C. Bermejo et al. *Advanced Modeling and Simulation in Engineering Sciences(2025)12:27*
https://doi.org/10.1186/s40323-025-00311-8

Advanced Modeling and Simulation in Engineering Sciences

RESEARCH Open Access

Meshgraphnets informed locally by thermodynamics for the simulation of flows around arbitrarily shaped objects

Carlos Bermejo[1], Alberto Badías[2], David González[1] and Elías Cueto[1*]

*Correspondence: ecueto@unizar.es
[1]ESI Group-UZ Chair of the Spanish National Strategy on AI, Aragon Institute of Engineering Research, Universidad de Zaragoza, Zaragoza 50018, Spain
ETSIAE, Universidad Politécnica de Madrid, Madrid 28040, Spain

Abstract

We present a thermodynamics-informed graph neural network framework for learning the time evolution of complex physical systems, incorporating thermodynamic structure via a nodal port-metriplectic formulation. Built upon the MeshGraphNet architecture, our method replaces the standard decoder with multiple specialized decoders that predict local energy and entropy gradients, along with Poisson and dissipative operators. These components are assembled at each graph node according to the GENERIC formalism, enforcing the first and second laws of thermodynamics. The framework is evaluated on two examples involving incompressible fluid flow past obstacles: one with varying cylindrical obstacles and another with obstacles of different types, not seen during training. The proposed model shows accurate long-term predictions, robust generalization to unseen geometries, and substantial speedups compared to traditional numerical solvers.

Keywords: Deep learning, Graph Neural Networks, Thermodynamics, GENERIC, Port-metriplectic structure

Introduction

Simulating physical systems has long been a central goal in science and engineering. From modelling fluid flows and viscoelastic behaviours to simulating interactions between systems, traditional numerical methods such as the Finite Element Method (FEM) and Computational Fluid Dynamics (CFD), based on the discretization of partial differential equations (PDEs), have provided a robust framework for understanding complex dynamics. However, as systems become more intricate, with complex irregular geometries and fast-evolving phenomena, these traditional solvers often become computationally expensive. This limits their applicability in large-scale simulations, and makes them unsuitable for real-time tasks, such as predictive digital twins [1–3].

In recent years, we have seen the rise of data-driven methods and machine learning, which has opened new possibilities for accelerating or even replacing conventional solvers [4–7]. By learning from simulation or experimental data, surrogate models aim to reduce the cost of numerical integration, enabling faster predictions while maintaining acceptable levels of accuracy [8–10]. Regarding these methods, deep neural networks have shown

strong capabilities to model nonlinear dynamics. However, these approaches, usually referred as black boxes, often fall short when modelling physical systems, as they lack in interpretability, exhibit poor extrapolation behaviour, fail to perform stable long-term predictions, and typically violate fundamental physical laws.

To mitigate these shortcomings, physics-informed deep learning has emerged, embedding prior physical knowledge into the learning procedure [11,12]. Several strategies have been proposed, with Physics-Informed Neural Networks (PINNs) [13] being one of the most well-known. PINNs incorporate the PDEs that govern the system into the learning process, including a residual term in the loss function that enforces the satisfaction of the PDEs. While these method offer increased interpretability and more stable long-term predictions, they assume known governing equations and fixed initial and boundary conditions, limiting their applicability in scenarios where such equations are unknown or cannot be easily applied.

To address these limitations, alternative approaches infer time evolution from data, enabling to incorporate known physical properties into the learning process. For instance, Hamiltonian [14–17] and Lagrangian [18–20] neural networks enforce conservative mechanics laws. However, these approaches do not account for the dissipative nature of most dynamical systems. The GENERIC formalism extends to dissipative systems by incorporating a metriplectic structure [21,22]. Recent works have shown the strength of GENERIC for modelling evolution of complex systems [1,23–27], leading the development of Thermodynamics-Informed Neural Networks (TINNS) [28]. TINNs incorporate energy and entropy constraints into learning, leading to the fulfillment of the first and second laws of thermodynamics, achieving a thermodynamically compliant framework.

In parallel, the advances in geometric deep learning and, in particular, Graph Neural Networks (GNNs), have introduced new possibilities for modelling physical systems over unstructured domains [29–32]. This allows to accurately represent complex geometries, an important advantage over Convolutional Neural Networks (CNNs), which require a grid-shaped domain [33,34]. By representing the system as a graph, GNNs can efficiently capture local interactions and generalize across different topologies. Approaches like MeshGraphNets, based on the message passing procedure, have shown strong performance in various physics-based tasks such as fluid mechanics and solid mechanics [35]. However, generally these approaches remain agnostic to the underlying physics of the system and do not explicitly enforce any physical constraints to the solution.

Recent advances have explored the gap between geometric and physics-informed deep learning by incorporating inductive biases into graph-based models [3,36]. These approaches have demonstrated strong performance for both fluid and solid mechanics, as well as in systems involving multi-body interactions [37].

The aim of this work is to further explore the generalization capabilities of thermodynamics-informed graph-based models for predicting the time evolution of physical systems. In particular, the problem of shape parametrisation is one of the most important and most difficult in the field of computational mechanics. To be able to develop a methodology that allows to obtain real-time simulations on arbitrary geometries is of utmost interest. Scientific machine learning techniques based on the use of graph networks offer unprecedented opportunities in this respect. The fact that such networks contain a network at each of the nodes of the mesh (and possibly at each of the edges) offers very important advantages. The position of the nodes can be varied without affecting the accuracy of the

predictions, thanks to the local character of the nodal networks. This offers the possibility of using graph networks in the same way as finite element or finite difference meshes, for example.

We propose a methodology that integrates a port-metriplectic structure with a graph-based learning framework, enabling physically-consistent predictions that comply with the first and second laws of thermodynamics. While the formulation is general and can be applied to a wide range of dynamical systems, we have used as a model problem the flow of an incompressible fluid around objects of arbitrary shapes and positions. By encoding local energy and entropy gradients along with conservative and dissipative dynamics at the node level, the method offers a scalable approach to model complex physical behaviours.

The structure of the paper is as follows. A description of the problem setup is presented in "Problem statement" section. The methodology is presented in "Methodology" section, where both the geometric and metriplectic biases are described. "Results" section draws the numerical results, including performance comparisons. Finally, the conclusions of the paper are discussed at "Conclusions" section.

Problem statement

In this work, we propose a framework to estimate the temporal evolution of a system governed by a set of independent state variables $\boldsymbol{z} \in \mathcal{M} \subseteq \mathbb{R}^D$, where $\mathcal{M}$ denotes the state space of these variables, assumed to have the structure of a differentiable manifold in $\mathbb{R}^D$. The full-order model of a physical phenomenon can be expressed as a system of differential equations governing the temporal evolution of the state $\boldsymbol{z}$,

$$\dot{\boldsymbol{z}} = \frac{d\boldsymbol{z}}{dt} = F(\boldsymbol{z}, t), \; t \in \mathcal{I} = (0, T], \; \boldsymbol{z}(0) = \boldsymbol{z}_0, \tag{1}$$

where t refers to the time coordinate in the time interval $\mathcal{I}$ and $F(\boldsymbol{z}, t)$ is an arbitrary unknown nonlinear function representing the system dynamics.

The goal of this work is to provide accurate and efficient predictions of the time evolution of complex systems with previously unseen geometries, while incorporating some information about the underlying physics governing their dynamics. Specifically, we aim to ensure that the model respects physics at a fundamental level by enforcing the first and second principles of thermodynamics: energy conservation and non-negative entropy production.

To achieve this, as in previous works [36,38], we incorporate two biases into the model. First, a geometric bias is introduced via Graph Neural Networks, which encode the spatial structure and local interactions through a graph-based representation, adding a geometric prior to the prediction. Second, a thermodynamic bias is applied by enforcing the GENERIC formalism into the learned dynamics, ensuring that basics principles of thermodynamics are satisfied. This combination allows the framework to be expressive and generalizable, while respecting the physics inherent to the system.

