Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten schriftlichen Arbeit. Eine der folgenden drei Optionen ist in Absprache mit der verantwortlichen Betreuungsperson verbindlich auszuwählen:

◉ Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Es wurden keine Technologien der generativen künstlichen Intelligenz[1] verwendet.

☐ Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Als Hilfsmittel wurden Technologien der generativen künstlichen Intelligenz[2] verwendet und gekennzeichnet.

☐ Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Als Hilfsmittel wurden Technologien der generativen künstlichen Intelligenz[3] verwendet. Der Einsatz wurde, in Absprache mit der Betreuungsperson, nicht gekennzeichnet.

Titel der Arbeit:

Masked PDE-Learning

Verfasst von:
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Chen	Shengdi

Ich bestätige mit meiner Unterschrift:
- Ich habe mich an die Regeln des «Zitierleitfadens» gehalten.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu und vollständig dokumentiert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Eigenständigkeit überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Zürich, 14. Juni, 2024	

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie grundsätzlich gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.

[1] z. B. ChatGPT, DALL E 2, Google Bard
[2] z. B. ChatGPT, DALL E 2, Google Bard
[3] z. B. ChatGPT, DALL E 2, Google Bard

MASKED PDE-LEARNING

SHENGDI CHEN[α]

Abstract. PDEs have historically been solved by numerical methods distilled from traditional applied mathematics. With the advent of the field of machine-learning as a whole, toolchains for PDE research now include the operator learning methodology structured around deep neural networks. Pertaining to general machine-learning, on the other hand, much effort is expended to study algorithmic resistance to the intrinsic noisiness of data. Masking, as one established method to this end, intentionally contaminates portions of a clean dataset by setting certain patches of its data channels to some noise value, effectively concealing parts of the underlying dataset from the learning model.

This project studies the effect of masking in the specific context of PDE Learning: the procedure of generating PDE datasets incorporating multi-channel masking is streamlined, allowing the formalization of a full experimentation pipeline for masked PDE learning. A tailored evaluation framework is subsequently established, differentiating in and out-of-distribution regimes by identifying and varying three key configuration vectors of the masking procedure. Peripheral aspects of the masked learning paradigm are also explored, including the deliberate usage of complimentary masking strategies targeting mutually exclusive candidates for training and inference, as well as the deployment of a dynamic termination and scheduling mechanism. Performance analyses are carried out with flexible inspection tooling and plotting utilities to enable experimentation-friendly prototyping.

This project, conducted under the supervision of Prof. Dr. S. Mishra, is executed within the framework of a Master-Thesis under the catalogue identifier `401-4990-01L` at ETH Zürich. Bogdan Raonić is the project advisor and primary correspondent. The project commenced on **04.12.2023**, with a nonnegotiable duration of six (**6**) months.

RW/CSE, D-MATH, ETH Zürich: `shenchen␣AT␣ethz␣DOT␣ch | 17-943-358`.

Contents

Part 1. Prelude 5
1. Quick Start 5
1.1. Overview 5
1.2. Thesis structure 5
1.3. Related research 5
2. Technical Details 6
2.1. Programming 6
2.2. Formatting and Typesetting 6
2.3. Version Tracking 6

Part 2. Background and Setup 7
3. Discretization 7
3.1. Spatial discretization 7
3.2. Temporal discretization 8
3.3. API 8
4. Masking 9
4.1. Introduction 9
4.2. Masking intensity 9
4.3. Masking strategies 10
4.4. Masking value 13
4.5. Masking overview 13
5. The PDE Prototypes 15
5.1. Setup 15
5.2. Poisson Equation 17
5.3. Heat Equation 20
5.4. Wave Equation 22
5.5. Summary 24

Part 3. Models and Datasets 25
6. Network candidates 25
6.1. FNO 25
6.2. UNet 25
6.3. CNO 26
6.4. KNO 27
6.5. Network Factory 27
7. Learning Pipeline 28
7.1. Masked Dataset 28
7.2. Prediction Model 31

Part 4. Evaluation and Comparison 33
8. Evaluation methodology 33
8.1. Dataset Construction 33
8.2. In-distribution VS Out-of-distribution 34
9. Full intensity-range 35
9.1. Single masks 35

9.2. Double masks 37
10. Specialized Ranges 41
10.1. The Poisson Equation 41
10.2. Heat Equation 45
10.3. Wave Equation 49
11. Other Evaluations 52
11.1. Dynamic remasking 52
11.2. Direct visual inspection 55
11.3. The island-ring juxtaposition 58
11.4. Non-standard masking 60
12. Executive Summary 62

Part 5. Epilogue 63
13. Review and Projection 63
13.1. Environment Setup 63
13.2. Timeline 64
13.3. Future work 65
14. Acknowledgements 65
References 66

Part 1. Prelude

1. Quick Start

1.1. **Overview.** Partial Differential Equations (PDEs) describe and model a multitude of phenomenon in diverse regimens of science and engineering. Historically, numerical methods distilled from traditional applied mathematics have been used to obtain their solutions [AH07]; [BF10]; [CK12]; [IK94]; [Kön03]; [QV09] with frequent references to analysis techniques [Kön03]; [QV09] . With the advent of machine-learning as a whole, the novel direction of research for solving PDEs has gravitated towards operator learning using deep neural networks. Notable success stories include the Unet [RFB15], DeepONet [Lu+21], DeepM&Mnet [Mao+21], KNO [Xio+23a]; [Xio+23b], FNO [Li+20]; [Kov+21] and CNO [Rao+23].

