AISE

Final Project

Edoardo Tarcisio Visconti

Fall 2024

1 Task 1

1.1 Task 1.1

To address Task 1.1, I implemented a vanilla Fourier Neural Operator (FNO) architecture. The model consists of 5 Fourier layers, each combining a spectral layer and a convolutional layer, with a width of 64 and 16 modes per layer. The final fully connected layer has 128 nodes. To enhance the model's performance on the validation set, I incorporated classical batch normalization after each Fourier layer and included the spatial coordinates as an additional input channel.

To further optimize the model, I experimented with various techniques, including fixed positional encoding, dropout, and expanding the network's capacity. However, none of these adjustments significantly reduced the error on the validation set. These alternative approaches are documented and commented in the corresponding Jupyter notebook.

This architecture achieves an error (relative L2 Norm) of 2.60% in the provided test dataset.

1.2 Task 1.2

The results in tabel below are consistent with empirical evidence and with what we have discussed during lectures: the FNO model has the smallest error when testing resolution equals training resolution (64 grid points). It has good generalization properties on finer resolutions, such as 96 and 128 grid points, where achieves a slightly worse performance than on the training mesh (but still comparable in this setting). On the other hand the model struggles on the coarsest mesh as the error grows to 13% (aliasing error arises).

32	64	96	128
13.06%	3.14%	3.42%	4.12%

Table 1: Relative L2 Norm for different resolutions.

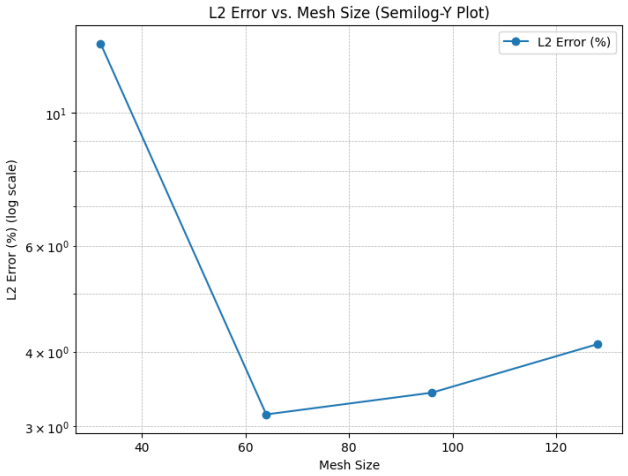

Figure 1: L2 error vs mesh size

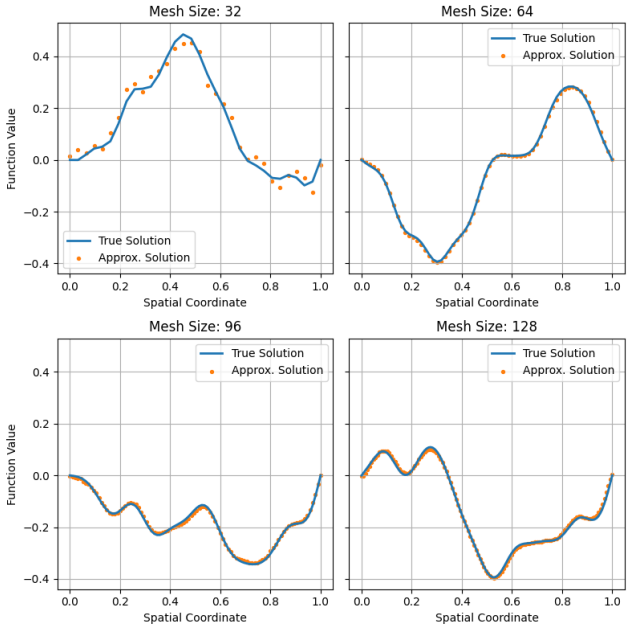

Figure 2: Plot functions for 1.2

1.3 Task 1.3

The error in the dataset generated under initial OOD conditions is 7.51%. This aligns with the expectation that the FNO was trained on functions with fewer frequency components, resulting in a limited ability to capture highly oscillatory behavior. Addressing this requires incorporating diverse training examples and potentially expanding the modes (and thereby the width) of the model architecture for broader generalization.

1.4 Task 1.4 + bonus

To complete the task, I first generated the dataset using 64 of the 128 trajectories in the training set, obtaining 960 input/target couples on which the training of my model was performed.

To train the network, I experimented with a variety of techniques and configurations, which can be summarized as follows:

- **Conditional Batch Normalization:** Applied conditional batch normalization based on time, both without any embedding and with an 2/4/8-dimensional learnable embedding.
- **Stacking Inputs:** Tested stacking the input with time and with spatial coordinates. These experiments were conducted both without embedding and with either an 2/4/8-dimensional learnable embedding (for time) or a positional embedding (for space).
- **Loss Functions:** Used different loss functions, including L^1-norm, L^2-norm, and H^1-seminorm. Finite differences were used to approximate derivatives for the H^1-seminorm.
- **Network Architecture:** Introduced skip connections within the Fourier layers and experimented with different network sizes and numbers of Fourier modes.
- **Optimization Techniques:** Tested various learning rate schedulers and optimizers. Additionally, hyperparameter optimization and fine-tuning were performed.

