrahedral meshes is deeply explored in the literature [48]. However, there is no shared consensus on the characterization of good quality for polygonal and polyhedral meshes, which is still an object of ongoing research [49, 50]. For example, a standard assumption on shape regularity for FEMs on polytopal meshes is that each element of the mesh is star-shaped with respect to a point; however, numerical experiments have shown that these methods can reliably solve differential problems also on much more irregular meshes [51], so the practical requirements are much weaker. `MAGNET` is flexible in this regard and allows for easy implementation of new quality metrics and integrates them with the rest of the framework. The following quality metrics described in [1, 51], which are defined element-wise and take values in [0,1], are provided in `MAGNET`:

- Circle Ratio (CR): the ratio between the radius of the smallest sphere inscribed in the element P and the biggest sphere containing it:

$$\mathrm{CR}(P) = \frac{\max\limits_{B(r)\subset P} r}{\min\limits_{B(r)\supset P} r} \tag{2}$$

  where $B(r)$ is the ball of radius r. It is a measure of the roundness of the elements: the closer the circle ratio is to 1, the closer the element is to a sphere.

- Area to Perimeter Ratio (APR): the ratio between the area of the polygon P and its perimeter.

$$\mathrm{APR}(P) = \frac{4\pi|P|}{|\partial P|^2} \tag{3}$$

  It is also known as the iso-perimetric quotient because it is the ratio between the area of P and that of a circle with the same perimeter P. This metric is a measure of the

compactness of the polygon: it has a maximum value of 1 for the circle and decreases for less compact polygons (i.e. the shape is more spread out). An equivalent metric in three-dimensions is sphericity:

$$\Psi = \frac{\sqrt[3]{36\pi|P|^2}}{|\partial P|} \tag{4}$$

- Uniformity Factor (UF): the ratio between the diameter of the element and the mesh size:

$$\mathrm{UF}(P) = \frac{\mathrm{diam}(P)}{h}, \quad h = \max_{P\in\tau_h} \mathrm{diam}(P), \tag{5}$$

where $\mathrm{diam}(P)$ indicates the maximum among the distances of any pair of vertices of P. Values closer to one denote that mesh elements have similar diameters.

- Volumes Difference (VD): the relative difference between the volume of the cell P and the volume that each cell should have if they all had the same volume $\hat{V}$:

$$\mathrm{VD}(P) = \frac{|\mathrm{Vol}(P) - \hat{V}|}{\hat{V}}, \quad \hat{V} = \frac{\sum_P \mathrm{Vol}(P)}{N} \tag{6}$$

This quantity can take any positive value, so to have a metric between zero and one we consider instead:

$$\widetilde{\mathrm{VD}}(P) = \frac{1}{1+\mathrm{VD}(P)} \tag{7}$$

A value of one denotes that all elements have the same volume, while lower values correspond to a less homogeneous volume distribution across the cells.

4 Numerical Results

In this section, we provide a brief step-by-step guide, showing how to train a GNN and how to agglomerate meshes with `MAGNET`, starting from the two-dimensional homogeneous case and moving on to more complex applications. We also use these examples to comment on the performance and main features of the considered methods, comparing the GNN strategies with METIS and k-means using the metrics of Section 3.5.

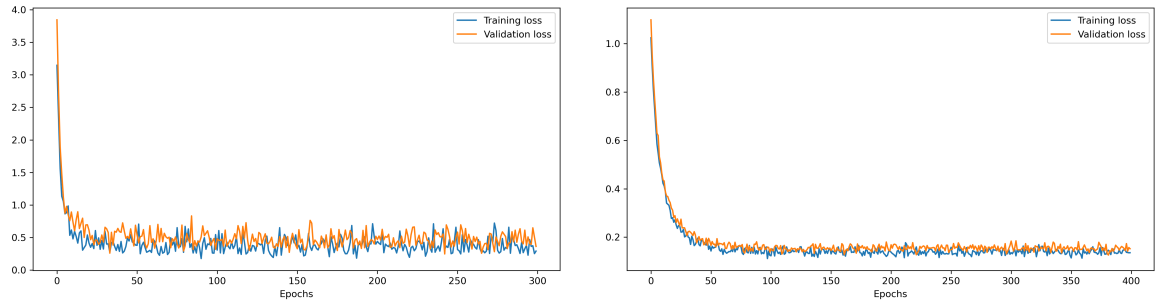
4.1 GNN Training

To train a GNN, we need a training dataset and a validation dataset to evaluate the model's performance during the training. To speed up training, mesh graphs (adjacency matrix, centroid coordinates, volumes) need to be pre-computed and stored. `MAGNET` provides a `generate` Submodule that generates meshes while computing their graph data; it is also possible to create a dataset from a folder of existing meshes by using `create_dataset`. After instantiating the GNN model, training can be initiated with the `train_GNN` method; Listing 1 gives an example of how the whole pipeline looks in `MAGNET`.

```
import magnet
# create a training dataset and load it
magnet.generate.dataset_2D({'random_delaunauy':200},'datasets/trainig_set')
training_dataset = magnet.io.load_dataset('datasets/trainig_set')
# initialize GNN
NN = magnet.aggmodels.SageBase(64,32,3,2).to(magnet.aggmodels.DEVICE)
NN.train_GNN(training_dataset, epochs=200, batch=4, lr=0.0001)# training
loop
NN.save_model('models/test_training.pt')

```

Listing 1: Example of the code for GNN training.



(a) 2D SAGE-Base (b) 3D SAGE-Base

Figure 5: Training and validation losses history plot of the two-dimensional and three-dimensional versions of the SAGE-Base model of Section 2.2.1.

For the reinforcement learning approach, the method `A2C_train` is used instead. When training is completed, you have the option to save a training summary log file and the loss history; Figure 5 shows as an example the loss history plots for the SAGE-Base models of Section 2.2.1.

4.2 Test Case 1: Two-Dimensional Brain Slice

To showcase the agglomeration of two-dimensional homogeneous meshes and the generalization capabilities of the presented GNN models, we consider as test cases two human brain sections coming from Magnetic Resonance Imaging (MRI); these domains are very complex, have narrow sections, and also include holes in the second case. The code for Test Case 1 is reported in Listing 2. First, we load the mesh by using `io.load_mesh`, which also extracts the mesh graph needed for agglomeration. Then, we initialize our GNN agglomeration model and load it from a state dictionary. We can then proceed to agglomerate the mesh using one of the modes introduced in Section 3.2 and plot it. Finally, using the `get_quality_metrics` method, the quality metrics described in Section 3.5 are computed and plotted.

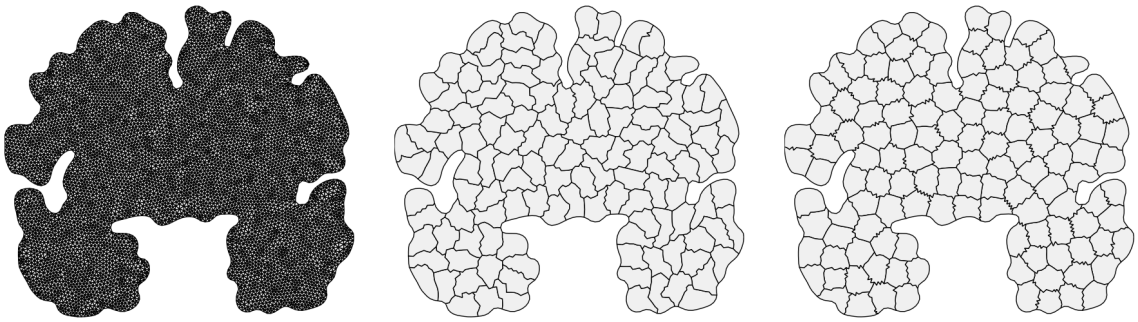
```
from magnet import io, aggmodels
brain_mesh = io.load_mesh('datasets/BrainCoronal.vtu')
NN = aggmodels.SageBase(64,32,3,2).to(aggmodels.DEVICE)  # initialize GNN
agglomerated_brain = NN.agglomerate(brain_mesh, mode='Nref', nref=7)
agglomerated_brain.view(colors='grey')
agglomerated_brain.get_quality_metrics(boxplot=True)

```

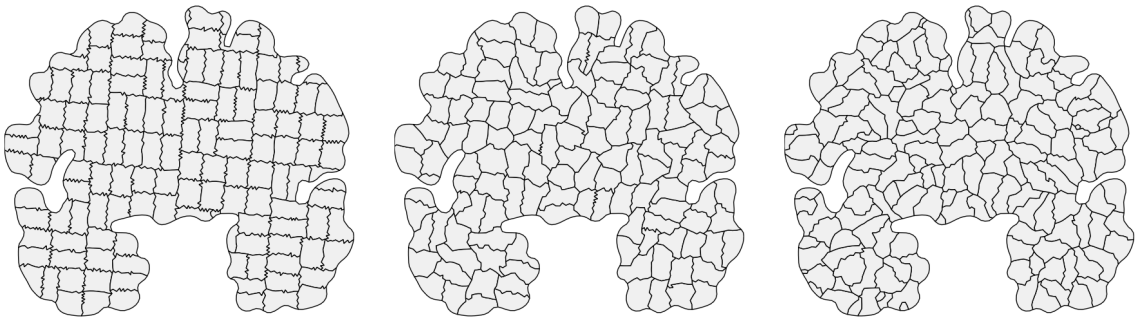
Listing 2: Example of the code for agglomerating one mesh.