Methodology

Geometric bias: Graph Neural Networks

As we have explained in the previous section, the geometric bias is applied to the network by the use of GNNs. We work with a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, ..., n\}$ is a set of $|\mathcal{V}| = n$ vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of $|\mathcal{E}| = e$ edges. Each vertex and

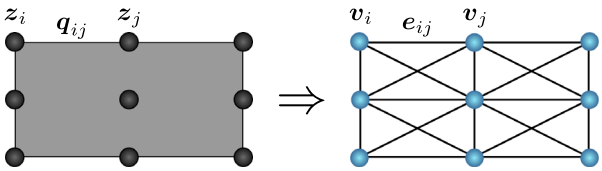


Fig. 1 Discretized domain of a physical system represented as a mesh, with node state variables $\boldsymbol{z}_i$ and relative nodal distances $\boldsymbol{q}_{ij}$ (left). The same system is represented as a graph (right), composed of a set nodes and edges, each associated with attributes: $\boldsymbol{v}_i$ and $\boldsymbol{e}_{ij}$, respectively

edge in the graph represents a node and a pairwise interaction in a discretized physical system. For each vertex $i \in \mathcal{V}$, we associate a feature vector $\boldsymbol{v}_i \in \mathbb{R}^{F_v}$, representing the physical properties of each node. Similarly, for each edge $(i,j) \in \mathcal{E}$, we associate an edge feature vector $\boldsymbol{e}_{i,j} \in \mathbb{R}^{F_e}$. As in previous approaches [35], when working with synthetic data coming from numerical simulations (finite element or finite volumes methods), we assume one-to-one correspondence between the vertices on the mesh and their connectivity and the nodes and edges of the graph.

In this setup, the positional variables of the system ($\boldsymbol{q}_i$) are assigned to the edge feature vector $\boldsymbol{e}_{i,j}$. This implies that the edge features represent relative distances between nodes $\left(\boldsymbol{q}_{ij} = \boldsymbol{q}_i - \boldsymbol{q}_j\right)$ along with their Euclidean norm, $\left\|\boldsymbol{q}_i - \boldsymbol{q}_j\right\|_2$, allowing to the graph neural network to learn distance-based interaction while maintaining translation and rotation invariance. The remaining state variables are assigned to the node feature vector $\boldsymbol{v}_i$. A simplified graph representation of a physical system is shown in Fig. 1, highlighting the node state variables and relative distances between nodes.

Mesh Graph Neural Networks

These features are processed using an encode-process-decode scheme [35]. This scheme consist on several multilayer perceptrons (MLPs) shared between all the nodes and edges of the graph. The algorithm used in the original MeshGraphNets (MGN) work consists of four steps (Fig. 2):

Encoder: The encoder transforms the vertex and edge features into higher dimensional embeddings. We use two MLPs: one for nodes (ε_v) and one for edges (ε_e). For each node, we encode its physical state variables along with a one-hot vector indicating the node type, enabling to define different boundary conditions. For the edges, we encode relative position vectors between connected nodes, as well as their Euclidean norm.

Processor: The processor is the core of the algorithm, as it performs the message passing, sharing nodal information between each vertex and its neighbours. It consist of M identical message passing blocks. Each block is conformed by two MLPs, the first updates edge features, while the later updates node features, and a permutation-invariant aggregation function.

First, for each edge (i,j), the MLP (π_e) updates the latent representation of the edge by taking into account the features of the sender and receiver nodes and the edge itself.

Then, for each node the messages are aggregated with a permutation invariant function ϕ base on the neighbourhood $\mathcal{N}_i = \left\{j \in \mathcal{V} \mid (i,j) \in \mathcal{E}\right\}$ of the node i. Finally, the node features are updated with a second MLP (π_v) which takes the current node features and the aggregated messages.

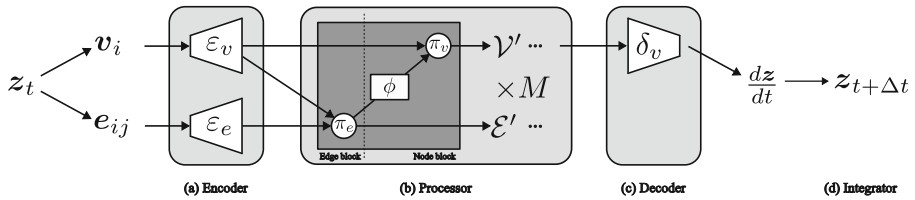


Fig. 2 Algorithm block scheme used in MeshGraphNets to predict a single-step variable change in time. **a** The encoder transforms the node and edge features to a latent representation. **b** The processor shares and updates edge and nodal information through the graph by applying M message passing blocks. **c** The decoder predicts the derivative of the input state variables. **d** The integrator obtains the next step state by using a forward-Euler integration scheme

By applying the processor block recurrently M times, the network increases the receptive field of each node, allowing it to better capture long distances dependencies in the graph. In this work, we use unshared parameters across the M processing blocks and incorporate residual connections to each message passing block, improving stability. As aggregation function ϕ we use summation over the incoming messages.

Decoder: In order to predict the system state at time $t + \Delta t$ from the state at time t, the decoder uses an MLP (δ_v) to transform the latent node features $\boldsymbol{v}_i$ into the output features $\boldsymbol{p}_i$.

Integrator: The last step of the algorithm implies updating the state of the system. The output features obtained by the decoder are interpreted as the derivatives of the state variables of our system, $\boldsymbol{p}_i \approx \frac{d\boldsymbol{z}}{dt}$. In order to obtain the next step of the system, we use a forward-Euler integration scheme, updating the system as: $\boldsymbol{z}_{t+\Delta t} = \boldsymbol{z}_t + \Delta t \cdot \frac{d\boldsymbol{z}}{dt}$

Inductive bias: metriplectic systems

One of our main goals is to develop a scheme that satisfies fundamental physical principles. This is a key aspect for our framework, as we aim to provide accurate and credible predictions, and when dealing with physical systems, this can only be achieved by fulfilling the basics of physics. To this end, we can introduce some physics knowledge to the network as an inductive bias [32]. Briefly, an inductive bias is a set of assumptions about the data that guide the learning process, prioritizing one solution over others.

Some of the most well-known methods for including physics into model predictions include the Physics-Informed Neural Networks (PINNs) [13]. These approaches enforce the structure of specific partial differential equations (PDEs) governing the system, leading to accurate solutions that are physics compliant. However, there are situations where the governing equations of the system are either partially known or difficult to apply in practice. In those cases, we need to find a formalism that enforces the physical consistency of the solution without relying on any particular governing equation.

In this sense, using thermodynamics as inductive bias makes sense, as we are enforcing the physical consistency of the prediction while using a very general framework that can be applied to a wide range of systems. Previous works in this direction include Thermodynamics-Informed Neural Networks (TINNs) [26,28,39] and Thermodynamics-Informed Graph Neural Networks (TIGNNs) [36]. These methods are built to satisfy known properties of the system, such as energy conservation or entropy dissipation, implying that they can be applied to both conservative and dissipative systems, while ensuring that principles of thermodynamics are fulfilled by construction.

GENERIC formalism

To ensure that the solution remains physically meaningful, we adopt the "General Equation for Non-Equilibrium Reversible–Irreversible Coupling" formalism, generally known as GENERIC [21,22]. This formalism extend the traditional Hamiltonian formulation to complex systems involving both conservative and dissipative dynamics. The reversible part of the system is governed by a Hamiltonian structure, defined by an energy functional and a Poisson bracket, while the irreversible contribution is driven by a non-equilibrium entropy functional and a dissipative or friction bracket [40,41].