In the end, the model that performed best on the validation test made use of:

- learnable parameters skip connections;
- 2 Fourier layers (to address overfitting issues);
- input stacked with a 2-dimensional positional embedding of time and a 2-dimensional learnable embedding of time;
- batch normalisation conditional on time (using FILM);
- training using the H1 loss to control the gradients of the predictions;
- AdamW optimizer and CyclicLR scheduler, which made a significant difference in reducing the error.

The error on the testset was:

Time	0.0	0.25	0.5	0.75	1.0	1.0 (OOD)
L2 Error	4.25	39.09	36.51	27.77	20.24	46.41

Table 2: Relative L2 norm at different times

I believe that further experimentation, even focusing solely on optimizers and learning rate schedulers, could potentially reduce the validation error. This belief stems from repeated attempts to resume training from the point where the model achieved its best validation performance. In each instance, the training loss plateaued or experienced exploding gradients, suggesting that fine-tuning the optimization process could yield further improvements.

While implementing a U-Net architecture might significantly lower the error, I decided not to pursue this approach because the task specifically required the use of an FNO and an encoder-decoder architecture would not align with the task's requirements.

The error at $t = 1.0$ is approximately 20%. This is higher than in Task 1.2, as the model in this case is not yet fully optimized. However, this result is not entirely unexpected, even if the model had been at its optimum, the error would have probably been higher, as it is generally more challenging to generalize across different timeframes compared to focusing on a single timeframe.

The decreasing trend of the error as time progresses might initially appear counterintuitive. For smaller time intervals, there are more input-target pairs available for training (e.g., 512 pairs for $t = 0.25$, 384 for $t = 0.5$, 256 for $t = 0.75$, and 128 for $t = 1.0$) and one would expect higher errors for predictions at larger times due to fewer training samples and the increasing difficulty of extrapolation.

However, I think, this pattern in the error can be justified by considering the nature of the dynamics at different time intervals. Specifically, the dynamics between $t = 0.00$ and $t = 0.25$ may differ significantly from those between $t = 0.75$ and $t = 1.00$, even though the lag is the same. This discrepancy suggests that the underlying system's behavior could be more complex or harder to approximate in earlier intervals, leading to higher errors despite the greater number of training pairs available.

The error on the OOD dataset is 46%, which is approximately 230% of the in-distribution (ID) error at the same time. Similarly, in Task 1.3, the OOD error was also 230% of the ID error (7.50% compared to 3.14%). This consistency suggests that observing evolutions over different time lags does not significantly help the model generalize to unseen initial conditions (ICs). This highlights a systematic limitation in the model's ability to extrapolate beyond the training distribution.

Note: given the longer training, model parameters are saved in .pth files so that results can be verified.

2 Task 2

To solve this task, I implemented functions to evaluate derivatives and a regression function in the file `aux_for_PDE_find.py`. I decided to approximate the coefficients up to the third decimal figure.

2.1 Guess for the first dataset: Burger's equation

$$\frac{\partial u}{\partial t} = 0.101 \cdot \frac{\partial^2 u}{\partial x^2} - 0.998 \cdot \left(u \cdot \frac{\partial u}{\partial x} \right)$$

2.2 Guess for the second dataset: KdV equation

$$\frac{\partial u}{\partial t} = -0.995 \cdot \frac{\partial^3 u}{\partial x^3} - 5.978 \cdot \left(u \cdot \frac{\partial u}{\partial x} \right)$$

2.3 Guess for the third dataset: coupled reaction diffusion equations

$$\begin{cases} \frac{\partial u}{\partial t} = 0.949 \cdot u + 0.202 \cdot v + 0.095 \cdot \frac{\partial^2 u}{\partial x^2} + 0.099 \cdot \frac{\partial^2 u}{\partial y^2} - 0.947 \cdot u^3 + 0.766 \cdot v^3 - 0.948 \cdot (u \cdot v^2) + 0.766 \cdot (u^2 \cdot v) \\ \frac{\partial v}{\partial t} = -0.206 \cdot u + 0.917 \cdot v + 0.093 \cdot \frac{\partial^2 v}{\partial x^2} + 0.093 \cdot \frac{\partial^2 v}{\partial y^2} - 0.763 \cdot u^3 - 0.915 \cdot v^3 - 0.760 \cdot (u \cdot v^2) - 0.916 \cdot (u^2 \cdot v) \end{cases}$$

2.4 Regression

I will discuss the computation of derivatives in the next paragraph. For now, let us assume the derivatives have already been computed. When performing regression, classical methods like Ordinary Least Squares or Ridge/Lasso are not effective because many features are highly correlated with each other. Additionally, PCA is unsuitable since we are interested in retaining the original features as they are.