We agglomerate the two meshes using five different models (METIS, k-means, SAGE-Base, multilevel approach with RL Refiner, and either SAGE-Base or RL Partitioner as coarse partitioner) using the template of Listing 2. For the GNN models, *number of refinements* mode was used with `nref=7`, creating a total of 128 elements, while METIS and k-means were used with *direct k-way* mode and `k=128` to have a fair comparison. We report the results in Figure 6 and Figure 7. In particular, Figure 6g and Figure 7g show the quality metrics computed for each agglomerated mesh, denoting meshes obtained with different agglomeration models with distinct colors; since these metrics are defined element-wise, each box plot shows their distribution in a single mesh.

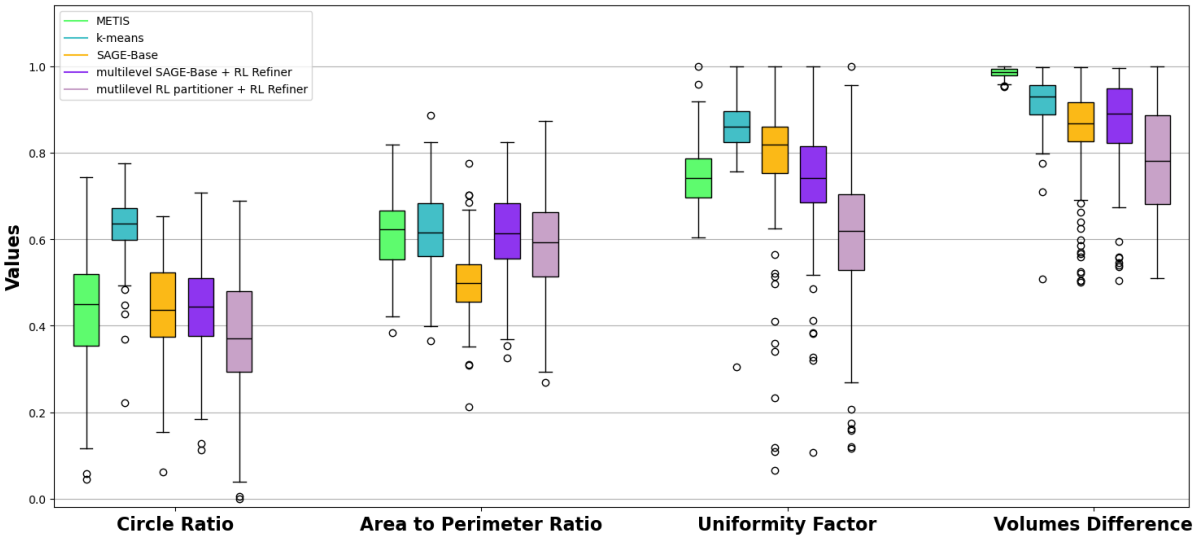
We observe that k-means is by far the best one in terms of both circle ratio and uniformity factor, which is not surprising considering that it is a clustering algorithm that exploits the geometric information of the mesh. METIS performs extremely well in terms of volume uniformity due to its strict algorithmic constraint on partition balance and the fact that we are using cell volumes themselves as node weights. The other models perform roughly the same in terms of the other metrics, except the RL coarse partitioner model, which is slightly worse across the board.



(a) Original mesh (b) METIS (c) k-means

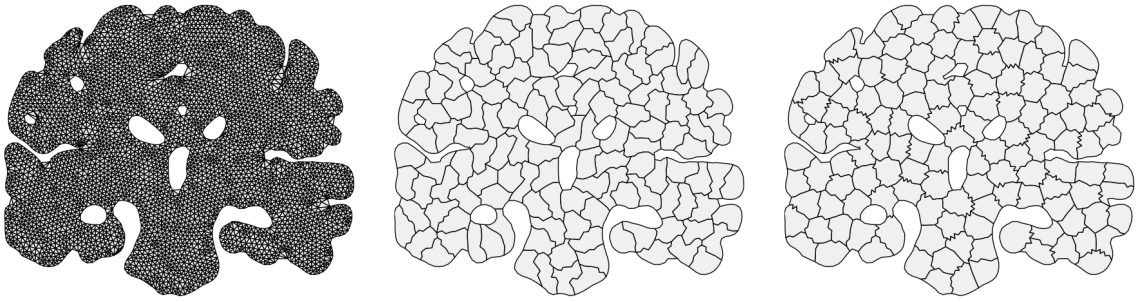


(d) SAGE-Base (e) SAGE-Base + RL Refiner (f) RL Partitioner + RL Refiner

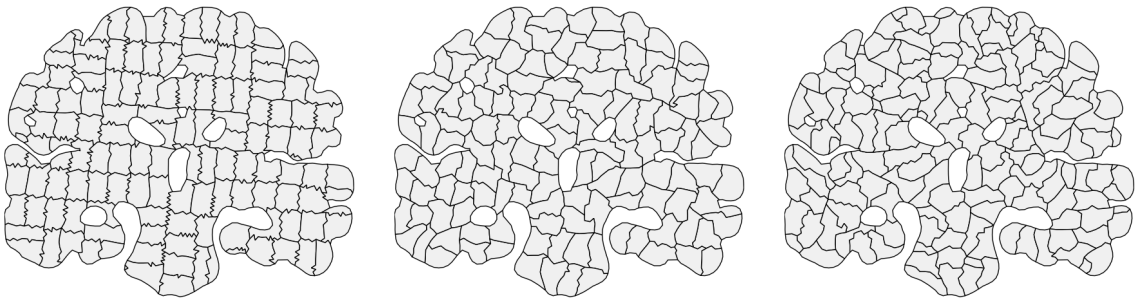


(g) Quality metrics box plots

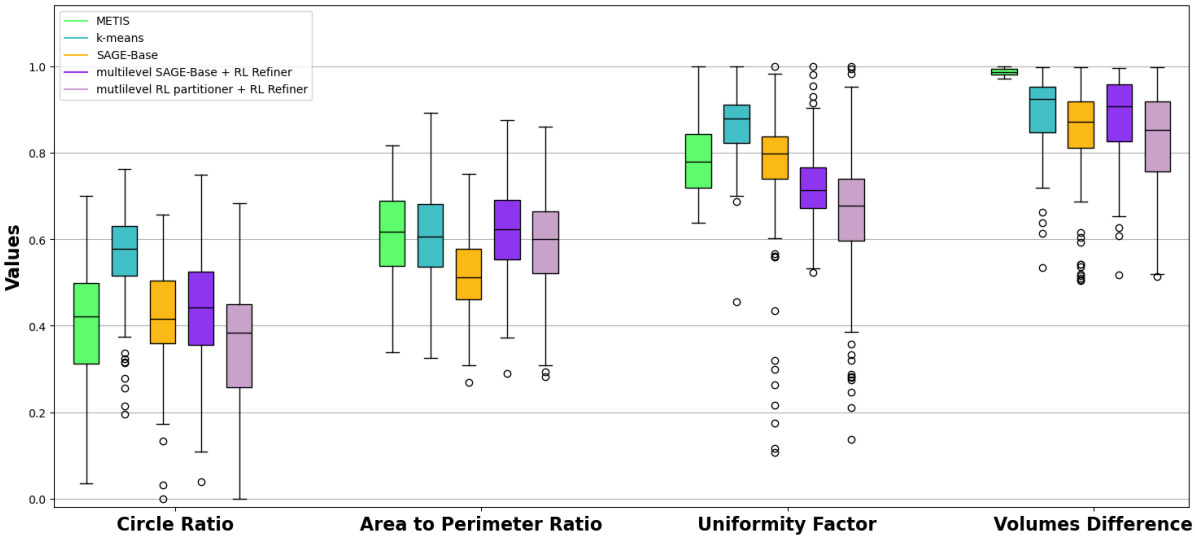
Figure 6: Test Case 1.a: unstructured mesh of a human brain section consisting of 14372 triangles, agglomerated using different methods (METIS, k-means, SAGE-Base, SAGE-Base and RL coarse partitioner in multilevel framework with RL Refiner), together with the box plots of the computed quality metrics (CR, APR, UF and $\widetilde{\text{VD}}$, defined as in Eq. (2)-(7)).