The dynamic evolution for non-equilibrium systems, described by a set of state variables $\boldsymbol{z}$ is given by:

$$\frac{d\boldsymbol{z}}{dt} = \{\boldsymbol{z}, E\} + [\boldsymbol{z}, S]\,. \tag{2}$$

For computational purposes, it is convenient to reformulate the bracket notation using two linear operators:

$$\boldsymbol{L} : T^*\mathcal{M} \to T\mathcal{M}, \qquad \boldsymbol{M} : T^*\mathcal{M} \to T\mathcal{M}, \tag{3}$$

where $T^*\mathcal{M}$ and $T\mathcal{M}$ represent the cotangent and tangent bundles of the state manifold $\mathcal{M}$, respectively. The operator $\boldsymbol{L}$, associated with the Poisson bracket, is required to be skew-symmetric, which ensures the conservative character of reversible dynamics. The operator $\boldsymbol{M}$, associated with the friction matrix, governs the irreversible part of the system and must be be positive semidefinite in order to ensure that entropy production remains non-negative. By replacing the original bracket expressions in Eq. (2) with their respective operators, we obtain the time evolution equation for the state variables z,

$$\frac{d\boldsymbol{z}}{dt} = \boldsymbol{L}\frac{\partial E}{\partial \boldsymbol{z}} + \boldsymbol{M}\frac{\partial S}{\partial \boldsymbol{z}}. \tag{4}$$

To complete the GENERIC formulation, two degeneracy conditions must be imposed:

$$\{S, \boldsymbol{z}\} = [E, \boldsymbol{z}] = \boldsymbol{0}. \tag{5}$$

The first condition implies that entropy is a degenerate functional of the Poisson bracket, showing the reversible nature of the conservative part to the dynamics. The second condition indicates the energy is a degenerate functional with respect to the friction matrix, ensuring that the total energy of the system is conserved. These bracket-based conditions can be reformulated into a matrix form, using the $\boldsymbol{L}$ and $\boldsymbol{M}$ operators previously defined, resulting in the following degeneracy conditions:

$$\boldsymbol{L}\frac{\partial S}{\partial \boldsymbol{z}} = \boldsymbol{M}\frac{\partial E}{\partial \boldsymbol{z}} = \boldsymbol{0}. \tag{6}$$

Together with the requirement that the dissipation bracket is non-negative, these conditions ensure that the system satisfies the first and second principles of thermodynamics.

$$\frac{dE}{dt} = 0, \quad \frac{dS}{dt} \geq 0. \tag{7}$$

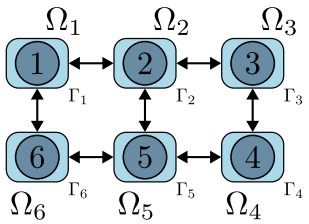


Fig. 3 Schematic representation of a port-metriplectic system. Each subsystem Ω_i exchanges information with its neighbours across its boundary Γ_i, through ports (arrows) that correspond to the edges in the graph

Port-metriplectic implementation

The GENERIC formulation described in "GENERIC formalism" section is suited for closed systems, where neither energy nor entropy is exchanged with the environment across the system boundary. However, this assumption becomes inadequate in the context of graph-based neural networks. In these models, each node evolves based on its own local information and its interaction with neighbouring nodes. This means that each node can be treated as an open system that exchanges information with its neighbours. Thus, including these inter-nodal interactions enables a consistent application of the thermodynamic bias in global systems composed of local subsystems interacting though well defined boundaries.

To account for this, we adopt a port-metriplectic formulation, as introduced in [37,38]. This framework is an extension of the GENERIC structure to open systems, composed of interacting subsystems [42]. In this setting, each node of the graph is treated as a local open subsystem that exchanges energy and entropy with its neighbours through its boundary, referred to as a port. In a graph-based setup, each edge is treated as a port. This formulation allows to impose a thermodynamic bias that is physically consistent and compatible with the graph structure.

Figure 3 illustrates the structure of the port-metriplectic approach. The global system is decomposed into a set of smaller subsystems, Ω_i, each associated with a graph node. Each subsystem is bounded by a region Γ_i, through which it interacts with neighbouring subsystems. The exchange of information occurs through ports, defined by arrows, that correspond to edges in the graph, defining local interaction interfaces.

By adopting this formulation, we are able to preserve the MGN architecture described in "Mesh Graph Neural Networks" section, while applying the thermodynamic structure at a local level. In this context, the Poisson and dissipative brackets can be decomposed into bulk and boundary contributions:

$$\{\cdot,\cdot\} = \{\cdot,\cdot\}_{\text{bulk}} + \{\cdot,\cdot\}_{\text{bound}}, \quad [\cdot,\cdot] = [\cdot,\cdot]_{\text{bulk}} + [\cdot,\cdot]_{\text{bound}}. \tag{8}$$

By applying this decomposition, the GENERIC formulation described in Eq. 2 for an open system is defined as

$$\frac{d\boldsymbol{z}}{dt} = \{\boldsymbol{z}, E\}_{\text{bulk}} + [\boldsymbol{z}, S]_{\text{bulk}} = \{\boldsymbol{z}, E\} + [\boldsymbol{z}, S] - \{\boldsymbol{z}, E\}_{\text{bound}} - [\boldsymbol{z}, S]_{\text{bound}}. \tag{9}$$

As we did in the previous section, we can reformulate Eq. 9 to matrix form:

$$\frac{d\boldsymbol{z}}{dt} = \boldsymbol{L}\frac{\partial E}{\partial \boldsymbol{z}} + \boldsymbol{M}\frac{\partial S}{\partial \boldsymbol{z}} - \boldsymbol{L}_{\text{bound}}\frac{\partial E_{\text{bound}}}{\partial \boldsymbol{z}} - \boldsymbol{M}_{\text{bound}}\frac{\partial S_{\text{bound}}}{\partial \boldsymbol{z}}. \tag{10}$$

The degeneracy conditions, which ensure thermodynamic consistency, still hold at bulk level:

$$\boldsymbol{L}_{\text{bulk}} \frac{\partial S_{\text{bulk}}}{\partial \boldsymbol{z}} = \boldsymbol{M}_{\text{bulk}} \frac{\partial E_{\text{bulk}}}{\partial \boldsymbol{z}} = \boldsymbol{0}. \tag{11}$$

In the graph-based formulation, this translates into a port-metriplectic approach at node level, in which we assume that each node exchanges energy and entropy with its neighbours according to the graph connectivity. The matrix formulation defined in Eq. 10 is applied to each node i, resulting in:

$$\frac{d\boldsymbol{z}_i}{dt} = \boldsymbol{L}_i \frac{\partial E_i}{\partial \boldsymbol{z}_i} + \boldsymbol{M}_i \frac{\partial S_i}{\partial \boldsymbol{z}_i} - \sum_{j \in \mathcal{N}(i)} \left(\boldsymbol{L}_{ij} \frac{\partial E_j}{\partial \boldsymbol{z}_j} + \boldsymbol{M}_{ij} \frac{\partial S_j}{\partial \boldsymbol{z}_j} \right), \tag{12}$$

where $\mathcal{N}(i)$ denotes the neighbouring nodes of i, with $j = 1, ..., n_{neigh}$ representing the nodes connected to i through graph edges.

Then, the degeneracy conditions are applied at node level,

$$\boldsymbol{L}_i \frac{\partial S_i}{\partial \boldsymbol{z}} = \boldsymbol{M}_i \frac{\partial E_i}{\partial \boldsymbol{z}} = \boldsymbol{0}, \tag{13}$$

for all $i = 1, ..., n$.

Learning procedure

In this section we illustrate the complete architecture of our framework, which builds upon the MGN algorithm defined in "Mesh Graph Neural Networks" section, but introduces modifications to incorporate the thermodynamic bias. A schematic overview of the proposed architecture is shown in Fig. 4, illustrating the modifications introduce with respect to the MGN baseline.