Following the methodology outlined in the referenced paper, I implemented a custom regression function named `myregression`. This function iteratively performs Ridge regression and selects relevant features based on the magnitude of their coefficients, retaining only those that exceed a predefined threshold. The number of iterations, the Ridge regression alphas, and the threshold are input parameters that require tuning. The parameters that performed well across all three datasets were:

- Alphas = [0.1, 0.01, 0.0] (corresponding to three regression iterations)
- Threshold = 0.05.

However, for the second dataset, this approach resulted in too many features being selected in the final output. To address this, I raised the threshold to 0.5. While effective, this approach lacks generality, as the threshold must be manually adjusted based on the output and the number of features selected in each iteration.

To overcome this limitation, I extended the approach in a new function called `myregression2`. Instead of using a fixed threshold on the coefficients, I applied the threshold to the product of the norm of each feature and its coefficient. With this modification, the same hyperparameters —threshold = 4 and the previously mentioned alphas— could successfully handle all three datasets. However, a key limitation of this method is that the norm-based threshold is sensitive to the sample size. In this case, a threshold of 4 works because the sample size for all three problems is on the order of 10^5 (using all data for the first and second datasets and subsampling for the third).

To achieve even greater generalization, I attempted to use metrics that are agnostic to sample size in a third function, `myregression3`. This approach aimed to select features based on their statistical significance by examining p-values. Unfortunately, this method was ineffective because many p-values were either exactly 0 or smaller than 10^{-10}.

To further extend this approach, one could explore statistical metrics such as AIC, BIC, or R^2 performance combined with stepwise selection, applying a strong penalty for model complexity. However, these metrics are generally only applicable to classical linear regression models, which struggle to select features in the initial iterations due to multicollinearity and they often produce large coefficients, which complicates feature selection.

2.5 Derivatives

In order to compute the derivatives, I tried different approaches. By analyzing the dataset and plotting some of the trajectories, I observed that the values were equally spaced in time and space. From the plots, I also inferred that the data were not noisy (or the noise was negligible). Moreover, the step size (in both time and space) was reasonably small, on the order of 10^{-1}. These conditions made the problem suitable for solving with finite differences (FDs).

Therefore, I solved all tasks using FDs (see `aux_for_PDE_find.py` for the specific implementation) and above guesses are based on this method. Under the given assumptions, this method also performed best in terms of memory and CPU usage.

However, finite differences have significant limitations. They work well only on equally spaced data and under low noise conditions. For this reason, I explored alternative methods to solve the problem, using the FD solution as a benchmark. While these alternative methods can address noisy and non-uniform data, they tend to be less precise, as errors accumulate in higher-order derivatives.

For the first and second tasks, I used a neural network approach and obtained results equal with the FD method. However, this approach requires careful fine-tuning of hyperparameters and the choice of activation functions to replicate the underlying solution to the PDE without interpolating the points. Minimizing the mean squared error (MSE) alone is not a useful metric when training the neural network; instead, we need to balance the error on the function while avoiding interpolation. Interpolation can lead to large errors in higher-order derivatives. To mitigate this, it helps to keep the network small, analogous to using a lower degree in polynomial interpolation.

Accurately estimating the error in derivatives becomes significantly more challenging when FDs cannot be used as a benchmark or incorporated into the loss function, such as in computing an H^P loss. To evaluate how neural networks approximate derivatives, I analyzed the relative L^2 norm error against FD approximations in the second problem. While the error on the function remained low (no more than 3%), the error on the derivatives varied widely, ranging from 10% to 60% when the network size was merely doubled. Moreover, there is often a trade-off: achieving low error on the function tends to result in interpolation and higher error on the derivatives.

Given these challenges, I approached the neural network method cautiously. My solution relied heavily on comparing results with those obtained using the FD approach. I believe regularization techniques could potentially mitigate these issues.

Given these observations, I experimented with an alternative approach—spline interpolation—to address the third problem. Specifically, I utilized quintic splines, which offer a high degree of flexibility and are well-suited for accurately capturing the behavior of functions.

To balance the trade-off between overfitting and handling noisy data, I introduced the smoothing coefficient s as a hyperparameter. This parameter plays a critical role because:

- A low value of s allows the spline to closely follow the function and replicate the data, making it ideal for smooth, noise-free datasets.
- A higher value of s introduces greater smoothing, which reduces sensitivity to noise but may compromise accuracy. This can, however, improve the estimation of derivatives in noisy scenarios.

