LEARNING THE DYNAMICS OF PHYSICAL SYSTEMS FROM SPARSE OBSERVATIONS WITH FINITE ELEMENT NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a new method for spatio-temporal forecasting on arbitrarily distributed points. Assuming that the observed system follows an unknown partial differential equation, we derive a continuous-time model for the dynamics of the data via the finite element method. The resulting graph neural network estimates the instantaneous effects of the unknown dynamics on each cell in a meshing of the spatial domain. Our model can incorporate prior knowledge via assumptions on the form of the unknown PDE, which induce a structural bias towards learning specific processes. Through this mechanism, we derive a transport variant of our model from the convection equation and show that it improves the predictive accuracy on sea surface temperature and gas flow forecasting against baseline models representing a selection of spatio-temporal forecasting methods. A qualitative analysis shows that our model disentangles the data dynamics into their constituent parts, which makes it uniquely interpretable.

1 INTRODUCTION

The laws driving the physical world are often best described by partial differential equations (PDEs) that relate how a magnitude of interest changes in time with its change in space. They describe how the atmosphere and oceans circulate and interact, how structures deform under load and how electromagnetic waves propagate (Courant & Hilbert, 2008). Knowledge of these equations lets us predict the weather (Coiffier, 2011), build sturdier structures, and communicate wirelessly. Yet, in many cases we only know the PDEs governing a system partially (Isakov, 2006) or not at all, or solving them is too computationally costly to be practical (Ames, 2014).

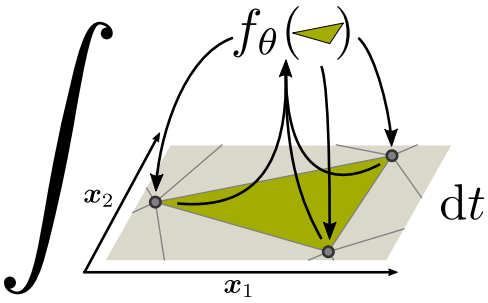


Figure 1: Finite Element Networks predict the instantaneous change of each node by estimating the effect of the unknown generating dynamics on the domain volume that the node shares with its neighbors.

Machine learning researchers try to fill in these gaps with models trained on collected data. For example, neural networks have been trained for weather forecasts (Shi et al., 2015) and fluid flow simulations (Belbute-Peres et al., 2020), both of which are traditionally outcomes of PDE solvers. Even the dynamics of discrete dynamical systems such as traffic (Li et al., 2018) and crowds (Zhang et al., 2017) have been learned from data. A challenge facing these models is the high cost of acquiring training data, so the data is usually only available sparsely distributed in space. Since graphs are a natural way to structure sparse data, models incorporating graph neural networks (GNNs) have been particularly successful for spatio-temporal forecasting (Yu et al., 2018; Wu et al., 2019).

In the domain of physical processes we can reasonably assume that the observed system follows a PDE. There are mainly two ways to incorporate this assumption as a-priori knowledge into a model. First, we can encode a known PDE into a loss function that encourages the model to fulfill the equation (Raissi et al., 2019). Another way to go about this is to derive the model structure itself

from known laws such as the convection-diffusion equation (de Bézenac et al., 2018). In this paper we will follow the second approach.

Consider a dynamical system on a bounded domain $\Omega \subset \mathbb{R}^d$ that is governed by the PDE

$$\partial_t u = F\left(t, \boldsymbol{x}, u, \partial_{\boldsymbol{x}} u, \partial_{\boldsymbol{x}}^2 u, ...\right) \tag{1}$$

on functions $u : [0, T] \times \Omega \to \mathbb{R}^m$. If we have a dense measurement $u_0 : \Omega \to \mathbb{R}^m$ of the current state of the system and a solution u that satisfies Eq. (1) for all $t \in [0, T]$ and also fulfills the initial condition $u(0, \boldsymbol{x}) = u_0(\boldsymbol{x})$ at all points $\boldsymbol{x} \in \Omega$, we can use u as a forecast for the state of the system until time T. From a spatio-temporal forecasting perspective, this means that we can forecast the evolution of the system if we have a continuous measurement of the state, know the dynamics F, and can find solutions of Eq. (1) efficiently. Unfortunately, in practice we only have a finite number of measurements at arbitrary points and only know the dynamics partially or not at all.

Contributions. An established numerical method for forecasts in systems with fully specified dynamics is the finite element method (FEM) (Brenner et al., 2008). In this paper, we introduce the first graph-based model for spatio-temporal forecasting that is derived from FEM in a principled way. Our derivation establishes a direct connection between the form of the unknown dynamics and the structure of the model. Through this connection our model can incorporate prior knowledge on the governing physical processes via assumptions on the form of the underlying dynamics. We employ this mechanism to derive a specialized model for transport problems from the convection equation. The way that the model structure arises from the underlying equation makes our models uniquely interpretable. We show that our transport model disentangles convection and the remainder of the learned dynamics such as source/sink behavior, and that the activations of the model correspond to a learned flow field, which can be visualized and analyzed. In experiments on multi-step forecasting of sea surface temperature and gas flow, our model improves upon baselines from recurrent, temporal-convolutional, and continuous-time model classes with further improvement by the transport model.

2 Background

2.1 Finite Element Method

In the following, we will outline how to approximate a solution u to the dynamics in Eq. (1) from an initial value u_0 by discretizing u in space using finite elements. Let $\mathcal{X}$ be a set of points with a triangulation $\mathcal{T}$ of d-dimensional, non-overlapping simplices

$$\mathcal{X} = \{\boldsymbol{x}^{(i)} \in \mathbb{R}^d\}_{i=1}^N \qquad \mathcal{T} = \{\Delta^{(j)} \mid \Delta^{(j)} \subset \mathcal{X},\ |\Delta^{(j)}| = d+1\}_{j=1}^{N_\mathcal{T}} \tag{2}$$

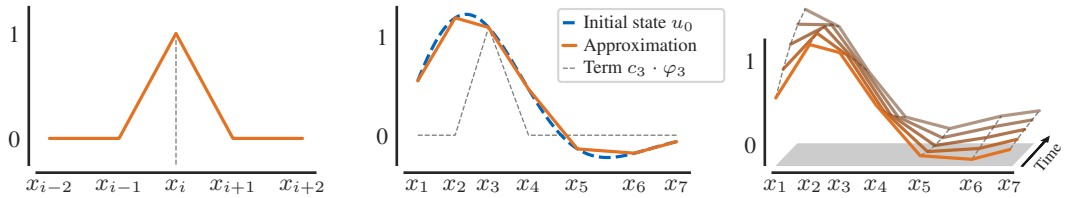
such that $\cup_{\Delta \in \mathcal{T}} \mathrm{CH}(\Delta)$ equals the domain Ω where $\mathrm{CH}(\Delta)$ is the convex hull of simplex Δ. So we define a simplex $\Delta^{(j)} \in \mathcal{T}$ representing the j-th mesh cell as the set of vertices of the cell and denote the domain volume covered by the cell by the convex hull $\mathrm{CH}(\Delta^{(j)})$ of the vertices. We will assume u to be a scalar field, i.e. $u : [0, T] \times \Omega \to \mathbb{R}$. If u is a vector field, we treat it as a system of m scalar fields instead. For a detailed introduction to FEM, we refer the reader to Igel (2017).

Basis Functions. A priori, we assume that the unknown solution u to our problem lies in an infinite-dimensional function space $\mathcal{U}$. The first step in FEM to make the problem numerically feasible is to approximate $\mathcal{U}$ with a finite-dimensional linear subspace $\tilde{\mathcal{U}}$. This subspace can then be written in terms of linear combinations of basis functions $\tilde{\mathcal{U}} = \mathrm{span}\left\{\varphi^{(1)}, ..., \varphi^{(N)}\right\}$. There are many possible bases and the choice determines various qualities of the resulting procedure such as continuity of the approximation and the sparsity pattern of the mass matrix in Eq. (7).