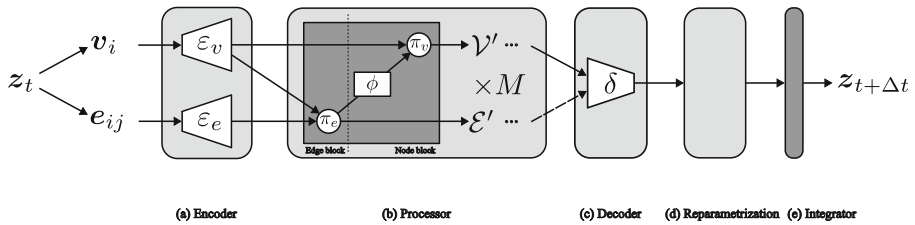
The **encoder** and **processor** blocks remain unchanged with respect to the original MGN architecture. However, the **decoder** block is modified to predict the components of the nodal GENERIC formalism. The single decoder is replaced by six small decoders, each defined as a MLP that acts on latent node and edge features. These decoders produce the energy $\frac{\partial E_i}{\partial \boldsymbol{z}_i}$ and entropy $\frac{\partial S_i}{\partial \boldsymbol{z}_i}$ gradients for each node, nodal flattened $\boldsymbol{l}_i$ and $\boldsymbol{m}_i$ operators, and edge flattened $\boldsymbol{l}_{ij}$, $\boldsymbol{m}_{ij}$ operators. For edge operators, the corresponding decoders use both nodal and edge latent features.

As these operators are predicted as flattened vectors, they must be reshaped into their corresponding matrix form, The reshaping is done by the **reparametrization** block, an additional step with respect to the MGN scheme. As shown previously, the Poisson operators $\boldsymbol{L}_i$ and $\boldsymbol{L}_{ij}$ are required to be skew-symmetric:

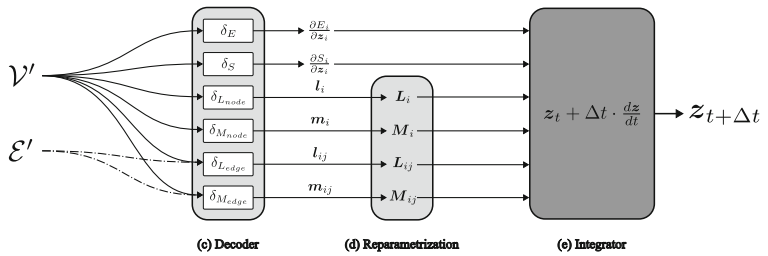
$$\boldsymbol{L}_i = \boldsymbol{l}_i - \boldsymbol{l}_i^{\mathrm{T}}, \qquad \boldsymbol{L}_{ij} = \boldsymbol{l}_{ij} = \boldsymbol{l}_{ij}^{\mathrm{T}}, \tag{14}$$

while the dissipative operators $\boldsymbol{M}_i$ and $\boldsymbol{M}_{ij}$ are defined to be positive semi-definite:

$$\boldsymbol{M}_i = \boldsymbol{m}_i \boldsymbol{m}_i^{\mathrm{T}}, \qquad \boldsymbol{M}_{ij} = \boldsymbol{m}_{ij} = \boldsymbol{m}_{ij}^{\mathrm{T}}. \tag{15}$$



General block scheme for the proposed algorithm.



Decoder, reparametrization and integrator blocks.

Fig. 4 Top: Proposed algorithm block scheme. **a** The encoder transforms the node and edge features to a latent representation. **b** The processor shares and updates edge and nodal information through the graph by applying *M* message passing blocks. **c** A set of six decoders predict the energy entropy gradients, and the flattened Poisson and dissipative operators both at node and edge levels. **d** The reparametrization blocks reshapes the flattened operators into matrices with the appropriate structure. **e** The integrator computes the next system state using a forward-Euler scheme based on port-metriplectic dynamics. Bottom: Detail of the decoder, reparametrization and integration blocks in the proposed scheme

Similar to the original MGN scheme, our last step is an **integrator** that updates the system state. This is done using a forward-Euler scheme:

$$\boldsymbol{z}_{i,t+\Delta t} = \boldsymbol{z}_{i,t} + \Delta t \cdot \frac{d\boldsymbol{z}_i}{dt}, \tag{16}$$

where, for each node i, the system derivative $\frac{d\boldsymbol{z}_i}{dt}$ is obtained from the nodal GENERIC formalism defined in Eq. 12.

The loss function used during the training process is composed by two different terms:

- Data loss: This term measures the accuracy of the network prediction. It is computed as the Mean Squared Error (MSE) between the predicted and the ground-truth time derivatives of the state variables over the graph nodes:

$$\mathcal{L}^{\text{data}} = \left\| \frac{d\boldsymbol{z}}{dt}^{\text{GT}} - \frac{d\boldsymbol{z}}{dt}^{\text{net}} \right\|_2^2, \tag{17}$$

  where $\|\cdot\|_2$ denotes the L2-norm. Using the time derivatives helps to normalize the loss components to similar orders of magnitude.
- Degeneracy conditions loss: This term enforces the thermodynamic consistency of the learned dynamics by penalizing violations of the degeneracy conditions:

$$\mathcal{L}^{\text{degen}} = \left\| \boldsymbol{L}_i \frac{\partial S_i}{\boldsymbol{z}_i} \right\|_2^2 + \left\| \boldsymbol{M}_i \frac{\partial E_i}{\boldsymbol{z}_i} \right\|_2^2. \tag{18}$$

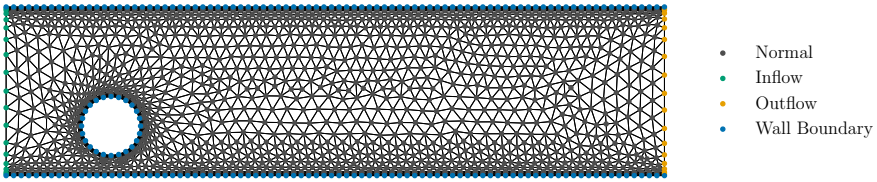


Fig. 5 Mesh example showing the different node types used in the graph: normal, inflow, outflow, and wall boundary. Each category encodes distinct boundary conditions relevant to the simulation

The total loss term is composed of a weighted sum of both terms. A hyperparameter λ^{degen} is introduced to balance their influence,

$$\mathcal{L} = \mathcal{L}^{\text{data}} + \lambda^{\text{degen}} \cdot \mathcal{L}^{\text{degen}}. \tag{19}$$

Results

We have evaluated the proposed method on two different examples involving fluid mechanics, both based on the simulation of unsteady flow around previously unknown solid obstacles. The first example corresponds to unsteady flow-past-a cylinder, where the obstacle geometry varies in size and position, but always maintain a cylindrical shape. The second example involves diverse geometries, where the obstacle shape itself changes across simulations.

In botch cases, the underlying dynamics are governed by the incompressible Navier–Stokes equations. The state variables for describing the state for the flow-past-a cylinder consist of the velocity $\boldsymbol{u}$ and pressure P fields at each node of the graph:

$$\mathcal{S} = \{\boldsymbol{z} = (\boldsymbol{u}, P) \in \mathbb{R}^2 \times \mathbb{R}\}. \tag{20}$$

The simulations are defined on Eulerian triangular meshes, where nodal coordinates remain fixed in space and are encoded as edge features. The state variables are assigned to the node features. In addition, a one-hot vector $\boldsymbol{n}$ is added to the node features to represent the different types of node: inflow, outflow, wall boundary or fluid (normal) nodes, as shown in Fig. 5. This node-type encoding allows us to impose appropriate boundary conditions at the inflow, outflow and wall modes, while predictions are carried out on fluid nodes,

$$\boldsymbol{v}_i = (\boldsymbol{u}, P, \boldsymbol{n}), \quad \boldsymbol{e}_{ij} = \left(\boldsymbol{q}_i - \boldsymbol{q}_j, \left\|\boldsymbol{q}_i - \boldsymbol{q}_j\right\|_2\right), \tag{21}$$

where the dimension of node features is $F_v = 12$, while edge features dimension is $F_e = 3$. Before training, all variables are normalized using global statistics computed from the training test, ensuring consistent scaling between variables and simulations.