For this problem, I selected a small smoothing coefficient, $s = 0.01$, since the data were relatively noise-free. In cases involving noisier datasets, a larger s could be chosen to prevent overfitting.

Using this spline-based approach, I successfully identified the same terms as those obtained with the finite difference (FD) method. However, the coefficients from the spline interpolation differed, likely due to error accumulation in higher-order derivatives—a challenge more pronounced in spline methods compared to finite difference approximations.

2.6 Size of the library

For the first two datasets, I used the library and included all terms that can be obtained by multiplying u, u^2, and u^3 with $\frac{\partial u}{\partial x}$, $\frac{\partial^2 u}{\partial x^2}$, and $\frac{\partial^2 u}{\partial y^2}$, as well as including time t and a bias term. I did not include any higher-order derivatives in time or space, as I followed the given hint. Additionally, I avoided multiplying derivatives with each other, as such terms are uncommon in typical Partial Differential Equations. In this way the library has a size of 17.

By plotting the dataset, it already becomes possible to identify which terms are significant (diffusion, advection...) and whether strong nonlinearities should be considered. For example, if evident from the plots, one could consider adding nonlinear terms such as $\log u$, $\exp(u)$, or other similar transformations to better capture the behavior of the system.

To address the third problem, I began with a smaller dataset consisting of 30 features. This dataset excluded all cross-products of u^2, v^2, u^3, and v^3 between themselves and the derivatives.

At the conclusion of the project, after identifying a method that worked consistently across all cases, I extended the feature set in the Θ matrix to include 51 features. This extension incorporated the omitted terms. Remarkably, I obtained the same results using the same hyperparameters (see `PDEfind3_bigds.ipynb`).

This outcome demonstrates the robustness and effectiveness of the regression function I implemented.

3 Task 3

3.1 The dataset

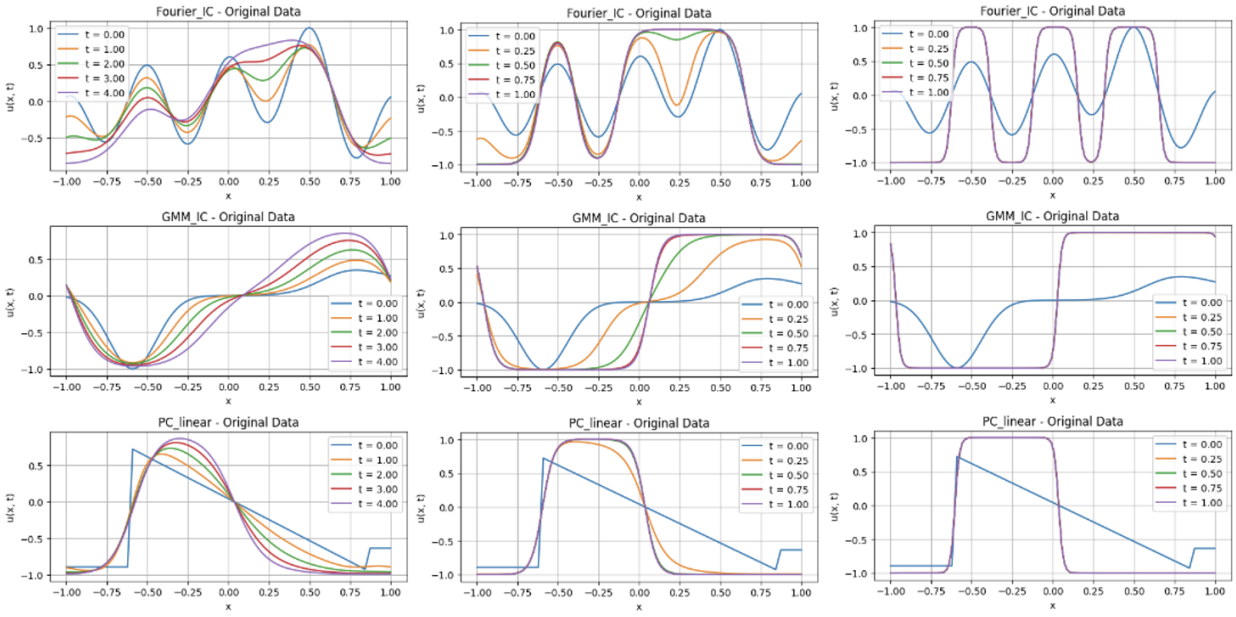

Figure 3: AC equation with different IC and ϵ

To generate the dataset, I decided to keep the values of ϵ as suggested, that is, equal to 0.1, 0.05, and 0.02. Moreover, I rescaled the time by a factor of 50; from now on, all times are considered divided by 50. As we can see from the plots, this choice of time and ϵ values allows us to capture the evolution of the Allen-Cahn equation under different regimes. For $\epsilon = 0.1$, the model does not reach convergence but evolves slowly from the initial condition. For $\epsilon = 0.05$, the model slowly reaches convergence, and we can observe the transition from the initial condition to convergence. For $\epsilon = 0.02$, the evolution is too fast to be captured by the model, and for any time different from zero, the model is already at convergence. The value of ϵ also influences the dynamics of the interface, the smaller ϵ si the sharper the interface between the two potentials.

