Neural Operators and Time-Dependent PDEs

Vaishak Kumar, Antoine Weishaupt, Patrizia Apelt

AG Numerische Mathematik und Angewandte Analysis

Contents

1 Theoretical Background **1**
- 1.1 Theoretical Framework . . . 1
  - 1.1.1 Theory of the Time-Dependent Heat Equation . . . 1
  - 1.1.2 Partial Differential Equations and the Operator Setting . . . 2
  - 1.1.3 Green's Function as an Integral Operator . . . 2
  - 1.1.4 Numerical Data Setting & Quadrature Loss . . . 3
- 1.2 Infinite-Dimensional Operator Learning . . . 3
  - 1.2.1 General Neural Operator Pipeline . . . 3
- 1.3 The Fourier Neural Operator (FNO) . . . 4
  - 1.3.1 Spectral Action of the Spatial Laplacian . . . 4
  - 1.3.2 Fourier Integral Operator Formulation . . . 4
  - 1.3.3 Spectral Truncation, Nyquist Limit, and Aliasing . . . 4
- 1.4 Alternative Neural Operator Architectures . . . 5
  - 1.4.1 Graph Neural Operator (GNO) . . . 5
  - 1.4.2 Multipole Graph Kernel Network (MGKN) . . . 5
  - 1.4.3 Deep Operator Network (DeepONet) . . . 6
  - 1.4.4 Geometry-Informed Operators (Geo-FNO / GINO) . . . 6
  - 1.4.5 Suitability Summary for Time-Dependent Parabolic Problems . . . 6

2 Data Generation for the Heat equation **7**

3 Approaches **9**
- 3.1 One Shot . . . 9
- 3.2 Autoregressive . . . 9
- 3.3 Comparison . . . 10
- 3.4 Evaluation and Results . . . 11
- 3.5 Conclusion . . . 20

4 Allen-Cahn Equation **21**
- 4.1 System and Data Generation . . . 21
- 4.2 Approach . . . 22
- 4.3 Results . . . 22
- 4.4 Conclusion . . . 28

1 Theoretical Background

While classical finite-dimensional deep learning models (such as standard CNNs or FCNs) have advanced scientific machine learning, they remain fundamentally bottlenecked by their mesh-dependent nature. Models trained on a specific grid resolution fail to generalize when evaluated on different discretizations, necessitating costly retraining cycles. The Neural Operator framework remedies this limitation by shifting the learning paradigm from finite-dimensional Euclidean spaces to infinite-dimensional function spaces (Banach spaces).

Specifically, the Fourier Neural Operator (FNO) learns a single set of grid-invariant network parameters capable of operating across varied discretizations. This intrinsic mesh independence enables the direct transfer of solutions between disparate computational grids, unlocking the ability to evaluate complex physical systems at arbitrary resolutions without a loss in accuracy (zero-shot super-resolution). Obtaining a solution for a new instance of a parameter field or initial condition requires only a single forward pass of the network, alleviating the major computational costs incurred by traditional numerical solvers and single-instance methods like Physics-Informed Neural Networks (PINNs).

The Fourier transform is frequently used in spectral methods for solving differential equations, as spatial differentiation simplifies to pointwise multiplication in the Fourier domain. The Fourier Neural Operator learns mappings between function spaces by restricting the integral kernel to a convolution and instantiating it through a linear transformation in frequency space. In this work, we study operator learning of time-dependent parabolic PDEs, with a primary focus on the Heat and Allen–Cahn equation.

1.1 Theoretical Framework

1.1.1 Theory of the Time-Dependent Heat Equation

Let $\Omega \subset \mathbb{R}^d$ be a bounded spatial domain, and let $T > 0$ define the time horizon. The evolution of the temperature field $u(x,t) : \Omega \times [0,T] \to \mathbb{R}$ is governed by the PDE:

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = f(x,t), \quad x \in \Omega,\ t \in (0,T] \tag{1}$$

where:

- $\frac{\partial u}{\partial t}$ is the rate of change of temperature at a given point over time.
- $\alpha > 0$ represents the thermal diffusivity of the material ($\alpha = \frac{\kappa}{\rho c_p}$, where κ is thermal conductivity, ρ is density, and c_p is specific heat capacity).
- ∇^2 (or Δ) is the spatial Laplacian operator, defined in Cartesian coordinates as $\sum_i \frac{\partial^2}{\partial x_i^2}$. It measures the local spatial variation of the field.
- $f(x,t)$ represents an external heat source or sink term.

Initial and Boundary Conditions

To yield a well-posed initial-boundary value problem with a unique solution, the PDE must be supplemented with:

- **Initial Condition (IC):** The temperature profile across the domain at $t = 0$:

$$u(x,0) = u_0(x), \quad \forall x \in \Omega \tag{2}$$

- **Boundary Conditions (BCs) on $\partial\Omega$:** Typically parameterized as:

- **Dirichlet:** $u(x,t) = g(x,t)$ (fixed boundary temperature).
- **Neumann:** $\frac{\partial u}{\partial n} = \nabla u \cdot n = h(x,t)$ (specified heat flux across the boundary, where n is the outward normal vector).

1.1.2 Partial Differential Equations and the Operator Setting

Consider a general boundary value problem for a differential operator $\mathcal{L}_a$ parameterized by an input coefficient field or initial condition $a \in \mathcal{A}$:

$$\begin{cases} (\mathcal{L}_a u)(x) = f(x), & x \in D \\ u(x) = 0, & x \in \partial D \end{cases} \tag{3}$$

where $D \subset \mathbb{R}^d$ is a bounded open domain, f is a driving source function, and $u \in \mathcal{U}$ is the unknown target solution.

In classical numerical analysis, the primary task is instance-based: given one fixed instance of $a(x)$ and $f(x)$, compute the corresponding solution vector $u(x)$. In contrast, operator learning seeks to directly approximate the continuous non-linear mapping:

$$G^\dagger : \mathcal{A} \to \mathcal{U}, \quad a \mapsto u \tag{4}$$

between two infinite-dimensional function spaces.

1.1.3 Green's Function as an Integral Operator

When the differential operator $\mathcal{L}_a$ is linear and uniformly elliptic, the exact solution $u(x)$ to Eq. (3) can be expressed in continuous form via a Green's function $G_a(x,y)$:

$$u(x) = \int_D G_a(x,y) f(y)\, dy \tag{5}$$

Physically, the Green's function acts as the impulse response of the system, satisfying $\mathcal{L}_a G_a(\cdot, y) = \delta_y$ in D with $G_a(\cdot, y) = 0$ on ∂D.

From an operator-theoretic perspective:

1. The Green's function $G_a(x,y)$ serves as an **integral kernel** defining a linear operator $T_g f(x) = \int_D G_a(x,y) f(y)\, dy$.
2. The integral operator T_g acts as the explicit continuous inverse: $T_g = \mathcal{L}_a^{-1}$.
3. By **Mercer's Theorem**, if $\mathcal{L}_a$ possesses positive eigenvalues $0 < \lambda_1 \le \lambda_2 \le \dots$ and orthonormal eigenfunctions $\{\phi_k\}_{k=1}^\infty$, the Green's kernel admits the exact spectral expansion:

$$G_a(x,y) = \sum_{k=1}^{\infty} \frac{\phi_k(x)\phi_k(y)}{\lambda_k} \tag{6}$$

Approximating $g : \Omega \times \Omega \to \mathbb{R}$ with a parametric neural network $g_\theta(x,y)$ allows us to evaluate the PDE solution via:

$$u(x) \approx \int_\Omega g_\theta(x,y) f(y)\, dy = (G_\theta f)(x) \tag{7}$$

1.1.4 Numerical Data Setting & Quadrature Loss

Given a collection of data pairs $\{f^i, u^i\}_{i=1}^N$ where $f^i \in L^2(\Omega)$ represents the forcing function and $u^i \in H^1(\Omega)$ represents the target solution:

- For the 1D homogeneous Dirichlet eigenvalue problem $-\frac{d^2v}{dx^2} = \lambda v$, non-trivial solutions ($\lambda > 0$) have the general second-order ODE form:

$$v(x) = A\cos(\sqrt{\lambda}x) + B\sin(\sqrt{\lambda}x) \tag{8}$$

- On domain $\Omega = (0, 1)$, random input forcing functions $f^i(x)$ can be generated via trigonometric series:

$$f^i(x) = \sum_{j=1}^{N_m} \alpha_j^i \sin(2\pi x + \phi_j^i) \tag{9}$$

- Evaluating continuous integration via numerical quadrature over discretization mesh points $(x_k^i, f_k^i, u_k^i)_{0\le k\le n_i+1}$ yields:

$$(G_\theta f^i)(x) \approx \sum_{k=0}^{n_i+1} g_\theta(x, x_k^i) f_k^i w_k^i \tag{10}$$

- The discretized normalized L^2 loss functional $L(\theta)$ is given by:

$$L(\theta) := \frac{1}{N}\sum_{i=1}^{N} \frac{\sum_{l=0}^{n_i+1} \left|u_l^i - \sum_{k=0}^{n_i+1} g_\theta(x_l^i, x_k^i) f_k^i w_k^i\right|^2 w_l^i}{\sum_{l=0}^{n_i+1} |u_l^i|^2 w_l^i} \tag{11}$$

1.2 Infinite-Dimensional Operator Learning

To generalize beyond linear Green's functions, let $D \subset \mathbb{R}^d$ be a bounded open set, and let $\mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_a})$ and $\mathcal{U} = \mathcal{U}(D; \mathbb{R}^{d_u})$ be separable Banach spaces. Given observations $\{a_j, u_j\}_{j=1}^N$ with $a_j \sim \mu$ and $u_j = G^\dagger(a_j)$, we construct a parametric map $G_\theta : \mathcal{A} \to \mathcal{U}$ by minimizing the expected empirical risk:

$$\min_{\theta\in\Theta} \mathbb{E}_{a\sim\mu}\Big[C\Big(G(a, \theta), G^\dagger(a)\Big)\Big] \tag{12}$$

1.2.1 General Neural Operator Pipeline

The general forward pass of a Neural Operator consists of three main steps:

1. **Lifting (P):** A local transformation $P : \mathbb{R}^{d_a} \to \mathbb{R}^{d_v}$ lifts the input function pointwise into a higher-dimensional channel space: $v_0(x) = P(a(x))$.
2. **Iterative Kernel Updates ($v_t \mapsto v_{t+1}$):** For layers $t = 0, \ldots, T-1$, hidden state representations update via:

$$v_{t+1}(x) = \sigma\Big(Wv_t(x) + \Big(\mathcal{K}(a; \phi)v_t\Big)(x)\Big) \tag{13}$$

   where $W \in \mathbb{R}^{d_v \times d_v}$ is a local linear transformation (acting as a residual shortcut), $\mathcal{K}$ is a non-local integral operator, and σ is an element-wise non-linear activation function.
3. **Projection (Q):** A local transformation $Q : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$ projects the deep features back to the physical output dimension: $u(x) = Q(v_T(x))$.

1.3 The Fourier Neural Operator (FNO)

1.3.1 Spectral Action of the Spatial Laplacian

FNO is exceptionally well-suited for parabolic problems (such as the Heat equation) because the spatial Laplacian ∇^2 diagonalizes in the Fourier domain. Expressing $u(x,t)$ via a Fourier expansion over spatial modes f:

$$u(x,t) = \sum_f \hat{u}(f,t) e^{i\langle f,x\rangle} \tag{14}$$

Substituting this expansion into the homogeneous heat equation $\frac{\partial u}{\partial t} = \alpha \nabla^2 u$ yields:

$$\nabla^2 u(x,t) = \sum_f \hat{u}(f,t)\left(-|f|^2\right) e^{i\langle f,x\rangle} \implies \hat{u}(f,t) = \hat{u}_0(f) e^{-\alpha |f|^2 t} \tag{15}$$

In Fourier space, spatial differentiation reduces to algebraic mode-wise multiplication by $-|f|^2$. By parameterizing complex weights directly over truncated low-frequency modes $k_{\max}$, the FNO layer naturally learns this exponential decay operator directly from observational data.

1.3.2 Fourier Integral Operator Formulation

Assuming spatial translation invariance $\kappa(x,y) = \kappa(x-y)$, the non-local kernel integration simplifies to a continuous convolution. By the Convolution Theorem, this is evaluated efficiently using the Fast Fourier Transform (FFT).