In our case, we choose the so-called P1 basis of piecewise linear functions (hat functions), see Fig. 2a (Igel, 2017). There are as many basis functions as there are points and each is uniquely defined by being linear when restricted to a single cell $\Delta \in \mathcal{T}$ and the constraint

$$\varphi^{(j)}(\boldsymbol{x}^{(i)}) = \begin{cases} 1 & \text{if } \boldsymbol{x}^{(i)} = \boldsymbol{x}^{(j)} \\ 0 & \text{otherwise} \end{cases} \qquad \forall \boldsymbol{x}^{(i)} \in \mathcal{X}. \tag{3}$$

So the basis function $\varphi^{(j)}$ is 1 at $\boldsymbol{x}^{(j)}$, falls linearly to 0 on mesh cells adjacent to $\boldsymbol{x}^{(j)}$ and is 0 everywhere else. The resulting finite-dimensional function space $\tilde{\mathcal{U}}$ is the space of linear interpolators



(a) Choice of basis functions (b) Discretization of state and dynamics (c) Solving forward in time

Figure 2: Solving a PDE with the Galerkin method and method of lines consists of three steps.

between values at the vertices, see Fig. 2b. An important property is that if we expand $u \in \tilde{\mathcal{U}}$ in this basis, the value of u at the i-th node is just its i-th coefficient.

$$u(\boldsymbol{x}^{(i)}) = \sum\nolimits_{j=1}^{N} c_j \varphi^{(j)}(\boldsymbol{x}^{(i)}) = c_i \tag{4}$$

Galerkin Method. A piecewise linear approximation $u \in \tilde{\mathcal{U}}$ is not differentiable everywhere and therefore cannot fulfill Eq. (1) exactly. So instead of requiring an exact solution, we ask that the residual $R(u) = \partial_t u - F(t, \boldsymbol{x}, u, ...)$ be orthogonal to the approximation space $\tilde{\mathcal{U}}$ with respect to the inner product $\langle u, v \rangle_\Omega = \int_\Omega u(\boldsymbol{x}) \cdot v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$ at any fixed time t. In effect we are looking for the best possible solution within $\tilde{\mathcal{U}}$. Because $\tilde{\mathcal{U}}$ is generated by a finite basis, the orthogonality requirement decomposes into N equations, one for each basis function.

$$\langle R(u), v \rangle_\Omega = 0 \quad \forall v \in \tilde{\mathcal{U}} \qquad \Longleftrightarrow \qquad \langle R(u), \varphi^{(i)} \rangle_\Omega = 0 \quad \forall i = 1, ..., N \tag{5}$$

Plugging the residual back in and using the linearity of the inner product, we can reconstruct a system of equations that resemble the PDE that we started with.

$$\langle \partial_t u, \varphi^{(i)} \rangle = \langle F(t, \boldsymbol{x}, u, ...), \varphi^{(i)} \rangle_\Omega \quad \forall i = 1, ..., N \tag{6}$$

At this point we can stack the system of N equations into a vector equation. If we plug in the basis expansion $\sum_{j=1}^{N} c_j \varphi^{(j)}$ for u into the left hand side, we get a linear system

$$\boldsymbol{A} \partial_t \boldsymbol{c} = \boldsymbol{m} \tag{7}$$

where $A_{ij} = \langle \varphi^{(i)}, \varphi^{(j)} \rangle_\Omega$ is the so called mass matrix, $\boldsymbol{c}$ is the vector of basis coefficients of u, and $m_i = \langle F(t, \boldsymbol{x}, u, ...), \varphi^{(i)} \rangle_\Omega$ captures the effect of the dynamics F. The left hand side evaluates to $\boldsymbol{A} \partial_t \boldsymbol{c}$, because the basis functions are constant with respect to time. The right hand side cannot be further simplified without additional assumptions on F.

Method of Lines. If we can evaluate the right hand side $\boldsymbol{m}$, we can solve the linear system in Eq. (7) for the temporal derivatives of the coefficients of u at each point in time. In fact we have converted the PDE into a system of ordinary differential equations (ODEs) which we can solve with an arbitrary ODE solver given an initial value $\boldsymbol{c}^{(0)}$ as in Fig. 2c. This is known as the method of lines because we solve for u along parallel lines in time.

To find a vector field $u : [0, T] \times \Omega \to \mathbb{R}^m$ instead of a scalar field, we treat the m dimensions of u as a system of m scalar fields. This results in m copies of Eq. (7), which we need to solve simultaneously. Because the mass matrix $\boldsymbol{A}$ is constant with respect to u, we can combine the system into a matrix equation

$$\boldsymbol{A} \partial_t \boldsymbol{C} = \boldsymbol{M} \tag{8}$$

where $\boldsymbol{C}, \boldsymbol{M} \in \mathbb{R}^{N \times m}$ are the stacked $\boldsymbol{c}$ and $\boldsymbol{m}$ vectors, respectively. In summary, the spatial discretization with finite elements allows us to turn the PDE (1) into the matrix ODE (8).

2.2 Message Passing Neural Networks

Message-Passing Neural Networks (MPNNs) are a general framework for learning on graphs that encompass many variants of graph neural networks (Gilmer et al., 2017). It prescribes that nodes in a graph iteratively exchange messages and update their state based on the received messages for P

steps. For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$ and edges $\mathcal{E}$, and initial node states $\boldsymbol{h}_v^{(0)}\ \forall v \in \mathcal{V}$, the p-th propagation step is

$$\boldsymbol{h}_v^{(p)} = f_{\text{upd}}\left(\boldsymbol{h}_v^{(p-1)}, \sum\nolimits_{\{u,v\}\in\mathcal{E}} f_{\text{msg}}\left(\boldsymbol{h}_u^{(p-1)}, \boldsymbol{h}_v^{(p-1)}\right)\right), \tag{9}$$

where f_{msg} maps node states and edge attributes to messages and f_{upd} updates a node's state with the aggregated incoming messages. The final node states $\boldsymbol{h}_v^{(P)}$ can then be interpreted as per-node predictions directly or passed as node embeddings to downstream systems.

In this work, we employ a slight generalization of the above to undirected hypergraphs, i.e. graphs where the edges are sets of an arbitrary number of nodes instead of having a cardinality of exactly 2. For such a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$ and hyperedges $\varepsilon = \{u, v, w, ...\} \in \mathcal{E}$, and initial node states $\boldsymbol{h}_v^{(0)}\ \forall v \in \mathcal{V}$, the p-th propagation step is

$$\boldsymbol{h}_v^{(p)} = f_{\text{upd}}\left(\boldsymbol{h}_v^{(p-1)}, \sum\nolimits_{\substack{\varepsilon\in\mathcal{E}\\ \text{s.t.}\, v\in\varepsilon}} f_{\text{msg}}\left(\left\{\boldsymbol{h}_u^{(p-1)} \mid u \in \varepsilon\right\}\right)_v\right). \tag{10}$$

Note that f_{msg} jointly computes a separate message for each node v participating in a hyperedge ε.

3 Finite Element Networks

Consider a set of nodes $\mathcal{X} = \{\boldsymbol{x}^{(i)} \in \mathbb{R}^d\}_{i=1}^N$ representing N points in d-dimensional space. At each node we measure m features $\boldsymbol{y}^{(t_0,i)} \in \mathbb{R}^m$. Our goal is to predict $\boldsymbol{y}^{(t_j)}$ at future timesteps t_j by solving PDE (1). For it to be well-defined, we need a continuous domain encompassing the nodes $\mathcal{X}$.

By default, we construct such a domain by Delaunay triangulation. The Delaunay algorithm triangulates the convex hull of the nodes into a set of mesh cells – d-dimensional simplices – $\mathcal{T} = \{\Delta^{(i)} \subset \mathcal{X} \mid |\Delta^{(i)}| = d+1\}_{i=1}^{N_\mathcal{T}}$ which we represent as sets of nodes denoting the cell's vertices. The PDE will then be defined on the domain $\Omega = \cup_{\Delta\in\mathcal{T}} \text{CH}(\Delta)$. One shortcoming of this algorithm is that it can produce highly acute, sliver-like cells on the boundaries of the domain if interior nodes are very close to the boundary of the convex hull. We remove these slivers in a post-processing step, because they have small area and thus contribute negligible amounts to the domain volume, but can connect nodes which are far apart. See Appendix F for our algorithm. As an alternative to Delaunay triangulation, a mesh can be explicitly specified to convey complex geometries such as disconnected domains, holes in the domain and non-convex shapes in general.

As a first step towards solving the PDE, we need to represent $\boldsymbol{y}^{(t_0)}$ as a function on Ω. We define the function $u_0(\boldsymbol{x})_k = \sum_{i=1}^N y_k^{(t_0)} \varphi^{(i)}(\boldsymbol{x})$, such that its coefficients as a vector in the P1 basis encode the data and evaluating it at the nodes $u_0(\boldsymbol{x}^{(i)}) = \boldsymbol{y}^{(t_0,i)}$ agrees with the data by construction.

In Section 2.1, we have seen that solving PDE (1) approximately under an initial condition $u(t_0, \boldsymbol{x}) = u_0(\boldsymbol{x})$ can be cast as an initial value ODE problem over the trajectory of the coefficients of a solution in time. Above, we encoded the features $\boldsymbol{y}^{(t_0)}$ in the coefficients of u_0, meaning that Eq. (8) exactly describes the trajectory of the features through time. The dynamics of the features are thus given by

$$\boldsymbol{A}\partial_t \boldsymbol{Y}^{(t)} = \boldsymbol{M} \tag{11}$$

where $Y_{ik}^{(t)} = y_k^{(t,i)}$ is the feature matrix and $A_{ij} = \langle \varphi^{(i)}, \varphi^{(j)} \rangle_\Omega$ is the mass matrix. We call the matrix $\boldsymbol{M}$ on the right hand side the message matrix and it is given by

$$M_{ik} = \langle F\left(t, \boldsymbol{x}, u, ...\right)_k, \varphi^{(i)} \rangle_\Omega \tag{12}$$

where u encodes the data $\boldsymbol{Y}$ at time t in its coefficients.