3.2 The model and ID Testing

To complete the task, I implemented a model that takes the initial condition (IC) as input and outputs the entire trajectory of the solution from $t = 0$ to $t = 1$. It is worth noting that, in the forward pass, I chose to generate each time snapshot starting directly from the IC, rather than using the previous condition as a starting point. This approach proved to be more effective in this specific setting, as it prevented the accumulation of errors over time. To enable the network to capture the behavior of the Allen-Cahn (AC) equation, I utilized an FNO with 4 layers, a width of 128, and 16 modes. Additionally, I employed a modified version of time batch normalization (FILM), which takes as input the sum of the 8-dimensional learnable embeddings of ϵ and time.

This technique significantly reduced the error on the test dataset. I also attempted to concatenate space and/or time (as a repetition of the time I wanted to simulate) to the input vector, but this made no significant difference on the validation set. To analyze the error under different regimes, we provide the following table 3 that highlights the performance across varying time steps and total averages. The largest error occurs when $\epsilon = 0.02$. This is likely because, in most cases within the training set, the model has already converged before $t = 0.25$. When this does not happen, the model makes mistakes (see 4); however, it correctly identifies that the equation has not yet converged. This issue could potentially be resolved by adding more examples of this type into the dataset. For the other two scenarios, the predictions are very close to the targets (see the notebook `Task3_testing.ipynb` for more plots.

Time	**Regime (ϵ)**			**Total**
	0.1	**0.05**	**0.02**	
0	1.91%	2.07%	2.68%	2.22%
0.25	1.35%	1.58%	9.34%	4.09%
0.5	1.31%	2.27%	10.93%	4.83%
0.75	1.49%	3.78%	10.80%	4.35%
1	1.76%	5.42%	10.99%	6.06%
Total	1.56%	3.03%	8.95%	4.51%

Table 3: Error Table, for t and(ϵ).

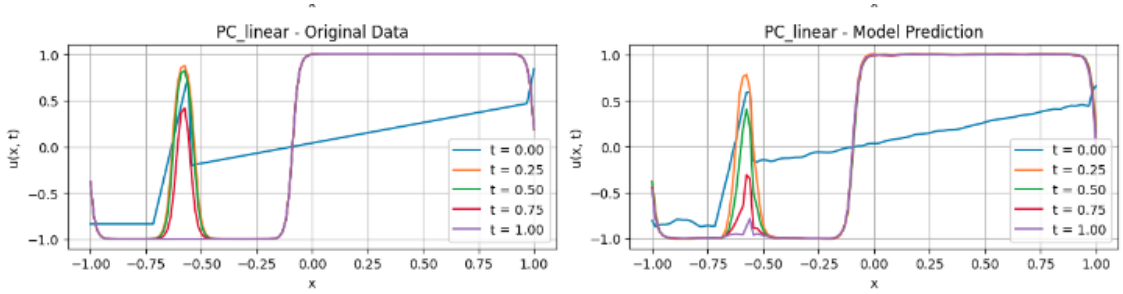

Figure 4: Worst case scenario for my model ID

3.3 OOD Testing

The error on intrapolation for ϵ is (obviously) lower than for extrapolation. This error could likely be reduced by adding more regimes on a finer scale (i.e., more ϵ values closer to each other) to the training set. When $\epsilon = 0.2$, the evolution is too slow, and the model fails to capture it as the approximation appears to move too fast compared to the true evolution. Conversely, when ϵ is too small, the model seems to "want" to continue the evolution even if one potential is reached. However both these error come with a trade-off, for example the first one is about 70% if we train the model for less epochs (600 instead of 1000). I believe the first issue can be addressed by incorporating slower regimes into the dataset. For the second issue, I would suggest trying again with an H^1 loss (using FDs for the derivative in space) to better control the behavior of the gradients.

ϵ	**Error (%)**
$\epsilon = 0.2$	243.43%
$\epsilon = 0.033$	12.17%
$\epsilon = 0.008$	22.83%

Table 4: OOD error on diverse ϵ

This model demonstrates good generalization properties on sharper or more oscillatory initial conditions (ICs). When tested with Fourier ICs containing more modes than those seen during training (trained on 4 modes and tested on 8 modes), the model exhibits an error of 15%. Similarly, when tested with piecewise linear ICs containing more breakpoints (and consequently more discontinuities) than those in the training set (trained on up to 7 breakpoints and tested on 15 breakpoints), the model shows an error of 9.64%. The lower error in this case, compared to the ϵ OOD scenarios, is likely due to the ability of the Allen-Cahn (AC) equation to quickly smooth out sharper or more oscillatory conditions.