The l-th feature channel update in an FNO layer is given by:

$$v_{t+1,l}(x) = \sigma\left((W v_t(x))_l + \mathcal{F}^{-1}\left(\sum_{j=1}^{d_v} R_{l,j}(k) \cdot (\mathcal{F} v_t)_j(k) \right)(x) \right) \tag{16}$$

where:

- $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the forward and inverse Fast Fourier Transforms.
- $(\mathcal{F} v_t)_j(k)$ is the Fourier coefficient of the j-th channel for frequency mode $k \in \mathbb{Z}^d$.
- $R_{l,j}(k) \in \mathbb{C}$ is a learnable complex-valued tensor acting on lower modes $Z_{k_{\max}} = \{k \in \mathbb{Z}^d : |k_i| \leq k_{\max,i}\}$.

Because modes higher than $k_{\max}$ are truncated, the spatial integration complexity is dominated by FFT evaluation, requiring only $O(n \log n)$ operations.

1.3.3 Spectral Truncation, Nyquist Limit, and Aliasing

To remain computationally efficient and resolution-invariant, the Fourier Neural Operator truncates the Fourier series along each spatial dimension m, restricting the complex learnable weight tensor $R_\phi(k) \in \mathbb{C}^{d_v \times d_v}$ to act exclusively on a small subset of low-frequency modes ($|k_m| \leq K_m$):

$$[(\mathcal{K}_\phi v)(x)]_l = \mathcal{F}^{-1}\left(\sum_{j=1}^{d_v} R_{\phi,l,j}(k) \cdot (\mathcal{F} v)_j(k) \right)(x), \quad \text{for } |k_m| \leq K_m \tag{17}$$

This truncation is physically justified for parabolic PDEs because dominant physical energy resides in low frequencies, while higher modes carry negligible energy. Restricting learnable contractions to K_m modes bounds parameter counts to $O\left(C^2_{\text{hidden}} \prod_{m=1}^{d} K_m\right)$, leaving total layer evaluations dominated by

the FFT ($O(N \log N)$). Although higher modes are truncated during spectral convolution, non-linear activations σ applied between layers act as pointwise frequency mixers that continuously regenerate higher-frequency details throughout deeper layers.

However, selecting K_m is constrained by the **Nyquist–Shannon Sampling Theorem**. For a continuous function $f(x)$ sampled on a uniform grid of N_m points with spacing Δx_m, the sampling frequency $f_s = \frac{1}{\Delta x_m}$ defines the Nyquist limit $f_{\text{Nyquist}} = \frac{N_m}{2}$ as the maximum frequency resolvable without information loss. Discrete sampling folds any signal components exceeding f_{Nyquist} back into lower modes, causing spectral distortion known as **aliasing**.

A subtle problem occurs when composing spectral layers with pointwise non-linear activations like $\sigma(x) = x^2$. If a field initially activates frequencies up to K_m, squaring it in physical space expands the active modes up to $2K_m$:

$$x_n^2 = \frac{1}{N} \sum_{p=-K_m}^{K_m} \sum_{q=-K_m}^{K_m} X_p X_q e^{2\pi i (p+q) n/N} \tag{18}$$

Because non-linearities expand the frequency bandwidth, setting K_m too close to $N_m/2$ causes generated modes beyond $N_m/2$ to fold back and contaminate low-frequency coefficients during coarse-resolution training. When evaluating on a fine grid during zero-shot super-resolution, this aliasing disappears, causing the contaminated low-frequency weights to fail. To guarantee stable zero-shot super-resolution the number of, K_m, is typically chosen conservatively below $N_m/2$ (e.g., $K_m \leq N_m/3$).

1.4 Alternative Neural Operator Architectures

While FNO operates in the spectral frequency domain, other neural operator paradigms parameterize integral kernels through spatial graphs or separate network branches.

1.4.1 Graph Neural Operator (GNO)

The Graph Kernel Network parameterizes the kernel $\kappa_\phi(x, y, a(x), a(y))$ as a Graph Neural Network (GNN). Non-local message passing is computed over local physical neighborhoods $N(x) = B(x, r)$:

$$v_{t+1}(x) = \sigma\left(W v_t(x) + \frac{1}{|N(x)|} \sum_{y \in N(x)} \kappa_\phi(x, y, a(x), a(y)) v_t(y) \right) \tag{19}$$

GNO naturally is applicable to unstructured meshes, but $O(K^2)$ edge complexity makes fine-grid training expensive, and local physical ball neighborhoods struggle to capture global spectral decay without multi-scale factorizations.

1.4.2 Multipole Graph Kernel Network (MGKN)

To address the quadratic scaling of GNOs without sacrificing geometry flexibility, MGKN adopts principles from the Fast Multipole Method (FMM). It factorizes the kernel matrix additively across spatial length scales:

$$K = K_1 + K_2 + \cdots + K_L \tag{20}$$

where K_1 represents local sparse graphs (near-field) and K_L represents low-rank coarse clusters (far-field). State representations are updated via a multi-resolution **Graph V-Cycle Algorithm** (combining restriction $K_{l+1,l}$ and prolongation $K_{l,l+1}$), achieving global receptive fields on arbitrary meshes in linear time complexity $O(n)$.

MGKN performs significantly better than standard GNO because its hierarchical V-cycle captures both near-field local diffusion and far-field long-range couplings in linear time $O(n)$. It performs well on non-uniform domains, though it is slightly more complex to construct than FNO on regular grids.

1.4.3 Deep Operator Network (DeepONet)

Rooted in the Universal Approximation Theorem for Operators (Chen & Chen, 1995), DeepONet decouples function encoding from target evaluation coordinates using two sub-networks:

$$G(u)(y) \approx \sum_{k=1}^{p} \underbrace{b_k(u(x_1), u(x_2), \ldots, u(x_m))}_{\text{Branch Net Output}} \cdot \underbrace{t_k(y)}_{\text{Trunk Net Output}} + b_0 \tag{21}$$

- **Branch Network:** Takes evaluations of input function u at fixed sensors $\{x_i\}_{i=1}^{m}$.
- **Trunk Network:** Takes target continuous coordinate $y \in \mathbb{R}^d$ as input.

DeepONet provides flexible output evaluations, though sensor inputs x_i remain fixed across dataset instances.

DeepONet can easily be applied by mapping initial sensor inputs $[u_0(x_1), \ldots, u_0(x_m)]$ (Branch) to target space-time points (x, t) (Trunk). While it handles unstructured output query locations easily, it requires fixed input sensor locations and typically yields higher generalization error on periodic/regular grids compared to FNO's direct spectral convolutions.

1.4.4 Geometry-Informed Operators (Geo-FNO / GINO)

To apply FNOs to irregular physical domains, Geo-FNO learns a coordinate deformation map $\varphi : D_c \to D_a$ pulling back irregular shapes to a regular computational grid D_c. Similarly, GINO uses a Graph Neural Operator (GNO) encoder to map point clouds onto a regular latent grid for FNO processing, followed by a GNO decoder.

Geo-FNO can be beneficial only on Complex Geometries.If the Heat or Allen–Cahn equation is evaluated on regular rectangular meshes with periodic boundary conditions, these geometric encoders add unnecessary computational steps. However, if solving phase separation inside irregular material boundaries or complex physical molds, GINO/Geo-FNO would be required.

1.4.5 Suitability Summary for Time-Dependent Parabolic Problems

For our time-dependent Heat and Allen–Cahn problems on regular periodic meshes, FNO is the most suitable architecture due to its fast $O(n \log n)$ FFT convolutions and exact alignment with Laplacian spectral decay. GNO and DeepONet would be less effective here due to quadratic scaling or fixed input sensor requirements, while Geo-FNO/GINO add unnecessary mapping complexity since our computational domain is already a simple regular grid.

2 Data Generation for the Heat equation

Before we can start learning the time dependent heat equation with neural operators we need to generate some data. This data will then be used to train the neural operator. The training data will be solutions of the heat equation that we calculate with the help of the finite element method (FEM). The reason why FEMs are used is, because generally there are not always analytical solutions.
The set up for the data generation is the following, we have the time dependent heat equation

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u + f(x,t) \tag{22}$$

on the spatial domain $\Omega = [0,1]$ and the time interval $t \in [0,1]$. We choose the thermal diffusivity as $\alpha = 0.1$ and the heat source as a stationary gaussian with $f(x) = e^{-100(x-0.5)^2}$. For our boundary conditions we chose homogeneous Dirichlet boundary conditions

$$u(0,t) = 0 = u(1,t) \quad , \tag{23}$$

while our initial condition is a random linear combination of the first three sine modes

$$u(x,0) = \sum_{n=1}^{3} a_n \sin\left(\frac{n\pi x}{L}\right) \quad . \tag{24}$$

The coefficients a_n are sampled independently from the uniform distribution on [-1, 1].

The spatial discretization was performed using the FEM. This requires the weak formulation of the heat equation. To derive it, we multiply the heat equation by a test function v and integrate over the domain. This yields

$$\int_\Omega \frac{\partial u}{\partial t} v \, dx + \alpha \int_\Omega \nabla u \, \nabla v \, dx = \int_\Omega f \, v \, dx \quad . \tag{25}$$

Now we choose a finite element space

$$V_h = \text{span}\{\phi_1, \phi_2, ..., \phi_N\} \quad , \tag{26}$$

where ϕ_i are finite element basis functions. The solutions are then approximated by

$$u_h(t,x) = \sum_{j=1}^{N} U_j(t) \, \phi_j(x) \quad . \tag{27}$$

Differentiating this in t gives

$$\frac{\partial u_h}{\partial t} = \sum_{j=1}^{N} \dot{U}_j(t) \, \phi_j(x) \quad . \tag{28}$$

Then we choose the test functions from the same space

$$v = \phi_i \qquad i = 1, ..., N \quad . \tag{29}$$

Substituting this into the weak formulation gives

$$\int_\Omega \left(\sum_{j=1}^{N} \dot{U}_j \, \phi_j \right) \phi_i \, dx + \alpha \int_\Omega \nabla \left(\sum_{j=1}^{N} U_j \, \phi_j \right) \nabla \phi_i \, dx = \int_\Omega f \, \phi_i \, dx \quad , \tag{30}$$

rearranging gives us

$$\sum_{j=1}^{N} \dot{U}_j \int_\Omega \phi_j \, \phi_i \, dx + \alpha \sum_{j=1}^{N} U_j \int_\Omega \nabla \phi_j \, \nabla \phi_i \, dx = \int_\Omega f \, \phi_i \, dx \quad . \tag{31}$$

Now we define the

$$\begin{aligned} \textbf{mass matrix:} \quad & M_{ij} = \int_\Omega \phi_i \, \phi_j \, dx \\ \textbf{stiffness matrix:} \quad & K_{ij} = \int_\Omega \nabla \phi_i \, \nabla \phi_j \, dx \\ \textbf{load vector:} \quad & F_i(t) = \int_\Omega f \, \phi_i \, dx \quad . \end{aligned}$$

Then we can write the heat equation as a matrix system with

$$U(t) = \begin{bmatrix} U_1(t) \\ . \\ . \\ . \\ U_N(t) \end{bmatrix} \tag{32}$$

and get

$$M \, \dot{U}(t) + \alpha \, K \, U(t) = F(t) \quad . \tag{33}$$

This is the ODE system obtained by the finite element discretization in space [11].

Applying the implicit backward Euler method for the time discretization results in

$$(M + \Delta t \, \alpha \, K) \, U^{n+1} = M \, U^n + \Delta t \, F \quad . \tag{34}$$

For the spatial discretization, we use a uniform one-dimensional linear mesh consisting of 64 equally spaced nodes. This mesh is used for all simulations.

To generate the data, we keep the same partial differential equation, mesh, source term, and boundary conditions for all simulations, while varying only the randomly generated initial conditions.
Thus, each data sample is generated by providing an initial temperature field $u(x, 0)$ and computing the corresponding full space-time solution $u(x, t)$ on a 64×64 grid.

3 Approaches

Our initial work with neural operators began with the Fourier Neural Operator (FNO) applied to the heat equation, as it provided an accessible starting point. As our work naturally progressed, we decided not to explore other operator architectures. Instead, we decided to focus on investigating two different approaches for handling the time-dependency of the heat equation with the FNO being our neural operator.

3.1 One Shot

The first way to handle the time dependency of the heat equation is with what we would call the "One Shot" method or for the model "One Shot model" (OSM). This comes from the fact that in this case the neural network only needs to perform one forward pass to calculate the solution across the entire space-time domain. This is achieved by treating time as just another coordinate along the space dimension.

