Project 3: Foundation Models for Phase-Field Dynamics

Wu, You
Department of Mathematics
youwuyou@ethz.ch

Problem Setup and Implementation Approach

The Allen-Cahn equation describes the order-disorder transition in a many-particle system [1]. It is a nonlinear parabolic equation that models the physics of reaction-diffusion phenomena:

$$\frac{\partial u}{\partial t} = \Delta_x u - \frac{1}{\varepsilon^2} f(u) \tag{1}$$

where the parameter ε controls the width of transition layers between phases. The right-hand side function $f(u)$ is the derivative of a non-negative function $F \in C^1(\mathbb{R})$ with two minima $F(\pm 1) = 0$, meaning that the solution of the system u will evolve and converge to two stable equilibria $u \in \{-1, 1\}$ at its energy equilibrium. Here, we set $f(u) := u^3 - u$ and seek for the solution of (1) defined on the spatio-temporal domain $\tilde{\Omega} := \Omega \times [0, T]$, where $T > 0$ is some final observation time.

Neural Foundation Model

Our foundation model `AllenCahnFNO` is built upon two important components. We used Fourier Neural Operator (FNO) [2] as our model backbone, and FiLM [3] layers are incorporated after each FNO block to ensure we have trainable general conditioning layers that can be used to **embed both the time-dependency, as well as the ε parameter** into our model. **Our model is capable of handling different spatial and temporal resolutions.** In a forward pass through our model, we pass in the initial condition u_0, the associated ε value, as well as the temporal discretization t. We automatically detect the number of spatial and temporal grid points from the shape of u_0 and t respectively. Previous work such as Subramanian et al. [4] has successfully shown the potentials of FNO to learn underlying physics for a wide range of downstream tasks. This further motivates our choice and has been proven successful in our experiments. We used the following parameters for our model:

Number of modes	Depth per Layer Type	Width	Activation Function
20	2	64	GELU

Table 1: Model Architecture Parameter for AllenCahnFNO.

Data Generation

The Allen-Cahn equation (1) exhibits different behaviors depending on the parameter ε. In order to capture full range of dynamics, we experimented with a wide range of ε parameters, and finally selected ε to be 0.1, 0.05, and 0.01 in our training dataset. At *large* ε values like 0.1, the system shows **smooth, diffusion-dominated** behavior. Conversely, at *small* values when $\varepsilon \to 0$ like 0.01, **rapid phase separation occurs with sharp interfaces**, **nonlinear behaviors dominates over diffusive behavior**. The 0.05 is selected is an appropriate transition value showing intermediate dynamics. Notably, when ε is very small, the system becomes stiff. Therefore, to cope with potential issues with stiffness, we employed `scipy.integrate.solve_ivp` with the `Radau` integrator for the implementation of the numerical solver, which can be seen under `data_generator.py`.

For generation of each trajectory across time, we fixed the spatial domain on the interval [−1.0, 1.0] with 128 grid points, as in the provided template code. To comply with the selected ε values, we used 5 uniformly spaced time points from 0.0 to 0.01, with a fixed timestep of $\Delta t = 0.0025$ for the temporal evolution. We emphasize that ε and Δt need to be selected in tandem. If we aim to capture the transient behavior of the system, but choose a

timestep that is too large, we will not be able to catch enough dynamics after the initial snapshot. We used three different types of initial conditions $u_0 = u(x, 0)$ and implemented the corresponding `FunctionSampler` class: 1. Piecewise Linear (PL); 2. Gaussian Mixture (GM); 3. Fourier Series (FS). We refer the readers to the appendix section to see more details about each function class. **For the training dataset, we generated 1000 trajectories across spatio-temporal domain.**

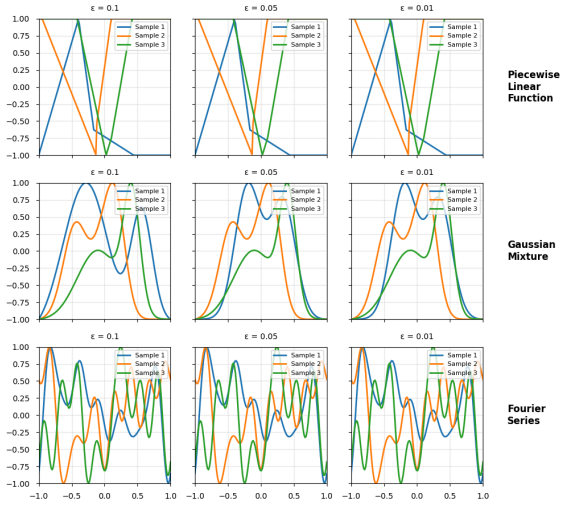

Figure 1: Selected three training trajectories at initial time $t = 0$. Periodic boundary condition is enforced and nodal values $u_0 \in [-1, 1]$ holds.