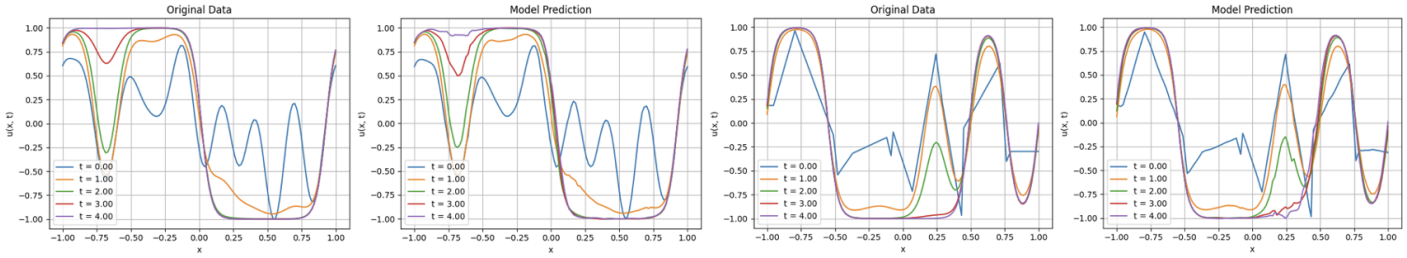

Figure 5: An example of OOD prediction (2nd and 4th image) compared with the OOD IC (1st and 3rd)

3.4 Generalization performances on training different than testing

To further evaluate the generalization capabilities and the dependency on the initial condition, I trained my model for 200 epochs on each different IC type. I then tested it in two scenarios: zero-shot and after fine-tuning (with 50 additional epochs). From this routine, we observe that the worst zero-shot performance occurs when testing on the the GMM IC. I believe this is because the GMM IC is the steadiest, and it is not clear which potential the solution will tend towards. Therefore, the model requires exposure to some of these cases during fine-tuning. On the other hand, when the model is trained on the sharpest transitions, i.e., the PC linear IC, it is capable of generalizing to smoother transitions. Conversely, when trained on the Fourier IC, which is closest to the smooth evolution of the AC equation, the model struggles to generalize to the other ICs.

			Testing IC		
			Fourier	GMM	PC linear
training IC	Fourier	zeroshot	2.52%	59.63%	31.92%
		finetuned	/	16.78%	8.76%
		improvement	/	-71.86%	-72.56%
	GMM	zeroshot	20.23%	12.23%	16.15%
		finetuned	9.58%	/	10.58%
		improvement	-52.64%	/	-34.49%
	PCl	zeroshot	4.80%	17.27%	4.72%
		finetuned	3.82%	14.35%	/
		improvement	-20.42%	-16.91%	/

Table 5: OOD error on ϵ

4 Bonus Task: Stability Analysis

Proof:

We know that u is a weak solution of the Allen-Cahn equation. Let us assume it satisfies the equation with homogeneous Neumann boundary conditions, which is the natural formulation for the Allen-Cahn equation due to its physical relevance (however the proof works for any boundary condition that allows the integral on the boundary to vanish when applying the divergence theorem). Specifically: $\frac{\partial u}{\partial n} = 0$ on $\partial\Omega$, and the initial condition: $u(t=0) = u_0$.

Under the conditions stated on u, for all test functions $v(x,t) = \psi(t)\phi(x)$, where $\psi \in L^2([0,T])$ and $\phi \in H^1(\Omega)$, the weak formulation of the Allen-Cahn equation is:

$$\int_0^T \langle \partial_t u, v\rangle_{H^{-1},H^1}\, dt + \int_0^T \int_\Omega \nabla u \cdot \nabla v\, dx\, dt + \int_0^T \int_\Omega \frac{1}{\varepsilon^2} f(u) v\, dx\, dt = 0.$$

Now, since $\partial_t u \in L^2(\Omega)$, the duality pairing simplifies to the standard L^2 inner product:

$$\langle \partial_t u, v\rangle_{H^{-1},H^1} = \langle \partial_t u, v\rangle_0 = \int_\Omega \partial_t u\, v\, dx.$$

And therefore, factoring out $\psi(t)$ and allowing all spatial terms to be grouped under a single temporal integral:

$$\int_0^T \psi(t) \left[\langle \partial_t u, \phi\rangle_0 + \langle \nabla u, \nabla\phi\rangle_0 + \left\langle \frac{1}{\varepsilon^2} f(u), \phi \right\rangle_0 \right] dt = 0.$$