Our goal is to solve the feature dynamics (11) with an ODE solver forward in time to compute the trajectory of $\boldsymbol{Y}$ and thereby predict $\boldsymbol{y}^{(t_j)}$. To apply an ODE solver, we have to compute $\partial_t \boldsymbol{Y}$ at each time t which in turn consists of, first, evaluating the right hand side and, second, solving the resulting sparse linear system.

Solving Eq. (11) for $\partial_t \boldsymbol{Y}$ with the exact sparse mass matrix $\boldsymbol{A}$ carries a performance penalty on GPUs, because sparse matrix solving is difficult to parallelize. To avoid this operation, we lump the

mass matrix. Lumping of the mass matrix is a standard approximation in FEM for time-dependent PDEs to reduce computational effort (Lapidus & Pinder, 1999), which leads to good performance in practice.

$$\tilde{\boldsymbol{A}}_{ii} = \sum\nolimits_{j=1}^{N} \langle \varphi^{(i)}, \varphi^{(j)} \rangle_{\Omega} \tag{13}$$

We employ the direct variant that diagonalizes the mass matrix by row-wise summation, making the inversion of $\boldsymbol{A}$ trivial.

The remaining step in solving Eq. (11) is evaluating the right hand side. Let $\mathcal{T}_i = \left\{ \Delta \in \mathcal{T} \mid \boldsymbol{x}^{(i)} \in \Delta \right\}$ be the set of all mesh cells that include node $\boldsymbol{x}^{(i)}$. Then we can break the elements of $\boldsymbol{M}$ up into the contributions from individual cells adjacent to each node.

$$M_{ik} = \langle F\left(t, \boldsymbol{x}, u, ...\right)_k, \varphi^{(i)} \rangle_{\Omega} = \sum\nolimits_{\Delta \in \mathcal{T}_i} \langle F\left(t, \boldsymbol{x}, u, ...\right)_k, \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)}. \tag{14}$$

The sum terms capture the effect that the dynamics have on the state of the system such as convection and chemical reaction processes given the geometry of the mesh cell, the values at the nodes, and the basis function to integrate against. However, the dynamics F are of course unknown and need to be learned from data.

Yet, if we were to introduce a deep model $f_\theta \approx F$ at this point, the model would be intolerably slow. Each evaluation of the model would require numerical integration of a neural network over each mesh cell against multiple basis functions and adaptive-step ODE solvers can require hundreds of evaluations. This would prohibit training even with the adjoint equation (Chen et al., 2018), because just a single solver step would incur too many forward passes.

We can eliminate this computational effort by assuming that F is cell-wise constant

$$F\left(t, \boldsymbol{x}, u, ...\right) = \sum\nolimits_{\Delta \in \mathcal{T}} F_\Delta\left(t, \Delta, \left\{u(\boldsymbol{x}) \mid \boldsymbol{x} \in \Delta\right\}, ...\right) \cdot \mathbb{I}_\Delta(\boldsymbol{x}), \tag{15}$$

where $\mathbb{I}_\Delta$ is the indicator function of cell Δ. We will omit the parameters of F_Δ for brevity from now on. Though this seems like a strong assumption at first, we still allow the dynamics to depend on the features, location, and time in arbitrary ways and the approximation becomes arbitrarily good as the mesh resolution increases. Under this constancy assumption, the message matrix simplifies to

$$M_{ik} = \sum\nolimits_{\Delta \in \mathcal{T}_i} F_{\Delta,k} \cdot \langle 1, \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)}. \tag{16}$$

Now we can introduce a deep model $f_\theta \approx F_\Delta$ and avoid the costly integration in training and application because the coefficients $\langle 1, \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)}$ can be precomputed once per domain.

3.1 Model

With these theoretical foundations in place, we can formally define **Finite Element Networks (FENs)**. Let $\mathcal{G} = (\mathcal{X}, \mathcal{T})$ be the undirected hypergraph with nodes $\mathcal{X}$ and hyperedges $\mathcal{T}$ corresponding to the meshing of the domain, so each mesh cell $\Delta \in \mathcal{T}$ forms a hyperedge between its vertices. Let f_θ be a neural network that estimates the cell-wise dynamics F_Δ from time, cell location, cell shape and the values at the vertices

$$\tilde{F}(\Delta) = f_\theta\left(t, \boldsymbol{\mu}_\Delta, \left\{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_\Delta \mid \boldsymbol{x}^{(i)} \in \Delta\right\}, \left\{\boldsymbol{y}^{(t,i)} \mid \boldsymbol{x}^{(i)} \in \Delta\right\}\right) \tag{17}$$

where $\boldsymbol{\mu}_\Delta$ is the center of cell Δ. We call $\tilde{F}(\Delta)$ the reaction term in reference to the convection-diffusion-reaction model, because it is completely free-form and can in principle model anything such as reactions between features or sources and sinks. f_θ conditions on the cell centers $\boldsymbol{\mu}_\Delta$ and the difference vectors $\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_\Delta$ separately to help the model distinguish between cell position and shape, and improve spatial generalization. Define the following message and update functions in the MPNN framework

$$f_{\mathrm{msg}}\left(\Delta\right)_{\boldsymbol{x}^{(i)}} = \tilde{F}(\Delta) \cdot \langle 1, \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)} \qquad f_{\mathrm{upd}}\left(\boldsymbol{x}^{(i)}, \boldsymbol{m}^{(i)}\right) = \tilde{\boldsymbol{A}}_{ii}^{-1} \boldsymbol{m}^{(i)} \tag{18}$$

where $\boldsymbol{m}^{(i)} = \sum_{\Delta \in \mathcal{T}} f_{\mathrm{msg}}(\Delta)_{\boldsymbol{x}^{(i)}}$ are the aggregated messages for node $\boldsymbol{x}^{(i)}$.

Performing one message passing step ($P = 1$) in this model is exactly equivalent to solving for the feature derivatives $\partial_t \boldsymbol{Y}^{(t)}$ in Eq. (11) with the lumped mass matrix from Eq. (13) and the simplified

message matrix in Eq. (16) with f_θ as a model for F_Δ. So making a forecast with FEN for T points in time $\boldsymbol{t} = (t_0, t_1, ..., t_T)$ based on a sparse measurement $\boldsymbol{y}^{(t_0)}$ means solving an ODE where at each evaluation step t_e we run one message passing step in the MPNN described above to compute the current time derivatives of the features $\partial_t \boldsymbol{y}|_{t=t_e}$.

$$\hat{\boldsymbol{y}}^{(t_0,t_1,...,t_T)} = \text{ODESolve}(\boldsymbol{y}^{(t_0)}, \partial_t \boldsymbol{y}, \boldsymbol{t}) \tag{19}$$

Because FENs model the continuous-time dynamics of the data, they can also be trained on and make predictions at irregular timesteps.

By controlling the information available to f_θ, we can equip the model with inductive biases. If we omit the current time t, the learned dynamics become necessarily autonomous. Similarly, if we do not pass the cell position $\boldsymbol{\mu}_\Delta$, the model will learn stationary dynamics.

An orthogonal and at least as powerful mechanism to induce inductive biases into the model is through assumptions on the form of the unknown dynamics F. Let us assume that the dynamics are not completely unknown, but that a domain expert told us that the generating dynamics of the data include a convective component. So the dynamics for feature k are of the form

$$F\left(t, \boldsymbol{x}, u, ...\right)_k = -\nabla \cdot \left(v^{(k)}(t, \boldsymbol{x}, u, ...)u_k\right) + F'\left(u, \partial_{\boldsymbol{x}} u, ...\right)_k \tag{20}$$

where $v^{(k)}(t, \boldsymbol{x}, u, ...) \in \mathbb{R}^d$ is the velocity field of that feature and F' represents the still unknown remainder of the dynamics. The only way to reconcile the convective assumption with the basic assumption of cell-wise constancy of F, is to conclude that the velocity field is likewise constant on each cell.