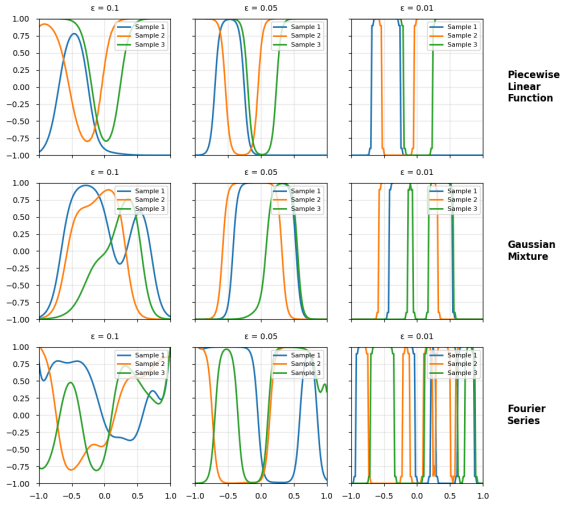

Figure 2: Selected three training trajectories at final observation time $T = 0.01$ after advancing 4 time steps of size $\Delta t = 0.0025$

Generating Testing Data & OOD Data & ε-Test Data

We generated the standard testing set (`test_sol.npy`) in the same way we generated the training data, making it also in-distribution. Furthermore, we aim to evaluated our model's robustness through two more test sets. **The out-of-distribution (OOD) test set** (`test_sol_OOD.npy`) challenges the model with more complex patterns than seen during training: additional breakpoints in piecewise linear functions, more components in Gaussian mixtures, and higher-frequency terms in Fourier series. **To test if our model is capable of extra- interpolating ε parameters**, we generated another test set (`test_sol_eps.npy`) using the standard initial conditions but with epsilon values ranging from 10.0 to 0.006, extending well beyond our training range of 0.1 to 0.01. In the following figure, one can see the generated initial data at $t = 0$. **For each testing dataset, we generated 200 trajectories across spatio-temporal domain.**

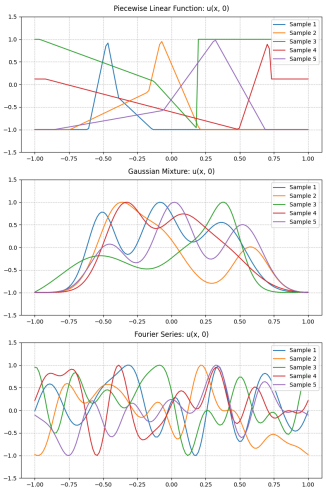

Figure 3: Testing Data (`test_sol.npy`)

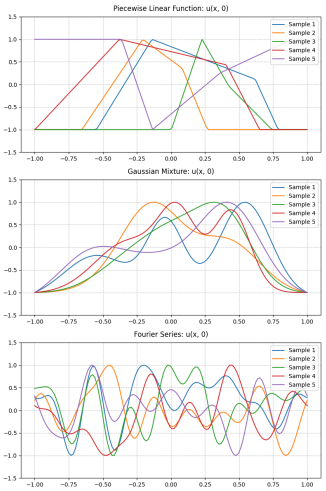

Figure 4: ε Testing Data (`test_sol_eps.npy`)

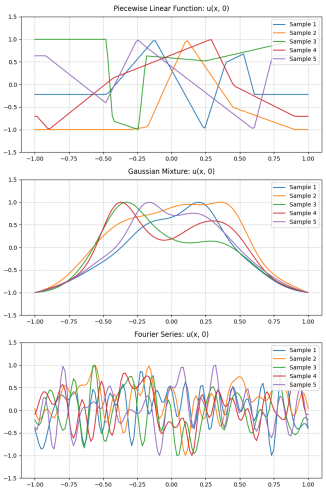

Figure 5: OOD Testing Data (`test_sol_OOD.npy`)

Training & Fine-Tuning

For training, we **only** trained **one base model** on the comprehensive `train_sol.npy` dataset across various initial conditions and ε values. Recall that we have generated 1000 trajectory across spatio-temporal domain for each data category by setting `n_train = 1000`, we have also three ε values where $\varepsilon \in \{0.1, 0.05, 0.01\}$ as well as three function classes where the initial condition u_0 belongs to. This gives us 9000 samples in all for the `train_sol.npy` dataset.[1] For more details about each function class, see **Appendix A**. We used all training samples at once, without subdividing them based on the initial condition used or the magnitudes of ε values. This is because *Curriculum Learning* [5] easily introduce biases for our `AllenCahnFNO` model, which we further elaborated in **Appendix B**.

Training Samples	Validation Samples	Maximal Number of Epochs	Early stopping
8100	900	800	After 50 epochs
Curriculum Learning	**Optimizer**	**Batch Size**	**Learning Rate**
False	AdamW	512	0.0001

Table 2: Training configuration for AllenCahnFNO.

To examine if our `AllenCahnFNO` can indeed serve as a foundation model, we **fine-tune** the base model by passing it a couple of few-shot samples before inference. The fine-tuning is very similar as the training process, but with the difference that we now **freeze all other layers, but leave the FiLM layers unfrozen**. This way, we could perform very lightweight fine-tuning in which **only 0.3%** of the parameters will be trained. We set the fine-tuning epochs to 100 and used only 20 fine-tune samples with a 16/4 split for training/validation.

Fine-Tune Samples	Total Model Parameters	Trainable parameters	Frozen parameters
20	387,009	**1,024 (0.3%)**	385,985 (99.7%)
Curriculum Learning	**Optimizer**	**Batch Size**	**Learning Rate**
False	AdamW	2	0.01

Table 3: Fine-tuning configuration for AllenCahnFNO.

Results

We report the average relative L2 error across 200 testing trajectories per testing category on datasets `test_sol.npy` and `test_sol_OOD.npy` in Table 4 & 5. The error is computed by comparing prediction at a time snapshot against ground truth for all trajectories. The predictions are obtained via evaluating our base model and fine-tuned model on the **original ε values**. The OOD dataset tests model's capabilities to resolve high-frequencies.