While traditional researches in machine-learning have generally been based on the assumption of data afflicted by some distribution-governed noise, an emerging trend is the attempt of incorporating such noisy data further deliberated tainted with *masks*. In the field of image classification and object detection, such masks would translate to patches of value 0 (zero) randomly applied to sections of the input channels, as seen in researches such as [Bac+22]; [Ton+22b]; [Ton+22a].

This project attempts to unify existing established frameworks PDE learning with the general machine learning practice of masking. By varying three configuration options of the masking procedure during various stages of the learning pipeline, the impact of masking on the trained models is studied under diverse settings using both in and out-of-distribution evaluation regimes.

1.2. **Thesis structure.** This thesis opens with generic preludial material in Part 1, outlining the project as a whole and providing some technical specifications.

Part 2 commences with a run-down of the fundamental concepts of this project, including discretization techniques and masking procedures. Also, the three fundamental PDE prototypes of this thesis are presented along with their representative respective datasets.

The four network models deployed in this project are introduced in the beginning sections of Part 3. Then, the construction procedure of the masked datasets is introduced. With focus on the single and double-masked tasks, the full learning pipeline is then specified, covering the process from dataset assembly to model evaluation.

Part 4 collects the results from the masked PDE learning problem statement and presents them in order of masking types and PDE models. In addition, a specific training scheduling technique is studied, along with alternative masking practices to justify and contrast with the default choices. Part 4 concludes with an executive summary, collecting all findings of this thesis in a centralized manner.

To round up the findings of this project, Part 5 provides extra documentation on general project execution, including time allocation and development setup.

1.3. **Related research.** As this project treads on both the field of machine learning and that of applied mathematics, a considerable number of researches from tangent areas are referenced during its execution. A selection of representative sources is listed here:

Methods from traditional numerical methods are mostly sourced from standard textbooks, such as [AH07]; [BF10]; [CK12]; [IK94]; [Kön03]; [QV09], including the PDE models and their respective solving methods. Fundamental analysis methods are referenced from [Kön03]; [AE09].

Generic concepts of machine learning are taken from [Mur22]; [HTF09], whereas techniques specific to deep learning are mostly consulted from [GBC16]. Masking as a general means for inflicting noise onto dataset is also studied in [Bac+22]; [Ton+22b]; [Ton+22a].

Operator-Learning as machine learning paradigm is introduced in [BT23]. Algorithms representative of this paradigm studied include [Li+20]; [Kov+21]; [Rao+23], all of which are used in this thesis as networks candidates.

2. Technical Details

2.1. **Programming.** The source-code is packaged in one monolithic repository, including the dataset generation, test suites, as well as statistical and visual analysis modules.

The programming is performed with the `Python` language with global type-hinting. Python versions no earlier than `3.11` is required. The virtual environment setup is managed with `poetry` [ET]. Automatic module testing is provided with `pytest` [Kre+04]. The source-code is consistently processed by the `black`-formatter [Wil+19] and implicitly conforms to the `PEP8` standard. A run-down of other software and hardware requirements are found in the closing portions of this thesis, notably in Section 13.1.

Based on the published code of their original authors, custom implementations for the `FNO`, `CNO`, `UNet` and `KNO` algorithms are provided. Implementation and configurations details are discussed when introducing the respective models in Chapter in 6. All other source code modules are implemented from scratch by the author.

2.2. **Formatting and Typesetting.** This paper is compiled by the XeTEX typesetting-engine featuring the open-source font-packages `Libertinus` and `Source-Code-Pro`.

2.3. **Version Tracking.** The source-code of this project is distributed under the following repositories:

- The main repository contains all code modules necessary for inspecting and reproducing the results of this study and is published under: GitLab [ETH-Domain] and GitHub.
- The thesis is tracked in a separate GitHub repository. This very document is also available directly as a pre-built PDF file.

Part 2. Background and Setup

3. Discretization

Numerical solutions of PDEs defined on continuous structures rely on the discretization of these domains. This section explains the discretization strategy of both the spatial and the temporal domains deployed in this thesis.

3.1. **Spatial discretization.** As argued in [Rao+23] in sections establishing its benchmarking setup[1], the two-dimensional space boasts of distinct merits for the purpose of gauging the performance of operator-learning: in essence, the learning landscape is sufficiently complex to challenge algorithms without inducing fearsome costs during training and evaluation, which is critical for rapid iteration and prototyping.

In this spirit, this project focuses on the two-dimensional space as well, with specifics of its discretization defined below:

Definition 1. Space mesh

Methods from the traditional discipline of numerical analysis stipulates the discretization as the transformation of any continuous space into a discrete mesh structure, as demonstrated in textbooks such as:[AH07]; [BF10]; [CK12]; [IK94]. While the quest for domain-specific, high-quality mesh generation is intensely contested in applied mathematics, this thesis, similar to other researches for PDE Learning such as [Li+20]; [Kov+21]; [Rao+23], uses the generic grid-shaped mesh. For the purpose of the two-dimensional space that this thesis dwells on, this translates to *rectangular* elements. Henceforth in this thesis, the terms »grid« and »rectangular mesh« are used interchangeably.

Implementation of such a rectangular mesh requires as basis the discretization of both dimensions x_1 and x_2, specified fully with the following four values:

$$x_{1,2}^{\text{discrete}} := \left(\begin{bmatrix} \text{start} \triangleq \texttt{float} \\ \text{step-size} \triangleq \texttt{float} \\ \text{end} \triangleq \texttt{float} \\ \#\text{steps} \triangleq \texttt{int} \end{bmatrix} \right)$$

where »#steps« denotes the number of grid-points for the dimension, also referred to as »resolution« in researches such as [Li+20]; [Kov+21]; [Rao+23].