By following the same steps with the convection term $-\nabla \cdot \left(v^{(k)}(t, \boldsymbol{x}, u, ...)u_k\right)$ as for the unknown dynamics, we arrive at a specialized term for modeling convection. Let g_θ be a neural network that estimates the cell-wise per-attribute flow field

$$\tilde{v}(\Delta) = g_\theta\left(t, \boldsymbol{\mu}_\Delta, \left\{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_\Delta \mid \boldsymbol{x}^{(i)} \in \Delta\right\}, \left\{\boldsymbol{y}^{(t,i)} \mid \boldsymbol{x}^{(i)} \in \Delta\right\}\right) \in \mathbb{R}^{m \times d}. \tag{21}$$

We derive the corresponding message function in Appendix A.

$$f^v_{\text{msg}}\left(\Delta\right)_{\boldsymbol{x}^{(i)}} = \sum\nolimits_{\boldsymbol{x}^{(j)} \in \Delta} \boldsymbol{y}^{(t,j)} \odot \left(\tilde{v}(\Delta) \cdot \langle \varphi^{(j)}, \nabla \varphi^{(i)} \rangle_{\text{CH}(\Delta)}\right) \tag{22}$$

With this we can extend FEN with specialized transport-modeling capabilities by adding the two message functions and define **Transport-FEN (T-FEN)** as a FEN with the message function

$$f^{\text{TFEN}}_{\text{msg}}\left(\Delta\right)_{\boldsymbol{x}^{(i)}} = f_{\text{msg}}\left(\Delta\right)_{\boldsymbol{x}^{(i)}} + f^v_{\text{msg}}\left(\Delta\right)_{\boldsymbol{x}^{(i)}}. \tag{23}$$

Consequently, T-FEN learns both a velocity field to capture convection and a reaction term that fits the unknown remainder of the dynamics.

Network Architecture. We instantiate both f_θ and g_θ as multilayer perceptrons (MLPs) with $\tanh$ non-linearities. The input arguments are concatenated after sorting the cell vertices by the angle of $\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_\Delta$ in polar coordinates. This improves generalization because f_θ and g_θ do not have to learn order invariance of the nodes. Both MLPs have an input dimension of $1 + d + (d+1) \cdot (m+d)$ in the non-autonomous, non-stationary configuration since $|\Delta| = d+1$. The reaction term f_θ computes one coefficient per cell and attribute and therefore has an output dimension of m whereas the transport term estimates a velocity vector for each cell and attribute resulting in an output dimension of md. The number of hidden layers and other configuration details for each experiment are listed in Appendix C. We use `scikit-fem` (Gustafsson & McBain, 2020) to compute the inner products between basis functions over the mesh cells and the dopri5 solver in `torchdiffeq` (Chen et al., 2018) to solve the resulting ODE.

Training. We train our models with the Adam optimizer (Kingma & Ba, 2015) by minimizing the multi-step prediction error in terms of the mean L_1 distance between predicted and actual node features

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{j=1}^{T} \|\hat{\boldsymbol{y}}^{(t_j,i)} - \boldsymbol{y}^{(t_j,i)}\|_1 \tag{24}$$

where $\hat{\boldsymbol{y}}^{(t_j,\cdot)}$ is the prediction at time t_j. To accumulate gradients, we backpropagate through the operations of the ODE solver.

We observed that training our models to predict complete sequences from the beginning can get them stuck in local minima that take many epochs to escape. To stabilize training, we begin by training on subsequences of length $s = 3$ and iteratively increase s by 1 per epoch until it reaches the maximum training sequence length.

4 Related Work

Spatio-Temporal Forecasting. Various DNNs, and in particular GNNs, have been designed to model complex spatial and temporal correlations in data, with applications ranging from forecasting traffic flow (see (Jiang & Luo, 2021) and (Yin et al., 2021) for a comprehensive survey), to forecasting COVID-19 (Kapoor et al., 2020) and sea-surface temperature (de Bézenac et al., 2018). For a detailed review of both classical and deep learning approaches, which can be roughly categorized as based on point-cloud trajectories, regular grids, or irregular grids, see Shi & Yeung (2018). As one representative of GNN-based methods, which tend to be SOTA, we compare FENs with Graph WaveNet (GWN) (Wu et al., 2019). GWN uses a combination of graph convolution and 1D dilated temporal convolutions to learn spatio-temporal properties of traffic networks.

Neural Networks and Differential Equations. In 1998 Lagaris et al. (1998) introduced the first neural network approach to solve ODEs and PDEs. They formulate the optimization goal as fulfilling the properties of the differential equation. Raissi et al. (2019) extend this idea and propose a framework to respect any given general nonlinear PDE for continuous- and discrete-time models. Another neural approach to learn PDEs from data is PDE-Net (Long et al., 2018). Its two main components are: learning the differential operators of PDEs through convolutional kernels and applying a neural network to approximate the nonlinear response functions. Building on that work, PDE-Net 2.0 (Long et al., 2019) uses a symbolic network to approximate the nonlinear response function. de Bézenac et al. (2018) derive a model from the convection-diffusion equation and estimate the flow field on a regular grid with a CNN. Chen et al. (2018) leverage ODE solvers in their neural architecture to create a model family of continuous depth called neural ODEs. Ayed et al. (2019) build upon neural ODEs to make spatio-temporal forecasts by learning the dynamics of all data points jointly with the structure of the domain encoded via the node positions as additional features. GNODE (Poli et al., 2019) brings the framework of neural ODEs of Chen et al. (2018) into the graph domain and provides continuous depth GNNs to make traffic predictions. In dynamical systems the variable depth component coincides with the time dimension which allows the network to work with observations at varying timesteps. Iakovl et al. (2021) extend GNODE with distance vectors between nodes as edge features and use it for PDE prediction. Mitusch et al. (2021) derive a model from FEM similar to our work but integrate the learned dynamics against the basis functions numerically and disregard the mesh to avoid designing the discretization into the model.

5 Experiments

Baselines. We evaluate FEN and T-FEN against a selection of models representing a variety of temporal prediction mechanisms: GWN combines temporal and graph convolutions (Wu et al., 2019); Physics-aware Difference Graph Network (PA-DGN) estimates spatial derivatives as additional features for a recurrent graph network (Seo et al., 2020); the continuous-time MPNN-based model by (Iakovl et al., 2021) which we will call CT-MPNN, uses a general MPNN to learn the continuous-time dynamics of the data.[1] We describe each model's configuration details in Appendix C.

GWN and PA-DGN use 12 and 5 timesteps respectively as input while the ODE-based models make their predictions based on a single timestep. To ensure a fair comparison, we do not evaluate the models on the first 12 timesteps of the test set.

Datasets. For our experiments we have chosen two real-world datasets. The Black Sea dataset provides data on daily mean sea surface temperature and water flow velocities on the Black Sea over several years. ScalarFlow is a collection of 104 reconstructions of smoke plumes from 2 seconds long, multi-view camera recordings. In each recording a fog machine releases fog over a heating element in the center of the domain which then rises up with the hot air through convection and buoyancy.

[1] GWN and PA-DGN condition their predictions on multiple timesteps, giving them a possible advantage over ODE-based models since they can use the additional input for denoising or to infer latent temporal information.

Table 1: Mean absolute error and the number of function evaluations for continuous-time models on 10-step ahead prediction on the test set. We subsampled both datasets to 1000 sparsely distributed nodes and trained each model three times.

	Black Sea		ScalarFlow	
	MAE	NFE	MAE	NFE
PA-DGN	0.996 ± 0.001	-	0.130 ± 0.003	-
GWN	0.983 ± 0.006	-	0.102 ± 0.000	-
CT-MPNN	0.944 ± 0.003	159.727 ± 24.119	0.093 ± 0.001	128.293 ± 59.879
FEN	$\mathbf{0.938 \pm 0.005}$	119.808 ± 7.077	0.092 ± 0.001	147.375 ± 43.691
T-FEN	$\mathbf{0.934 \pm 0.002}$	117.151 ± 10.130	$\mathbf{0.090 \pm 0.000}$	111.651 ± 3.459

All our experiments are run on hold-out datasets, i.e. the year 2019 of the Black Sea data and the last 20 recordings of ScalarFlow. See Appendix B for a detailed description of the datasets, our sparse subsampling procedure, choice of normalization, and train-val-test splits.

Multi-step Forecasting. The main objective of our models as well as the baselines are accurate forecasts and we train them to minimize the mean absolute error over 10 prediction steps. We chose 10 steps because it is long enough that a model needs to learn non-trivial dynamics to perform well and more accurate dynamics lead to better performance. At the same time it is short enough that not too much uncertainty accumulates. For example, if we train any of the models on Black Sea on much longer training sequences, they learn only diffusive smoothing because the correlation between the temperature distribution today and in 30 days is basically nil, so that the only winning move is not too play – similar to how forecasting the weather for more than 10 days is basically impossible (Mailier, 2010).

For Table 1 we have evaluated the models on all possible subsequences of length 10 of the respective test set of both datasets. The results show that FEN and T-FEN provide the smallest prediction error on both datasets with a further boost due to the separate transport term in T-FEN. Additionally, T-FEN learns dynamics that are simpler to solve and require fewer function evaluations than the more general FEN and CT-MPNN models. These results highlight that including prior knowledge in the model is a key to increased performance in accuracy as well as efficiency.