IC Type	ε	Zero-shot	Fine-tuned
PL	0.10	10.1411%	10.0359%
	0.05	8.1278%	8.0712%
	0.01	5.0139%	5.2801%
FS	0.10	26.8162%	26.7704%
	0.05	27.4113%	27.3574%
	0.01	16.3927%	16.0335%
GM	0.10	3.9864%	3.7835%
	0.05	6.4224%	6.3270%
	0.01	2.8208%	2.7567%

Table 4: Average relative L2 error between zero-shot and fine-tuned models on in-distribution test set

IC Type	ε	Zero-shot	Fine-tuned
PL	0.10	16.5216%	16.3927%
	0.05	13.1615%	13.4098%
	0.01	3.6672%	4.2901%
FS	0.10	87.3301%	83.1482%
	0.05	47.4966%	45.9286%
	0.01	28.0266%	27.0400%
GM	0.10	3.7367%	4.7908%
	0.05	4.9008%	5.1821%
	0.01	2.0659%	2.5415%

Table 5: Average relative L2 error between zero-shot and fine-tuned models on OOD test set

[1] Note that each sample can be further broken down into 5 time snapshots, but in our implementation, the model predicts 4 further time snapshots at once, thus we arrive at a train/validation split of 8100/900 samples.

In the tables above, we colored the fine-tuned results with green if it shows lower error percentage comparing to the zero-shot inference obtained directly via evaluating on the base model, and red if it becomes worse. We observe that, **our model is capable to resolve phase-field dynamics across different ε values**. Moreover, our model is also capable to capture nuances in different initial conditions. **We obtained less than 10% relative L2 errors on most subsets of PL and GM function classes.** For FS we obtained notably worse results, this may due to the fact that choosing `n_modes = 10` allows too many high-frequency features in our dataset, which is not very comparible with other two initial condition types that only mildly have around 4 breakpoints or components. **GM is remarkably the easiest category out of all function classes**, demonstrating errors $< 5\%$.

Furthermore, we highlight that our model is capable of transfer learning. For in-distribution dataset, all fine-tuned results, besides the $\varepsilon = 0.01$ for PL function class, improved after fine-tuning on only 20 samples. However, the OOD dataset worsens its results in both PL and GM datasets after fine-tuning, leaving only the FS category with some improvements. This again reflects our previous observation that **learning the observations from FS dataset is inherently harder due to its high-frequency features**. We increased the Fourier mode for OOD dataset from 10 to 20, this further leads to the inbalance in dataset difficulty. Interestingly, increasing the number of Gaussian components leads to better performance on the OOD dataset. This is due to the fact that the superposition of more Gaussian components give a more homogeneous profile of the initial condition.

Generalization Across Different ε Values

Next, we report results on the ε-test dataset (`test_sol_eps.npy`). This dataset consists of 6 unseen ε values for each IC type. We can subdivide the tested ε values into three categories:

IC Type	ε	Zero-shot	Fine-tuned
PL	10.000	9409.0869%	64.7127%
	0.500	233.5755%	26.3383%
	0.075	9.6927%	15.7740%
	0.025	5.7023%	7.0492%
	0.008	3.4510%	5.6850%
	0.006	3.8525%	5.7232%
FS	10.000	30882.6602%	61.9866%
	0.500	1394.6853%	53.6279%
	0.075	34.1827%	54.2362%
	0.025	25.8429%	31.3667%
	0.008	32.3054%	30.9287%
	0.006	30.1576%	28.5304%
GM	10.000	12466.9521%	14.3580%
	0.500	310.5461%	15.3666%
	0.075	6.4313%	13.3130%
	0.025	5.9716%	7.6806%
	0.008	2.9431%	6.3092%
	0.006	3.3061%	6.2021%

Table 6: Average relative L2 error between zero-shot and fine-tuned models on ε-test set showing extra- and interpolation capabilities.

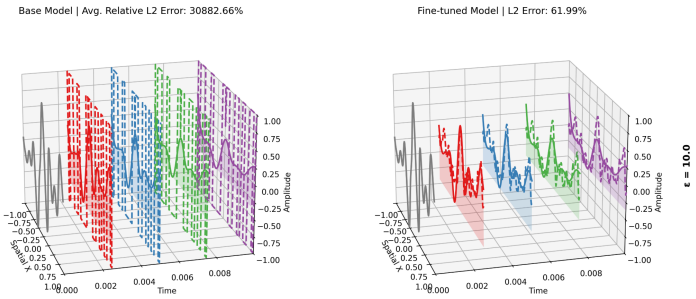

Figure 6: ε-test with Fourier Series (FS).

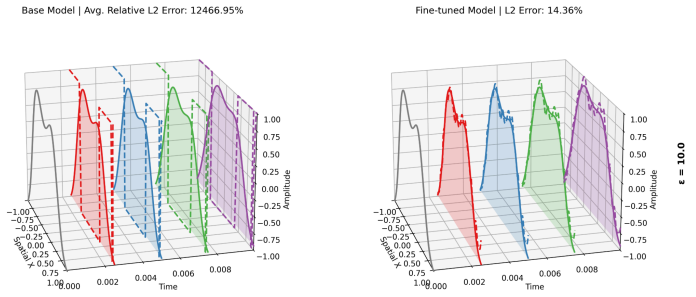

Figure 7: ε-test with Gaussian Mixture (GM).