Note 1. Construction of rectangular mesh

It should be noted that the knowledge of three of any of the four variables in the definition above suffices to reconstruct the remaining one, enabling thus the spatial discretization as a rectangular mesh. This thesis uses the following two typical construction methods to describe spatial discretization:

- the starting value, the step-size, and the number of steps;
- the starting and the end value, and the number of steps,

each with its own merit: the knowledge of the step-size allows direct control over specific constraints such as the CFL condition [AH07]; [BF10]; [CK12]; [IK94], whereas explicit end value aides the tuning of spatial boundary conditions. In this project, specification of the spatial grid is provided by the more appropriate of these two construction variants specific to the PDE prototype at-hand.

Note 2. Quality of grid discretization

This square mesh, though primitive compared to specialized mesh generation as deployed in the context of traditional numerical methods, is, as pointed out previously, nevertheless common-place for

[1] referred to as »Representative PDE Benchmarks (RPB)«, Page 8

the status-quo of PDE Learning. A potential direction for future research involves investigation of more sophisticated discretization strategies, as discussed in the closing section of this thesis in Section 13.3.

Example 1. Space-grids in action

This thesis uses a constant resolution of 64 for both for the purpose of spatial discretization, as also seen in existing researches including [Li+20]; [Kov+21]; [Rao+23]. This value is chosen consciously: the exact 2-exponential facilitates efficient computation of the discrete Fourier transformation, as explained in [Li+20]; [Kov+21]; [Rao+23].

On the other hand, the other three parameters of the grid are variable, chosen specifically for each individual PDE model. As an example, the mesh used for solving the Poisson Equation, seen later in Section 5.2, is defined as follows:

$$\mathbb{R}^{64} \ni x_{1,2}^{\text{Poisson}} := \begin{cases} \text{start} := 0.00 \\ \text{step-size} := 0.01 \end{cases}$$

which translates to an end value of 0.63: be careful to avoid the off-by-one error!

The reader is referred to chapters dedicated to the specific PDE prototypes for their space grid specifications.

3.2. **Temporal discretization.** A similar procedure is applied to discretize the time axis, with the apparent additional constraint that the starting value is non-negative. As is typical for temporal discretization, the initial time is set to 0.0.

Note 3. Application of temporal discretization

It should be noted that the temporal discretization is not used to steady-state, time-independent PDEs, including, in particular, the Poisson Equation, introduced in Section 5.2. On the other hand, PDEs such as Heat Equation and the Wave Equation, introduced in Section 5.3 and Section 5.4 respectively, require temporal discretization as discussed in this section.

3.3. **API.**

Definition 2. Key API

Listing 1 provides an overview of the `Grid` class.

```
# src/numerics/grid.py

class Grid:
    def __init__(self, n_pts: int, stepsize: float, start: float = 0.0):
        ... # detail omitted

    @classmethod
    def from_start_end(cls, n_pts: int, start: float, end: float) -> "...
    ...Grid":
        return cls(n_pts, (end - start) / (n_pts - 1), start)

    def step(self, with_start: True, with_end: True) -> list[float]:
        ... # detail omitted

    def is_on_boundary(self, val: float) -> bool:
        ... # detail omitted
```

Listing 1. Essential API for grid

4. Masking

4.1. **Introduction.** With the formality of recalling discretization out of the way, »masking«, the very namesake of this thesis, can now be introduced. This section formalizes the concept of masking, identifies its configuration options and enumerates the three distinct masking flavors deployed in this thesis.

Definition 3. General masking procedure

The output of applying masking to a matrix-like object $\mathbf{R}$ is another matrix-like object $\mathbf{R}^{\text{mask}}$ of the same shape:

$$\mathbb{R}^{d_1 \times d_2 \times \cdots} \ni \mathbf{R} \xrightarrow{\mathbf{M}} \mathbf{R}^{\mathbf{M}} \in \mathbb{R}^{d_1 \times d_2 \times \cdots}$$

The masked object $\mathbf{R}^{\mathbf{M}}$ is said to have been masked with mask $\mathbf{M}$ from the original, unmasked $\mathbf{R}$. Specifically, the masking procedure is understood as the following three-step process:

(1) obtain the masking *intensity*, either as a constant value or from sampled from some random distribution;
(2) determine, according to some masking *strategy*, the indexes `idx` of the mask-candidates, possibly with some sampling routine as well;
(3) set the selected masking indexes to some masking *value* v:

$$\mathbf{R}^{\mathbf{M}} := \Big(\mathbf{R}\texttt{[idx]} \leftarrow v \Big)$$

The remainder of this section elaborates on the masking procedure nominated above.

4.2. **Masking intensity.**

Definition 4. Masking intensity

The masking intensity i denotes the »strength« of the mask: the higher the intensity i, the more masking indexes `idx` are selected:

$$\#\texttt{idx} \propto i$$

Definition 5. Intensity sampling

To infuse the masking operation with extra entropy, the masking intensities i may be sampled from a uniform distribution to capture a *range* of acceptable intensity values:

$$i \simeq \text{Unif}\left[i_{\min}; i_{\max}\right]$$

Similar to the convention of `scipy` [Vir+20], the masking intensity can also be specified with a (center) location and a spread value[2], captured by the following (hand-wavy) notation:

$$i \simeq \begin{cases} \text{Unif}\left[i_{\min}; i_{\max}\right] \\ \text{Unif}\left[i_{\mu}; i_{\sigma}\right] \end{cases}$$

Note that an exact, deterministic intensity value is naturally obtained by specifying a vanishing spread value $i_{\sigma} \equiv 0.0$.

Note 4. Availability of intensity sampling

All three masking strategies of this thesis (to be introduced in the following section) support the intensity-sampling procedure above. The reference implementation included in the source code provides a function that encapsulates the intensity resampling logic from the underlying uniform distribution, see Listing 2.