It is implemented via constructing spatial and temporal coordinate tensors that span the entire space-time mesh. The spatial tensor repeats the spatial grid across all n_t time steps, resulting in a shape of $[n_t, n_x]$ for a 1D spatial domain, $[n_t, n_x, n_y]$ for 2D, and so on. Conversely, the temporal tensor repeats the time vector across all spatial points. This provides the network with a coordinate pair for every point in the space-time domain, and ensuring that the tensor shapes work out correctly during calculations.
The initial condition input must also match this grid size. While one could theoretically input the initial condition at $t = 0$ and pad the remaining time steps with zeros, it is more effective to "broadcast" (repeat) the initial condition values across the entire time dimension. Because the initial state of a point heavily influences its evolution, broadcasting this state across all time steps gives the neural network local access to the initial conditions at every point in time, which leads to better performance. One could think of it this way that the initial conditions across the entire domain get "shaped" into the solution by the model, instead of the solution being built step by step. The broadcasted initial condition and the coordinate tensors are concatenated along the channel dimension to form a single input tensor. The Fourier layers then perform multi-dimensional transforms (such as a 2D FFT for 1D space + time) across both space and time. This provides the model with information not only across space but also globally across all times simultaneously, resulting in the final solution after a single forward pass. The training data for this model consists of samples with the broadcasted initial condition and the target being the numerical solution across the entire domain.

The main advantage of this architecture is its computational efficiency when it comes to predicting a solution, where it outperforms numerical methods and the Autoregressive approach we looked at. Additionally, there is no error accumulation from each time step to the next as it is the case for iterative numerical and autoregressive methods. However, drawbacks also exist. Since the input and thus also the training data contain the temporal tensor the model is inherently limited by the final time T of the data. While varying end times in the training data is possible, the model will fail when it tries to predict past the maximum time horizon T it has seen. On the other hand, this allows for the training data to contain times steps of different sizes. For example the interval $[0, 1]$ could be discretized into 64 points while another sample goes from $[0, 2]$ containing 64 points as well. Technically, different amounts of grid points in time are also possible, but it requires samples to be batched together accordingly during training to prevent tensor shape mismatches from occurring. Because the FNO is used and both time and space are being Fourier transformed, the OSM does not only have grid independence in space but also in time.

3.2 Autoregressive

The second approach is the Autoregressive model (ARM), which closely mimics classical numerical methods by solving the system iteratively, step-by-step. The initial condition is used to predict the

solution at the next time step, which is then recursively fed back into the model as the input for the subsequent forward pass. Only the spatial domain is Fourier transformed in this case and the model only has global information across the spatial dimension while being local in time. At each step, the model receives the current solution values together with the space and time grid, here not converted into global tensors. This means the training data consists of pairs containing the solution at a current time step t_i as the input, and the target solution at the subsequent step t_{i+1}.

The primary advantage of the ARM is that it is not limited by a time horizon and can predict up to any arbitrary time T. While the ARM is overall smaller (less parameters) than the OSM because it only calculates a single step with each forward pass, the overall inference speed for the solution across the entire domain is slower, as it has to do the forward pass many times. This iterative scheme also makes it susceptible to error accumulation, where the small prediction errors at each step add up over time. Additionally, the model is constrained to a fixed stepsize which it inherits from the discretization of the training data. The samples cannot have varying stepsize. On the other hand, the maximum time horizon T can differ but the interval has to be discretized accordingly with different number of points in time such that the stepsize stays the same. Lastly, the ARM requires considerably less memory for inference where only one step at a time is computed, compared to the OSM which calculates the entire solution at once.

3.3 Comparison

Although both models are based on the Fourier Neural Operator (FNO), they process data differently. This raises the question of how we can compare these two architectures in a fair way. One option would be to train both models to ultimate convergence and see which model can deliver the best performance. However, performance in this context can mean different things like total error, speed or data utilization. Each of which might be prioritized, depending on the physical system and the intended application. Because different scenarios naturally favor one architecture over the other, a universally fair comparison is difficult to define.

Consequently, we approached our benchmark from an efficiency based perspective. Rather than training both models to their theoretical limits, which would require long training times, we decided on limiting the training budget. This approach allowed for faster evaluations, enabling us to explore a wider variety of features and behaviors of the two models.

With this perspective, both models were given the exact same 200 solutions across the entire domain, ensuring they get access to the identical amount of physical information, Testing was done with 50 samples. However, as described before, the models utilize the training data differently due to their architectures. The ARM slices these 200 solutions into $200(n_t - 1)$ step transitions, resulting in 12,600 training samples for a 64 point time grid. Comparatively, the OSM treats them as 200 complete training samples. This difference in sample count in itself is an inherent architectural feature rather than an unfair advantage.

But to handle both of these cases fairly, while getting a sense of their efficiency, we decided to train them for the same number of gradient steps (weight updates) rather than a fixed number of epochs. Additionally, we used different batch sizes (16 for the OSM and 64 for the ARM) to balance the amount of physical information they can process per step. A single One-Shot sample contains an entire space-time solution, whereas an Autoregressive sample is only a single step transition. Beyond balancing the data volume, setting the OSM's batch size to 16, kept its higher computations manageable for our many evaluations, while a batch size of 64 allowed the ARM to still train for enough epochs. This efficiency based approach tries to balance the differences between the two models, in terms of information and computation, while also allowing us to evaluate their effectiveness.

3.4 Evaluation and Results

The results overall support and showcase the aforementioned behavior, advantages and disadvantages of the two models. We evaluated the two in many different aspects: Size, Error, Time Horizon, Energy Evolution, Step functions with step gradients, Diffusion Strength, Inference Speed and Mesh size.

To implement the FNO we used the neural operator library by PyTorch, which provides an easy out of the box model together with a Data loader. We did not use the library's provided Trainer class due to the above mentioned fact of wanting to train on a fixed number of gradient steps. Instead, we implemented the standard PyTorch training loop, with forward pass, loss calculation, backpropagation, and gradient step, looping until we reach the desired steps. The chosen optimizer being AdamW.
If not specifically mentioned otherwise, the "default settings" that were used, are 32 hidden channels, keeping the first 16 Fourier modes, 1800 gradient steps, 0.001 learning rate, 64 x 64 time-space mesh with final time T=1 in the training data, and $\alpha = 0.1$. The number of hidden channels refers to the width of the network. We restricted space to 1D because our primary focus was on investigating how the two different approaches deal with the time-dependency of the physical systems. This simplification allowed us to focus our efforts on the following evaluation of the models' performance, rather than dedicating resources too early to 2D FEM data generation or 3D space-time plotting, which are often more difficult to visualize clearly in a report.

Overall Performance

The OSM performed better in terms of accuracy than the ARM. Its error was overall lower and did not get propagated in each time step, as it calculates the entire solution in one go.

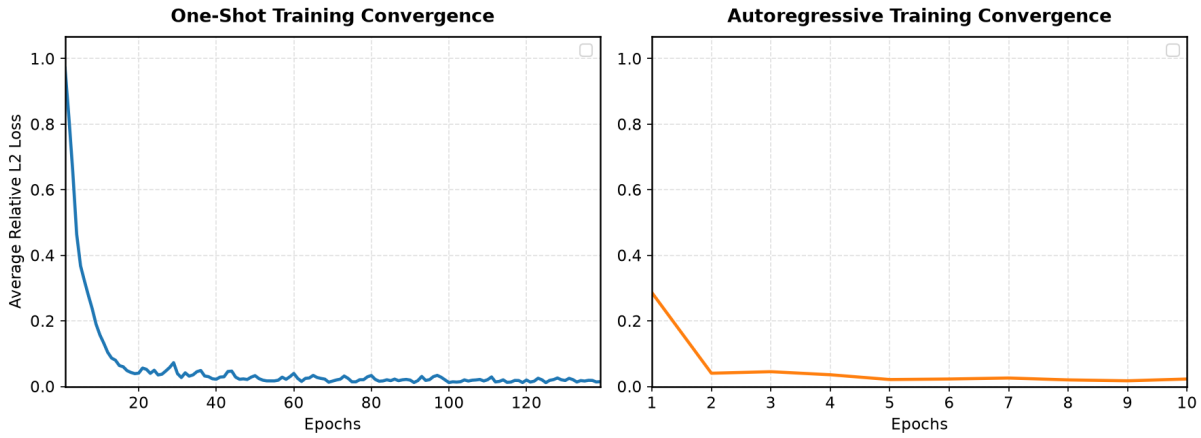


Figure 1: Avg. rel. epoch loss for OSM and ARM

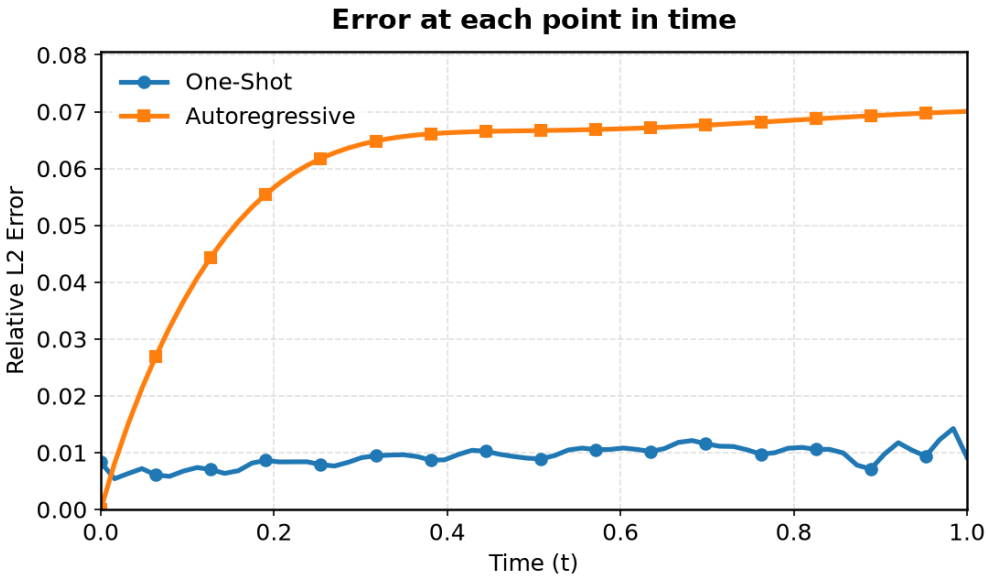


Figure 2: Error at t_i for OSM and ARM

With an average relative L^2 error per test sample of 0.0578 and RMSE of 0.0221 for the ARM and 0.0088 and 0.0034 for the OSM respectively. Because the OSM Fourier transforms time as well, the error at each point in time displays a more oscillating pattern. Compare this to the curve of the ARM, which propagates the error smoothly from one step to the next. With similar average relative L^2 losses in the final epoch for both models with 0.0226 ARM and 0.0145 OSM. The reason why the graph of the ARM starts down lower is that it already saw enough samples during the first epoch to make a somewhat decent prediction for a single step.

An important point has to be mentioned here about the ARM. Training on the FEM data, it never sees its own error which will occur during each time step prediction. This can result in the model being unstable. One possible fix is to introduce a tiny bit of noise in the training data.