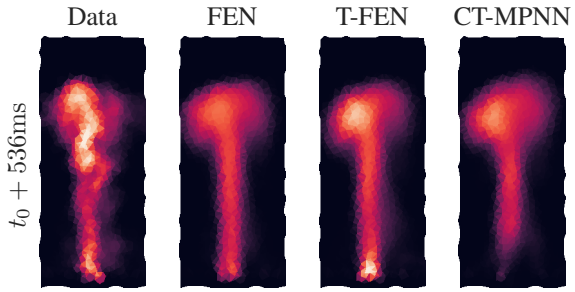


Figure 3: Evolving the learned dynamics of FEN models and the strongest baseline trained on 10 steps forward for 50 steps reveals the differences between these models to the human eye.

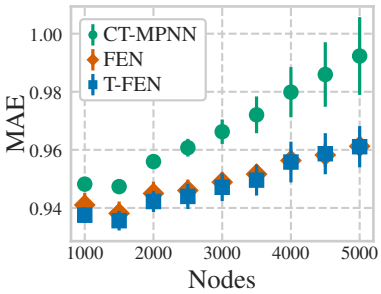


Figure 4: Predictive accuracy of models trained on 1000 nodes and evaluated on increasingly fine subsamplings of the test data.

Extrapolation. While we train the models on 10-step forecasting, it is instructive to compare their predictions on longer time horizons. Fig. 3 shows the predictions of the three strongest models on a smoke plume after 50 steps. Because these models are all ODE-based, they were conditioned on a single timestep and have been running for hundreds of solver steps at this point. First, we notice that all three models have the fog rise at about the same, correct speed. However, with CT-MPNN the fog disappears Extrapolationat the bottom because this model is by construction unable to learn non-stationary dynamics. FEN avoids this problem but overall the predicted density of the rising fog is more uniform than we see in the data. T-FEN with its separate transport term correctly predicts the

increased density at the bottom over the fog inlet as well as the rise of a high-density region without too much smoothing. See Appendix E for a comparison with all baselines at multiple time steps.

Super-Resolution. Graph-based spatio-temporal forecasting models make predictions on all nodes jointly. Training these models on many points or fine meshes therefore requires a prohibitive amount of memory to store the data points and large backwards graphs for backpropagation. One way around this problem is to train on coarse meshes and later evaluate the models on the full data. In Fig. 4 we compare FEN and T-FEN against the strongest baseline on the Black Sea dataset. While all three models increase in prediction error as the mesh resolution increases, FEN models deteriorate at a markedly slower rate than CT-MPNN. We ascribe this to the fact that in FEN and T-FEN estimation of the dynamics and its effect on the nodes are to a certain extent separated. The terms f_θ and g_θ estimate the dynamics on each cell, but the effect on the node values is then controlled by the inner products of basis functions which incorporate the shape of the mesh cells in a theoretically sound way. In CT-MPNN on the other hand, the model learns these aspects jointly which makes it more difficult to generalize to different mesh structures. See Appendix D for the complete results on both datasets and a visual comparison of low and high resolution data.

Interpretability. In FEN the reaction term f_θ models the dynamics F as a black box and as such its activations are equally opaque as the dynamics we derived it from. However, as we have seen in Section 3.1, by making assumptions on the structure of F we impose structure onto the model and this structure in turn assigns meaning to the model's components. For a T-FEN we see that the transport term corresponds to a velocity field for each attribute, while the reaction term absorbs the remainder of the dynamics. Fig. 5 shows that this disentanglement of dynamics works in practice. In the ScalarFlow data we have a fog inlet at the bottom, i.e. a source of new energy, and from there the fog rises up by convection. Plotting the contributions of f_θ and g_θ to $\partial_t \boldsymbol{y}$ separately shows that the model has learned that all fog is transported upwards and that there is a source at the bottom. Instead of inspecting the contributions to $\partial_t \boldsymbol{y}$, we can also investigate the learned parameter estimates directly. Fig. 6 shows that the transport term learns a smooth velocity field for the sea surface temperature on the Black Sea.

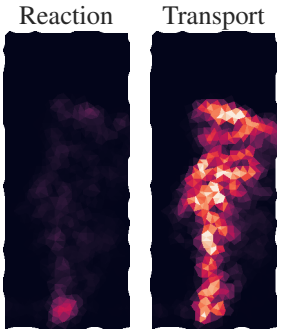


Figure 5: Contribution to $\partial_t \boldsymbol{y}$ from the reaction and transport term in a T-FEN on a snapshot of ScalarFlow.

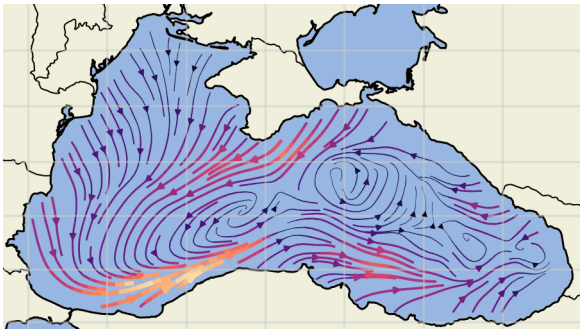


Figure 6: Learned temperature flow field in a T-FEN after 10 steps on Black Sea data. The model recognized the physical relationship between the features.

6 Conclusion

We have introduced Finite Element Networks as a new spatio-temporal forecasting model based on the established FEM for the solution of PDEs. In particular, we have shown how FENs can incorporate prior knowledge about the dynamics of the data into their structure through assumptions on the form of the underlying PDE and derived the specialized model T-FEN for data with convection dynamics. Our experiments have shown that FENs learn the short-term dynamics of sea surface temperature and gas flow more accurately than baseline models from three spatio-temporal forecasting model classes and generalize better to higher resolution meshes at test time. The T-FEN variant provides even stronger performance and additionally allows meaningful introspection of the model. Its structure is directly derived from the assumed form of the underlying PDE and as such its transport and reaction terms disentangle the dynamics into a convection component and the remainder such as inlets and sinks.

ETHICS STATEMENT

We propose a general model for spatio-temporal forecasting which can be applied in a variety of settings. Among these are traffic and crowd flow predictions which have the potential for abuse. However, due to the close connection to PDEs, we see it primarily as a model for physical systems in earth sciences and engineering, with impacts in the study of climate change and understanding of complex systems. Finally, we can also imagine its application in modeling the spread of infectious diseases which also employs PDEs today (Viguerie et al., 2021).

REPRODUCIBILITY

To maximize the reproducibility of our experimental results, we have fixed seeds for every domain subsampling, training run and evaluation and recorded these in configuration files, so that everything can be re-run. We obtained all seed values from `numpy.random.SeedSequence` to guarantee that the streams of random numbers from the random number generators (RNGs) are independent and have high sample quality. While we fixed all seeds, model training is still non-deterministic because the message passing step in graph-based models relies on a scatter operation. Scattering is implemented via atomic operations on GPUs, which can re-order floating point operations such as additions between runs with the same seed inducing non-determinism (Fey & Lenssen, 2019). Nonetheless, we have observed that multiple runs from the same seed only diverge slowly as training progresses and the variability in performance between runs from different seeds is small for all models anyway as can be seen in the standard deviation of the mean absolute error over three runs in Table 1.

We will provide the source code for our models as well as the baselines, experiment configurations and a pre-processed version of the raw Black Sea data to the reviewers through OpenReview at the beginning of the discussion period.

We have used an NVIDIA GeForce GTX 1080 Ti for all experiments and evaluations.

REFERENCES

William F Ames. *Numerical methods for partial differential equations*. Academic press, 2014.

Ibrahim Ayed, Emmanuel de Bézenac, Arthur Pajot, Julien Brajard, and Patrick Gallinari. Learning dynamical systems from partial observations. *CoRR*, abs/1902.11136, 2019.

Filipe de Avila Belbute-Peres, Thomas Economon, and Zico Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *International Conference on Machine Learning*, pp. 2402–2411. PMLR, 2020.

Susanne C Brenner, L Ridgway Scott, and L Ridgway Scott. *The mathematical theory of finite element methods*, volume 3. Springer, 2008.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31*, pp. 6572–6583, 2018.

Jean Coiffier. *Fundamentals of numerical weather prediction*. Cambridge University Press, 2011.

Richard Courant and David Hilbert. *Methods of Mathematical Physics: Partial Differential Equations*. John Wiley & Sons, 2008.

Emmanuel de Bézenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net, 2018.

Marie-Lena Eckert, Kiwon Um, and Nils Thuerey. Scalarflow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Trans. Graph.*, 38(6):239:1–239:16, 2019.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 2017.