[2] API for uniform distribution by `scipy`

Definition 6. Masking intensity revisited

Given some matrix defined on a space-grid, once the masking intensity i is known, the exact number of masking indexes n is calculated as follows:

$$n := \texttt{\#idx} := \texttt{round}\left(i \times \left\| \text{grid} \right\|\right)$$

In other words, a higher intensity value i translates to proportionally larger number of entries n tainted after masking, with the proportionality factor understood as the number of grid-points of the spatial discretization.

It is now up to the masking *strategy* to obtain these n masking indexes `idx`. Three such *strategies* are implemented and utilized in this thesis, to be introduced in the upcoming section:

4.3. **Masking strategies.**

4.3.1. *Random style.* As suggested by its namesake, the mask indexes are selected randomly. This style of masking is widely used, in particular, as seen in publications such as [Ton+22a]; [Ton+22b]; [Bac+22].

Example 2. Visualization of random-masking

Figure 4.1 provides visualization of random-style masking at 50% intensity. The intensity-spread is set to 0.0 to guarantee the exact intensity for this particular instance.

The particular raw, unmasked matrix $\mathbf{R}$ is defined as:

$$\mathbf{R}_{x_1,\,x_2} := \cos\left(\frac{1}{2}\pi \cdot x_1\right) \times \cos\left(\frac{1}{2}\pi \cdot x_2\right) \quad , \quad x_{1,2} \in \Big[-1.0; +1.0\Big]^2$$

One observes that entries of $\mathbf{R}$ is strictly within the range of $[0; 1]$ and form a perfect dome-like shape.

Figure 4.1 is organized as follows: every matrix is shown in a 2D contour-plot in the top row and a 3D surface-plot in the bottom row. The raw, unmasked $\mathbf{R}$ is shown as the first column on the left, whereas three outputs of this 50% random-mask as shown in the other three columns to the right: from the second to the fourth column, a masking value of 0.0, 0.25 and 0.5 is used to calculate the masked $\mathbf{R}^{\mathbf{M}}$ respectively.

4.3.2. *Island style.* The island-style masking strategy keeps a centered portion, or an »*island*«, of the input matrix in-tact, while targeting its surrounding entries. The masked candidates form a rectangular »annulus«, or »ring«

Note 5. Alternative interpretation of masking intensity

In the case of island-masks, the masking intensity i can also be understood as:

$$i \equiv \frac{\text{Area}\,(\text{annulus})}{\text{Area}\,(\text{grid})} \equiv 1 - \frac{\text{Area}\,(\text{island})}{\text{Area}\,(\text{grid})}$$

On a *square* grid, given some desired masking intensity i, the proportion p of the side-length of the island over that of the entire grid is calculated as follows:

$$p = (1 - i)^{1/2}$$

Example 3. Visualization of island-masking

Figure 4.2 provides visualization of the island masks with the *same* underlying unmasked, raw matrix $\mathbf{R} := \cos(\cdot) \times \cos(\cdot)$. The masking intensity shown is 75%. Following the calculation in Note 5, the side-length of the center island is thus exactly half of that of the whole grid.

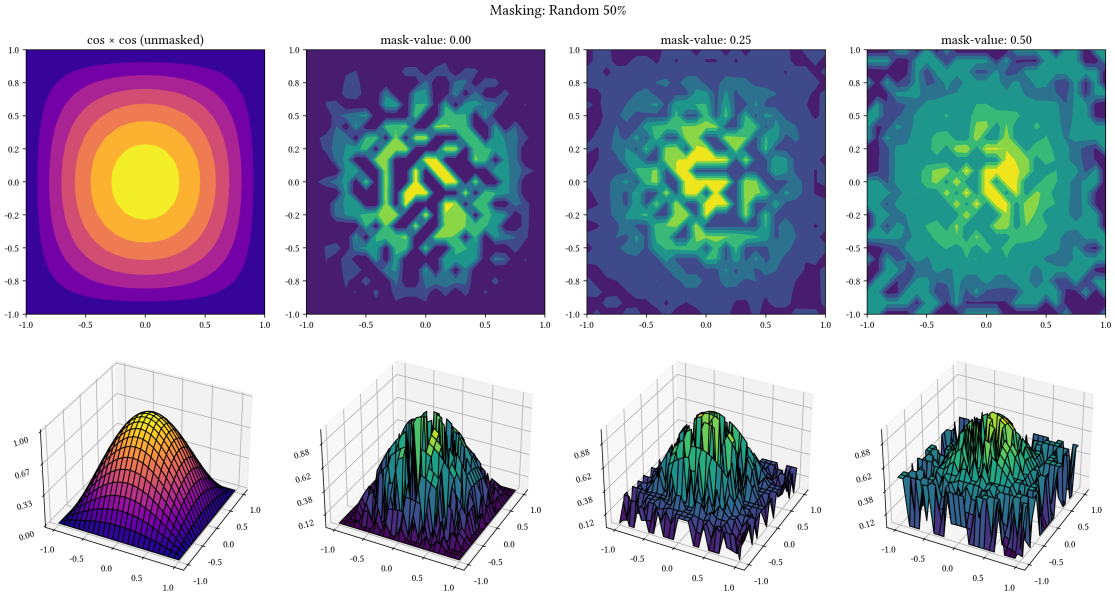


FIGURE 4.1. Visualization of random masking of intensity 50%, using $\mathbf{R} := \cos(\cdot) \times \cos(\cdot)$ on $[-1;1]^2$ in Definition 2. Top row: 2D contour-plots; bottom row: the corresponding 3D surface-plots. From left to right: the raw, unmasked $\mathbf{R}$, followed by the masked results $\mathbf{R}^{\mathbf{M}}$ with masking value 0.0, 0.25 and 0.50 respectively, plotted using a different color-theme.