(a) Original mesh (b) METIS (c) k-means



(d) SAGE-Base (e) SAGE-Base + RL Refiner (f) RL Partitioner + RL Refiner



(g) Quality metrics box plots

Figure 7: Test Case 1.b: unstructured mesh of a human brain section consisting of 8597 triangles, agglomerated using different methods (METIS, k-means, SAGE-Base, SAGE-Base and RL coarse partitioner in multilevel framework with RL Refiner), together with the box plots of the computed quality metrics (CR, APR, UF and $\widetilde{\text{VD}}$, defined as in Eq. (2)-(7)).

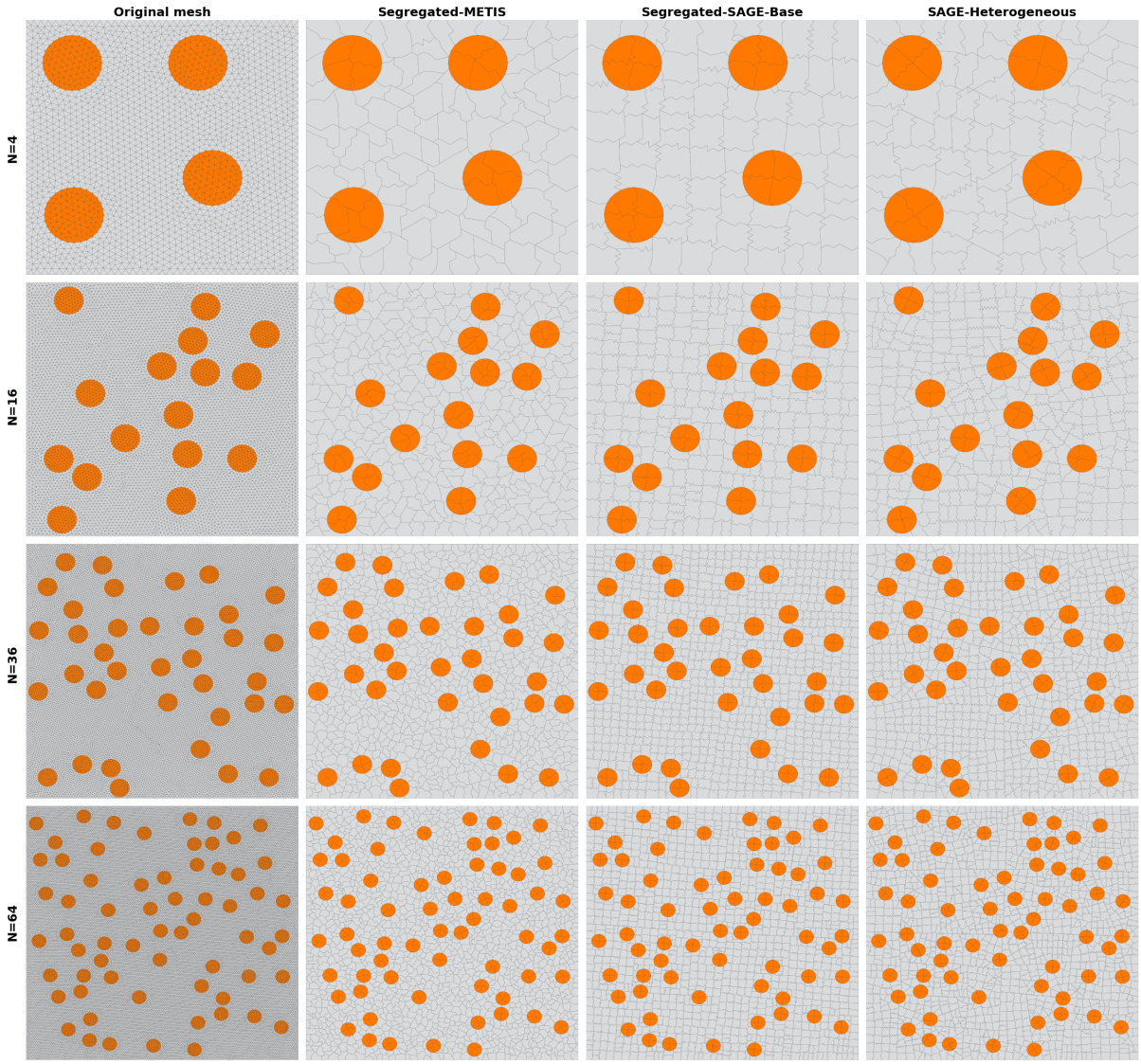
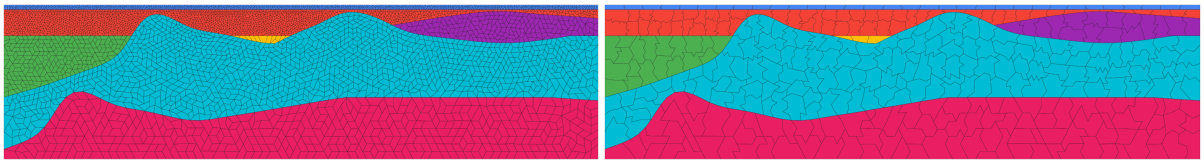


Figure 8: Test Case 2: four meshes of the unit square containing 4, 16, 36, 64 circular inclusions representing microstructures in the domain, consisting of 3796, 15374, 35132, 62766 triangles respectively. The meshes have been agglomerated with METIS and SAGE-Base in *segregated* mode and with SAGE-Heterogeneous, using a target size equal to 3/4 of the circle's diameter.

Finally, we notice that SAGE-Base tends to bisect the mesh along straight lines, leading to the formation of elements with squared corners and a lower area-to-perimeter ratio as a consequence.

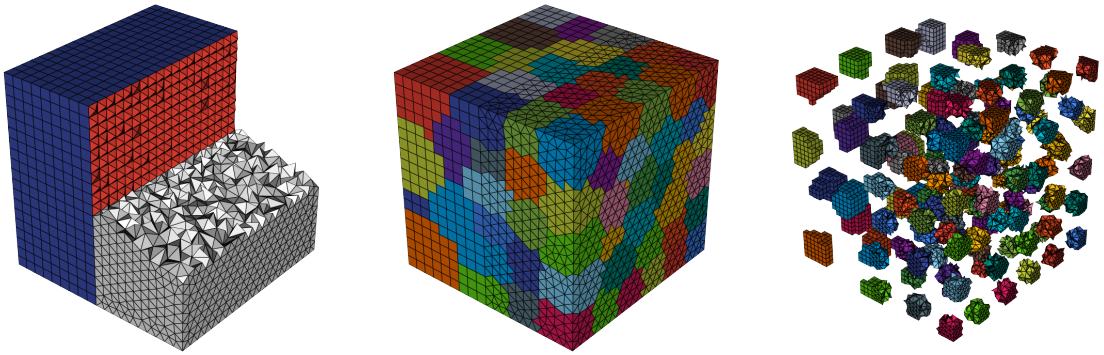
4.3 Test Case 2: Two-Dimensional Domain with Inclusions

We consider a set of four triangular meshes of the unit square with an increasing number of circular inclusions that represent microstructures in the underlying domain, with respectively 4, 16, 36, and 64 inclusions each. The radii of the circles have been chosen so that they always cover 15% of the total area. We agglomerate them both with the SAGE-Heterogeneous model and by using *segregated* mode with both METIS and SAGE-Base using a target size equal to 3/4 of the circle diameter, so that the agglomerated elements have a size comparable to that of the inclusions. The results are reported in Figure 8. In this case, the SAGE-Heterogeneous model correctly separates the inclusions; however, since from the second bisection onward only one of the two physical groups appears, k-means is called instead.



(a) Original mesh (b) Agglomerated mesh

Figure 9: Test Case 3: an unstructured polygonal grid of a layered medium; materials with different physical properties are denoted with different colors. On the left, the original mesh with 4870 elements; on the right, the corresponding agglomerated mesh obtained with the SAGE-Base model using *segregated* mode and multiplicative factor of 0.04, resulting in 363 elements.