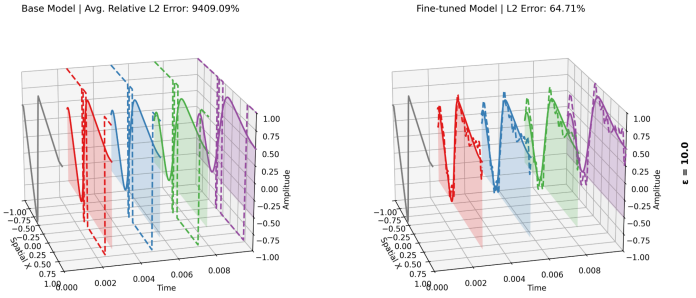

Figure 8: ε-test with Piecewise Linear (PL).

The first category consists of ε values in $\{0.075, 0.025\}$, it tests the interpolation capability of our model. Next is $10.0, 0.5$, which tests the extrapolation to higher values. At last, $0.008, 0.006$ tests the extrapolation to lower values. As we can observe in Table 6, **our base model is capable of both interpolating, as well as extrapolating to lower values.** In both mentioned ε subsets, we got similar results as in the previous two datasets. **However, extrapolating ε to higher values poses a significant challenge for our models.** Our base model fatally failed to predict at higher unseen values $\varepsilon \in \{10, 0.5\}$.

Success of Lightweight Fine-Tuning

While our base model failed to predict higher ε values, the fine-tuned model gives remarkable results by learning from only 20 samples. For all IC classes, we significantly improve the quality of prediction. The GM class performs the best in which it improved from 12466.9521% and 310.5461% of average relative L2 error to 14.3580% ($\varepsilon = 10.0$) and 15.3666% ($\varepsilon = 0.5$). Similar trends can be seen for PL and FS classes too. **We highlight the fact that this success is obtained by activating only 1,024 parameters in fine-tuning, which is equivalent of 0.3% of the total parameters $387,009$ of our `AllenCahnFNO` model.**

Despite the success, we also point out that fine-tuning does harm performance on other categories. For interpolated value such as $\varepsilon = 0.025$, our fine-tuned model downgraded its performance in which it consistently gives larger errors such as $5.7023\% \rightarrow 7.0492\%$ for PL class, $25.8429\% \rightarrow 31.3667\%$ for FS class and $5.9716\% \rightarrow 7.6806\%$ for GM class.

For completeness, in Figure 6 - 8 we also reported sample trajectory at index 5 from each IC function class, the initial condition is reported in grey, and the predicted results are reported in dashed colored lines, and ground truth are marked by color-filled bold lines. The title reports the average relative L2 error in percentage. One can again see clearly how just fine-tuning FiLM layers is already powerful enough to give us reasonable predictions across time for all IC types.

Observation

To summarize, we developed a lightweight yet effective model with 387,009 parameters for resolving a wide range of physics governed by Allen-Cahn equation (1). The most remarkable results from our model is that by fine-tuning only the FiLM layers that consist of 0.3% of model parameters, we were capable to reduce errors to reasonable range for all initial conditions. Moreover, zero-shot results from our base model are already quite good for most ε values and IC types.

However, our model is currently still limited in several aspects:

1. Fine-tuning on in-distribution and OOD datasets is not as effective as fine-tuning for the ε test set.
2. We have not evaluated our model for other spatial grid sizes, presumably the inherent weakness of FNO will bring significant performance degrades for coarser grid sizes as it is not ReNO [6].
3. Our approach highly rely on the quality of data for both training and fine-tuning.
4. We did not fully make use of all time series generated, we only leveraged $O(n)$ sample pairs per trajectory

For future development, using more advanced architectural component such as scalable Operator Transformer (scOT) as backbone and training with All2All technique [7] could help us to overcome these challenges.

Appendix A: Function Space Specifications

For the dataset generation, we ensured the periodic boundary condition is always enforced and the generated data span over x-axis from −1 to 1 and the nodal values $u_0 \in [-1, 1]$ as well. In all, we used the following three function classes for data generation. We distinguish between default samplers (used for `train_sol.npy`, `test_sol.npy` and `test_sol_eps.npy`) and special samplers (used for `test_sol_OOD.npy`).

A.1 Piecewise Linear Functions (PL)

Let $[a, b]$ be a closed interval and $\{x_0, x_1, ..., x_k\}$ be a partition where $a = x_0 < x_1 < ... < x_k = b$. A piecewise linear function $f : [a, b] \to \mathbb{R}$ is defined as:

$$f(x) = \begin{cases} \alpha_1 x + \beta_1 x \in [x_0, x_1] \\ \vdots \\ \alpha_k x + \beta_k x \in [x_{k-1}, x_k] \end{cases} \tag{2}$$

where $\alpha_i, \beta_i \in \mathbb{R}$ and continuity is enforced at breakpoints: $\alpha_i x_i + \beta_i = \alpha_{i+1} x_i + \beta_{i+1}$ for $i = 1, ..., k-1$.

For **default samplers**, we set number of breakpoints to 4 and for the **OOD dataset** `test_sol_OOD.npy` we increased it to 6. It is worth to note that some initial data from the PL function class shows spurious oscillations during the solution process. This indicates that one needs to extra care when selecting proper training data for piecewise linear functions in the future to avoid learning solutions with inherent approximation errors in themselves.

A.2 Fourier Series (FS)

The Fourier series for an L-periodic function on $[0, L]$ is similarly given by:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos\left(\frac{2\pi k x}{L}\right) + b_k \sin\left(\frac{2\pi k x}{L}\right) \right), \tag{3}$$

with coefficients a_k and b_k given by $a_k = \left(\frac{2}{L}\right) \int_0^L f(x) \cos\left(\frac{2\pi k x}{L}\right) dx$ and $b_k = \left(\frac{2}{L}\right) \int_0^L f(x) \sin\left(\frac{2\pi k x}{L}\right) dx$.