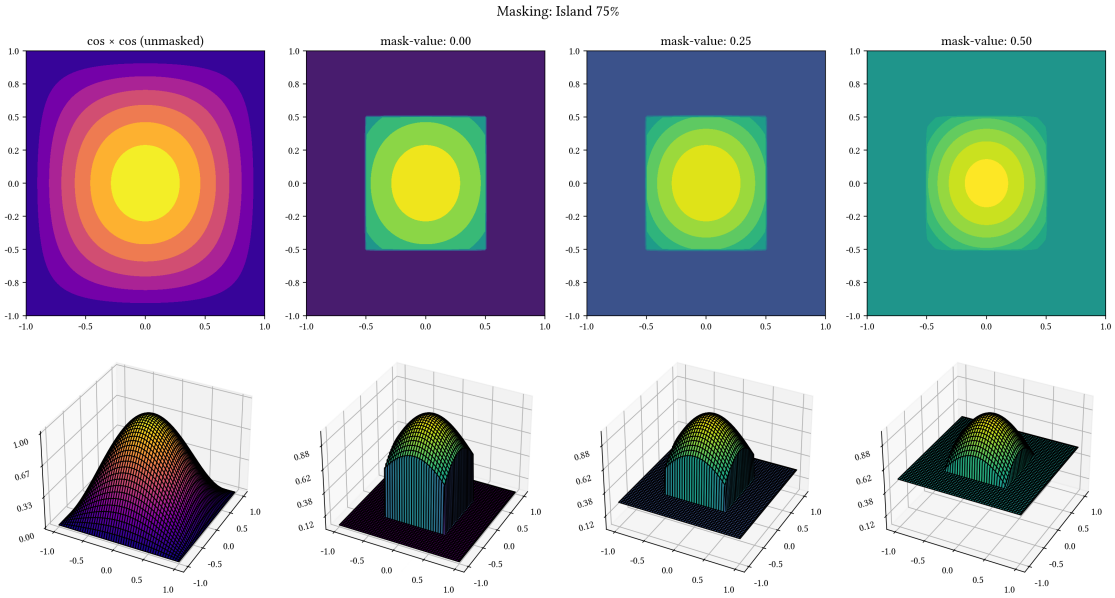


FIGURE 4.2. Visualization of the island-style mask at 75% intensity. The same underlying, unmasked matrix $\mathbf{R} := \cos(\cdot) \times \cos(\cdot)$ is used, and all plotting conventions follow those of Figure 4.1 for random-masks. Note that for the 75% overall intensity, the side-length of the (center) island is half of that of the entire (outer) grid.

4.3.3. *Ring style.* Conceived as the mirror-image of the island-masks, ring-style masks apply the masking-value to the center proportion of the grid, while leaving the surrounding ring untouched.

Note 6. Interpretation of masking intensity

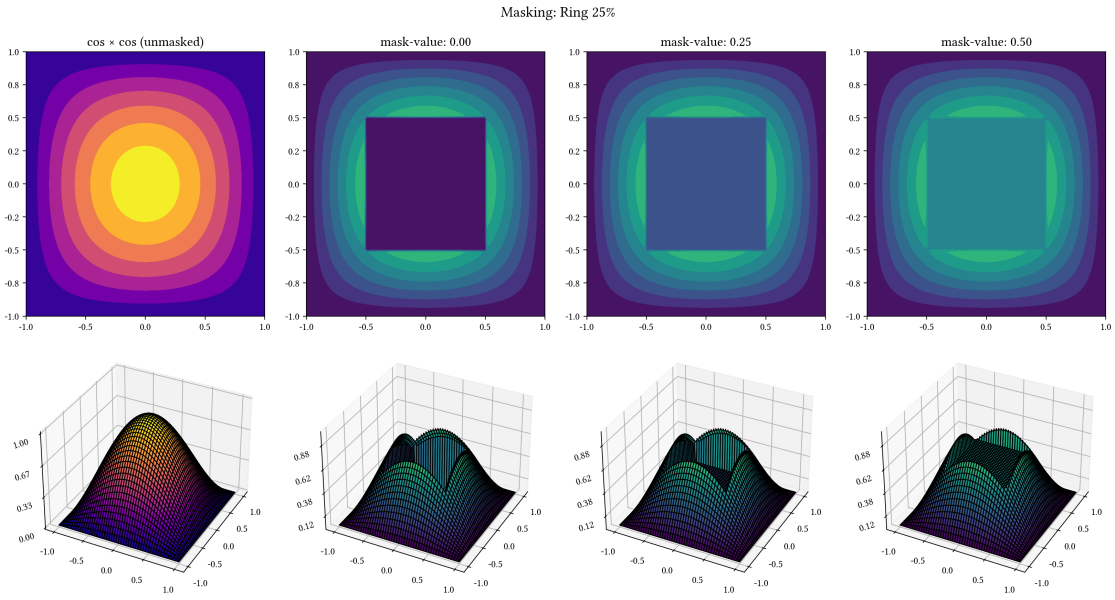


FIGURE 4.3. Visualization of the ring-style masks. The underlying, unmasked matrix, as well as all plotting conventions follows those of Figure 4.1.

The masking intensity of ring-style masks can also be understood as the ratio of the area of the (masked) center island over the area of the grid:

$$i \equiv \frac{\text{Area (island)}}{\text{Area (grid)}} \equiv 1 - \frac{\text{Area (ring)}}{\text{Area (grid)}}$$

Again, on a square two-dimensional grid, the proportion p of the side-length of the center island over that of the entire grid is calculated as:

$$p = i^{1/2}$$

Example 4. Visualization of ring-masking

Figure 4.3 visualizes a 25% ring-style mask using the *same* underlying unmasked, raw matrix $\mathbf{R} := \cos(\cdot) \times \cos(\cdot)$ seen previously. Per Note 6, the side-length of the center island is exactly half of that of the whole grid.