By the arbitrariety of $\psi(t)$, we have:

$$\langle \partial_t u, \phi\rangle_0 + \langle \nabla u, \nabla\phi\rangle_0 = -\left\langle \frac{1}{\varepsilon^2} f(u), \phi \right\rangle_0, \quad \forall t \in [0,T].$$

Moreover, from the hypotheses, for all $\phi \in H^1(\Omega)$, we have:

$$\langle \partial_t \tilde{u}, \phi\rangle_0 + \langle \nabla \tilde{u}, \nabla\phi\rangle_0 = -\left\langle \frac{1}{\varepsilon^2} f(\tilde{u}), \phi \right\rangle_0, \quad \forall t \in [0,T].$$

and subtracting side by side, we obtain:

$$\langle \partial_t (u-\tilde{u}), \phi\rangle_0 + \langle \nabla (u-\tilde{u}), \nabla\phi\rangle_0 = -\left\langle \frac{1}{\varepsilon^2} \big(f(u) - f(\tilde{u})\big), \phi \right\rangle_0, \quad \forall \phi \in H^1(\Omega),\, \forall t \in [0,T].$$

a. Nonlinear term

We analyze the term $f(u) - f(\tilde{u})$. By the mean value theorem, we can write: $f(u) - f(\tilde{u}) = f'(z)(u-\tilde{u})$, where z lies on the segment joining u and $\tilde{u}$, i.e., $z = \theta u + (1-\theta)\tilde{u}$ for some $\theta \in [0,1]$.

The derivative of f is given by: $f'(u) = 3u^2 - 1$. Since $|u| \le 1$ and $|\tilde{u}| \le 1$, $|z| \le 1$.

For $z \in [-1,1]$, we compute the bound on $f'(z)$:

$$f'(z) = 3z^2 - 1, \quad \text{so} \quad \max_{z\in[-1,1]} |f'(z)| = 2 = c_f$$

Using this bound, we estimate:

$$f(u) - f(\tilde{u}) = f'(z)(u-\tilde{u}) \quad \Longrightarrow \quad |f(u) - f(\tilde{u})| \le c_f |u - \tilde{u}|.$$

Now, substituting this into the weak formulation and considering the test function ϕ, we have:

$$\left\langle \frac{1}{\varepsilon^2} \big(f(u) - f(\tilde{u})\big), \phi \right\rangle \le \frac{c_f}{\varepsilon^2} \langle u - \tilde{u}, \phi\rangle_0.$$

This result holds for any $f \in H^1([0,T]; W^{1,\infty}_{\text{loc}}(\Omega))$. Indeed, the assumption in the hp. guarantees that $f'(u) \in L^\infty_{\text{loc}}$. Since $u, \tilde{u} \in [-1,1]$, z is restricted to the finite interval $[-1,1]$. Consequently, the boundedness of $f'(z)$ in $[-1,1]$ is guaranteed, allowing the argument to extend to all functions satisfying the stated regularity assumptions.

Therefore with $w = u - \tilde{u}$:

$$\langle \partial_t w, \phi\rangle_0 + \langle \nabla w, \nabla\phi\rangle_0 \le \frac{c_f}{\varepsilon^2} \langle w, \phi\rangle_0, \quad \forall \phi \in H^1(\Omega),\, \forall t \in [0,T].$$

Since u and $\tilde{u}$ are both in $H^1(\Omega)$, we can choose w as the test function ϕ. Substituting w into the inequality, we obtain:

$$\langle \partial_t w, w\rangle_0 + \|\nabla w\|^2_{L^2(\Omega)} \le \frac{c_f}{\varepsilon^2} \|w\|^2_{L^2(\Omega)}, \quad \forall t \in [0,T].$$

Now, the first term becomes (all hypothesis to exchange integral and derivative are met):

$$\langle \partial_t w, w\rangle_0 = \int_\Omega \partial_t w\, w\, dx = \int_\Omega \frac{1}{2} \partial_t (w^2)\, dx = \frac{1}{2} \frac{d}{dt} \|w\|^2_{L^2(\Omega)}.$$

and we rewrite the equation:

$$\frac{1}{2} \frac{d}{dt} \|w\|^2_{L^2(\Omega)} + \|\nabla w\|^2_{L^2(\Omega)} \le \frac{c_f}{\varepsilon^2} \|w\|^2_{L^2(\Omega)}, \quad \forall t \in [0,T].$$

b. Gronwall's inequality

Now starting from the inequality (without the term with the gradient, which is positive):

$$\frac{1}{2}\frac{d}{dt}\|w(t)\|^2_{L^2(\Omega)} \leq \frac{c_f}{\varepsilon^2}\|w(t)\|^2_{L^2(\Omega)},$$

we integrate over $[0, s]$ to obtain:

$$\int_0^s \frac{d}{dt}\|w(t)\|^2_{L^2(\Omega)}\,dt \leq \frac{2c_f}{\varepsilon^2}\int_0^s \|w(t)\|^2_{L^2(\Omega)}\,dt.$$

so the left-hand side simplifies as:

$$\|w(s)\|^2_{L^2(\Omega)} - \|w(0)\|^2_{L^2(\Omega)} \leq \frac{2c_f}{\varepsilon^2}\int_0^s \|w(t)\|^2_{L^2(\Omega)}\,dt.$$

$$\|w(s)\|^2_{L^2(\Omega)} \leq \|w(0)\|^2_{L^2(\Omega)} + \frac{2c_f}{\varepsilon^2}\int_0^s \|w(t)\|^2_{L^2(\Omega)}\,dt.$$