To evaluate the performance of the model, we measure both Root Mean Squared Error (RMSE) and Relative Root Mean Squared Error (RRMSE) between the predicted and reference fields. These metrics are computed independently for each state variable and are averaged over all nodes. Long-range accuracy is measured by evualting predictions at 1-step, 50-steps, 200-steps and full sequence predictions.

The RMSE for a given variable is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left\| \hat{\boldsymbol{y}} - \boldsymbol{y} \right\|_2}, \tag{22}$$

while the RRMSE is taken as

$$\text{RRMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left(\frac{\left\| \hat{\boldsymbol{y}} - \boldsymbol{y} \right\|_2}{\left\| \boldsymbol{y} \right\|_\infty} \right)}, \tag{23}$$

where $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ denote the reference and predicted values.

Additionally, we compare the proposed method against two baselines trained using the same hyperparameters: (i) a vanilla MGN, following the same original formulation introduced in [35], which predicts velocity gradients directly and estimates pressure as an auxiliary variable, without incorporating any thermodynamical constraints; and (ii) a vanilla TINN [28], which includes the thermodynamic bias but does not exploit any geometric information.

The following subsections present a detailed description of each example, including dataset, training settings and an analysis of the results.

The network architecture is implemented in PyTorch and is publicly available online in GitHub.

Example 1: Flow past-a-cylinder

Database and hyperparameters

For the first example, we have utilized a publicly available dataset produced by Google Deepmind to test MeshGraphNets [35]. The dataset consists of 1200 simulations of unsteady flow past a cylindrical obstacle. The cylinder varies in diameter and position across different simulations. No-slip conditions are applied to the walls and the cylinder obstacle. The flow conditions are varied by modifying the free stream velocity. All cases are discretized in $N_t = 600$ time increments of $\Delta t = 10^{-2}$.

The chosen hyperparameters for training the model include a hidden dimension of $N_h = 128$ neurons for both node and edge latent features, with $L_h = 2$ hidden layers. The number of message passing iterations is set to $M = 15$. We use the Swish Activation function [43] for all layers in the model except the decoder heads final layer, which do not use any activation function. The Adam optimizer is used during the training process [44]. During training, gradients are computed after single-step updates, while multi-step rollouts are only employed for evaluation. The number of training epochs is $N_{\text{epochs}} = 34$, which is equivalent to training for approximately 10×10^6 steps using a batch size of 2. The learning rate is set to $l_r = 10^{-4}$, and decreased two orders of magnitude to $l_r = 10^{-6}$ after 8×10^6 steps following an exponential scheduler. Additionally, the training noise variance is set $\sigma_{\text{noise}} = 2 \times 10^{-2}$. The weight of the degeneracy loss term is set to $\lambda^{\text{degen}} = 10^{-2}$, selected from a hyperparameter optimization study using the Weight & Biases (W&B) framework [45], where values where explored within the range $\left[10^{-4}, 10^{-1}\right]$.

Table 1 Relative Root Mean Squared Error (RRMSE) for each variable across different rollout horizons

Variable	RRMSE [$\times 10^{-3}$]			
	1-step	50-step rollout	200-step rollout	Full rollout
u_x	2.23 ± 0.09	9.37 ± 0.55	14.17 ± 0.61	13.40 ± 0.52
u_y	2.98 ± 0.08	6.72 ± 0.32	10.04 ± 0.38	9.95 ± 0.45
P	1.57 ± 0.10	3.23 ± 0.13	2.87 ± 0.17	2.77 ± 0.22

All values are scaled by 10^{-3}

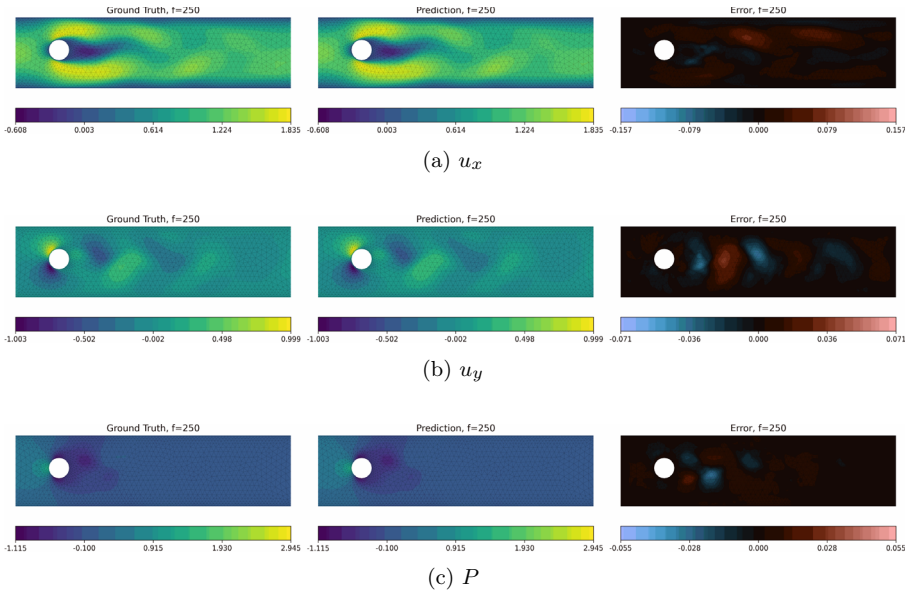


Fig. 6 Prediction results for a representative test snapshot (frame number f = 250, corresponding to time $t = 2.50$ s). Each row corresponds to one predicted variable: **a** u_x, **b** u_y, **c** P. For each, we show ground truth (left), predicted field (middle), and absolute error (right). The model accurately captures the main flow features across all variables

Results

The rollout prediction results are summarized in Table 1, which reports the RRMSE for each variable at different time horizons (1, 50, 200, and full-step rollout). All values are scaled by 10^{-3} and averaged over the test set. The results indicate stable error growth, demonstrating the ability of the model to maintain accurate predictions over long ranges.

Figure 6 shows a comparison between predicted and the ground truth velocity and pressure fields at a single snapshot from a test case. The corresponding absolute error fields are also shown. The prediction accurately captures the main flow features, with low discrepancies across the domain.

We compare our method against the baseline MGN and the vanilla TINN. Figure 7 shows the RMSE distribution across the test set for each variable over full rollouts. The proposed method achieves lower mean error and variance, indicating that combining both thermodynamic and geometric biases contributes to improve long-range predictions.