Note 7. Complementary masks

The intensities of 75% and 25% for visualizing the island-style and ring-style masks above are chosen consciously: one observes that the resulting, masked objects are perfect *complements* of each other, i.e., the masked portion of one corresponds directly to the unmasked portion of another.

In fact, any pairing of an island-mask with intensity i and a ring-mask with intensity $1 - i$ forms exact compliments of each other. This specific complementary property of masking is further studied in Section 11.3.

Definition 7. Obtaining masking candidates

Once the masking strategy is decided upon, the masking candidates are readily available:

- for the random-style masks, the candidates are calculated via random sampling from all indexes of the underlying matrix $\mathbf{R}$;
- for the island-style and ring-style masks, the candidates are known deterministically with the calculation formula for the side-length proportion p of the embedded center square as specified above.

The final step of masking corresponds to setting the masking candidates to some masking value v:

4.4. **Masking value.** A common masking value to apply to the masking candidates, as deployed in researches such as [Ton+22a]; [Ton+22b]; [Bac+22], is to uniformly set the values of these entries to 0.

This thesis continues upon this strategy of constant masking value, but consciously adjusts the value to 0.5. The justification of this decision stipulates a sneak-peak to the dataset normalization technique deployed in this project, discussed further in Section 7.1.3 and Definition 25:

Proposition 1. *Dataset normalization*

This thesis performs $[0, 1]$ *normalization of the dataset per [Dev24]; [HTF09]; [GBC16], such that every channel is linearly scaled within the range of* $[0, 1]$.

The adjustment of masking value from 0 to 0.5, though insignificant at first glance, brings forth a deep implication: the masked value v now corresponds to the *mean* of the value-range of the data channels:

$$v^{\text{mask}} := \text{Dataset}_{\mu}^{\text{norm}} \equiv 0.5$$

Masking performed with $v^{\text{mask}} \equiv 0.5$ can thus be considered »dataset-neutral«, as it does not implicitly skew the resultant, masked dataset towards the minimum, as would be in the case of $v = 0$.

Later analyses in Chapter 11.4 study the impact such »non-neutral« masking-values, using 0 and 1 as alternative masking values for comparison.

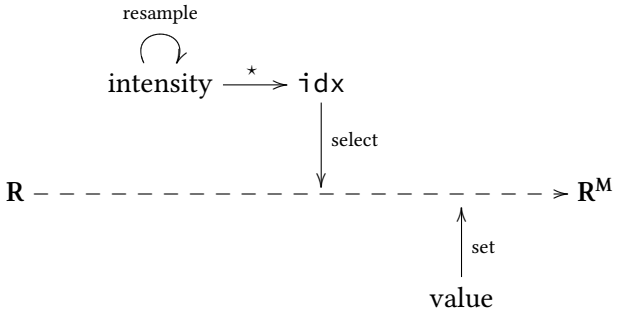
Note 8. Non-constant masking values

It should be noted that masking value v need not be constant: in fact, the author considers the non-constant masks a future research direction, see discussions in Section 13.3.

4.5. **Masking overview.**

Conclusion 1. Masking procedure

The following chart depicts a curt summary of the masking procedure explained in previous sections:



where $\star$ denotes the calculation step of the indexes `idx` of the masking candidates under application of the masking strategy of choice.

Key elements concerning the implementation of masks are listed as follows:

Definition 8. Masking API and Implementation

The base `Masker` class provides the abstract method `mask()`, which accepts as argument a raw `tensor` of `pytorch`, and returns the masked matrix, see Listing 2. Subclasses of `Masker` are expected to provide corresponding implementations of the `mask()` method.

A common utility `_sample_intensity()` is provided to obtain the masking intensity i sampled per uniform distribution as explained above.

Note that the default construction yield a masker object of intensity range:

$$\text{mask-intensity} \simeq \mathcal{P} := \text{Unif}\left[c_{\text{low}} = 0.4; c_{\text{high}} = 0.6\right]$$

paired with the neutral masking value $v = 0.5$ as discussed previously.

The three masking variants are implemented in the same module inheriting the common base `Masker`. The motivated reader is invited to implement its own class of masker class for alternative styles of masking other than those mentioned above.

```python
# src/util/dataset.py

class Masker:
    def __init__(
        self,
        intensity: float = 0.5,
        intensity_spread: float = 0.1,
        value_mask: float = 0.5,
    ):
        pass # detail omitted

    @abc.abstractmethod
    def mask(self, full: torch.Tensor) -> torch.Tensor:
        raise NotImplementedError # to be overloaded by subclasses

    def _sample_intensity(self) -> float:
        pass # detail omitted

class MaskerRandom(Masker): # direct inheritance
    ... # detail omitted

class MaskerIsland(Masker): # ditto
    ... # detail omitted

class MaskerRing(Masker): # ditto
    ... # detail omitted
```

Listing 2. Essential API for masking

Thus concludes this introductory chapter on masking, which provides concrete definitions of the masking procedure, showcasing three variants of masks with their respective configuration possibilities. To conclude this section and as a sneak-peak into the upcoming section where concrete datasets are formed with masking, the following masks are used in this thesis:

Example 5. Masking ranges for learning pipeline

- for *training*: four masking intensity-ranges, all of random-style: full-range $[0\%; 100\%]$, as well as low, mid and high regimes: $[20\%; 40\%]$, $[40\%; 60\%]$ and $[60\%; 80\%]$ (4 masking intensity-ranges in total);
- for *evaluation*: all three masking styles, each with 10 intensities ranges: $[0\%; 10\%]$, $[10\%; 20\%]$ till $[90\%; 100\%]$ (30 masks in total).