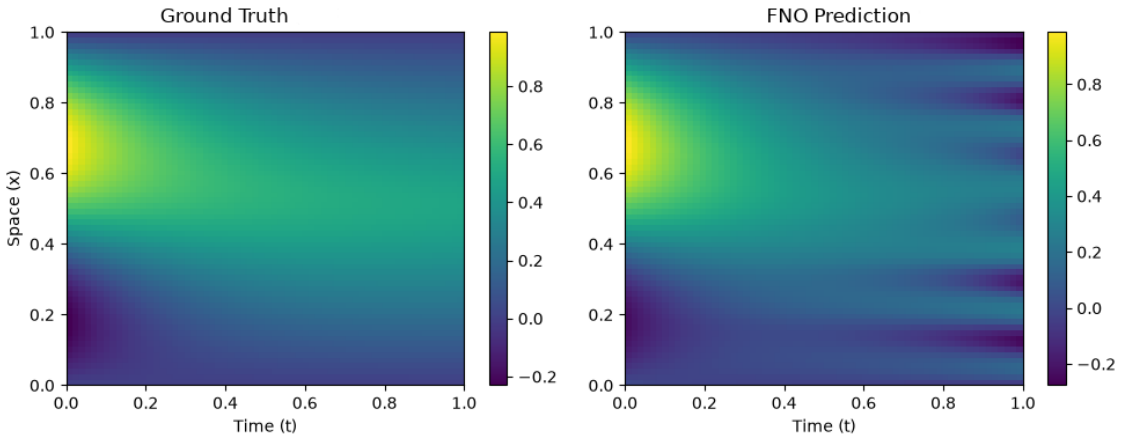


Figure 3: ARM without noise

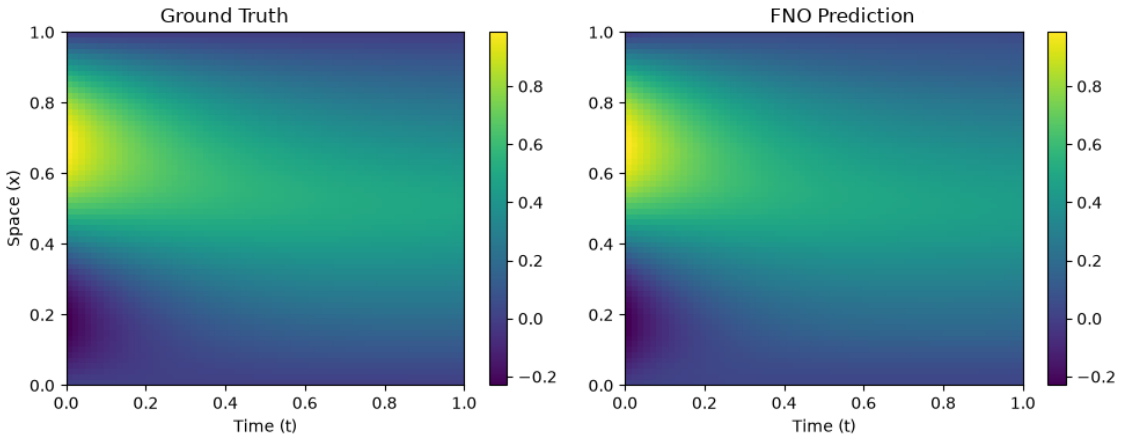


Figure 4: ARM with added noise

Without noise, the model is trained only on clean inputs and never sees the high-frequency errors it makes during the prediction. Because of this, the network does not know how to dampen these errors, which accumulate and result in the spatial oscillations towards the end. Adding noise significantly brought down the Avg. Rel. L^2 error from 0.3639 to 0.0578. The error comparison between the two models at the beginning was done with this added noise in the ARM. But not in every case does added noise actually help. We observed that for a "better" model with more parameters or more training time, the noise actually resulted in worse performance. The following plot shows the stability of more training time and noise/no noise:

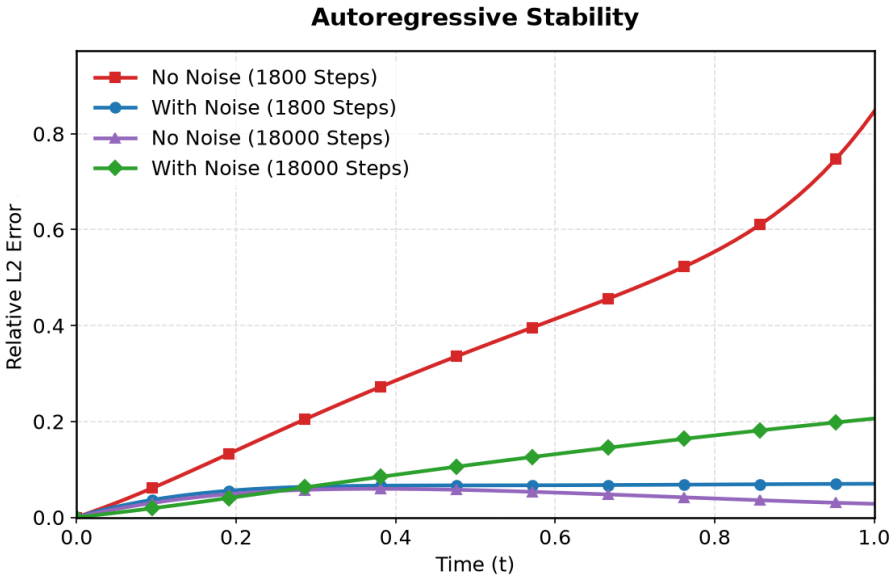


Figure 5: Stability of the ARM

The more parameter case is not included here because the ARM generally performed worse with it compared to with less parameters, but also exhibited the same unstable behavior with added noise.

This leads directly into the next point of the comparison, concerning the size of the models.

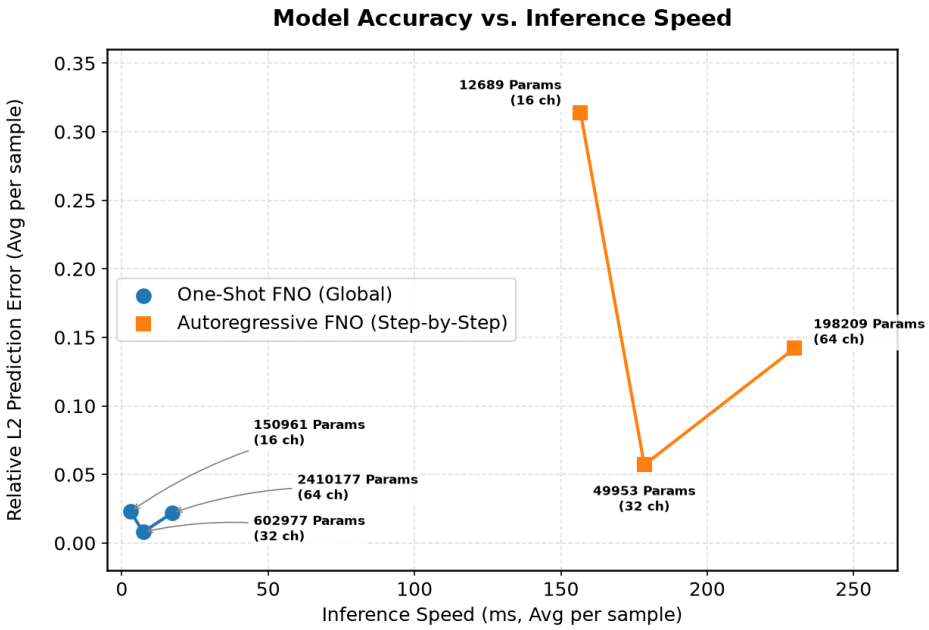


Figure 6: Inference speed vs Accuracy (Relative Error) with models of different size

Both models did the best with 32 hidden channels. This is most likely down to the fact that within 1800 gradient steps these bigger models did not have enough time to fully converge. It is also possible that with 64 channels the models have too many parameters, an amount that is simply not needed to capture the evolution of the heat equation in 1D.

Time Horizon

In the following we take a look at the time horizon advantages and disadvantages of the two models. As stated already, the OSM cannot predict beyond the time horizon featured in its training data. Conversely, the ARM can go to any final time T.

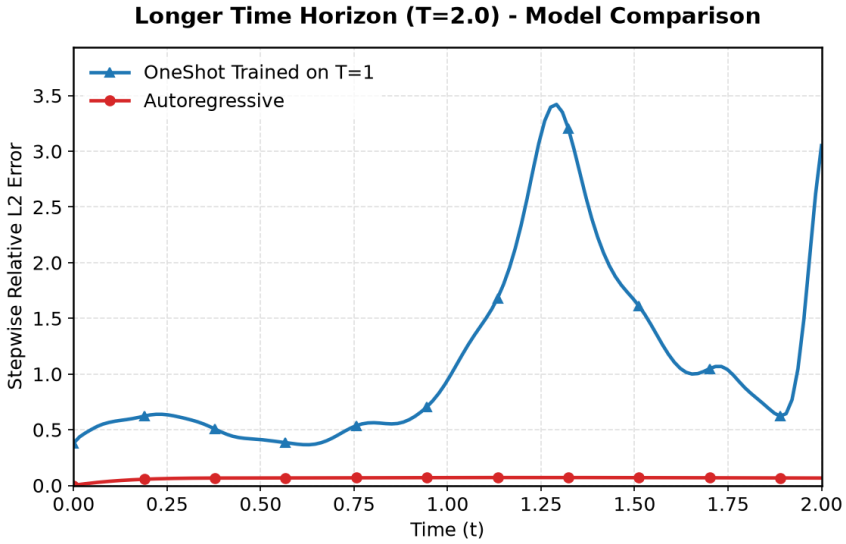


Figure 7: ARM and OSM trained until T=1; Evaluated until T=2

Interestingly enough, the prediction of the OSM does not only break down in $t \in [1, 2]$ but its prediction in the interval of $t \in [0, 1]$ is also considerably worse. This is due to the fact that it tries

to calculate the solution for the entire interval [1,2] in one forward pass and the unseen interval thus has a negative effect on the prediction of the learned interval as well.

We also looked at how including samples with a larger time horizon can affect the performance of the OSM.

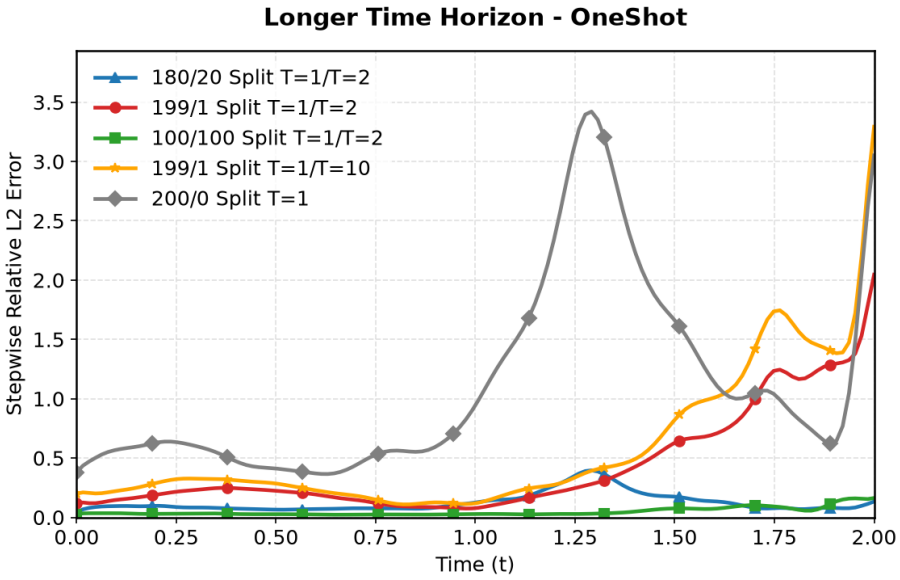


Figure 8: OSM trained on different splits of training data with T=1 and T=2/T=10

It can be seen that including just one sample with time horizon T=2 already improved the performance by a lot, although not remotely enough to be usable. The 180/20 split continues this trend and at 100/100 the model performs nearly the same as in the T=1 case. The slight error increase towards the end in this split is most likely due to 100 samples being not being quite enough to learn a time interval of 1. Introducing a single sample but with an even larger time horizon yielded actually a slight performance decrease compared to the single sample with T=2. This is most likely due to the fact the model tried to learn the evolution between the larger interval $t \in [2, 10]$ from this sample as well, which impacted the rest of the training negatively.
Next we want to look at the performance regarding the physical properties of the heat equation.

Energy

As heat dissipates and leaves the system due to the boundaries being kept at zero or the heat increasing from the source term, the total energy of the system decreases or increases as well. To check how well the models adhere to the physics we integrated our obtained solution using the Simpson rule, yielding the total heat content of the system. The energy is proportional to this integral up to a physical scaling factor. As we are comparing our prediction with the (numerical) solution this constant scaling factor is identical for both and does not affect the comparison and we can make comments about how the models predict the energy evolution. The following plot shows how the "energy" changes based on the prediction by the two models:

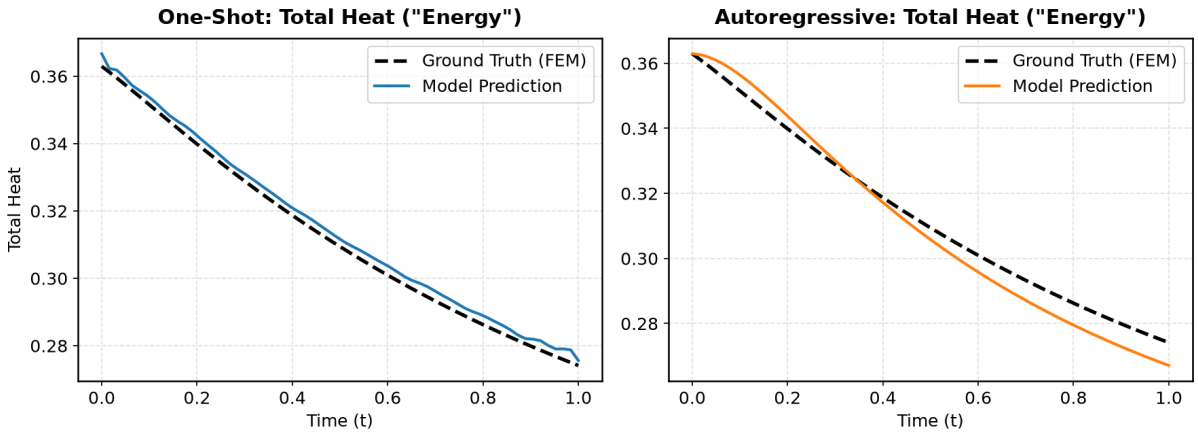


Figure 9: "Energy" of the System; Model Prediction and Ground Truth

The OSM's prediction replicates the Energy decay well and is globally stable, deviating from the exact energy equally at all times. This is again down to the solution being calculated all at once. On the right, the ARM overshoots initially and then predicts the energy to decay faster. The bump at the beginning is caused by spatial truncation, which slightly smooths out the gradients and results in slower diffusion. The OSM's energy is also always a bit higher for the same reason, but the amount being much less because the model is performing better. Due to the added noise, the ARM learns to smoothen and dampen it. This learned dampening causes the predicted energy to undershoot at around $t = 0.4$. Without noise this does not happen and the ARM only predicts more energy than there actually is. Another thing that can be observed again in the curves is the smooth prediction of the ARM compared to the oscillating pattern in the OSM due to the Fourier transform in time.