Instead of taking an infinite summation, we used only 10 Fourier modes for **default samplers**, and for the **OOD dataset** `test_sol_OOD.npy` we increased the Fourier modes used to 20 in order to explore our model performance on high-frequency data.

A.3 Gaussian Mixture Model (GM)

For the GM class, we obtain the initial condition through the weighted sum of Gaussian components such that $\sum_{i=1}^{K} w_i \overset{!}{=} 1$:

$$u(x, 0) = \sum_{j=1}^{K} w_j \overbrace{\mathcal{N}(\boldsymbol{x}; \mu_j, \sigma_j)}^{\text{probability of j-th Gaussian}} \tag{4}$$

The number of Gaussian components K is determined randomly between 2 - 5 for **default sampler**. For the **OOD dataset** `test_sol_OOD.npy`, K is fixed to 10. We separated the interval depending on the number of components are used in the sample and then uniformly randomly draw mean μ and σ within the range of the subinterval.

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \tag{5}$$

Appendix B: Pitfalls of Curriculum Learning

We explored different training strategies for our Allen-Cahn model, with a particular focus on curriculum learning [5]. For curriculum trainings, we divided the first half of the training period into equal segments based on our ε values, introducing them sequentially from largest to smallest. On the contrary, for anti-curriculum training, we introduce new ε values by decreasing difficulty.

The **default training method**, which exposes the model to all ε values at once from the beginning, **obviously outperforms both curriculum alternatives** in our experimental results. The default method exhibits **smooth convergence** and a lower final validation loss as shown in Figure 9. Performance instability is evident in the *curriculum learning technique* as seen in Figure 10, with **noticeable jumps in loss whenever new ε values are introduced**. Similarly, bad performance was shown by the *anti-curriculum technique* in Figure 11, which started with the most difficult small ε values and showed extremely unstable training dynamics.

This comparison of validation loss of different training strategies indicates that the complexity of curriculum learning does not significantly improve the Allen-Cahn system over ordinary training techniques. **Default training strategy exhibits best convergence** in minimizing validation errors.

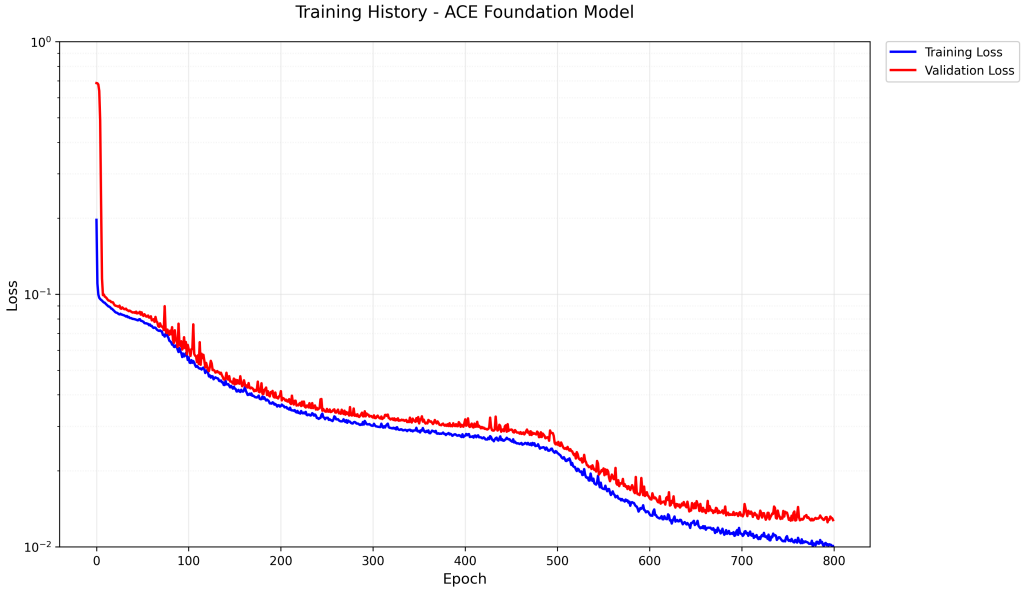

Figure 9: Default training strategy

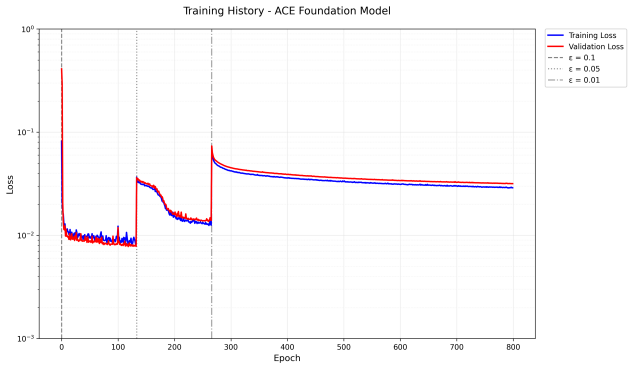

Figure 10: Training with curriculum learning (easiest samples first)

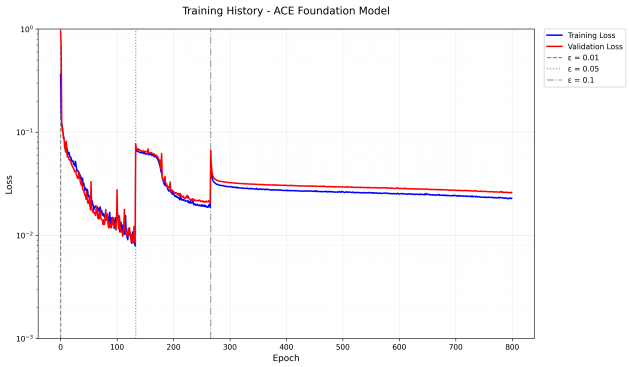

Figure 11: Training with anti-curriculum (hardest samples first)