Finally, we measure the computational efficiency of the method. Training the model takes 140 h on a single NVIDIA RTX 3090. Inference times are shown in Table 2, where we compare the rollout time for the baseline MGN, our approach and running the CFD simulation. We can see that the baseline MGN approach outperforms our method, which

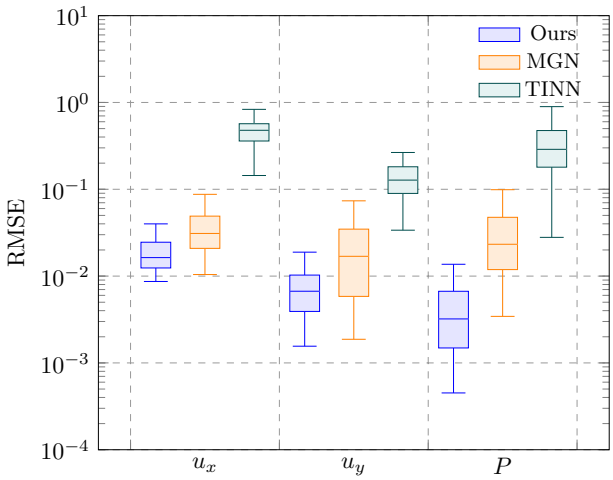


Fig. 7 Log-scale RMSE distribution across test set for each variable over full rollouts. Comparison between our method, the baseline MGN, and the vanilla TINN

Table 2 Computational performance comparison

Metric	CFD (GT)	Baseline MGN	Ours
Full rollout time (s)	492	3.84	4.56
Time per step (ms)	820	6.4	7.6

makes sense, considering that it only requires one decoder while our is composed by 6 decoder heads. Eitherway, both methods are considerably faster than computing the ground-truth simulation, achieving 107× speedups.

Example 2: Flow past-an-obstacle—varying geometries

Database and hyperparameters

For the second example, the ground-truth simulations are computed by solving the 2D incompressible Navier–Stokes equations. Six different types of obstacles are considered, using the following primitive shapes: circle, triangle, square, rectangle, pentagon, and hexagon. Each shape is randomly varied in position, orientation, and size, following a similar procedure to [46]. This results in a total of 1200 simulations, each with an average of 2600 mesh nodes. No-slip boundary conditions are applied at the channel walls and around the obstacle. The fluid has a density of $\rho = 1$ and a dynamic viscosity of $\mu = 10^{-3}$. The freestream velocity is varied across simulations, sampled uniformly $\mathcal{U}$ [0.5, 1.5]. The dataset is split into training, validation and test sets, containing 1000, 100, and 100 simulations, respectively. All cases are discretized in $N_t = 600$ time increments of $\Delta t = 10^{-2}$.

As for the hyperparameters, we keep the same latent dimension of $N_h = 128$ neurons for both nodes and edges, with $L_h = 2$ hidden layers, and $M = 15$ message passing steps. Swish Activation function [43] is employed for all layers in the model, except in the final decoder layers which are left unactivated. Optimization is performed using Adam optimizer for $N_{\text{epochs}} = 35$ (about 10×10^6 steps with batch size 2) [44]. During training, the loss is evaluated after single-step time integrations, whereas multi-step evaluations are reserved for testing. The learning rate is set to $l_r = 10^{-4}$, and decays to $l_r = 10^{-6}$ after 8×10^6 steps following an exponential scheduler. To improve robustness, noise with

Table 3 Variable-wise Relative Root Mean Squared Error (RRMSE) for different rollout horizons

Variable	RRMSE [$\times 10^{-3}$]			
	1-step	50-step rollout	200-step rollout	Full rollout
u_x	1.04 ± 0.03	2.93 ± 0.14	5.32 ± 0.25	5.65 ± 0.29
u_y	1.32 ± 0.04	3.79 ± 0.26	6.74 ± 0.40	7.10 ± 0.45
P	1.63 ± 0.14	2.04 ± 0.11	3.28 ± 0.32	3.76 ± 0.44

All values are multiplied by a factor of 10^{-3}

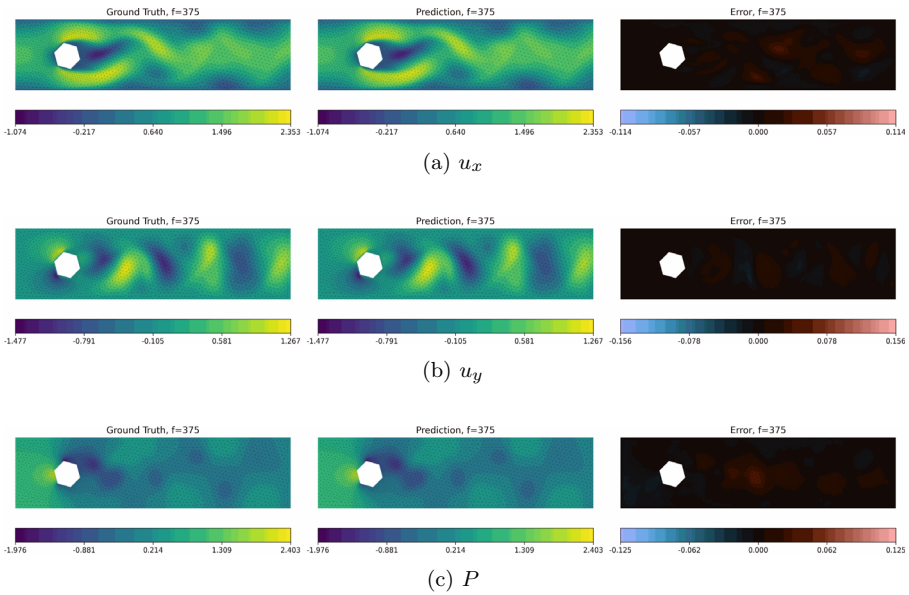


Fig. 8 Prediction results for a test snapshot (frame number f = 375, corresponding to time $t = 3.75$ s) with an unseen obstacle geometry (hexagon-shape based). Each row corresponds to one predicted variable: **a** u_x, **b** u_y, **c** P. The ground truth (left), predicted field (middle), and absolute error (right) show good generalization to this geometry

variance $\sigma_{\text{noise}} = 2 \times 10^{-2}$ is added during training. The degeneracy loss weight is set to $\lambda^{\text{degen}} = 10^{-1}$, selected using the W&B framework [45], from a hyperparameter search in the range $\left[10^{-4}, 10^{-1}\right]$.

Results

Table 3 presents the RRMSE values for the rollout prediction, evaluated at multiple prediction horizons (1, 50, 200 and full sequence). The values, scaled by 10^{-3} and averaged over the test set, present a controlled growth in error as rollout advances, showing the model's ability to remain accurate over long-range predictions.

Figures 8 and 9 display the model predictions for two test snapshots and different obstacle geometries. Each figure shows the ground truth and predicted velocity and pressure fields, and their corresponding absolute error. These examples demonstrate the ability of the model to generalize to unseen geometries while maintaining good accuracy.

To further assess performance, we compare our model against two baselines: a vanilla MGN and a vanilla TINN. Figure 10 presents the RMSE error for all predicted variables over full rollouts. Our method achieves lower average errors and reduced variability across

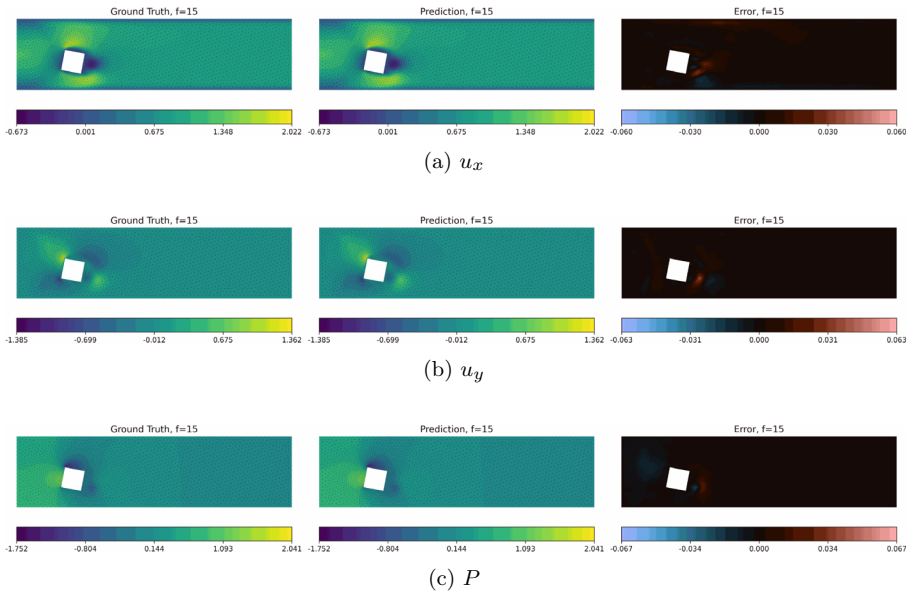


Fig. 9 Prediction results for another test snapshot (frame number f = 15, corresponding to time $t = 0.15$ s) featuring a different unseen obstacle geometry (rectangle-based shape). The rows correspond to: **a** u_x, **b** u_y, **c** P, each showing ground truth, prediction, and absolute error. The model successfully predicts the flow for unseen obstacles