Diffiusion and Gradients

Regarding the gradients, both models struggle more when they are steeper, i.e. when the diffusion factor α of the heat equation is larger.

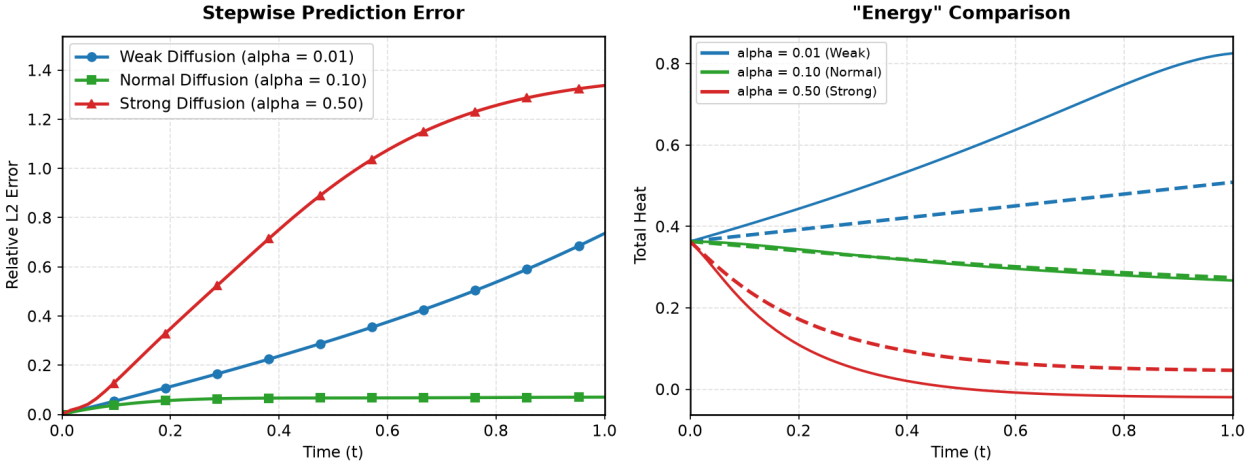


Figure 10: ARM trained with different alphas

For $\alpha = 0.5$ the system reaches zero quickly. Due to this, the relative error is large because while the denominator with the target solution approaches zero, the ARM still contains its propagation error. Despite it predicting the solution to go to zero, its own error leaves the overall relative error large. The bad prediction for weak diffusion is also down to the same idea of the error propagation. For weak diffusion the solution is nearly the same from one time step to the next. The model's error is larger than the actual change in the system and the reason why it performs worse. All three cases

used the same learning rate and noise which was well suited for $\alpha = 0.1$. Increasing or decreasing them helped the other two cases to perform better, but this was not explored further.

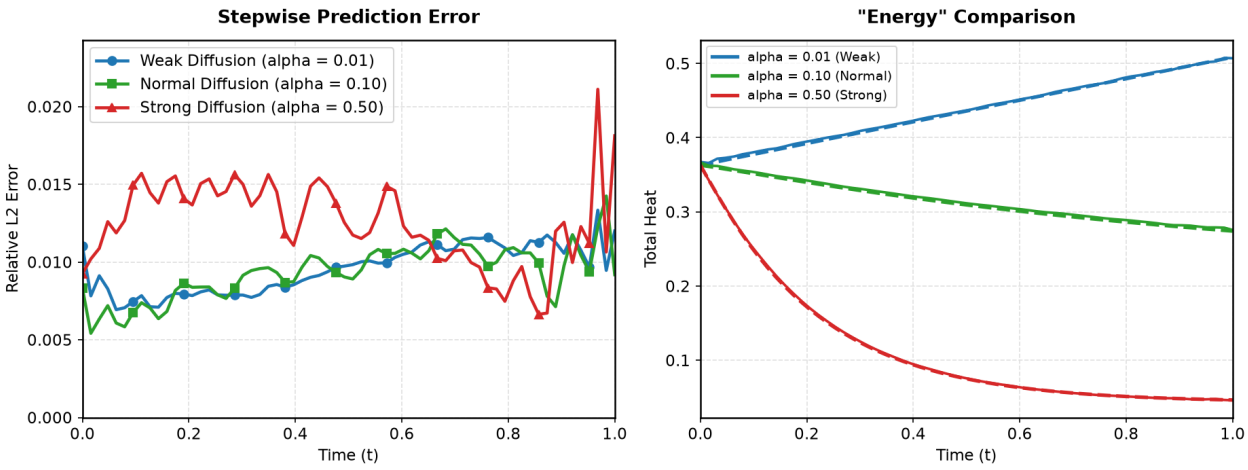


Figure 11: OSM trained with different alphas

The OSM also struggled a bit more with the steeper gradients of the strong diffusion but overall it did perform well, also in terms of the change in energy. The smooth vs oscillating error of the two models can be seen once more.

The struggle with steeper gradients is even more apparent when the FNO is evaluated on a step-function.

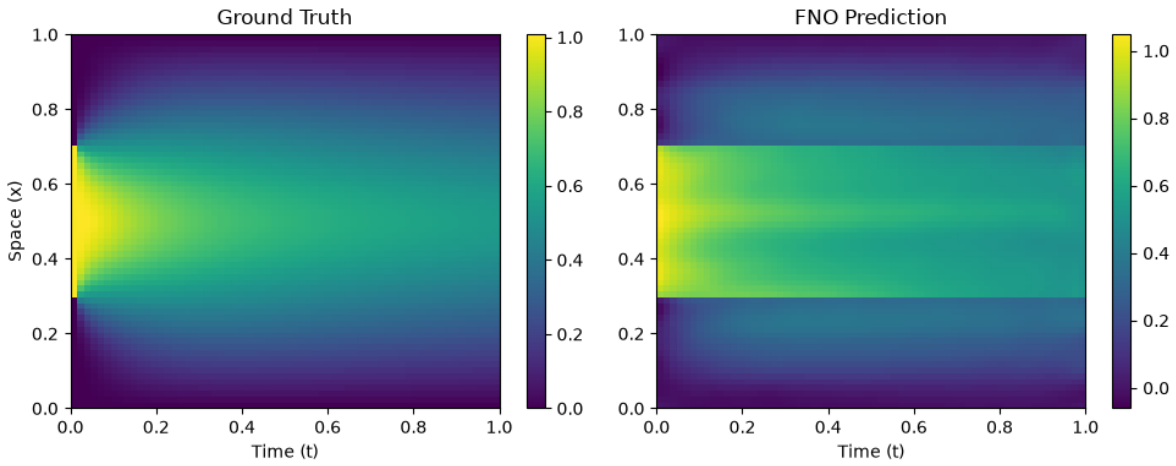


Figure 12: Default OSM evaluated with a step-function

Here the OSM fails to diffuse the heat contained in the step-function properly. Oscillations next to the cutoff of the step-function can also be seen. These are created by the Gibbs phenomenon, where the finite amount of Fourier modes cannot represent a jump accurately enough and leave behind large peaks around the jump.

Grid properties

As mentioned before the ARM cannot deal with a step-size different from the one it was trained on. The following plots show the ARM trained on the interval $t \in [0, 1]$ with 64 points in time, thus a step-size of $\Delta t = \frac{1}{64} \approx 0,015$, evaluated on 32 and 128 points in time on $t \in [0, 1]$.

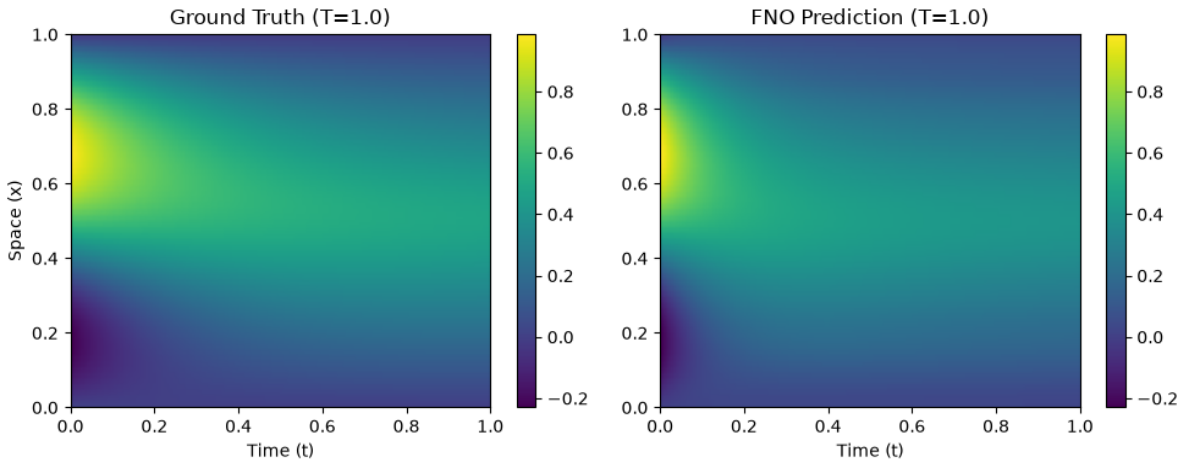


Figure 13: ARM trained on $\Delta t = \frac{1}{64}$, evaluated on $\Delta t = \frac{1}{128} \approx 0,007$

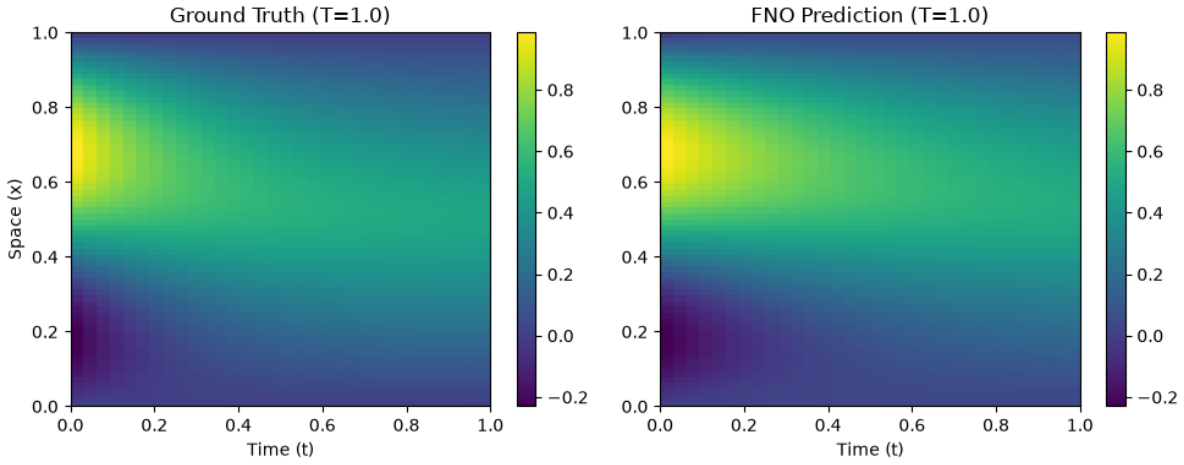


Figure 14: ARM trained on $\Delta t = \frac{1}{64}$, evaluated on $\Delta t = \frac{1}{32} \approx 0,031$

Because of the smaller step-size the model calculates the decay of the solution too fast, as it does two predictions in the time interval where it only learned to do one. Spatially the solution is correct but it is at the wrong point in time. For larger steps it is the other way around and the model's solution is too slow in time. Technically, this means that the same model could be used to calculate solutions with different time steps, as long as the solution is scaled correctly in the end on the time dimension. But if this is actually a good work around, instead of training the model again, would need to be tested. We also had the idea of possibly dealing with the OSM's disadvantage of a fixed time horizon, by using the solution at the final time T again as input for another forward pass and thus predict for another time interval of the same size. We did not explore this further either, the reason in both cases being time constraints, as it would have required adapting our implementation.