Appendix C.1: In-Distribution Test Results (`test_sol.npy`)

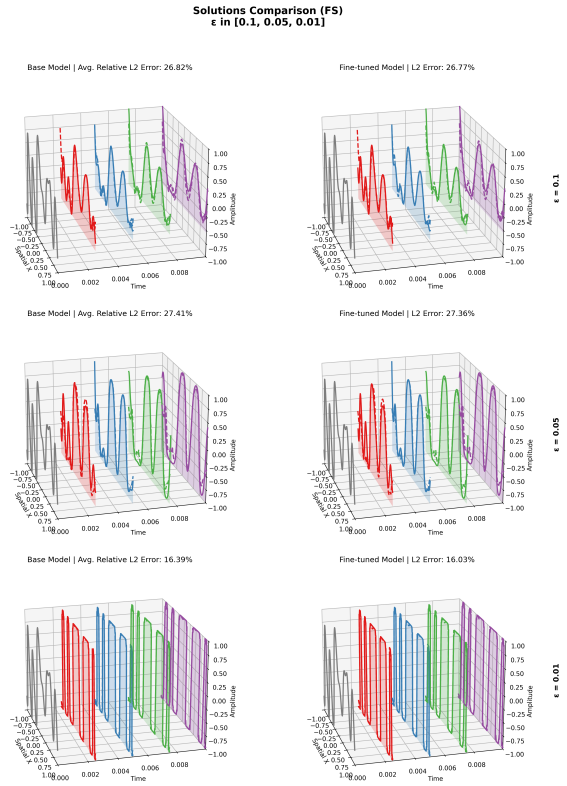

Figure 12: Fourier Series (FS)

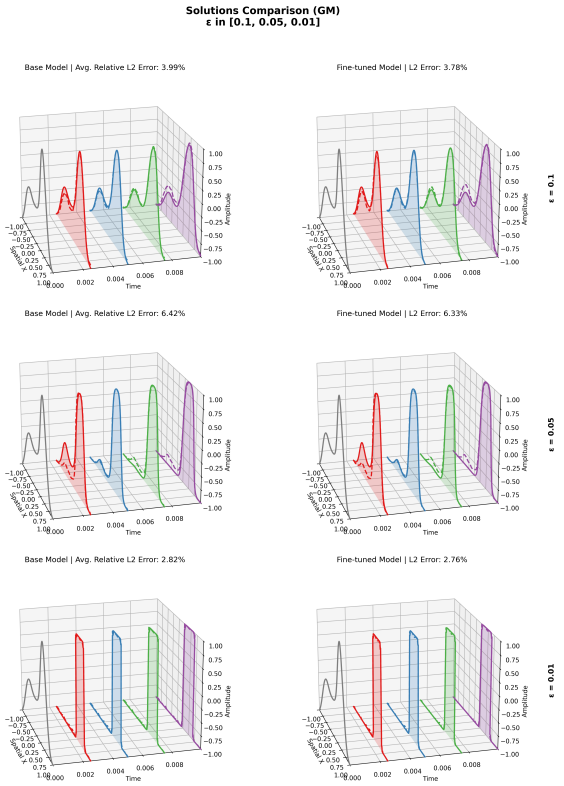

Figure 13: Gaussian Mixture (GM)

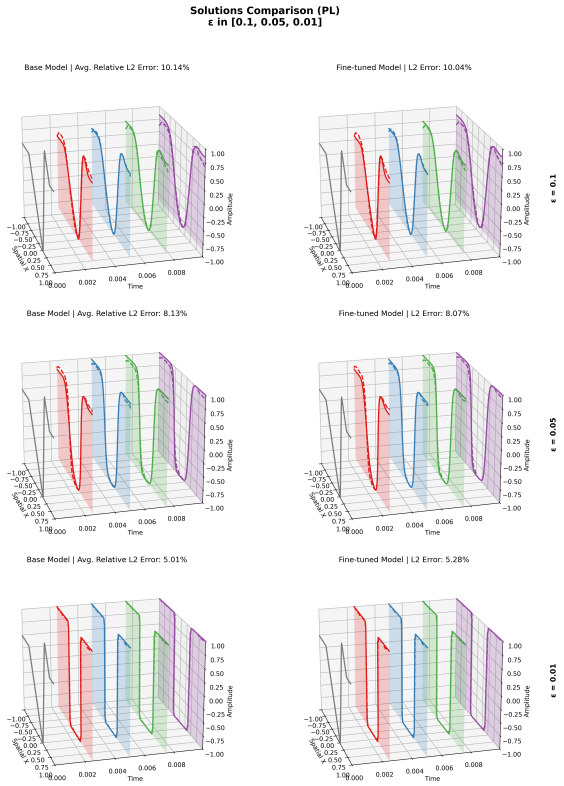

Figure 14: Piecewise Linear (PL)

Appendix C.2: Out-of-Distribution Test Results (`test_sol_OOD.npy`)