(a) Section of the original mesh (b) Agglomerated mesh (c) Exploded view

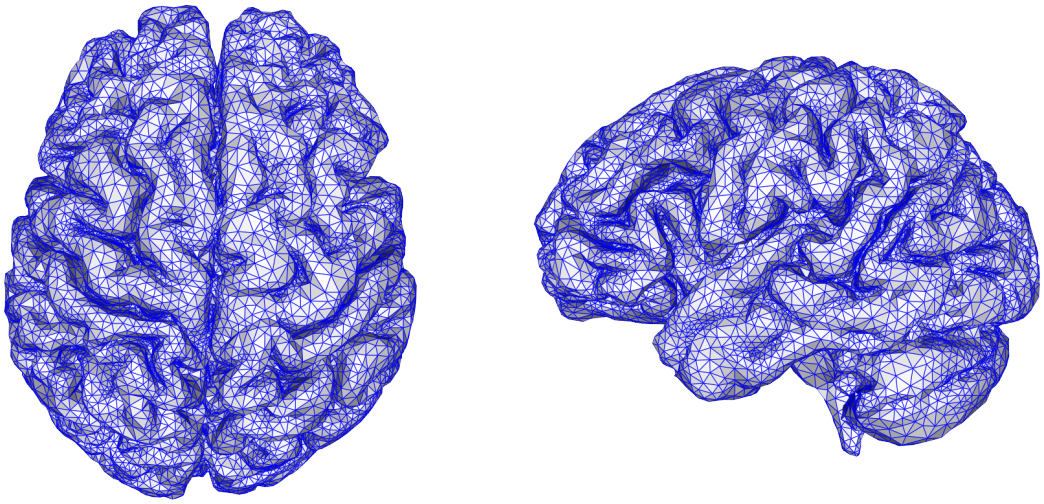
Figure 10: Test Case 4: mesh formed by 27484 cells, comprised of 23874 tetrahedra, 3249 hexahedra, and 361 pyramids. On the left, a cut of the original mesh highlighting in red the pyramids for the transition between the white tetrahedra and blue hexahedra. On the center and right, the mesh agglomerated by k-means with `k=128`, and its exploded view.

4.4 Test Case 3: Two-Dimensional Layered Medium

In this test case, we showcase the ability of `MAGNET` to handle physical domains associated with multiple physical quantities. As explained in Section 2.2.2, SAGE-Heterogeneous can only handle two distinct physical parts at most. If we consider a mesh with more than two physical parameters, the only option is to use the *segregated* mode. We demonstrate this by agglomerating a mesh of a layered medium, including seven different physical parameters, using the SAGE-Base model. The mesh represens an idealized bidimensional Earth's cross-section $\Omega = (0, 38.4)\text{km} \times (0, 10)\text{km}$ used for the simulation of seismic wave propagation [12]. The results are reported in Figure 9.

4.5 Test Case 4: Hybrid Mesh of a Three-Dimensional Domain

In the following test case, we want to highlight that `MAGNET` can correctly handle three-dimensional meshes that include more than one type of cell. To this end, we consider a mesh of the unit cube formed by tetrahedra and hexahedra, with pyramids for the transition between the two, that has been generated using `Gmsh`. The mesh was agglomerated using k-means with $k = 128$; the result is shown in Figure 10.



(a) View from above (b) Lateral view

Figure 11: Test Case 5: Original mesh of the whole brain coming from Magnetic Resonance Imaging.

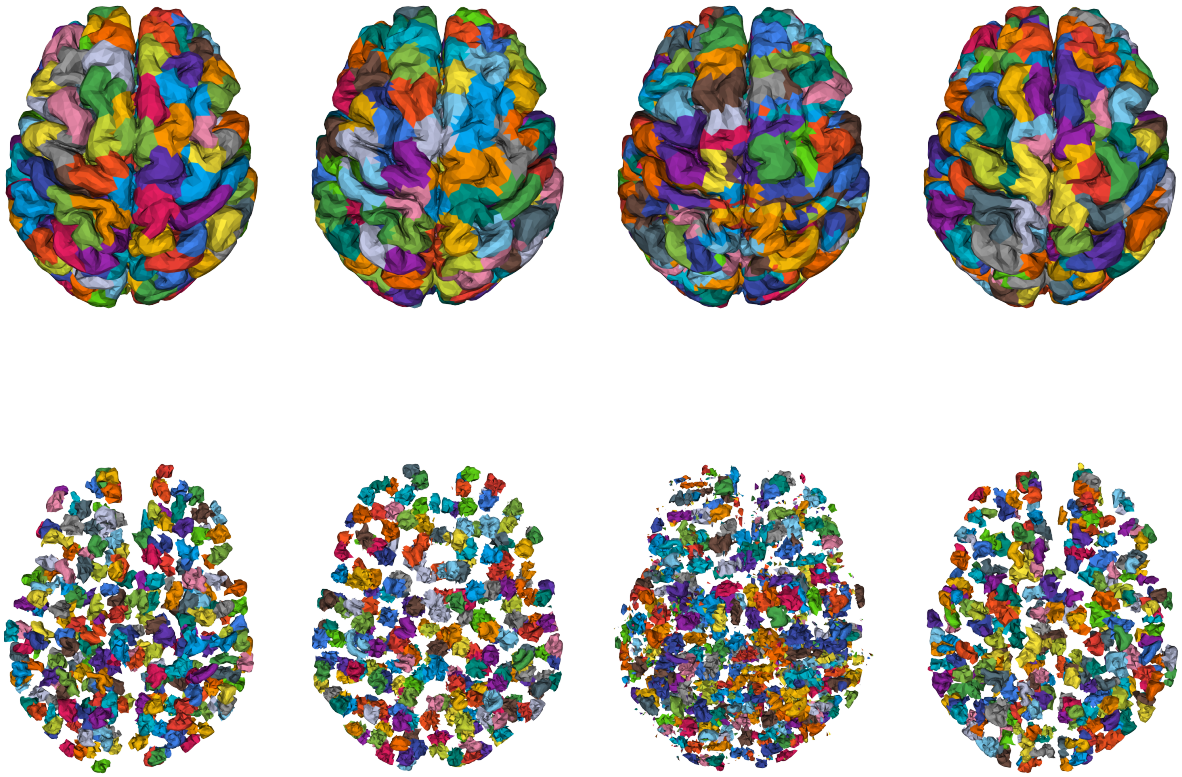
4.6 Test Case 5: Three-Dimensional Brain Geometry

To demonstrate the generalization capabilities of the models in the three-dimensional case, we consider once again a mesh coming from an MRI scan of a human brain, shown in Figure 11. We agglomerate the mesh using four different models (METIS, k-means, SAGE-Base, and SAGE-Base as coarse partitioner in a multilevel framework with RL Refiner). For the GNN models, we used *number of refinements* mode with `nref=8`, while for the other two *direct k-way* with `k=256` was used. In Figure 12 we report the agglomerated meshes and their exploded view, together with their quality metrics box plots. Most of the same considerations of the two-dimensional case hold, with some differences: due to the highly corrugated surface of the brain, k-means and SAGE-Base struggle to create fully connected elements, often leaving a few tetrahedra separated from the rest. As a consequence, the corresponding elements will be much smaller, leading to a lower uniformity factor. This issue can be alleviated in the k-means case by properly choosing a higher k, while not much can be done for SAGE-Base since the problem arises from the very first bisection. On the other hand, the multilevel approach with the RL refinement mostly solves the issue, improving the quality of the partition.

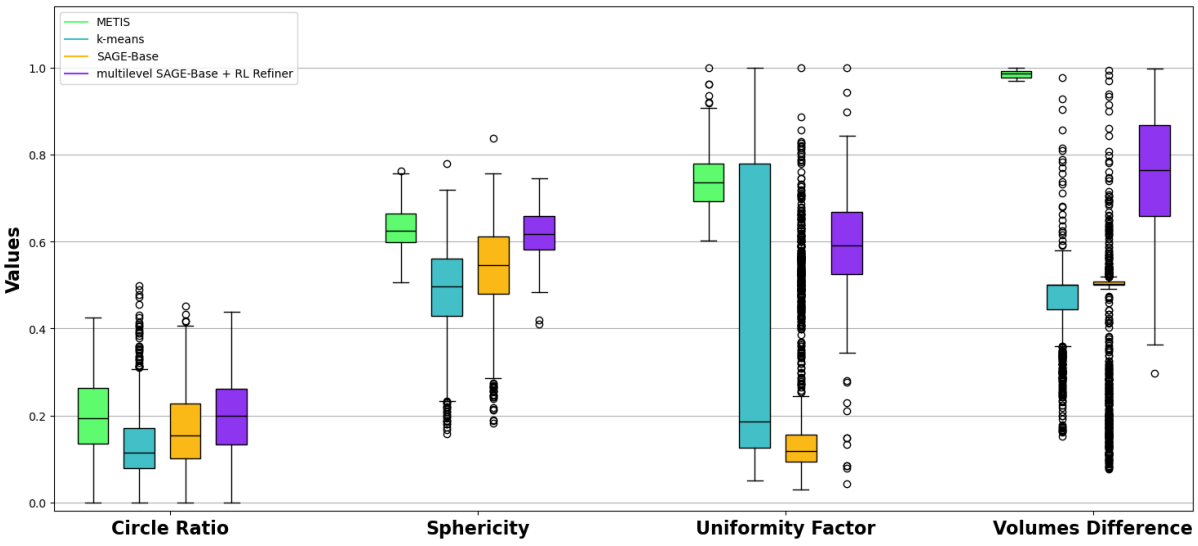
4.7 Test Case 6: Three-Dimensional Statue of Garuda and Vishnu

In this test case, we use a scan of a wooden statue of Garuda and Vishnu, which has very fine geometry and many holes. Thus, it is particularly challenging due to its topology. The tetrahedral mesh has been generated using `Gmsh` starting from an STL file coming from a 3D scan of the object (the original STL of Garuda and Vishnu, which is shown in Figure 13, is by Artec3D). We agglomerate it using SAGE-Base in a multilevel framework with RL Refiner and `nref=9`, while *direct k-way* with `k=512` was used for METIS and k-means; the results are reported in Figure 14. Listing 3 shows the code for the whole agglomeration pipeline of this example when using the multilevel approach.