Thus, for every model on each dataset, a total of 4 masking iterations are performed for training, and 30 for evaluation.

5. The PDE Prototypes

Another central component of the setup work is the dataset generated by the PDEs, typically revolving around solving the PDEs themselves. This section introduces the three core PDE prototypes of the project and the respective datasets induced by them. Some necessary setup steps are introduced in the following section:

5.1. **Setup.**

5.1.1. *Sample Super-positioning.*

One frequently seen paradigm for generating the PDE is discussed below:

Definition 9. Super-positioning construction

Assuming linearity of the underlying PDE model, multiple *samples* can be linearly combined to construct one *instance*, as seen in [Rao+23]; [AH07].

Such super-positioning technique provides entropy critical to dataset generation while guaranteeing a common underlying distribution. The critical relevance of these dataset properties is studied at length in textbooks of general machine-learning such as [HTF09]; [Mur22] as well as those of traditional statistics including [Was13].

The following example illustrates this technique:

Example 6. Sample super-positioning

Each instance of the PDE solution u_i is constructed via a linear combination of k samples u_i^k:

$$u_i := \left[\text{weights}\right] \circ \left[\text{samples}\right] = \left[\mathsf{R}_i^k\right] \circ \left[u_i^k\right]$$

where the weights R_i^k are drawn from some distribution $\mathcal{P}$ chosen a priori, which typically corresponds to a uniform distribution:

$$\mathsf{R}_i^k \simeq \mathcal{P} := \text{Unif}\left[c_{\text{low}}; c_{\text{high}}\right]$$

in which case the configurations c_{low} and $c_{\text{high}}$are considered tunable hyper-parameters.

Note 9. Scope of super-positioning

Apparently, such super-positioning construction shall be applied to all *channels* induced by the PDE dataset. As an example, for the case of the Poisson equation as introduced in Section 5.2, both the solution u and the source term f must be linearly combined if super-positioning is used.

Note 10. Prerequisite for super-positioning

The underlying PDE must be linear for the super-positioning technique above to be applicable [Kön03]; [AE09]. Thankfully, the three PDE prototypes used in this thesis are all linear by nature, leading handily to the applicability of this technique. Reader should be also aware that more involved PDE models, including the prominent Navier–Stokes equations, are likely non-linear.

5.1.2. *Design and Implementation.*

Note 11. The programmer's perspective

The remaining logic, such as the randomly weighted linear combination and the final dataset assembly with the coordinates matrices, can be reused to a substantial extent.

This stipulates a clean overload structure: the base-class offers those reusable utility functions, and individual dataset providers shall overload this base to provide concrete implementation for the generation of the instance-samples of the underlying PDE prototype. This design pattern is used throughout

the code-base of this project, and the interested reader is invited to inspect the details in the source repository. Listing 3 showcases this base-class `DatasetPDE`.

Central to the API is the abstract method `solve_instance()`, which returns a sequence of matrices[3] of some PDE. Specific PDE models are to inherit this API and provide implementation of this method.

The length of this sequence denotes the number of channels relevant to fully capture the PDE problem and saved in member `N_CHANNELS`. Continuing with the previous example of the Poisson equation, its implementation would set `N_CHANNELS` to 2, and have `solve_instance()` return the solution u and the source f as the two channels.

The `DatasetPDE.as_dataset()` method handles the low-level logic to assemble multiple instances into one dataset for `pytorch`.

```
# src/pde/dataset.py

class DatasetPDE:
    N_CHANNELS: int # to be overwritten by individual PDEs

    def as_dataset(self, n_instances: int) -> T_DATASET:
        # assemble instances into dataset (common to all PDEs)
        ... # implementation detail not shown

    @abc.abstractmethod
    def solve_instance(self) -> Sequence[torch.Tensor]:
        raise NotImplementedError # to be implemented by individual PDEs
```

Listing 3. Key methods of PDE-Dataset API

Note 12. Where abstraction stops

From the perspective of API design, the super-positioning pipeline described above would suggest further (that we further abstract the super-positioning logic)

that the construction of such datasets requires only the implementation of generating one such sample of the instances. However, to allow extendability to other non-linear PDE models, the abstraction stops at the instance-level: the boilerplate of implementing super-positioning is fairly minimal with the natural matrix-operation capabilities of `pytorch` [Pas+19] and `numpy` [Har+20]. Alternatively, non super-positioned datasets, as justified in the discussion above, are relieved from such unnecessary inheritance.

Note 13. Other functionalities of the dataset API

In addition to the core instance-solving and dataset-exporting methods mentioned above, several peripheral functions are also instrumental to its workings. These include, among others, a save-load utility. In particular, the save-load path is calculated as follows:

`<BIN_DIR> / <PDE_name> / <dataset_name>-<type>_<n_instances>.<suffix>`

where `<BIN_DIR>` is a project-wide constant for the path all binary outputs and `<type>` denotes the designated usage mode of training or evaluation for the dataset. For the »sum-of-sines« dataset as introduced in Definition 10, an evaluation dataset of 300 samples is saved at:

[3] here, »matrices« refer to `python` objects of type `torch.Tensor`

```
bin
|-- poisson
   |-- sum_of_sine-eval_300.pth
   |-- ...
|-- heat
   |-- ...
```

Other essential utilities of the PDE dataset API include those related to visualization: which will be further detailed with concrete plot examples when the individual PDE prototypes are studied.