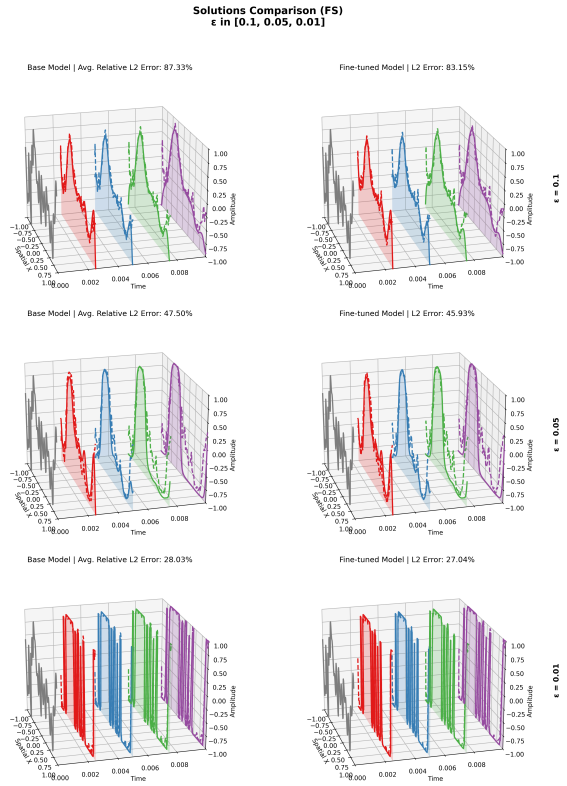

Figure 15: OOD Fourier Series (FS)

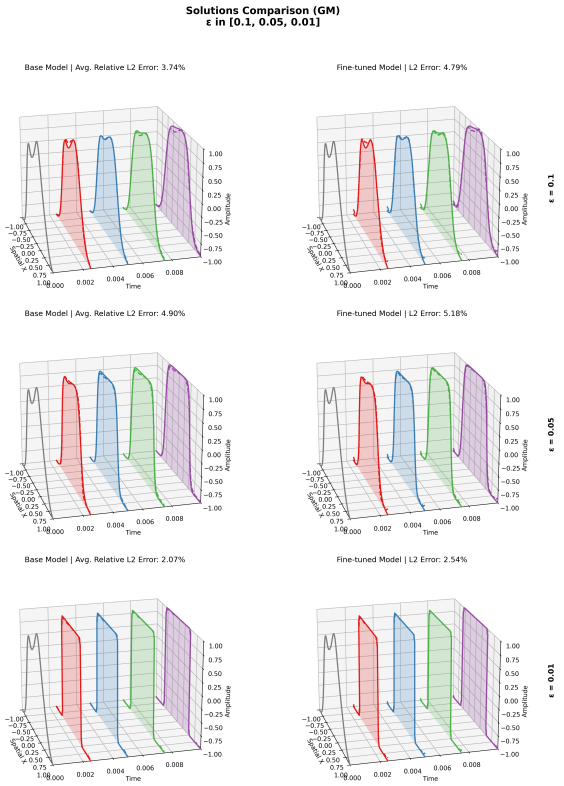

Figure 16: OOD Gaussian Mixture (GM)

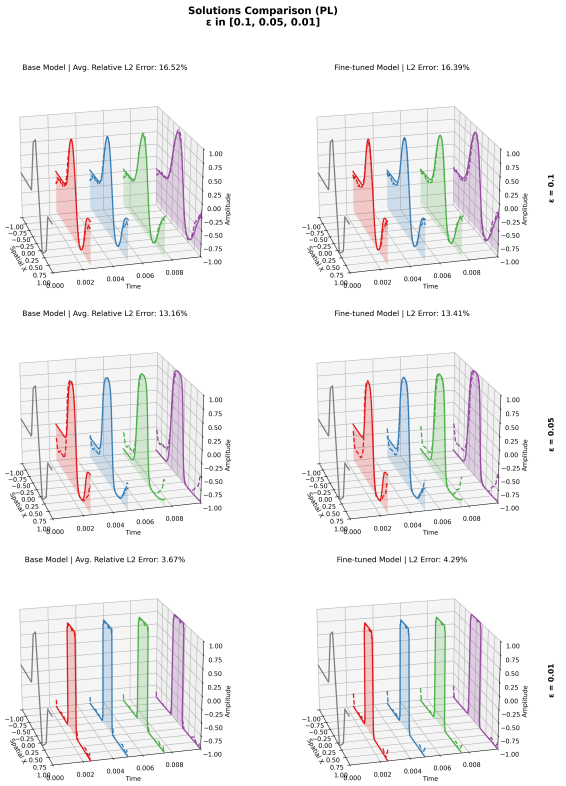

Figure 17: OOD Piecewise Linear (PL)

Appendix C.3: Epsilon Variation Test Results (`test_sol_eps.npy`)

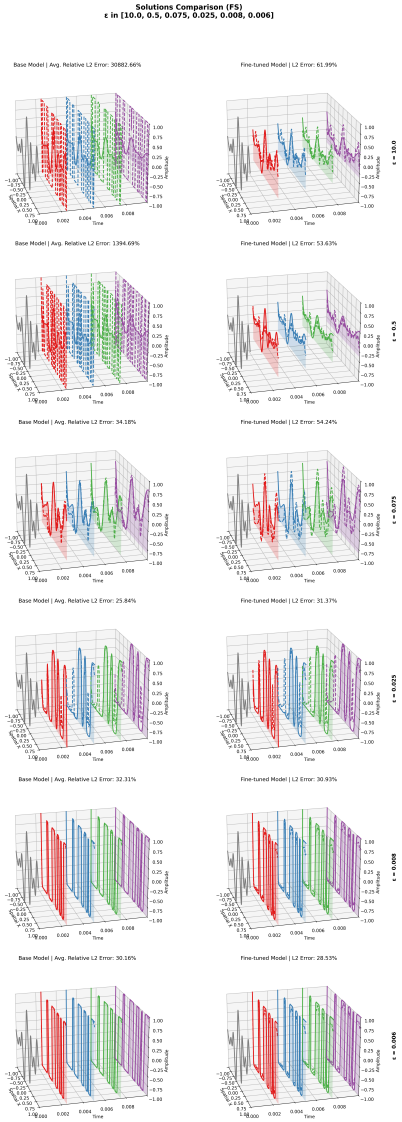

Figure 18: ε-Test Fourier Series (FS)