Tom Gustafsson and G. D. McBain. scikit-fem: A python package for finite element assembly. *Journal of Open Source Software*, 5(52):2369, 2020. doi: 10.21105/joss.02369. URL `https://doi.org/10.21105/joss.02369`.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3): 90–95, 2007. doi: 10.1109/MCSE.2007.55.

Valerii Iakovl, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time pdes from sparse data with graph neural networks. In *International Conference on Learning Representations*, 2021.

Heiner Igel. *Computational Seismology: A Practical Introduction*. Oxford University Press, 2017.

Victor Isakov. *Inverse problems for partial differential equations*, volume 127. Springer, 2006.

Weiwei Jiang and J. Luo. Graph neural network for traffic forecasting: A survey. *ArXiv*, abs/2101.11174, 2021.

Amol Kapoor, Xue Ben, Luyang Liu, Bryan Perozzi, Matt Barnes, Martin Blais, and Shawn O'Banion. Examining covid-19 forecasting using spatio-temporal graph neural networks. *arXiv preprint arXiv:2007.03113*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

Georg Kohl, Kiwon Um, and Nils Thuerey. Learning similarity metrics for numerical simulations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5349–5360. PMLR, 2020.

Isaac Lagaris, Aristidis Likas, and Dimitrios Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *Neural Networks, IEEE Transactions on*, pp. 987 – 1000, 10 1998.

Leon Lapidus and George F Pinder. *Numerical solution of partial differential equations in science and engineering*. John Wiley & Sons, 1999.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net, 2018.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3214–3222. PMLR, 2018.

Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *ArXiv*, abs/1812.04426, 2019.

Pascal J Mailier. Can we trust long-range weather forecasts? *Management of weather and climate risk in the energy industry*, pp. 227–239, 2010.

Met Office. *Cartopy: a cartographic python library with a matplotlib interface*. Exeter, Devon, 2010 - 2015. URL http://scitools.org.uk/cartopy.

Sebastian K. Mitusch, Simon W. Funke, and Miroslav Kuchta. Hybrid FEM-NN models: Combining artificial neural networks with the finite element method. *CoRR*, abs/2101.00962, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *CoRR*, abs/1911.07532, 2019.

Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.

Sungyong Seo, Chuizheng Meng, and Yan Liu. Physics-aware difference graph networks for sparsely-observed dynamics. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, 2020.

Xingjian Shi and D. Yeung. Machine learning for spatiotemporal sequence forecasting: A survey. *ArXiv*, abs/1808.06865, 2018.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems 28*, pp. 802–810, 2015.

Sam Shleifer, Clara McCreery, and Vamsi Chitters. Incrementally improving graph wavenet performance on traffic prediction. *CoRR*, abs/1912.07390, 2019.

Alex Viguerie, Guillermo Lorenzo, Ferdinando Auricchio, Davide Baroli, Thomas JR Hughes, Alessia Patton, Alessandro Reali, Thomas E Yankeelov, and Alessandro Veneziani. Simulating the spread of covid-19 via a spatially-resolved susceptible–exposed–infected–recovered–deceased (seird) model with heterogeneous diffusion. *Applied Mathematics Letters*, 111:106617, 2021.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pp. 1907–1913. ijcai.org, 2019.

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

Xueyan Yin, Genze Wu, Jinze Wei, Yanming Shen, Heng Qi, and Baocai Yin. Deep learning on traffic prediction: Methods, analysis and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In Jérôme Lang (ed.), *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pp. 3634–3640. ijcai.org, 2018.

Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In Satinder P. Singh and Shaul Markovitch (eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 1655–1661. AAAI Press, 2017.

A Derivation of the Transport Term

Let F be just a convection term. Because of the linearity of the inner product $\langle \cdot, \cdot \rangle_\Omega$, we can ignore other additive components of F in this derivation.

$$F\left(t, \boldsymbol{x}, u, ...\right)_k = -\nabla \cdot \left(v^{(k)}(t, \boldsymbol{x}, u, ...) u_k\right) \tag{25}$$

We begin by plugging this into the message matrix.

$$M_{ik} = -\langle \nabla \cdot \left(v^{(k)}(t, \boldsymbol{x}, u, ...) u_k\right), \varphi^{(i)} \rangle_\Omega \tag{26}$$

Next, we split the domain into the mesh cells.

$$= -\sum\nolimits_{\Delta \in \mathcal{T}_i} \langle \nabla \cdot \left(v^{(k)}(t, \boldsymbol{x}, u, ...) u_k\right), \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)} \tag{27}$$

Assuming the velocity field to be cell-wise constant, we can pull it out of the inner product.

$$= -\sum\nolimits_{\Delta \in \mathcal{T}_i} v_{\Delta,k} \cdot \langle \nabla u_k, \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)} \tag{28}$$

Now we expand u in the P1 basis (remember that we have encoded the data $\boldsymbol{y}$ in the coefficients of u)

$$= -\sum\nolimits_{\Delta \in \mathcal{T}_i} v_{\Delta,k} \cdot \langle \nabla \sum\nolimits_{j=1}^{N} y_k^{(j)} \varphi^{(j)}, \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)} \tag{29}$$

and make use of the linearity of the inner product once more.

$$= -\sum\nolimits_{\Delta \in \mathcal{T}_i} \sum\nolimits_{j=1}^{N} y_k^{(j)} v_{\Delta,k} \cdot \langle \nabla \varphi^{(j)}, \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)} \tag{30}$$

Next, we use Gauss' theorem and a 0-Neumann boundary condition to move the gradient over to $\varphi^{(i)}$ and absorb the negative sign.

$$= \sum\nolimits_{\Delta \in \mathcal{T}_i} \sum\nolimits_{j=1}^{N} y_k^{(j)} v_{\Delta,k} \cdot \langle \varphi^{(j)}, \nabla \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)} \tag{31}$$

Because the inner product is restricted to Δ, we can restrict the inner sum to vertices of Δ.

$$= \sum\nolimits_{\Delta \in \mathcal{T}_i} \sum\nolimits_{\boldsymbol{x}^{(j)} \in \Delta} y_k^{(j)} v_{\Delta,k} \cdot \langle \varphi^{(j)}, \nabla \varphi^{(i)} \rangle_{\mathrm{CH}(\Delta)} \tag{32}$$

If we stack this expression over all features k and translate the result into the MPNN framework, we arrive at the transport term that we present as part of T-FEN in Section 3.1.

B Datasets

In this section, we give a brief overview over each of the datasets we used in Section 5 and how we pre-processed them.

Subsampling. Both datasets that we use provide large amounts of dense data on a 2D or 3D grid, which we subsample to simulate sparse measurements of a physical system and make the amount of data per timestep manageable. A non-trivial problem is the selection of sampling points and we decided to use the k-Medoids algorithm. This way we ensure that we use actual data points as sampling points and reach a roughly uniform but still random cover of the domain. A uniform covering is especially important for the Black Sea dataset because of the non-convex shape of the domain and small bays on the border that we still want to cover.

B.1 Black Sea

The Black Sea data is a reanalysis dataset of daily and monthly mean fields in the Black Sea of variables such as temperature, flow velocity, and salinity and 31 depth levels from 01/01/1992. The whole dataset is available as the BLKSEA_MULTIYEAR_PHY_007_004[2] product from the EU

[2]https://resources.marine.copernicus.eu/product-detail/BLKSEA_MULTIYEAR_PHY_007_004

Copernicus Marine Service under a custom license[3] granting permission to freely use the data in research under the condition of crediting the Copernicus Marine Environment Monitoring Service.

The spatial resolution of the raw data is 1/27° × 1/36° on a regular mesh of size 395×215 covering the Black Sea. Of the resulting 84,925 grid points, about 40,000 contain actual sea data and we subsample these valid points to create our sparse datasets. We mesh the chosen sample points with the Delaunay algorithm and filter acute mesh cells on the boundary as described in Appendix F. Additionally, we remove mesh cells that cover more land than water to avoid connecting nodes that are actually far apart within the domain of the water.

We use daily data from 01/01/2012 to 31/12/2017 for training, 01/01/2018 to 31/12/2018 for validation and 01/01/2019 to 31/12/2019 for testing. As features we choose the zonal velocities and the water temperature. Both features are taken at a depth of 12.54m because the dynamics at the surface are strongly influenced by wind, sun, and cloud cover, which are not part of the data and thus make the dynamics non-deterministic from the perspective of the model.

We normalize the node positions by their mean and average standard deviation in latitudinal and longitudinal direction. This way we ensure that the aspect ratio stays constant and mesh cells are not deformed. The velocities are normalized by their global mean and standard deviation over the training date range. For the temperature, we normalize each node with its daily mean over the training date range and daily standard deviation of all nodes in the training date range. We choose separate normalization schemes for the following reasons. Assume that the flow field would be perfectly stable. If we were to normalize node-wise, all normalized velocity features would be zero and no longer informative for the model. For the temperature, we chose node-wise normalization to emphasize modeling the dynamics of the heat that is in excess of the daily mean temperature field on each day. This puts more emphasis on accurate modeling of individual heat pockets and helps distinguish models that can learn this from models that mostly work through diffusion such as PA-DGN.