We remark that in this case, METIS fails to partition the graph when passing cell volumes as node weight, so the agglomerated mesh has been created using unitary weights instead. Consequently, the volume uniformity for METIS is much lower compared to previous cases. Other than that, the three methods are fairly comparable across all four metrics.



(e) METIS (f) k-means (g) SAGE-Base (h) SAGE-Base + RL Refiner



(i) Quality metrics box plots

Figure 12: Test Case 5: an unstructured mesh of a human brain consisting of 123383 tetrahedra, agglomerated using different methods (METIS, k-means, SAGE-Base, SAGE-Base in multilevel framework with RL Refiner) and their exploded view, together with the box plots of the computed quality metrics (CR, Ψ, UF and $\widetilde{\text{VD}}$, defined as in Eq. (2)-(7)).

(a) Frontal view (b) Lateral view

Figure 13: Test Case 6: Original mesh of the statue of Garuda and Vishnu coming from a 3D scan.

```
from magnet import io, generate, aggmodels
# Generate tetrahedral mesh from STL file using Gmsh:
generate.tetrahedra_from_stl('Garuda_Vishnu.stl', remesh=False)
GV = io.load_mesh('Garuda_Vishnu.vtk')
sage = aggmodels.SageBase(128,64,4,2).to(aggmodels.DEVICE)  # initialize GNN partitioner
sage.load_model('models/sage_base.pt')
lrefiner = aggmodels.RLRefiner(5, 10).to(aggmodels.DEVICE)  # initialize GNN refiner
lrefiner.load_model('models/rl_refiner.pt')
agglomerated_GV = sage.agglomerate(GV, mode='multilevel', nref=9, refiner=lrefiner, threshold=500)
io.exploded_view(agglomerated_GV, scale=0.75, orientation=(30, 50, 0))
```

Listing 3: Example code showing the whole agglomeration pipeline, from the generation of the mesh from the STL file to the creation of the exploded view plot.

5 Integration with the `lymph` Library

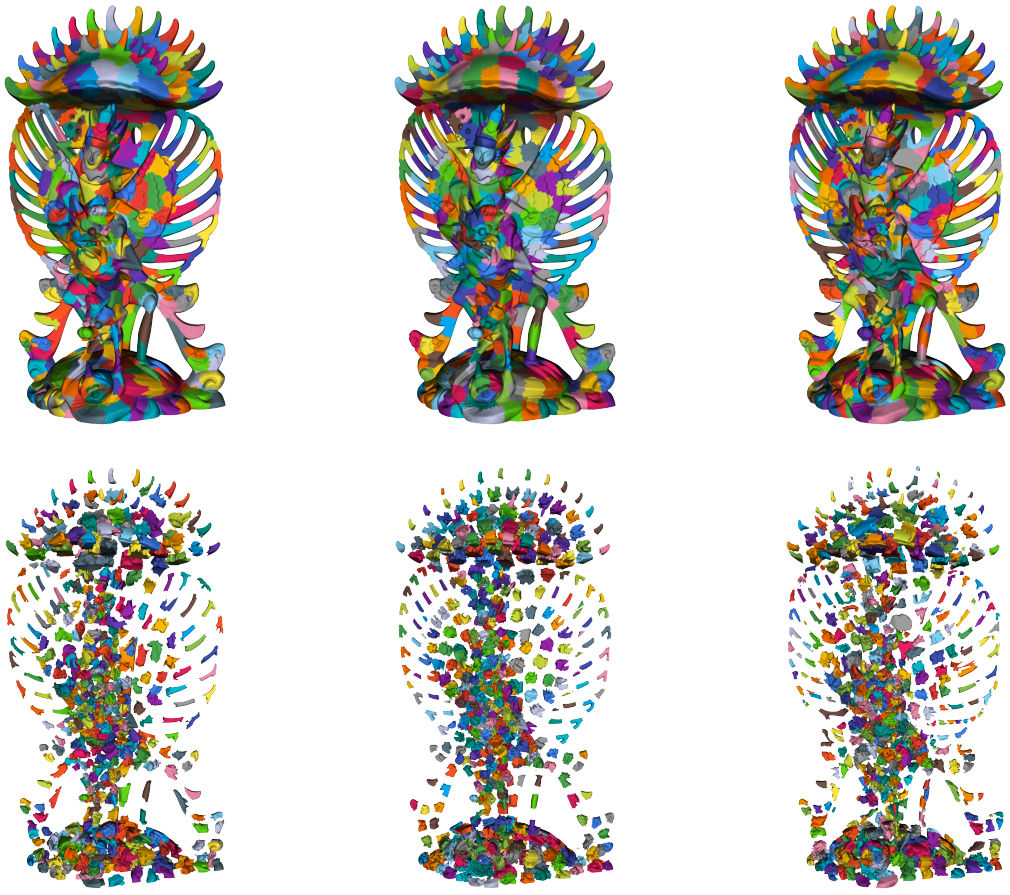
`lymph` is an open-source library for the discretization of multiphysics partial differential equations based on discontinuous Galerkin methods on polytopal grids [27]. `MAGNET` comes with `Agglomerate.m`, which is a function that allows to agglomerate a mesh by calling the `agglomerate.py` script and convert it to `lymph` format directly from Matlab, making it possible to easily solve differential problems on it. In this section, we give an example usage of this feature, showing that agglomerated meshes created by `MAGNET` are suitable for PolyDG discretization.

5.1 Verification Test 1: Poisson Problem on a Brain Slice

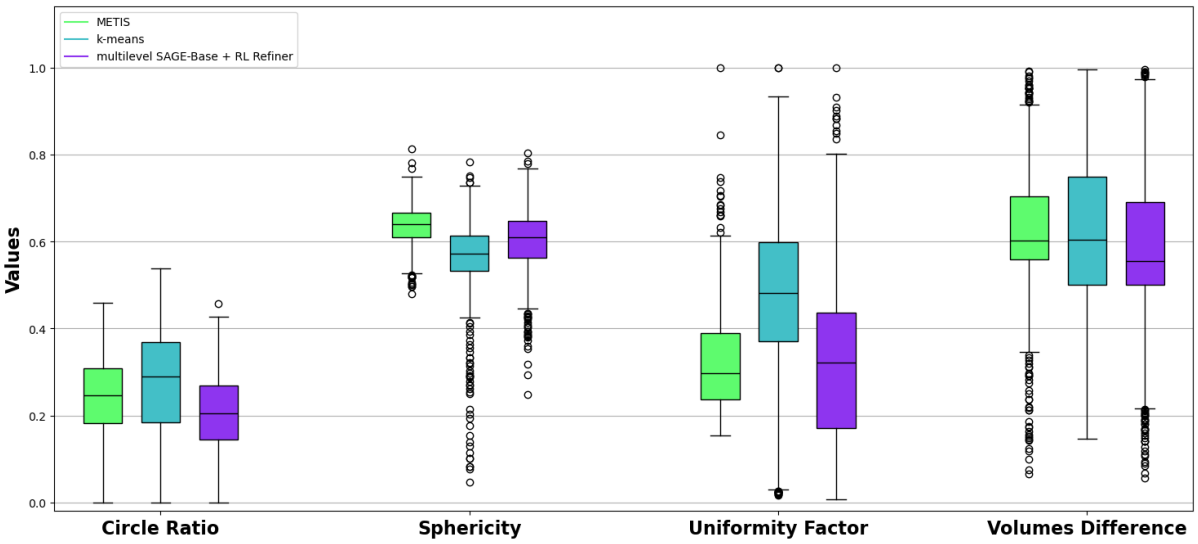
We first provide a brief guide on how to use the `lymph` API by considering the solution of a Poisson problem in an open and bounded domain $\Omega \subset (0,1)^2$, cf. Figure 15,

$$\begin{cases} -\Delta u = f(\mathbf{x}) & \text{in } \Omega, \\ u(\mathbf{x}) = g(\mathbf{x}) & \text{on } \partial\Omega. \end{cases} \tag{8}$$

We discretize problem (8) with a PolyDG method as shown in [27]. For simplicity, we consider as analytical solution $u(\mathbf{x}) = \sin(2\pi\mathbf{x}_1)\cos(2\pi\mathbf{x}_2)$ and compute the right-hand side f and boundary condition g accordingly. We solve this problem on the brain slice we used in Section 4.2 rescaled



(d) METIS (e) k-means (f) SAGE-Base + RL Refiner



(g) Quality metrics box plots

Figure 14: Test Case 6: a mesh of a statue of Garuda and Vishnu consisting of 615229 tetrahedra, agglomerated using different methods (METIS, k-means, SAGE-Base in multilevel framework with RL Refiner) and their exploded view, together with the box plots of the computed quality metrics (CR, Ψ, UF and $\widetilde{\text{VD}}$, defined as in (2)-(7)).

to fit in $(0,1)^2$. The mesh is agglomerated from Matlab by calling the `Agglomerate.m` function as reported in Listing 4.