Lastly, the FNO's spatial independence is also shown here for the OSM. It perfectly handled a space grid of 1024 points despite being trained on 64.

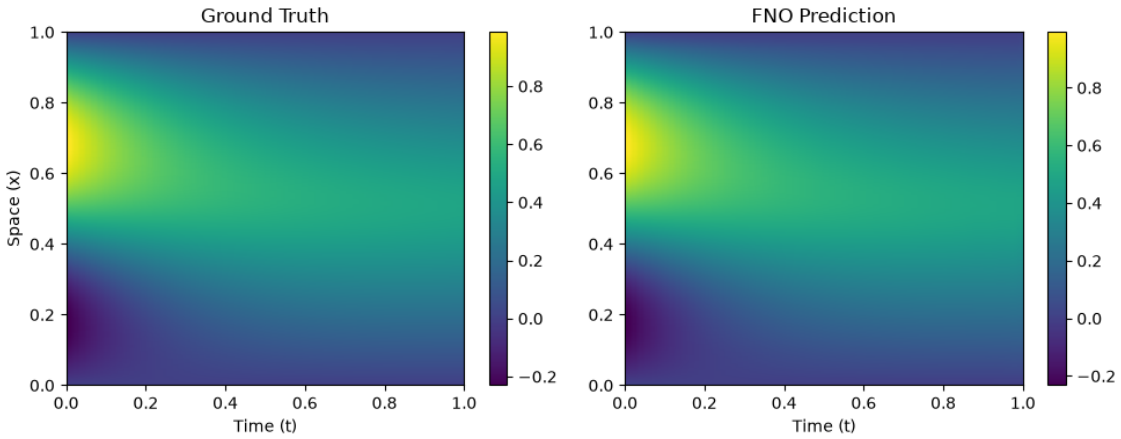


Figure 15: OSM trained on 64x64 and evaluated on 1024x64

With an average relative L^2 error per test sample of 0.0146 and RMSE of 0.0043 it slightly increased compared to the 64x64 test, possibly due to the numerical solution working better on the boundaries with a finer grid.

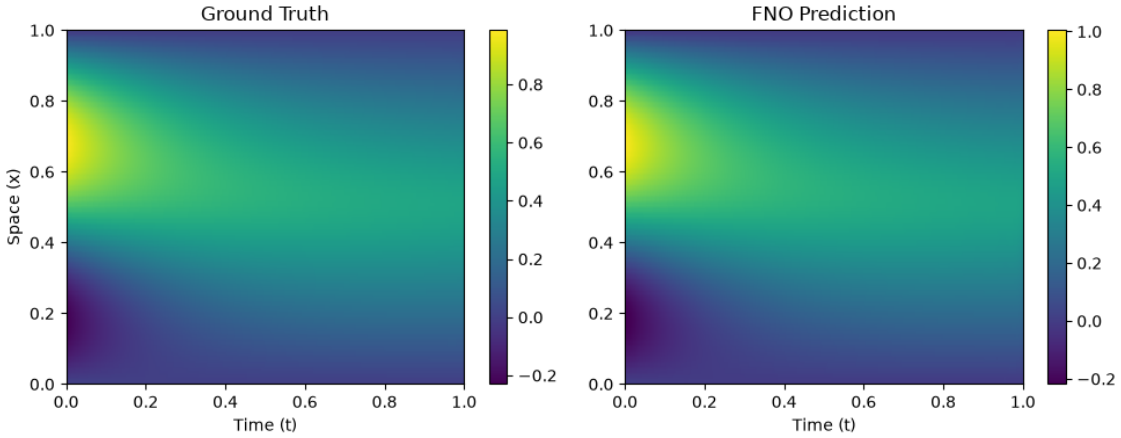


Figure 16: OSM trained on 64x64 and evaluated on 64x1024

In the time dimension the FNO also displays mesh independence but with a slightly larger relative error of 0.0346 and RMSE of 0.0081. The reason being that the FNO only kept the 16 first modes in Fourier space and discards the rest. While in the spacial dimension the higher frequencies are not of importance because they get diffused away quickly. In time that is not the case and the FNO is missing those. The low frequencies are too large to be resolved on the finer mesh for which the higher frequencies are needed.

Inference speeds vs Numerical calculation

One of the advantages of Neural Operators is that they can calculate a solution faster than numerical methods. Obviously, training time and time for numerical data generation has to be considered as well, but once they have been trained, they can be used to predict a solution for different initial data quickly. Meanwhile, the numerical method has to be applied again completely from scratch. For a simple system like the heat equation and a small and sparse domain as we have used throughout, the difference in speed is not much of a concern and the training time was the only noticeable thing. But for finer and more complex systems, which might also be applied in real time, this becomes way more important and the training time is not a point of focus. The following shows the aforementioned speed differences we encountered. The values were achieved on an AMD Ryzen 5 3600 6-core

processor.

For our default configuration of the OSM, it trained for 593 seconds and took an average of 7.27ms per 64x64 test sample to predict. Meanwhile the average numerical solve time per sample was 27.99ms. Going up to 1024 spatial grid points, the grid independence allows us to use the same model, resulting in an average inference speed of 138.99ms per test sample. The FEM solver handles a finer spatial grid well only increasing to 50.34ms per sample. Increasing the time discretization has a much bigger effect in the FEM.
The average sample took the numerical solver 404.16ms with 1024 points in time. Meanwhile the FNO of the OSM is only affected by the overall points in the domain but does not differentiate between time and space, achieving the nearly identical average inference speed of 138.74ms per sample. Because the ARM needs to do a forward pass recursively for all points in time, it is slower than the OSM with an average time of 178.52ms per 64x64 sample but having a total training time of only 57 seconds.

3.5 Conclusion

Our comparisons on the heat equation highlight the clear differences between the two models. Under the limited training budget, the OSM performed better in terms of both accuracy, inference speed and handling the physical properties of the heat equation. This makes sense because the heat equation is dissipative and smooths out over time. The global space-time convolutions of the OSM work well to capture this decay.

Of course, there are still many other comparisons that can be done or behaviors showcased which we mentioned but did not test. For example, we did not push both models to their absolute performance limits. We also did not test keeping different numbers of Fourier modes other than 16. Effects of higher spatial dimensions can be studied or the performance examined, if we do not broadcast the initial condition. Even so, our current results clearly highlight a lot of the main differences between the two.

4 Allen-Cahn Equation

4.1 System and Data Generation

Besides the time-dependent heat equation, we also considered the Allen-Cahn equation, which was originally introduced to describe the motion of anti-phase boundaries in crystalline solids [9].
We study this equation because it is structurally similar to the heat equation. The key differences are the additional nonlinear reaction term and the parameter ε, which replaces the diffusivity constant α. The equation is given by

$$\frac{\partial u}{\partial t} = \varepsilon^2 \Delta u - f(u) \tag{35}$$

$$u|_{t=0} = u_0(x), \quad x \in \Omega \tag{36}$$

$$u|_{\partial\Omega} = 0 \quad . \tag{37}$$

Here, u denotes the difference between the concentrations of the two components in a mixture, while ε is commonly referred to as the interfacial width. The nonlinear term f is the derivative of a smooth double-well potential, which attains its global minimum value of 0 at $u = \pm 1$ [10]. A typical choice for the non-linearity is $f(u) = u^3 - u$, which corresponds to the derivative of the Helmholtz free energy density [9].
Although the Allen–Cahn equation shares the diffusive nature of the heat equation, the nonlinear reaction term fundamentally changes the solution behavior. Rather than simply smoothing the initial condition over time, the reaction term drives the solution towards the stable phases $u = \pm 1$, resulting in the formation and motion of interfaces between these phases. The balance between diffusion and reaction determines the evolution of these interfaces, making the equation significantly more complex than the heat equation despite their similar mathematical structure.

To learn the solution operator of the Allen-Cahn equation, we first generate the training data. This is done in a similar manner to the heat equation.
We consider the spatial domain $\Omega = [0,1] \times [0,1]$ and the time interval $t \in [0,1]$. The interface parameter is chosen as $\varepsilon = 0.05$, and the non-linearity is given by the standard choice introduced above. We again impose homogeneous Dirichlet boundary conditions. The initial conditions consist of random smooth fields with values $u_0 \in [-0.2, 0.2]$. This interval is chosen because it lies in the vicinity of the unstable equilibrium $u = 0$, allowing small perturbations to grow into distinct phases separated by evolving interfaces, which are characteristic of the Allen-Cahn equation.

The spatial discretization is again performed using the FEM. For this purpose, we first derive the weak formulation of the Allen-Cahn equation, which is given by

$$\int_\Omega \frac{\partial u}{\partial t} v \, dx + \varepsilon^2 \int_\Omega \nabla u \, \nabla v \, dx = -\int_\Omega f(u) \, v \, dx \quad . \tag{38}$$

Using the same approach as in 2, with

$$V_h = \text{span}\{\phi_1, \phi_2, ..., \phi_N\} \quad , \tag{39}$$

$$u_h(t,x) = \sum_{j=1}^{N} U_j(t) \, \phi_j(x) \quad , \tag{40}$$

$$v = \phi_i \qquad i = 1, ..., N \quad , \tag{41}$$

the ODE system then looks like

$$M\,\dot{U}(t) + \varepsilon^2\,K\,U(t) = F(U) \quad . \tag{42}$$

Here, $F(U)$ represents the nonlinear contribution, with its i-th component given by

$$F_i(U) = \int_\Omega \sum_{j=1}^{N} (U_j - {U_j}^3)\,\phi_i\phi_j\,dx \quad . \tag{43}$$

Using the implicit backward Euler method, the time discretization is given by

$$(M + \Delta t\,\varepsilon^2\,K)\,U^{n+1} = M\,U^n + \Delta t\,F(U^n) \tag{44}$$

The FEM solutions are then interpolated from a triangular mesh onto a regular 64×64 Cartesian grid using linear interpolation.

Thus, the data generation process follows the same principle, where we vary only the randomly generated initial conditions $u(x, 0)$ and compute the corresponding space-time solutions $u(x, t)$.

4.2 Approach

Because the Allan-Cahn system evolves toward a steady-state equilibrium, the primary interest often lies in the final state at time T, rather than the time steps in between. While the FEM must calculate every time step to reach this final state, neural operators offer another alternative. Specifically, we can train the FNO to learn the direct operator mapping $\mathcal{G} : u_0(x, y) \to u_T(x, y)$, bypassing all intermediate computations. This approach represents a middle-ground between the OSM and ARM. We do not solve for the entire space-time domain, nor do we perform step-by-step iterations. Instead, we map the initial condition directly to the final state in a single forward pass.
It is worth noting that if the solution across the entire space-time domain were of interest, the ARM would likely be the better choice compared to the OSM in this case. This is because the rapid phase-separation dynamics at the beginning of the Allan-Cahn system would likely be smoothed out and by the global temporal Fourier transform used in the OSM.

In the following, we implemented this initial to final state mapping, projecting a 2D 64x64 spatial domain at $t = 0$ directly to the final state at $T = 1$, given as training data by the FEM. The FNO kept 16 Fourier modes across the two spatial dimensions with 32 hidden channels.
Unlike our previous benchmarking of the heat equation, the analysis of the Allen-Cahn system with the FNO was more exploratory. Since we were not performing a direct comparison between two architectures, we decided to experiment with varying learning rates and dataset sizes simply to investigate these behaviors, as it was something we had not done so in our previous tests. We trained the model for a larger number of epochs to observe how far the accuracy could be pushed, now without having to restrict the optimization to a fixed number of gradient steps. However, we observed that increasing the dataset size or the number of epochs quickly increased both training and data generation times. In particular, generating the ground-truth via FEM was highly time consuming, vastly exceeding the inference speed required by the trained FNO to predict the final state.