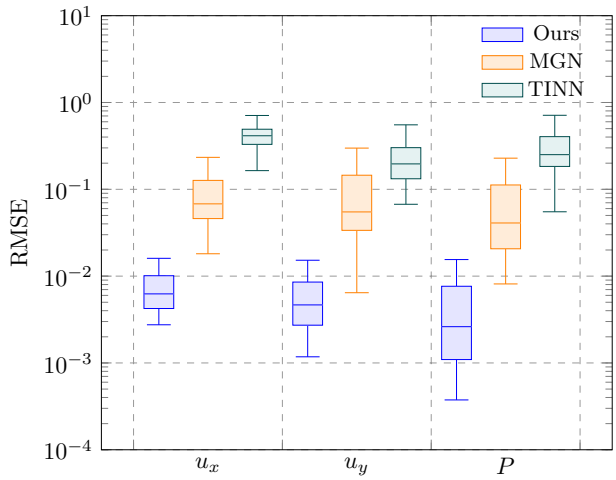


Fig. 10 Log-scale RMSE distribution across test set for each variable over full rollout predictions. Comparison between our method, a baseline MGN, and a baseline TINN

the test set, highlighting the benefit of combining geometric and thermodynamic biases for long-term time integration.

Regarding computational preformance, the model was trained for 76 h on a single NVIDIA RTX 4090 GPU. Table 4 compares the inference time per rollout to both the baseline MGN and the high-fidelity CFD solver. The ground truth solution was computed on a machine with an Intel i9-13900K CPU with 64 GB of RAM. While our method is slightly more expensive during inference than the vanilla-MGN due to its multi-head decoder structure, it remains orders of magnitude faster than the CFD simulations, with speedups up to 145×.

Table 4 Computational performance comparison

Metric	CFD (GT)	Baseline MGN	Ours
Full rollout time (s)	1250	4.53	8.58
Time per step (ms)	2083.33	7.55	14.3

Conclusions

In this work, we have presented a thermodynamically-informed graph neural network architecture for predicting the time evolution of complex dynamical systems while preserving the underlying thermodynamic structure. Our approach builds upon the MeshGraphNet architecture, replacing its standard decoder with six specialized heads that predict energy and entropy gradients, as well as flattened Poisson and dissipative operators. These components are assembled at the node level using a port-metriplectic formulation based on the GENERIC formalism, which has previously demonstrated strong performance [38]. This formulation ensures compliance with first and second principles of thermodynamics, while preserving the computational performance.

The proposed method has been evaluated on two fluid mechanics scenarios: flow past cylindrical obstacles with varying position and size, and a more general case involving obstacles with different primitive geometries. In both settings, the model shows high accuracy in long-term rollouts and improved robustness compared to a vanilla MGN. Moreover, inference is significantly faster than computing the ground truth using a traditional solver, making the method suitable for quasi-real time and real-time applications

However, the method still presents limitations related to scalability and computational cost. In simulations with fine meshes, a high number of message passing iterations or fine time discretization is required to capture the dynamics accurately, increasing the computational cost. To overcome these limitations, future approaches will explore multiscale and hierarchical graph neural networks [47,48] that allow to transfer information across different spatial scales, minimizing the number of message passing steps. Additionally, while the method shows good generalization capabilities, it has been trained and validated on synthetic CFD datasets. In this sense, future work could extend the framework to more complex behaviours, such as non-Newtonian fluids or turbulent regime, as well as integrating experimental or sensor-based data to enhance real-world applicability.

Acknowledgements

This work was supported by the Spanish Ministry of Science and Innovation, AEI/10.13039/501100011033, through Grant number PID2023-147373OB-I00, and by the Ministry for Digital Transformation and the Civil Service, through the ENIA 2022 Chairs for the creation of university-industry chairs in AI, through Grant TSI-100930-2023-1. This material is also based upon work supported in part by the Army Research Laboratory and the Army Research Office under contract/grant number W911NF2210271. The authors also acknowledge the support of ESI Group through the chair at the University of Zaragoza.

Author contributions

C.B. conducted research, wrote the code and the first draft of the paper. A.B. conducted research, reviewed the results, advised C.B. on the development of the method and reviewed the manuscript. D.G. conducted research, wrote code, reviewed the results, advised C.B. on the development of the method and reviewed the manuscript. E. C. conducted research, reviewed the results, advised C.B. on the development of the method and wrote the final version of the manuscript. Also got funding.

Data Availability

No datasets were generated or analysed during the current study.

Declarations

Competing interests

The authors declare no competing interests.

Received: 30 July 2025 Accepted: 2 September 2025
Published online: 30 September 2025