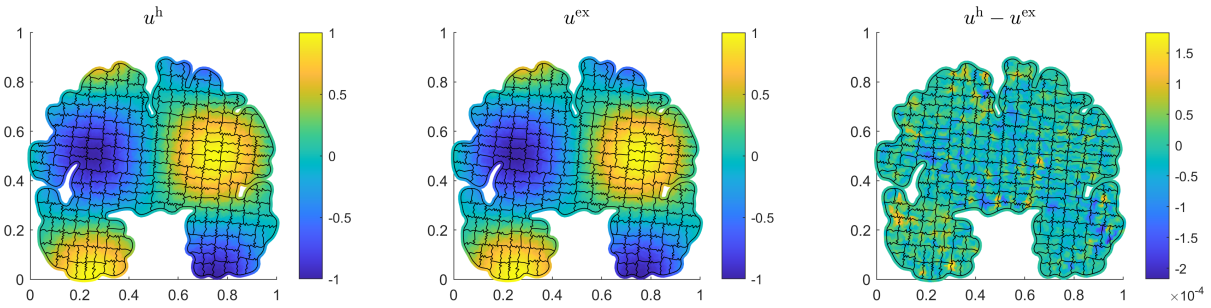


Figure 15: Verification Test 1: PolyDG solution of problem (8) computed on a grid of 256 agglomerated elements (*left*), plot of the exact solution (*center*) and computed numerical error (*right*).

```
run("lymph/Physics/Laplacian/InputData/LapAggTest.m")  % creates problem
Data structure
SimType = 'laplacian';
mesh_path = 'mesh.vtu';  % Input mesh to be agglomerated
output_path = fullfile('AggMeshTest.mat'); % Path where the mesh will be
saved
% agglomeration parameters
model = 'SageBase2D';  % Agglomeration model
mode = 'Nref';  % Agglomeration mode
param = 7;
Agglomerate(mesh_path, output_path, ...
Data, SimType, model, mode, param);

```

Listing 4: Example on how to use `MAGNET`'s `lymph` API.

`Data` and `SimType` are needed to correctly embed the boundary conditions into the FEM region data structure. The full list of accepted parameters can be found at the documentation web page; also, we redirect to it for details on the installation of a Python distribution compatible with Matlab. Once the mesh has been agglomerated, the problem can be solved by running the main function while indicating the appropriate mesh to use in the `Data` structure. In this case, we agglomerate the mesh using the SAGE-Base model with `nref=7` and solve the problem using polynomials of degree $\ell = 3$. The results are reported in Figure 15. To ensure that the agglomerated elements created by `MAGNET` are regular enough to achieve the theoretical order of convergence, we solve problem (8) with the same data in the unit square $\Omega = (0, 1)^2$, using six progressively finer meshes. Namely, we agglomerate the same initial mesh of 23264 elements with an increasing number of recursive bisections using the SAGE-Base model in *number of refinements* mode with N_{ref} going from five to ten. The polynomial degree for the discretization is fixed to $\ell = 3$. As seen in Figure 16, we observe the theoretical order of convergence of $\ell + 1$ for the error in L^2-norm and of ℓ for the dG-norm error [13]. However, we notice two facts: first, the empirical order of convergence is slightly higher than the theoretical one (4.40 and 3.34 versus 4 and 3, respectively). This is probably because in our setup, coarser meshes also have a larger number of edges per element, which is a source of irregularity and numerical errors that become less significant when we move to the finer ones. Second, the L^2 error seems to have an oscillatory behavior. Our suggested explanation is that in two-dimensional recursive bisection, if N_{ref} is even, then the agglomerated elements will tend to be square-shaped, while if N_{ref} is odd, the elements will be rectangular, with one side roughly double the other. The lower element quality for N_{ref} odd corresponds to a comparatively higher error for the same h, as is reflected by the graph. To confirm these suspicions, we perform a second convergence test, this time choosing six meshes with 500 to 16000 elements and agglomerating them with k-means in *direct k-way* mode, selecting k to be a tenth of the original number of elements. In this way, the shape, number of edges, and overall quality of the elements will be roughly the same across all meshes. Indeed, we observe that in this case, both errors are consistently lower than in the first

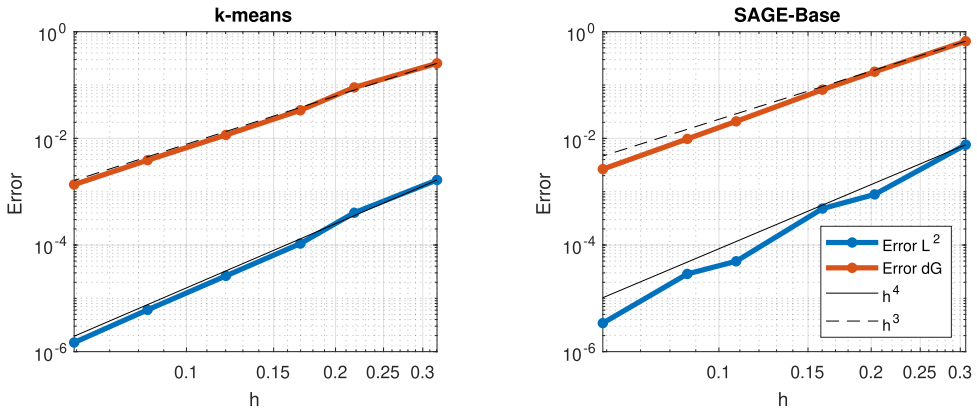


Figure 16: Verification Test 1: Computed errors $||u_h - u_{ex}||_{L^2(\Omega)}$ and $||u_h - u_{ex}||_{dG}$ as a function of the mesh size h by fixing the polynomial degree $\ell = 3$. On the left, the test is performed with k-means by agglomerating different meshes so that the ratio of original elements to agglomerated elements is the same; on the right, the test is performed by agglomerating the same mesh with SAGE-Base using an increasing number of recursive bisections.

test, and the empirical orders of convergence are much closer to the theoretical ones (4.13 and 3.08).

5.2 Verification Test 2: Heat Equation with Discontinuous Boundary Conditions

To assess the performance of differently agglomerated meshes in a PolyDG framework, we now consider the heat equation:

$$\begin{cases} \partial_t u - \nabla \cdot (\mu \nabla u) = 0 & \text{in } \Omega \times (0, T], \\ u(\mathbf{x}, t) = g(\mathbf{x}, t) & \text{on } \partial\Omega \times (0, T], \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}) & \text{on } \Omega \times \{t = 0\}. \end{cases} \tag{9}$$

We solve (9) through `lymph`, by combining a PolyDG method with a Crank-Nicolson time integration scheme, [27]. We consider $\Omega = B_{0.6}(0.5, 0) \cup B_{0.6}(-0.5, 0)$, i.e., the combination of two balls that intersect. We take $\mu = 0.1$, final time $T = 1$ and discontinuous Dirichlet boundary conditions $u(\mathbf{x}) = 1$ for $\mathbf{x}_1 \geq 0$, $u(\mathbf{x}) = 0$ for $\mathbf{x}_1 < 0$, with analogous discontinuous initial condition. We solve the problem on three meshes agglomerated by METIS, k-means, and SAGE-Base, respectively, starting from a very fine mesh of 35158 elements, using either *k-way* or *number of refinements* mode so that the resulting mesh has 256 elements. Additionally, we consider for comparison a polygonal mesh with the same number of elements generated using the software Polymesher [52], which tends to create very regular hexagonal cells. For each mesh, the problem is discretized using 3 different polynomial degrees $\ell = 1, 3, 5$ and time step $\Delta t = 10^{-3}$. The computed PolyDG solutions at the final time $T = 1$ are reported in Figure 17. We observe that all four meshes produce very similar results when considering the same polynomial degree, although the agglomerated meshes have a significantly higher number of edges per element compared to the Polymesher one. The numerical solutions are overall coherent with what is expected from the literature [53]. For higher polynomial degrees, some numerical oscillations appear in proximity to the boundary conditions discontinuity; this might be due to agglomerated elements not perfectly conforming to the two different sections of the border. Indeed, if a single element shares an edge with both sections, there will be great numerical oscillations because a continuous polynomial would need to approximate a discontinuous function. Unfortunately, `MAGNET` has no way to guarantee this does not happen since the boundary is not taken into account when agglomerating.