B.2 SCALARFLOW

ScalarFlow is a dataset of 104 smoke plumes published by Eckert et al. (2019) under the Creative Commons license CC-BY-NC-SA 4.0. (Eckert et al., 2019). Eckert et al. created a controlled environment in which fog from a fog machine rises in an air column over a heating element. They captured the resulting flow on video with a calibrated multi-camera setup and reconstructed the velocity and density fields of each recording on a dense 100×178×100 grid for 150 timesteps with their proposed method. At each grid point, the dataset provides 3-dimensional velocity vectors and the current fog density. The resulting dataset represents high-resolution measurements of a dynamical physical system that is mostly driven by buoyancy and transport processes but also evokes turbulence.

Of the 104 recordings we use the first 64 for training, the next 20 for validation and the final 20 as a test set. Before subsampling the grid points, we reduce the data to 2D by averaging over the depth (Kohl et al., 2020) and restrict the data to the central 60 points in x-direction and bottom 140 points in y-direction, because the fog does not reach points outside of that central box in almost any of the recordings. We normalize the coordinates of the sample points by their mean and average of the variances in x and y direction. This way we ensure that the aspect ratio of the domain does not change. We do not normalize the features.

C MODEL CONFIGURATIONS

C.1 FEN & T-FEN

Both f_θ and g_θ are MLPs with 4 hidden layers and $\tanh$ non-linearities. In FEN the layers of f_θ have a width of 128. For T-FEN we chose to reduce the width of the hidden layers of both f_θ and g_θ to 96, so that both FEN and T-FEN have a comparable number of parameters and we can trace back any difference in performance to difference in model structure and not capacity. See Table 2 for the parameter counts. We solve the feature dynamics with a `dopri5` solver with an absolute tolerance and relative tolerance of 10^{-6} on Black Sea and 10^{-7} on ScalarFlow. The models are non-autonomous and non-stationary on Black Sea; and non-stationary but autonomous on ScalarFlow.

[3] https://marine.copernicus.eu/user-corner/service-commitments-and-licence

On Black Sea, the models learn time-dependent dynamics and represent the yearly periodicity of the data, we map the absolute timestamp into yearly time values where 0 represents the 1st of January and 1 the 31st of December. On initialization, we set the last weights and bias of the last layers of both MLPs to 0 to start with the constant dynamics as a reasonable initialization for training. For parameter learning, we use the Adam optimizer with a learning rate of 10^{-3} (Kingma & Ba, 2015).

We always use a batch size of 6.

C.2 Baselines

CT-MPNN. Following Iakovl et al. (2021), our CT-MPNN implementation models both the message function and the update function as MLPs with $\tanh$ nonlinearities. Contrary to the default configuration used by Iakovl et al. (2021), we increased the widths of the MLPs from 60 to 128 and the message dimension from 40 to 128 to make the model capacity comparable to the other baselines. Instead of an implicit Adams-Bashforth solver we used the same `dopri5` solver as for FEN and T-FEN with an absolute tolerance of 10^{-6} and a relative tolerance of 10^{-7}. This both increased the models performance and avoided frequent convergence issues in the step function of the Adams-Bashforth solver. Finally, we exchanged the RPROP optimizer with a standard Adam optimizer and we compute the gradients the same in FEN and T-FEN by backpropagation through the ODE solver. The batch size is 6.

Table 2: Parameter counts for all models.

	Black Sea	ScalarFlow
PA-DGN	269140	269140
GWN	459750	466160
CT-MPNN	134403	134532
FEN	53129	53772
T-FEN	60975	61844

PA-DGN. For PA-DGN we choose two graph network layers of two layers each with a hidden size of 64. We condition the model on 5 timesteps. To represent the structure of the domain, we pass the 4-nearest neighbor graph of the nodes to the model following authors of the method (Seo et al., 2020). The batch size is 3 on both datasets.

GWN. For GWN we use the improved configuration described by Shleifer et al. (2019) with 40 convolution channels, extra skip connections and four of GWN's WaveNet inspired blocks with two layers each. The model is conditioned on 12 time steps of data and predicts 10 steps ahead. For the long-term extrapolation in Appendix E we apply the model auto-regressively to get predictions for more than 10 steps. We do not learn an adaptive adjacency matrix because that comes with a quadratic memory cost in the number of nodes and Wu et al. have shown that GWN only performs 1.2 % worse on METR-LA without the adaptive adjacency matrix. For the computation of the forward and backward transition matrices, we pass in the adjacency matrix derived from the mesh of the domain where the edges are weighted by the Gaussian kernel

$$e(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\frac{\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|_2^2}{\sigma^2}\right) \tag{33}$$

where σ is the standard deviation of $\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|_2$. The batch size is 6.

D Super-Resolution

Inspecting the full results in Fig. 7 reveals two things. While FEN based models perform best in super-resolution on both datasets, GWN and PA-DGN also only deteriorate slightly on finer meshes. However, this is offset by the fact that these models did not achieve such a good fit to begin with and are still out-performed by FEN-based models on every resolution. CT-MPNN achieved performance comparable to FEN and T-FEN on the original mesh but generalizes markedly worse to finer meshes. We suppose that this is due to the fact that CT-MPNN uses a general MPNN internally, while FEN and T-FEN have an inductive bias towards producing physically sensible predictions because of their structural connection to FEM and PDEs.

If we compare coarse and fine subsamplings of the two datasets in Fig. 8 visually, it is apparent that quintupling the number of nodes on ScalarFlow increases the resolution but the data is still qualitatively the same. On Black Sea on the other hand, the higher resolution plot reveals small-scale vortices that are lost at 1000 nodes. We believe that this transition between two qualitatively different

regimes is the reason why we observe a stronger deterioration on the Black Sea dataset than on ScalarFlow in Fig. 7.

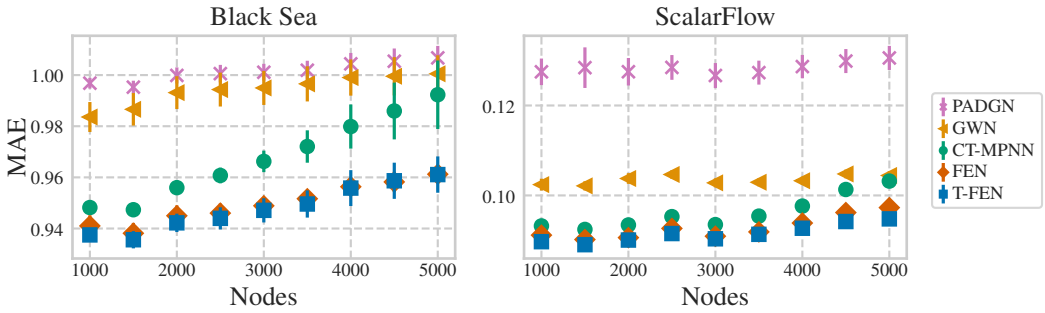


Figure 7: All models were trained on 1000 node subsamples of the data for 10-step prediction and then evaluated on the same task but on increasingly fine meshings of the domain.

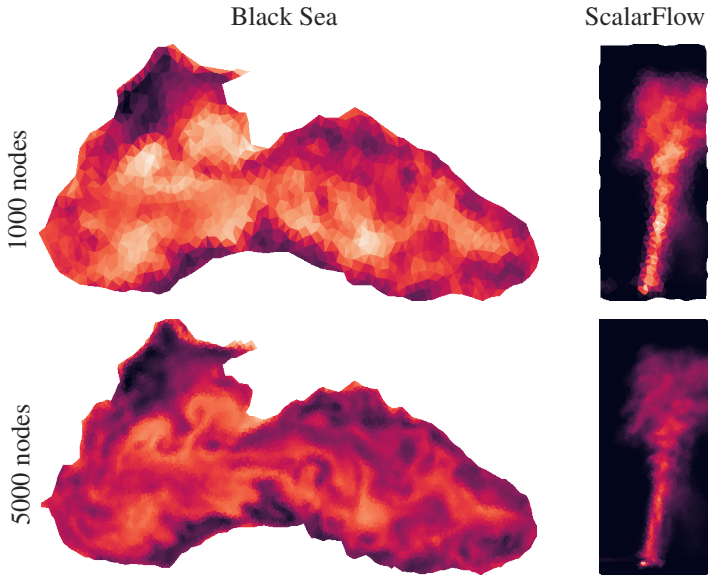


Figure 8: A visual comparison of the data when subsampled at 1000 nodes and at 5000 nodes.