4.3 Results

As described, we varied the different settings of the learning process to examine how their performance differs. The initial configuration consisted of 250 training samples, 50 test samples, and 500 training epochs. The resulting performance is presented below 17. The FNO prediction does not yet match the FEM solution accurately, although it successfully captures its overall structure. This

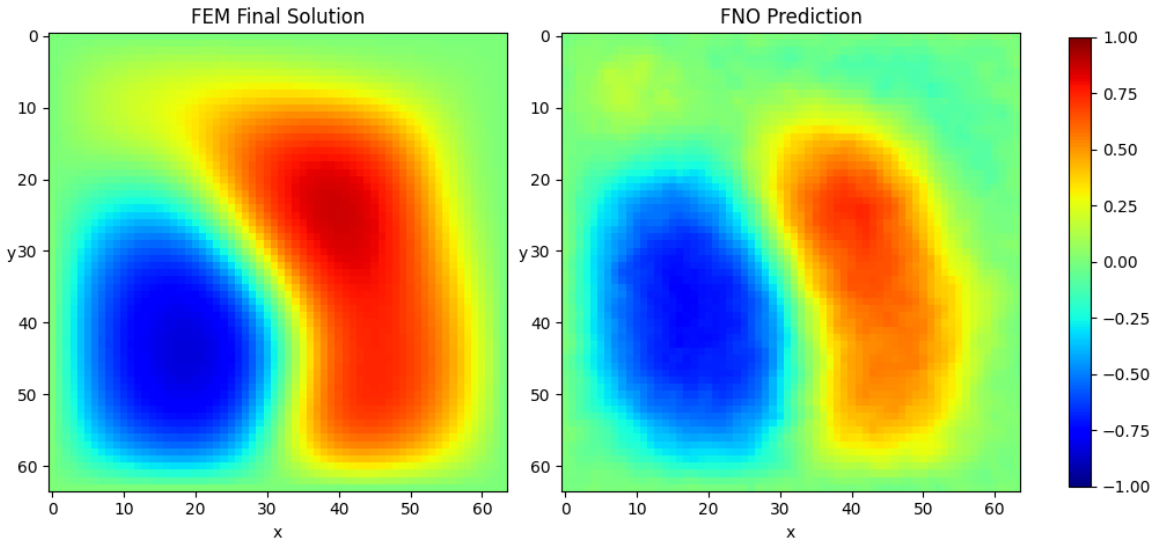


Figure 17: Results from 250 training samples, 50 test samples and 500 epochs.

observation becomes even clearer in the error plot shown in Figure 19a. Here, deviations from white indicate regions where the FNO prediction differs from the FEM solution. As can be seen, these deviations are quite pronounced, highlighting the areas where the prediction differs most from the reference solution.

For the next experiment, we reduced the number of training epochs to investigate their influence on the model's performance. Specifically, we decreased the number of epochs from 500 to 200 while keeping the number of training and test samples unchanged. The resulting predictions are shown in Figure 18. Here, the predictions still deviate from the FEM solution, although they are slightly more accurate than those obtained after 500 epochs. This improvement is also reflected in the error plot shown in Figure 19b, where the deviations from white appear to be somewhat smaller, indicating a reduction in the prediction error. The effect of the number of training epochs on the quality of the predictions requires further investigation before any firm conclusions can be drawn.

For the subsequent simulations, we adjusted the time step, as the value used in the first two simulations resulted in excessively long computation times (the runtime was over two hours with a 13th Gen Intel(R) Core(TM) i5-1335U processor). Specifically, we increased the time step from $dt = 0.001$ to $dt = 0.05$. In addition, we adjusted the color bar to improve the visualization of the results, making the behavior of the Allen–Cahn equation easier to observe. This led to a strong decrease of the runtime, which was 19 min and 35 s for 250 training samples, 50 test samples, 100 epochs for a step size of 0.05. However, increasing the time step reduces the accuracy of the FEM solution.

For the following simulations, we used 250 training samples and 50 test samples while evaluating the model for different numbers of training epochs. Specifically, we considered 100 and 300 epochs. The results for 100 epochs can be seen in 20 and the one for 300 in 21. Increasing the number of training epochs from 100 to 300 leads to small changes in the results. The solution plots show a slight improvement, while the error plots and the corresponding relative L^2 error indicate a small enhancement in the overall performance. This becomes more evident when observing the change in the color bar range between the 100-epoch case 20 and the 300-epoch case 21.The limited improvement could be related to the learning rate of the neural operator, which may be too high. A learning rate that is too large can prevent the model from converging smoothly, causing the loss to fluctuate or increase during later stages of training.

As the next test, the number of test samples was increased to 100. The resulting solution is shown in 23, with the corresponding error plot presented in 25a. In this case, no clear difference can be observed between the results, particularly when considering the relative L^2 errors, which remain identical. Only a slight variation in the predicted solution can be observed in the FNO output. This is expected, as the test samples are only used to evaluate the performance of the trained neural operator and do not influence the training process. Since the additional test samples represent a similar distribution of initial conditions, the overall performance remains nearly unchanged. Finally, the learning rate was adjusted from $5 \cdot 10^{-4}$ to $1 \cdot 10^{-4}$. The resulting prediction is shown in 24, together with the corresponding error plot in 25b. Compared to the original learning rate, the prediction shows a slight decrease in performance, which is reflected by a small increase in the error. This indicates that the lower learning rate did not improve the convergence of the neural operator for this specific case. The reduced learning rate did not improve the prediction accuracy because the original learning rate was already sufficiently stable for the optimization process. Furthermore, since the number of training epochs was kept constant, the smaller learning rate may not have allowed the model to reach the same level of convergence within the available training time.

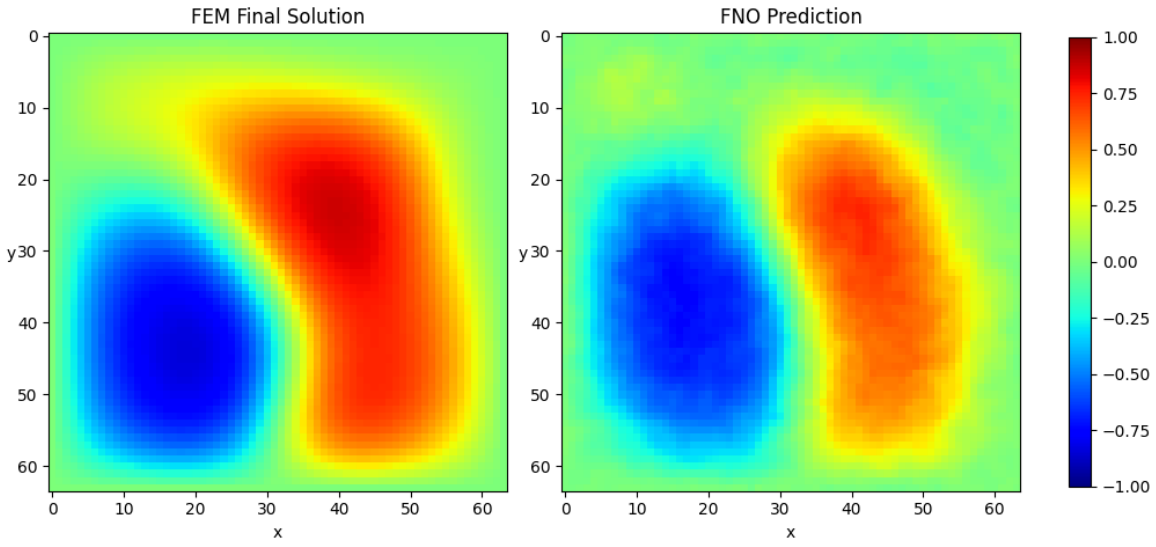
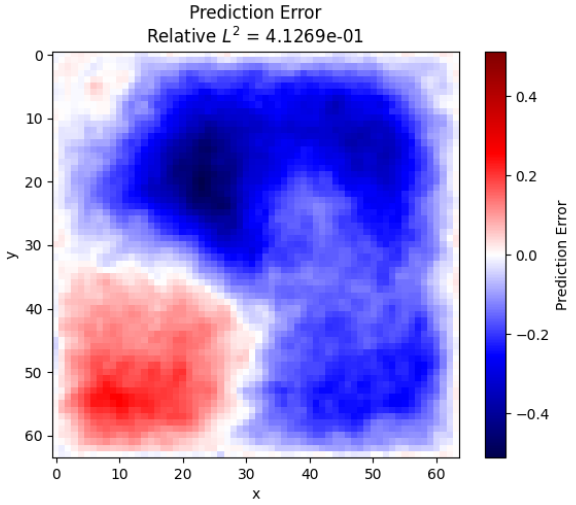
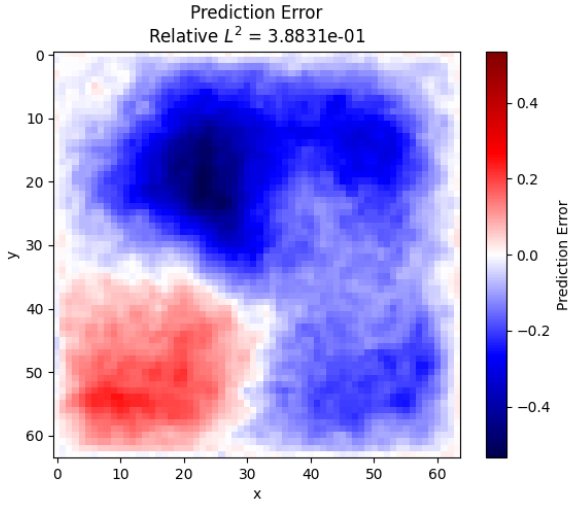


Figure 18: Results from 250 training samples, 50 test samples and 200 epochs.



(a) Error (difference) plot for 250 training samples, 50 test samples and 500 epochs.



(b) Error (difference) plot for 250 training samples, 50 test samples and 200 epochs.

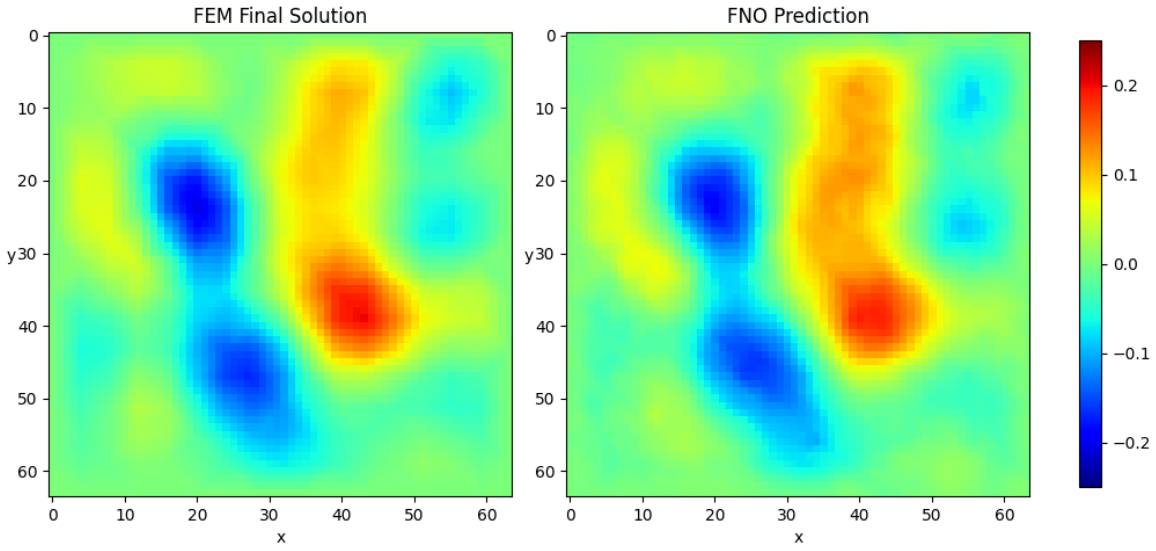


Figure 20: Results from 250 training samples, 50 test samples and 100 epochs for increased step size.