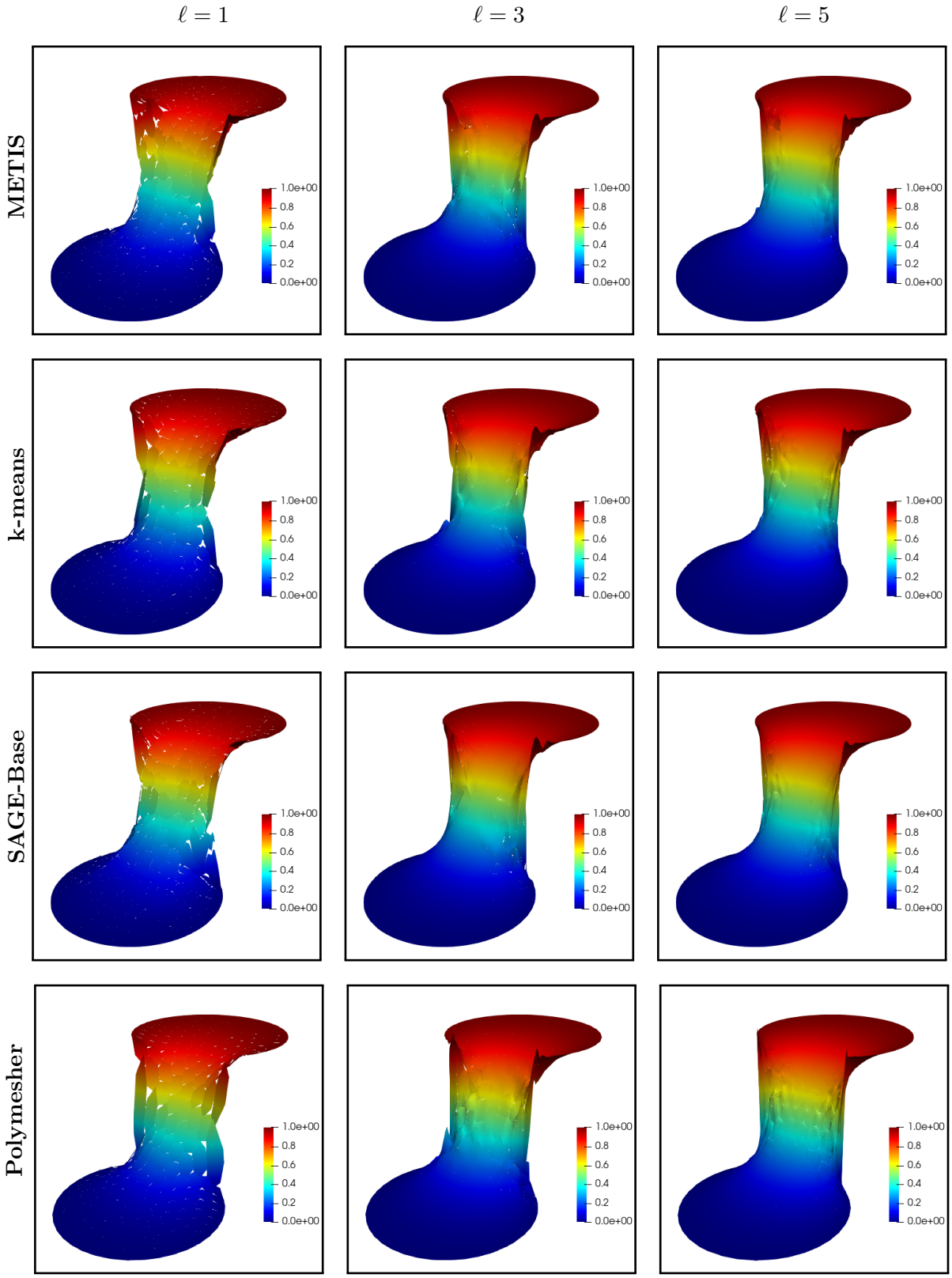


Figure 17: Verification Test 2: PolyDG solution of problem (9) on three meshes agglomerated by METIS, k-means, and SAGE-Base respectively, with the addition of a polygonal mesh generated with Polymesher, discretized with polynomial degrees $\ell = 1, 3, 5$

6 Conclusions

We have presented `MAGNET`, an open-source Python library for two and three-dimensional polytopal mesh agglomeration by Graph Neural Networks, and illustrated its core structure and main functionalities. Thanks to its flexibility, extensibility, straightforward integration with other software, and simple interface, we believe that `MAGNET` can be a useful tool to explore new ML approaches to mesh agglomeration. We have introduced the different GNN methods that are available within it. Namely, the SAGE-Base model can exploit the geometric information of the mesh together with its connectivity and the reinforcement learning partitioner and refiner models, which offer enhanced performance in applications to multilevel frameworks. We tested these agglomeration strategies against state-of-the-art methods like METIS and k-means (also readily available in `MAGNET`), showing that they produce agglomerated meshes of comparable quality. Finally, thanks to our flexible API, we employed `MAGNET` in conjunction with the Matlab library `lymph` for polytopal discontinuous Galerkin approximation. We proved that the agglomerate meshes we produce are suitable for discretizing PDEs. We have shown that the explored machine learning-based techniques can match state-of-the-art methods in terms of performance. Moreover, we are confident that the current implementation is still improvable. By exploiting the affinity of ML techniques for modern GPU architectures, we can achieve better speed and scalability and more easily incorporate the geometric and physical features of the problem at hand. Future developments include integrating the recursive bisection algorithm with multigrid methods by generating a hierarchy of nested coarser grids. Including edge and face coarsening routines might also be essential to avoid having agglomerated elements with many edges, which are associated with higher computational costs. Regarding the GNN models, the training datasets could be extended to improve their generalization capabilities, and different model architectures should be experimented with. In particular, bigger RL partitioner and refiner models should be trained to perform better in the three-dimensional case. Also, the RL Refiner model could be improved by further exploiting the geometric information. Finally, we mention that different combinations of these approaches could be possible to experiment with, e.g., using SageBase as a partitioner and a reinforcement learning model for refinement.

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

Acknowledgements. We thank Mattia Corti for segmenting and providing the meshes of the brain. The brain MRI images were provided by OASIS-3: Longitudinal Multimodal Neuroimaging: Principal Investigators: T. Benzinger, D. Marcus, J. Morris; NIH P30 AG066444, P50 AG00561, P30 NS09857781, P01 AG026276, P01 AG003991, R01 AG043434, UL1 TR000448, R01 EB009352. This work received funding from the European Union (ERC SyG, NEMESIS, project number 101115663). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. PFA, MC and IM are members of INdAM-GNCS. The present research is part of the activities of "Dipartimento di Eccellenza 2023-2027", MUR, Italy.

Code and data availability. The code and data used in this work are available can be accessed at https://github.com/lymphlib/magnet. Proper attribution should be given when reusing or distributing the materials.

References

[1] P. F. Antonietti and E. Manuzzi. Refinement of polygonal grids using convolutional neural networks with applications to polygonal discontinuous Galerkin and virtual element methods. *Journal of Computational Physics*, 452:110900, 2022.

[2] P. F. Antonietti, M. Corti, and G. Martinelli. Polytopal mesh agglomeration via geometrical deep learning for three-dimensional heterogeneous domains. *arXiv:2406.10587*, 2024.

[3] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. D. Marini, and A. Russo. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, 23(01):199–214, 2013.

[4] L. Beirão da Veiga, F. Brezzi, L. Marini, and A. Russo. Virtual element method for general second-order elliptic problems on polygonal meshes. *Mathematical Models and Methods in Applied Sciences*, 26(04): 729–750, 2016.

[5] L. Beirão da Veiga, F. Brezzi, F. Dassi, L. Marini, and A. Russo. A family of three-dimensional virtual elements with applications to magnetostatics. *SIAM Journal on Numerical Analysis*, 56:2940–2962, 01 2018.

[6] J. Tushar, A. Kumar, and S. Kumar. Virtual element methods for general linear elliptic interface problems on polygonal meshes with small edges. *Computers & Mathematics with Applications*, 122:61–75, 08 2022.