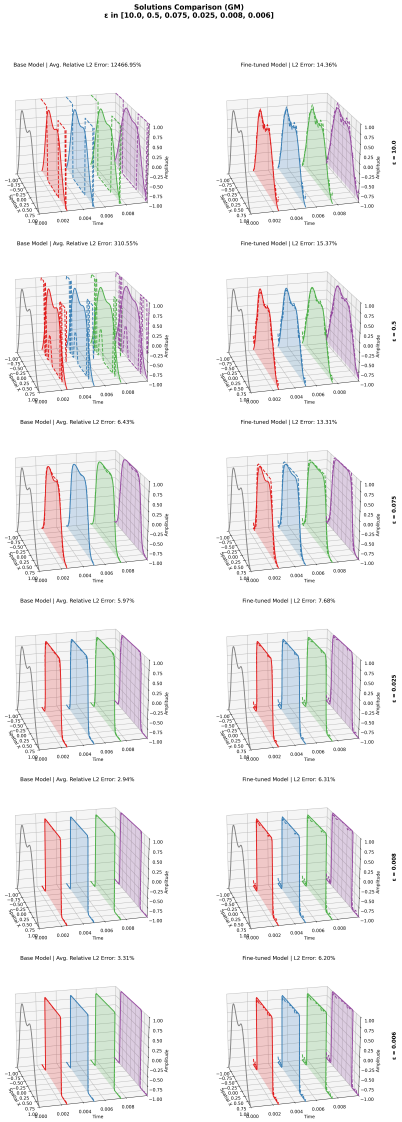

Figure 19: ε-Test Gaussian Mixture (GM)

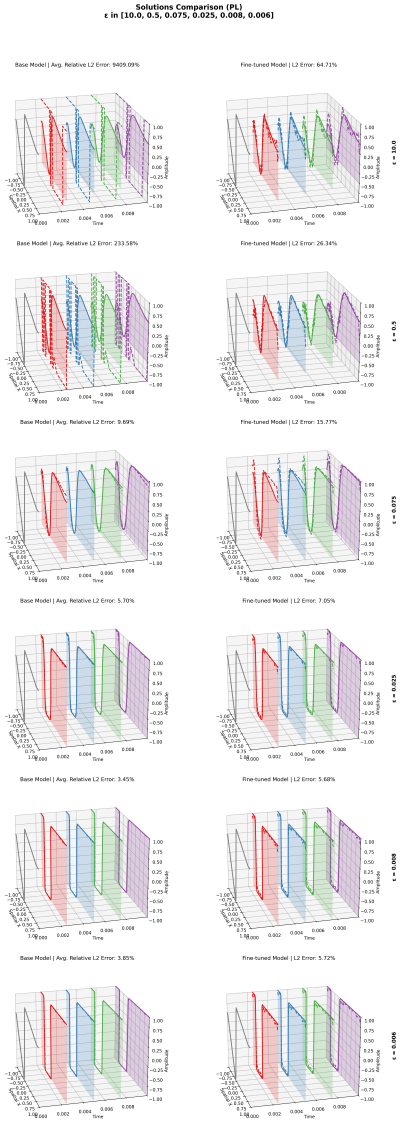

Figure 20: ε-Test Piecewise Linear (PL)

References

[1] S. M. Allen and J. W. Cahn, "A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening," *Acta Metallurgica*, vol. 27, no. 6, pp. 1085–1095, 1979, doi: 10.1016/0001-6160(79)90196-2.

[2] Z. Li *et al.*, "Fourier Neural Operator for Parametric Partial Differential Equations," *ICLR 2021 - 9th International Conference on Learning Representations*, 2020, [Online]. Available: https://arxiv.org/abs/2010.08895v3

[3] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "FiLM: Visual Reasoning with a General Conditioning Layer," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3942–3951, 2017, doi: 10.1609/aaai.v32i1.11671.

[4] S. Subramanian *et al.*, "Towards Foundation Models for Scientific Machine Learning: Characterizing Scaling and Transfer Behavior," *Advances in Neural Information Processing Systems*, vol. 36, 2023, [Online]. Available: https://arxiv.org/abs/2306.00258v1

[5] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," *ACM International Conference Proceeding Series*, vol. 382, 2009, doi: 10.1145/1553374.1553380.

[6] F. Bartolucci, E. de Bézenac, B. Raonić, R. Molinaro, S. Mishra, and R. Alaifari, "Representation Equivalent Neural Operators: a Framework for Alias-free Operator Learning," *Advances in Neural Information Processing Systems*, vol. 36, 2023, [Online]. Available: https://arxiv.org/abs/2305.19913v2

[7] M. Herde *et al.*, "Poseidon: Efficient Foundation Models for PDEs," 2024, [Online]. Available: https://arxiv.org/abs/2405.19101v2