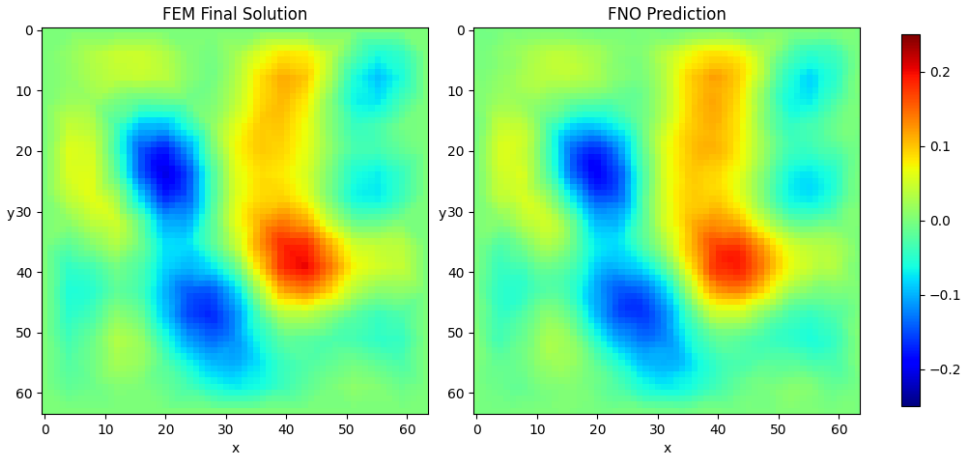
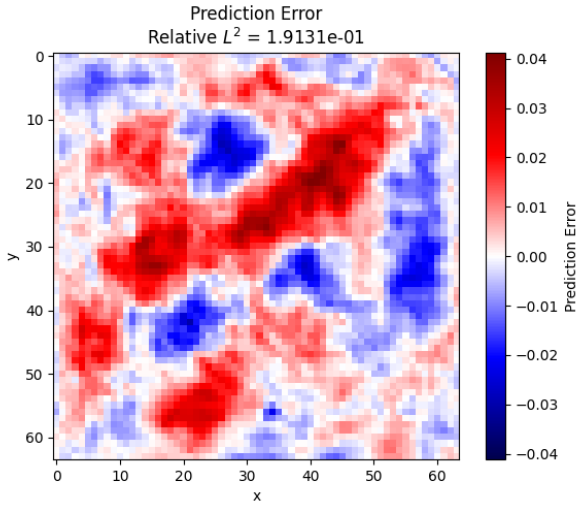
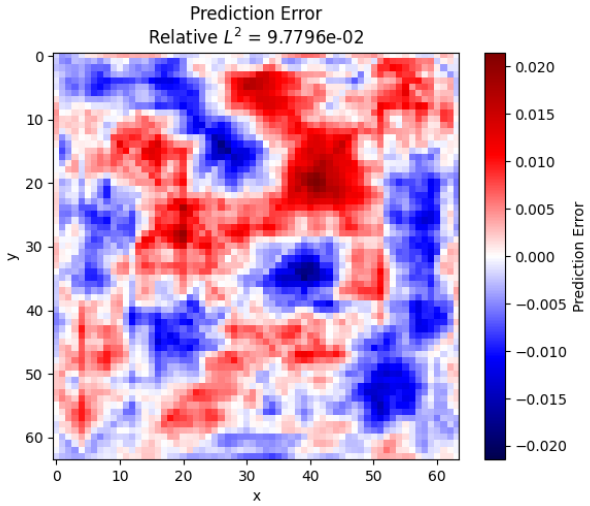


Figure 21: Results from 250 training samples, 50 test samples and 300 epochs for increased step size.



(a) Error (difference) plot for 250 training samples, 50 test samples and 100 epochs for increased step size.



(b) Error (difference) plot for 250 training samples, 50 test samples and 300 epochs for increased step size.

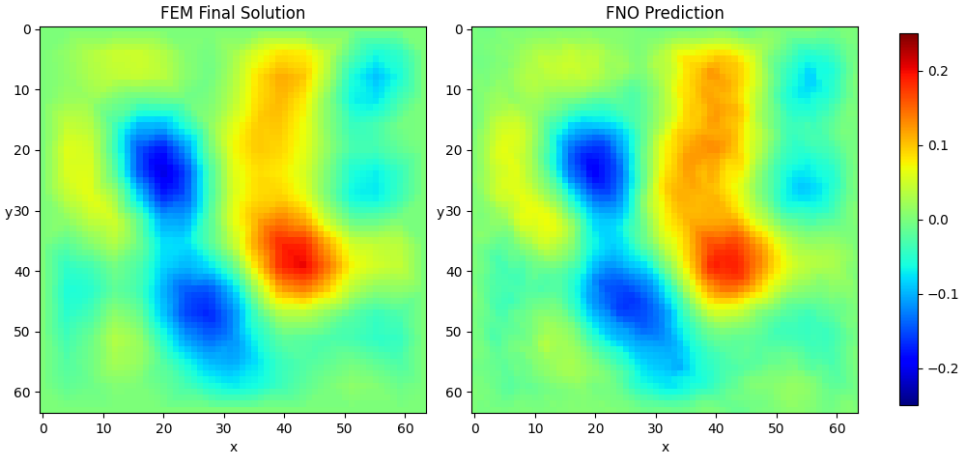


Figure 23: Results from 250 training samples, 100 test samples and 100 epochs for increased step size.

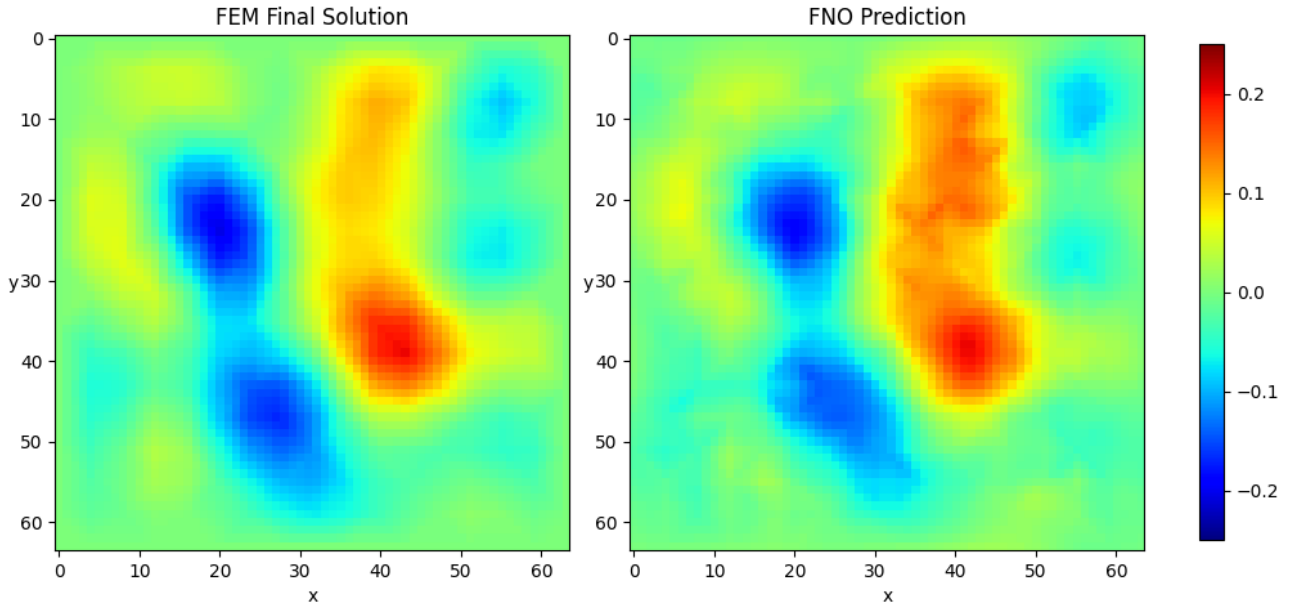
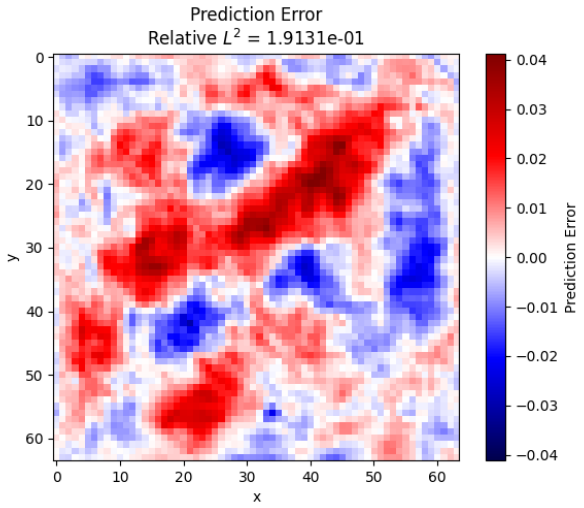
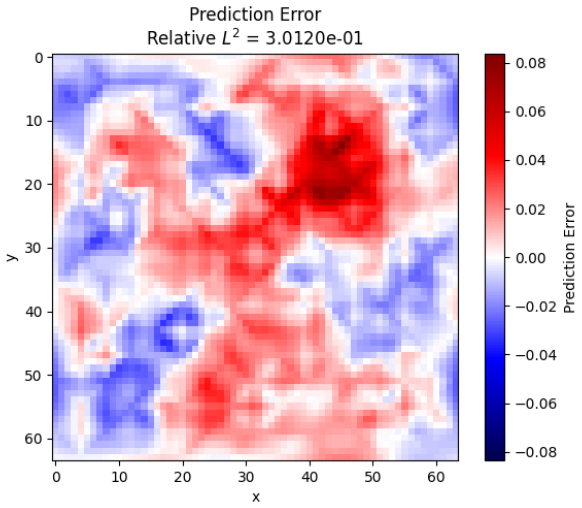


Figure 24: Results from 250 training samples, 50 test samples and 100 epochs for increased step size and smaller learning rate.



(a) Error (difference) plot for 250 training samples, 100 test samples and 100 epochs for increased step size.



(b) Error (difference) plot for 250 training samples, 50 test samples and 100 epochs for increased step size and smaller learning rate.

4.4 Conclusion

The Allen–Cahn equation is closely related to the heat equation. Both describe diffusion through a second-order spatial derivative, but the Allen–Cahn equation differs in two important aspects: it has a different coefficient multiplying the diffusion term and, more significantly, it includes an additional non-linear reaction term that is absent from the heat equation. Although these differences may appear minor, they fundamentally alter the behavior of the solution. In particular, the non-linear term introduces phase separation and interface dynamics, leading to solution characteristics that are not observed in the classical heat equation. When analyzing the FNO predictions, several observations can be made. First, the number of training epochs has a noticeable influence on the relative L^2 error: increasing the number of epochs generally leads to a lower error, although it is expected that this improvement will eventually plateau as the model converges. Second, increasing the number of test samples has little effect on the prediction error, since the test data are used only for evaluation and do not influence the training process. Finally, we investigated the impact of different learning rates and observed that, in this case, reducing the learning rate resulted in a higher relative L^2 error. A likely explanation is that, with the fixed number of training epochs, the smaller learning rate prevented the model from converging sufficiently, suggesting that additional training epochs would be required to fully benefit from the lower learning rate.

These observations may differ from those obtained for the heat equation because the Allen–Cahn equation is substantially more challenging to learn. While the heat equation is linear and primarily exhibits diffusive behavior, the Allen–Cahn equation includes a non-linear reaction term that introduces phase separation and the evolution of sharp interfaces. As a result, the solution contains more complex spatial features and stronger non-linear dynamics. These characteristics typically require a model with greater representational capacity or more extensive training to accurately capture the underlying behavior. As a consequence, the influence of parameters such as the number of training epochs and the learning rate can be more prominent than for the heat equation.

References

[1] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations (ICLR)*, 2021.

[2] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural Operator: Graph Kernel Network for Partial Differential Equations. *arXiv preprint arXiv:2003.03485*, 2020.

[3] Z. Li, K. Azizzadenesheli, K. Bhattacharya, N. Kovachki, B. Liu, A. Stuart, and A. Anandkumar. Multipole Graph Neural Operator for Parametric Partial Differential Equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 6755–6766, 2020.

[4] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning Nonlinear Operators via DeepONet Based on the Universal Approximation Theorem of Operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[5] T. Chen and H. Chen. Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[6] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations. *Journal of Computational Physics*, 378:686–707, 2019.

[7] V. Duruisseaux, J. Kossaifi, and A. Anandkumar. Fourier Neural Operators Explained: A Practical Perspective. *arXiv preprint arXiv:2512.01421*, 2025.

[8] NeuralOperator Developers. NeuralOperator: Learning in Infinite Dimension with Neural Operators. Available at `https://neuraloperator.github.io`, 2026.

[9] K. Poochinapan and B. Wongsaijai. Numerical Analysis for Solving Allen–Cahn Equation in 1D and 2D Based on Higher-Order Compact Structure-Preserving Difference Scheme. *Applied Mathematics and Computation*, 434:127374, 2022.

[10] X. Feng and A. Prohl. Numerical Analysis of the Allen–Cahn Equation and Approximation for Mean Curvature Flows. *Numerische Mathematik*, 94(1):33–65, 2003.

[11] J. Schöberl. Heat Equation. *Interactive Finite Elements (iFEM)*. Available at `https://jschoeberl.github.io/iFEM/timedependent/intro/heatequation.html`. Accessed: June 30, 2026.