[7] F. Brezzi, K. Lipnikov, and V. Simoncini. A family of mimetic finite difference methods on polygonal and polyhedral meshes. *Mathematical Models and Methods in Applied Sciences*, 15(10):1533–1551, 2005.

[8] F. Brezzi, K. Lipnikov, and M. Shashkov. Convergence of the mimetic finite difference method for diffusion problems on polyhedral meshes. *SIAM Journal on Numerical Analysis*, 43:1872–1896, 01 2005.

[9] L. Beirão da Veiga, K. Lipnikov, and G. Manzini. *The mimetic finite difference method for elliptic problems*, volume 11. Springer, 2014.

[10] F. Bassi, L. Botti, A. Colombo, D. Di Pietro, and P. Tesini. On the flexibility of agglomeration based physical space discontinuous Galerkin discretizations. *Journal of Computational Physics*, 231, 01 2012.

[11] P. F. Antonietti, S. Giani, and P. Houston. *hp*-Version composite discontinuous Galerkin methods for elliptic problems on complicated domains. *SIAM Journal on Scientific Computing*, 35(3):A1417–A1439, 2013.

[12] P. F. Antonietti, C. Facciolà, P. Houston, I. Mazzieri, G. Pennesi, and M. Verani. *High-order discontinuous Galerkin methods on polyhedral grids for geophysical applications: seismic wave propagation and fractured reservoir simulations*, pages 159–225. Springer, 06 2021.

[13] A. Cangiani, Z. Dong, E. Georgoulis, and P. Houston. *hp-Version Discontinuous Galerkin Methods on Polygonal and Polyhedral Meshes*. SpringerBriefs in Mathematics, 2017.

[14] D. A. D. Pietro and A. Ern. Hybrid high-order methods for variable-diffusion problems on general meshes. *Comptes Rendus Mathematique*, 353(1):31–34, 2015.

[15] D. A. Di Pietro, A. Ern, and S. Lemaire. *A review of hybrid high-order methods: formulations, computational aspects, comparison with other methods*, pages 205–236. Springer International Publishing, Cham, 2016.

[16] D. A. Di Pietro and J. Droniou. The hybrid high-order method for polytopal meshes. *Modeling, Simulation and Application*, 84, 2020.

[17] B. Cockburn, B. Dong, and J. Guzmán. A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems. *Mathematics of Computation*, 77(264):1887–1916, 2008.

[18] B. Cockburn, J. Gopalakrishnan, and R. Lazarov. Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems. *SIAM Journal on Numerical Analysis*, 47:1319–1365, 08 2009.

[19] B. Cockburn, J. Gopalakrishnan, and F.-J. Sayas. A projection-based error analysis of HDG methods. *Mathematics of Computation*, 79:1351–1367, 07 2010.

[20] P. F. Antonietti and G. Pennesi. V-cycle multigrid algorithms for discontinuous Galerkin methods on non-nested polytopic meshes. *Journal of Scientific Computing*, 78(1):625–652, 2019.

[21] S. Dargaville, A. G. Buchan, R. P. Smedley-Stevenson, P. N. Smith, and C. C. Pain. A comparison of element agglomeration algorithms for unstructured geometric multigrid. *Journal of Computational and Applied Mathematics*, 390:113379, 2021.

[22] M. Feder, A. Cangiani, and L. Heltai. R3MG: R-tree based agglomeration of polytopal grids with applications to multilevel methods. *Journal of Computational Physics*, 526:113773, 2025.

[23] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[24] C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6-8):318–331, July 2008.

[25] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[26] S. T. Barnard and H. D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and experience*, 6(2):101–117, 1994.

[27] P. F. Antonietti, S. Bonetti, M. Botti, M. Corti, I. Fumagalli, and I. Mazzieri. Lymph: discontinuous poLYtopal methods for Multi-PHysics differential problems. *ACM Transactions on Mathematical Software*, 2025.

[28] P. F. Antonietti, N. Farenga, E. Manuzzi, G. Martinelli, and L. Saverio. Agglomeration of polygonal grids using graph neural networks with applications to multigrid solvers. *Computers & Mathematics with Applications*, 154:45–57, 2024.

[29] A. Gatti, Z. Hu, T. Smidt, E. G. Ng, and P. Ghysels. Graph partitioning and sparse matrix ordering using reinforcement learning and graph neural networks. *Journal of Machine Learning Research*, 23:13675–13702, 2022.

[30] M. R. Garey and D. S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. W. H. Freeman & Co., USA, 1990.

[31] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[32] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[33] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai. Towards efficient large-scale graph neural network computing. *arXiv:1810.08403*, 2018.

[34] A. Gatti, Z. Hu, T. Smidt, E. G. Ng, and P. Ghysels. Deep learning and spectral embedding for graph partitioning. In *Proceedings of the 2022 SIAM conference on parallel processing for scientific computing*, pages 25–36. SIAM, 2022.

[35] A. Nazi, W. Hang, A. Goldie, S. Ravi, and A. Mirhoseini. Gap: Generalizable approximate graph partitioning framework. *arXiv:1903.00614*, 2019.

[36] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[37] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[38] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. MIT Press, 2nd edition, 2018.

[39] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PmLR, 2016.

[40] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[41] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pages 3835–3845. PMLR, 2019.

[42] J. Macqueen. Some methods for classification and analysis of multivariate observations. *Berkeley Symposium on Mathematical Statistics and Probability*, 5:281–297, 1967.

[43] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *arXiv:1903.02428*, 2019.

[44] N. Schlömer. meshio: Tools for mesh files, 2024. URL `https://zenodo.org/doi/10.5281/zenodo.1173115`.

[45] W. Schroeder, K. Martin, and B. Lorensen. *The visualization toolkit (4th ed.)*. Kitware, 2006.

[46] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering*, 79(11):1309–1331, 2009.

[47] D. J. Pearce. An improved algorithm for finding the strongly connected components of a directed graph. *Victoria University, Wellington, NZ, Tech. Rep*, 2005.

[48] J. R. Shewchuk. What is a good linear element? Interpolation, conditioning, and quality measures. *Proceedings of the 11th International Meshing Roundtable. Sandia National Laboratories*, 2002.

[49] T. Sorgente, S. Biasotti, G. Manzini, and M. Spagnuolo. Polyhedral mesh quality indicator for the virtual element method. *Computers & Mathematics with Applications*, 114:151–160, 2022.

[50] J. Zunic and P. Rosin. A new convexity measure for polygons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7):923–934, 2004.

[51] D. A. Di Pietro and J. Droniou. Benchmark of polygon quality metrics for polytopal element methods. *Modeling, Simulation and Application*, 84, 2020.

[52] C. Talischi, G. H. Paulino, A. Pereira, and I. F. M. Menezes. PolyMesher: A general-purpose mesh generator for polygonal elements written in matlab. *Structural and Multidisciplinary Optimization*, 45(3):309–328, 2012.

[53] A. Quarteroni. *Numerical models for differential problems*. Springer, 3rd edition, 2017.

MOX Technical Reports, last issues

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

22/2025 Leimer Saglio, C. B.; Pagani, S.; Antonietti P. F.
A p-adaptive polytopal discontinuous Galerkin method for simulating brain electrophysiology

21/2025 Caldera, L., Masci, C., Cappozzo, A., Forlani, M., Antonelli, B., Leoni, O., Ieva, F.
Uncovering mortality patterns and hospital effects in COVID-19 heart failure patients: a novel Multilevel logistic cluster-weighted modeling approach

20/2025 Botti, M.; Prada, D.; Scotti, A.; Visinoni, M.
Fully-Mixed Virtual Element Method for the Biot Problem

19/2025 Bortolotti, T.; Wang, Y. X. R.; Tong, X.; Menafoglio, A.; Vantini, S.; Sesia, M.
Noise-Adaptive Conformal Classification with Marginal Coverage

Bortolotti, T.; Wang, Y. X. R.; Tong X.; Menafoglio, A.; Vantini, S.; Sesia, M.
Noise-Adaptive Conformal Classification with Marginal Coverage

18/2025 Antonietti, P.F.; Corti, M.; Gómez, S.; Perugia, I.
A structure-preserving LDG discretization of the Fisher-Kolmogorov equation for modeling neurodegenerative diseases

17/2025 Botti, M.; Mascotto, L.
Sobolev--Poincaré inequalities for piecewise $W^{1,p}$ functions over general polytopic meshes

16/2025 Radisic, I.; Regazzoni, F.; Bucelli, M.; Pagani, S.; Dede', L.; Quarteroni, A.
Influence of cellular mechano-calcium feedback in numerical models of cardiac electromechanics

15/2025 Fois, M.; de Falco, C.; Formaggia L.
Efficient particle generation for depth-averaged and fully 3D MPM using TIFF image data

14/2025 Nicolussi, F.; Masci, C.
Stratified Multilevel Graphical Models: Examining Gender Dynamics in Education