E Long Term Extrapolation

In Fig. 9 we show 60-step extrapolations of the all models trained on 10-step forecasting. The first thing to note is that PA-DGN struggles with the directionality inherent in the ScalarFlow dataset. While its prediction evolves over time, it exhibits a mostly diffusive dynamic.

From a visual standpoint, GWN produces the most realistic looking forecasts with fine details and sharp contrasts, especially compared to the ODE-based models that beat GWN in short-term forecasting. We see two reasons for that. First, ODE-based models that learn the data dynamics directly, i.e. all three of FEN, T-FEN, and CT-MPNN, have to work with a tight information bottleneck. Because these models carry information through time only in the features themselves and have no latent state, they can condition their prediction only on a single time step and this bottleneck makes it difficult to conserve fine details over time. GWN on the other hand predicts 10 timesteps at a time and conditions each forecast on the past 12 timesteps. In this way, GWN can extract more information from the input data and also carry that information forward 10 steps at a time whereas ODE-base models have to conserve the information through roughly 100 solver steps to reach t_{10}, see the number of function evaluations in Table 1.

Second, FEN, T-FEN, and CT-MPNN are ODE models and as such are biased towards smooth predictions over time and over long time frames this leads to a smoothing in space. Yet, on short time frames, before smoothing sets in, these models are able to model these physical processes more accurately, because ODEs are natural models for these data. For a discussion of the differences between the FEN, T-FEN, and CT-MPNN predictions, see Section 5.

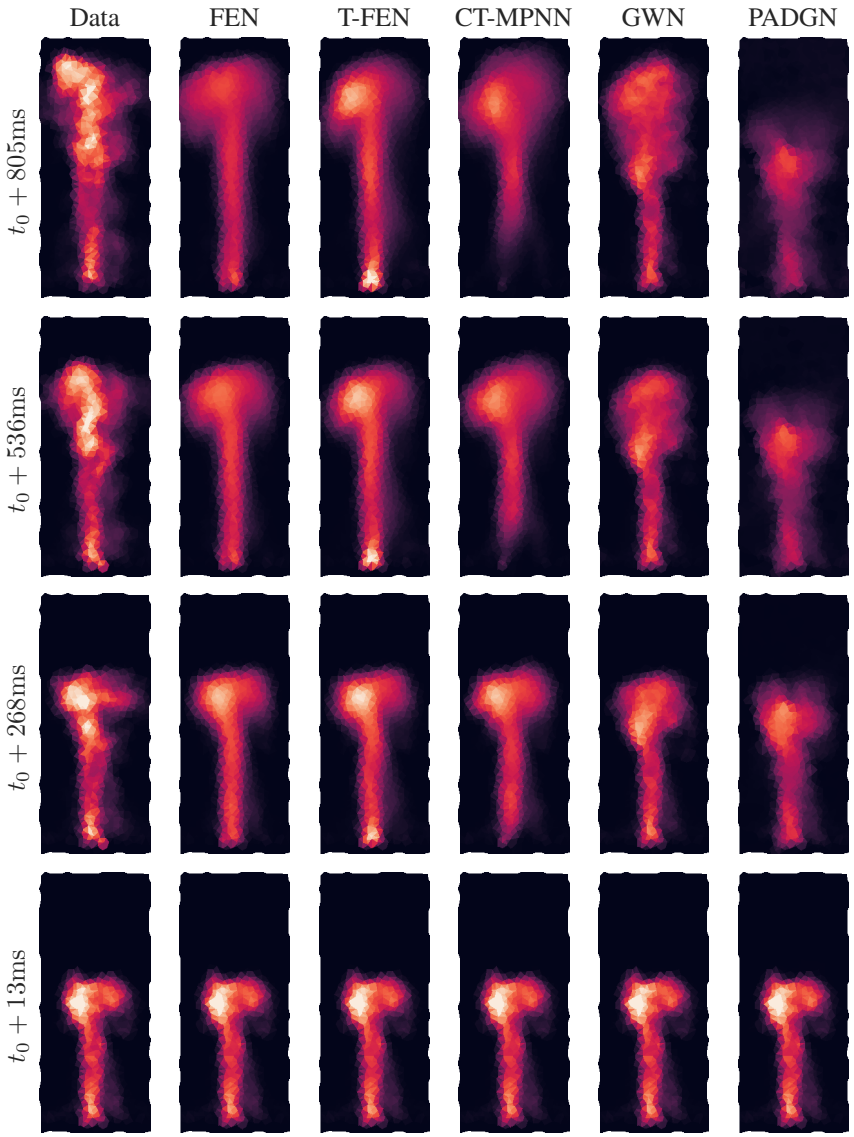


Figure 9: Predictions of all models for a rising smoke plume after 1, 20, 40 and 60 steps.

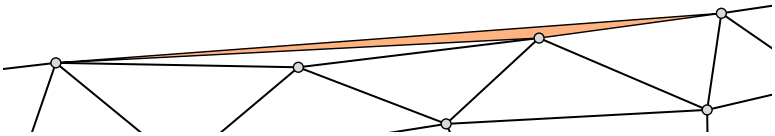
F Filtering Highly Acute Cells on the Boundary

Delaunay triangulation of a set of points can produce very long cells on the boundary as in Fig. 10 when several points close to the boundary are almost co-linear. This can just occur in the data but it can also happen during generation of synthetic data. If some boundary nodes in synthetic data are exactly co-linear such as in grid domains, they might lose the exact co-linearity due to the inexactness of floating point numbers and operations if we, for example, rotate the domain. To ensure that these artifacts do not impact our models, we pre-process the triangulation with Algorithm 1 and an angle threshold of $\varepsilon = \frac{10\pi}{180}$.

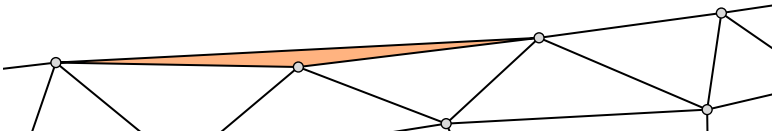
Algorithm 1 Filtering cells with small interior boundary-adjacent angles from a triangulation given a threshold angle and a set of $d-1$ dimensional boundary faces

function FILTERCELLS(triangulation $\mathcal{T}$, threshold $\varepsilon > 0$, boundary faces $\mathcal{B}$)
  for all $\Delta \in \mathcal{T}$ **do** ▷ Count number of faces on the boundary for each cell
    $c_\Delta \leftarrow |\{\tilde{\Delta} \mid \tilde{\Delta} \subset \Delta, |\tilde{\Delta}| = d\}|$
  $\mathcal{F} \leftarrow \{\Delta \mid \Delta \in \mathcal{T}, c_\Delta = 1\}$
  while $|\mathcal{F}| > 0$ **do**
    $\Delta \leftarrow \mathrm{pop}(\mathcal{F})$
    if $c_\Delta \neq 1$ **then** ▷ Check that Δ is not at an edge or corner of the domain
      continue
    $(\tilde{\Delta}, \xi) \leftarrow \mathrm{split}(\Delta)$ ▷ Split boundary face and interior node
    if MINBOUNDARYANGLE($\tilde{\Delta}$, ξ) < ε **then**
      $\mathcal{T} \leftarrow \mathcal{T} \setminus \{\Delta\}$ ▷ Remove cell
      for all $\hat{\Delta} \subset \tilde{\Delta} \mid |\hat{\Delta}| = d-1$ **do** ▷ Add newly become boundary faces to $\mathcal{B}$
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{\hat{\Delta} \cup \{\xi\}\}$
        for all $\Delta' \in \mathcal{T} \mid \hat{\Delta} \subset \Delta', \xi \in \Delta'$ **do**
          $c_{\Delta'} \leftarrow c_{\Delta'} + 1$
          $\mathcal{F} \leftarrow \mathcal{F} \cup \{\Delta'\}$ ▷ Add cells that are now boundary cells to $\mathcal{F}$
  return $\mathcal{T}$

function MINBOUNDARYANGLE($\tilde{\Delta}, \xi$)
  $B \leftarrow \mathrm{project}(\xi, \tilde{\Delta})$ ▷ Project ξ onto the hyperplane given by $\tilde{\Delta}$
  $\gamma \leftarrow \pi$
  for all $A \in \tilde{\Delta}$ **do**
    $\gamma \leftarrow \min(\gamma, \mathrm{angle}(AB, A\xi)$ ▷ Find the angle between the line segments AB and $A\xi$
  return γ



(a) The inside angles of the shaded boundary cell are below the threshold and it is removed.



(b) The removal has made this (previously) interior cell into a boundary cell that itself falls below the angle threshold and gets removed as well.

Figure 10: Delaunay triangulation can produce elongated, splinter-like cells when interior nodes are close to the boundary of the convex hull as in this example. We post-process the Delaunay triangulations with Algorithm 1 to filter these cells out.