Using Gronwall's inequality, this implies:

$$\|w(s)\|^2_{L^2(\Omega)} \leq \|w(0)\|^2_{L^2(\Omega)} \exp\left(\frac{2c_f}{\varepsilon^2}s\right).$$

Finally, taking the supremum over $t \in [0, T]$, we have:

$$\sup_{t\in[0,T]} \|w(t)\|^2_{L^2(\Omega)} \leq \|w(0)\|^2_{L^2(\Omega)} \exp\left(\frac{2c_f}{\varepsilon^2}T\right).$$

c. Dealing with the gradient term

Starting from the inequality:

$$\frac{1}{2}\frac{d}{dt}\|w(t)\|^2_{L^2(\Omega)} + \|\nabla w(t)\|^2_{L^2(\Omega)} \leq \frac{c_f}{\varepsilon^2}\|w(t)\|^2_{L^2(\Omega)}, \quad \forall t \in [0, T],$$

we rewrite it as:

$$\|\nabla w(t)\|^2_{L^2(\Omega)} \leq \frac{1}{2}\frac{d}{dt}\|w(t)\|^2_{L^2(\Omega)} + \frac{c_f}{\varepsilon^2}\|w(t)\|^2_{L^2(\Omega)}.$$

then, we integrate over $[0, T]$:

$$\int_0^T \|\nabla w(t)\|^2_{L^2(\Omega)}\,dt \leq \frac{1}{2}\|w(T)\|^2_{L^2(\Omega)} - \frac{1}{2}\|w(0)\|^2_{L^2(\Omega)} + \int_0^T \frac{c_f}{\varepsilon^2}\|w(t)\|^2_{L^2(\Omega)}\,dt.$$

Using the bound:

$$\|w(t)\|^2_{L^2(\Omega)} \leq \|w(0)\|^2_{L^2(\Omega)} \exp\left(\frac{2c_f}{\varepsilon^2}t\right),$$

we substitute into the inequality:

$$\int_0^T \|\nabla w(t)\|^2_{L^2(\Omega)}\,dt \leq \frac{1}{2}\|w(0)\|^2_{L^2(\Omega)} \exp\left(\frac{2c_f}{\varepsilon^2}T\right) - \frac{1}{2}\|w(0)\|^2_{L^2(\Omega)} + \frac{c_f}{\varepsilon^2}\int_0^T \|w(0)\|^2_{L^2(\Omega)} \exp\left(\frac{2c_f}{\varepsilon^2}t\right)\,dt.$$

Computing the integral, we have:

$$\int_0^T \|\nabla w(t)\|^2_{L^2(\Omega)}\,dt \leq \frac{1}{2}\|w(0)\|^2_{L^2(\Omega)} \exp\left(\frac{2c_f}{\varepsilon^2}T\right) + \frac{c_f}{\varepsilon^2}\frac{\varepsilon^2}{2c_f}\|w(0)\|^2_{L^2(\Omega)}\left[\exp\left(\frac{2c_f}{\varepsilon^2}T\right) - 1\right].$$

Thus, simplifying further:

$$\int_0^T \|\nabla w(t)\|^2_{L^2(\Omega)}\,dt \leq \|w(0)\|^2_{L^2(\Omega)} \exp\left(\frac{2c_f}{\varepsilon^2}T\right).$$

d. Conclusion: Combining the Results

Now, combining the results:

$$\sup_{t\in[0,T]} \|w(t)\|^2_{L^2(\Omega)} + \int_0^T \|\nabla w(t)\|^2_{L^2(\Omega)}\,dt \leq 2\|w(0)\|^2_{L^2(\Omega)} \exp\left(\frac{2c_f}{\varepsilon^2}T\right).$$

which is:

$$\sup_{t\in[0,T]} \|u(t) - \tilde{u}(t)\|^2_{L^2(\Omega)} + \int_0^T \|\nabla(u(t) - \tilde{u}(t))\|^2_{L^2(\Omega)}\,dt \leq 2\|u_0 - \tilde{u}_0\|^2_{L^2(\Omega)} \exp\left(\frac{2c_f}{\varepsilon^2}T\right).$$