References

1. Moya B, Alfaro I, Gonzalez D, Chinesta F, Cueto E. Physically sound, self-learning digital twins for sloshing fluids. PLoS ONE. 2020;15(6):0234569.
2. Moya B, Badias A, Alfaro I, Chinesta F, Cueto E. Digital twins that learn and correct themselves. Int J Numer Meth Eng. 2022;123(13):3034–44. https://doi.org/10.1002/nme.6535.
3. Hernandez Q, Badias A, Chinesta F, Cueto E. Thermodynamics-informed neural networks for physically realistic mixed reality. Comput Methods Appl Mech Eng. 2023;407:115912. https://doi.org/10.1016/j.cma.2023.115912.
4. Tierz A, Iparraguirre MM, Alfaro I, González D, Chinesta F, Cueto E. On the feasibility of foundational models for the simulation of physical phenomena. Int J Numer Meth Eng. 2025;126(6):70027. https://doi.org/10.1002/nme.70027.
5. Brunton SL, Proctor JL, Kutz JN. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc Natl Acad Sci. 2016;113(15):3932–7. https://doi.org/10.1073/pnas.1517384113.
6. Brunton SL, Kutz JN. Data-driven science and engineering: machine learning, dynamical systems, and control. 2nd ed. Cambridge: Cambridge University Press; 2022.
7. Vinuesa R, Brunton SL. Enhancing computational fluid dynamics with machine learning. Nat Comput Sci. 2022;2(6):358–66. https://doi.org/10.1038/s43588-022-00264-7.
8. Badias A, Curtit S, González D, Alfaro I, Chinesta F, Cueto E. An augmented reality platform for interactive aerodynamic design and analysis. Int J Numer Meth Eng. 2019;120(1):125–38. https://doi.org/10.1002/nme.6127.
9. Eivazi H, Clainche S, Hoyas S, Vinuesa R. Towards extraction of orthogonal and parsimonious non-linear modes from turbulent flows. Expert Syst Appl. 2022;202:117038. https://doi.org/10.1016/j.eswa.2022.117038.
10. Wang Y, Solera-Rico A, Sanmiguel Vila C, Vinuesa R. Towards optimal beta-variational autoencoders combined with transformers for reduced-order modelling of turbulent flows. Int J Heat Fluid Flow. 2024;105:109254. https://doi.org/10.1016/j.ijheatfluidflow.2023.109254.
11. Willard J, Jia X, Xu S, Steinbach M, Kumar V. Integrating physics-based modeling with machine learning: a survey. 2020; 1(1):1–34. arXiv preprint arXiv:2003.04919.
12. Cuomo S, Cola VS, Giampaolo F, Rozza G, Raissi M, Piccialli F. Scientific machine learning through physics-informed neural networks: where we are and what's next. J Sci Comput. 2022;92(3):88. https://doi.org/10.1007/s10915-022-01939-z.
13. Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys. 2019;378:686–707. https://doi.org/10.1016/j.jcp.2018.10.045.
14. Greydanus S, Dzamba M, Yosinski J. Hamiltonian Neural Networks. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R, editors. Advances in Neural Information Processing Systems (NeurIPS), vol. 32. Red Hook: Curran Associates Inc; 2019. p. 15379–89.
15. Mattheakis M, Sondak D, Dogra AS, Protopapas P. Hamiltonian neural networks for solving equations of motion. Phys Rev E. 2022;105:065305. https://doi.org/10.1103/PhysRevE.105.065305.
16. David M, Méhats F. Symplectic learning for Hamiltonian neural networks. J Comput Phys. 2023;494:112495. https://doi.org/10.1016/j.jcp.2023.112495.
17. Galimberti CL, Furieri L, Xu L, Ferrari-Trecate G. Hamiltonian deep neural networks guaranteeing nonvanishing gradients by design. IEEE Trans Autom Control. 2023;68(5):3155–62. https://doi.org/10.1109/TAC.2023.3239430.
18. Cranmer M, Greydanus S, Hoyer S, Battaglia P, Spergel D, Ho S. Lagrangian neural networks; 2020. arxiv:2003.04630.
19. Roehrl MA, Runkler TA, Brandtstetter V, Tokic M, Obermayer S. Modeling system dynamics with physics-informed neural networks based on Lagrangian mechanics. IFAC-PapersOnLine. 2020;53(2):9195–200. https://doi.org/10.1016/j.ifacol.2020.12.2182. (**21st IFAC World Congress**).
20. Bhattoo R, Ranu S, Krishnan NMA. Learning the dynamics of particle-based systems with Lagrangian graph neural networks. Mach Learn Sci Technol. 2023;4(1):015003. https://doi.org/10.1088/2632-2153/acb03e.
21. Grmela M, Öttinger HC. Dynamics and thermodynamics of complex fluids. I. Development of a general formalism. Phys Rev E. 1997;56:6620–32. https://doi.org/10.1103/PhysRevE.56.6620.
22. Öttinger HC, Grmela M. Dynamics and thermodynamics of complex fluids. II. Illustrations of a general formalism. Phys Rev E. 1997;56:6633–55. https://doi.org/10.1103/PhysRevE.56.6633.
23. Moya B, Badias A, Gonzalez D, Chinesta F, Cueto E. Physics perception in sloshing scenes with guaranteed thermodynamic consistency. IEEE Trans Pattern Anal Mach Intell. 2023;45(2):2136–50. https://doi.org/10.1109/TPAMI.2022.3160100.
24. González D, Chinesta F, Cueto E. Thermodynamically consistent data-driven computational mechanics. Continuum Mech Thermodyn. 2019;31(1):239–53. https://doi.org/10.1007/s00161-018-0677-z.
25. Ghnatios C, Alfaro I, González D, Chinesta F, Cueto E. Data-driven GENERIC modeling of poroviscoelastic materials. Entropy. 2019;21(12):1165. https://doi.org/10.3390/e21121165.
26. Zhang Z, Shin Y, Em Karniadakis G. GFINNs: GENERIC formalism informed neural networks for deterministic and stochastic dynamical systems. Phil Trans R Soc A. 2022;380(2229):20210207. https://doi.org/10.1098/rsta.2021.0207.
27. Bermejo-Barbanoj C, Moya B, Badias A, Chinesta F, Cueto E. Thermodynamics-informed super-resolution of scarce temporal dynamics data. Comput Methods Appl Mech Eng. 2024;430:117210. https://doi.org/10.1016/j.cma.2024.117210.
28. Hernandez Q, Badias A, González D, Chinesta F, Cueto E. Structure-preserving neural networks. J Comput Phys. 2021;426:109950. https://doi.org/10.1016/j.jcp.2020.109950.

29. Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE Trans Neural Netw. 2009;20(1):61–80. https://doi.org/10.1109/TNN.2008.2005605.
30. Hamilton WL, Ying R, Leskovec J. Representation learning on graphs: Methods and applications;2018. arXiv:1709.05584 [cs.SI].
31. Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P. Geometric deep learning: going beyond Euclidean data. IEEE Signal Process Mag. 2017;34(4):18–42. https://doi.org/10.1109/MSP.2017.2693418.
32. Battaglia PW, Hamrick JB, Bapst V, Sanchez-Gonzalez A, Zambaldi V, Malinowski M, Tacchetti A, Raposo D, Santoro A, Faulkner R, Gulcehre C, Song F, Ballard A, Gilmer J, Dahl G, Vaswani A, Allen K, Nash C, Langston V, Dyer C, Heess N, Wierstra D, Kohli P, Botvinick M, Vinyals O, Li Y, Pascanu R. Relational inductive biases, deep learning, and graph networks;2018.
33. Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE. 1998;86(11):2278–324. https://doi.org/10.1109/5.726791.
34. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436–44. https://doi.org/10.1038/nature14539.
35. Pfaff T, Fortunato M, Sanchez-Gonzalez A, Battaglia PW. Learning mesh-based simulation with graph networks;2021. arxiv:2010.03409.
36. Hernandez Q, Badias A, Chinesta F, Cueto E. Thermodynamics-informed Graph Neural Networks. IEEE Trans Artif Intell. 2022. https://doi.org/10.1109/TAI.2022.3179681.
37. Hernandez Q, Badias A, Chinesta F, Cueto E. Port-metriplectic neural networks: thermodynamics-informed machine learning of complex physical systems. Comput Mech. 2023;72(3):553–61. https://doi.org/10.1007/s00466-023-02296-w.
38. Tierz A, Alfaro I, González D, Chinesta F, Cueto E. Graph neural networks informed locally by thermodynamics. Eng Appl Artif Intell. 2025;144:110108. https://doi.org/10.1016/j.engappai.2025.110108.
39. Lee K, Trask NA, Stinis P. Machine learning structure preserving brackets for forecasting irreversible processes;2021.
40. Guha P. Metriplectic structure, Leibniz dynamics and dissipative systems. J Math Anal Appl. 2007;326(1):121–36. https://doi.org/10.1016/j.jmaa.2006.02.023.
41. Morrison PJ. A paradigm for joined Hamiltonian and dissipative systems. Physica D. 1986;18(1):410–9. https://doi.org/10.1016/0167-2789(86)90209-5.
42. Öttinger HC. Nonequilibrium thermodynamics for open systems. Phys Rev E. 2006;73:036126. https://doi.org/10.1103/PhysRevE.73.036126.
43. Ramachandran P, Zoph B, Le QV. Searching for activation functions;2017. arxiv:1710.05941.
44. Kingma DP, Ba J. Adam: a method for stochastic optimization;2017.
45. Biewald L. Experiment Tracking with Weights and Biases. Software available from wandb.com; 2020. https://www.wandb.com/.
46. Schmacker R, Henkes ARoth J, Wick T. Generalization capabilities of eshGraphNets to unseen geometries for fluid dynamics;2024. arxiv:2408.06101.
47. Fortunato M, Pfaff T, Wirnsberger P, Pritzel A, Battaglia P. MultiScale MeshGraphNets;2022. arxiv:2210.00612.
48. Li H, Zhao Y, Zhou H, Pfaff T, Li N. A new graph-based surrogate model for rapid prediction of crashworthiness performance of vehicle panel components;2025. arxiv:2503.17386.

Publisher's Note