With the setup work complete, the individual PDE prototypes are now introduced:

5.2. **Poisson Equation**. Used to describe the potential field generated by some density distribution, the Poisson equation generalizes the Laplace's equation with:

$$\begin{aligned} & \frac{\partial^2}{\partial x_1^2} u + \frac{\partial^2}{\partial x_2^2} u = -f \\ \implies \qquad & -\nabla^2 u = \quad f \end{aligned}$$

This thesis provides the following two datasets for the Poisson equation.

5.2.1. *Sum of Sines.*

Definition 10. The »sum-of-sines« dataset

Each instance of the solution u_i is constructed as follows:

$$u_i := \frac{c_c}{\pi} \cdot \underbrace{\sum_{k_1 \in [1; c_k]} \sum_{k_2 \in [1; c_k]}}_{=: \sum\sum} \left\{ \left(\mathsf{R}_{k_1,k_2} \left(k_1^2 + k_2^2 \right)^{-c_r} \right) \cdot \underbrace{\sin\left(k_1 \pi x_1\right) \sin\left(k_2 \pi x_2\right)}_{=: \underline{\sin}} \right\}$$

where:

- the constant $c_k \in \mathbb{N}^+$ determines the number of samples used to generate one such instance with the super-positioning technique introduced in Definition 5.1.1;
- the sample-weights R_{k_1,k_2} are drawn from a uniform distribution;
- a configurable scaling constant $c_c \in \mathbb{R}^+$ is provided to aid visualization and inspection of model data.

Due to Fact 1, the corresponding source term f_i is easily obtained by super-positioning the Laplacian ∇^2 of u_i:

$$\begin{cases} u_i := \frac{c_c}{\pi} \cdot \sum\sum \left\{ \left(\mathsf{R}_{k_1,k_2} \left(k_1^2 + k_2^2 \right)^{c_r - 1} \right) \cdot \underline{\sin} \right\} \\ f_i := c_c \pi \cdot \sum\sum \left\{ \left(\mathsf{R}_{k_1,k_2} \left(k_1^2 + k_2^2 \right)^{c_r} \right) \cdot \underline{\sin} \right\} \\ \underline{\sin} := \sin\left(k_1 \pi x_1\right) \sin\left(k_2 \pi x_2\right) \end{cases}$$

Fact 1. *Laplacian of double trig-product*

It is readily verifiable that:

$$\mathit{trig}\left(k_1 \pi x_1\right) \mathit{trig}\left(k_1 \pi x_2\right) \overset{\nabla^2}{\longmapsto} -\left(k_1^2 + k_2^2\right) \pi^2 \cdot \Big(\mathit{trig}\left(k_1 \pi x_1\right) \mathit{trig}\left(k_1 \pi x_2\right) \Big)$$

where $\mathit{trig}\left(\cdot\right)$ *denotes the sine or the cosine function.*

For this thesis, the following configuration of these constants is used for the sum-of-sines dataset:

Example 7. Configuration of »sum-of-sins« dataset

As previewed in Example 1, the following spatial grid of resolution 64 per axis is used:

$$\mathbb{R}^{64} \ni x_{1,2} := \begin{cases} \text{start} := 0.00 \\ \text{step-size} := 0.01 \end{cases}$$

Configurations pertaining to the dataset itself are as follows:

- each instance u_i is generated from $c_k := 4$ samples $u_i^{k \in \{1,2,3,4\}}$;
- the weights R_{k_1,k_2} are spawned uniformly from $\mathcal{U}\,[-1.0; +1.0]$;
- c_r is chosen as 0.85, which is the negative of the r constant in its original form as proposed in [Rao+23];
- manual scaling is kept neutral, i.e., $c_c := 1.0$.

Note 14. Properties of the sum-of-sines dataset

One notes that the solution u will uniformly evaluate to 0 when either coordinate x_1 or x_2 is 0; in the sense of the grid as chosen in this project, this means that u will vanish on the lower boundary of either dimension:

$$u\,(x_1 = 0, \forall x_2) \equiv u\,(\forall x_1, x_2 = 0) \equiv 0$$

This dataset will thus incorporate both fixed (vanishing) and varying boundary-condition (BC) on different segments of the spatial boundary. Note that this choice of mixed boundary condition is deliberate to contrast [Rao+23], which also used this dataset to represent the Poisson equation, albeit with a uniform boundary condition with the end-point set to 1.0.

Note 15. Visualization of the sum-of-sines dataset

Visualization of two instances of this dataset is provided in Figure 5.1, where the first and the second columns depict the first (u, f) pair, the third and fourth column the second respectively. The first row shows the target matrices in a 2D contour-plot, whereas the second in the corresponding 3D surface-plot.

Note that each subplot is scaled to its own dynamic range, as indicated by the z-axis tick-labels of the surface-plots in the second row.

5.2.2. *Sum of Gaussians.* Another dataset for the Poisson equation is constructed by super-positioning the probability density functions (PDFs) of Gaussian Distributions. This section describes its initialization and contrasts it with the »sum-of-sines« dataset introduced previously.

Note 16. Sign modification

In preparation of introducing this dataset, the target Poisson equation as seen previously in Section 5.2 is slightly modified such that the negative of the right-hand side is taken:

$$\nabla^2 u = f$$

The reason for this adjustment is purely cosmetic and follows from the very definition of this dataset:

Definition 11. The »sum-of-Gaussians« dataset

The following dataset is generated:

$$u_i := \sum_{k \in [1; c_k]} \left\{ \mathsf{R}^k \cdot u_i^k \right\} := \sum_{k \in [1; c_k]} \left\{ \mathsf{R}^k \cdot \mathcal{N}^k \right\}, \quad \mathsf{R} \in \mathbb{R}^{c_k}$$

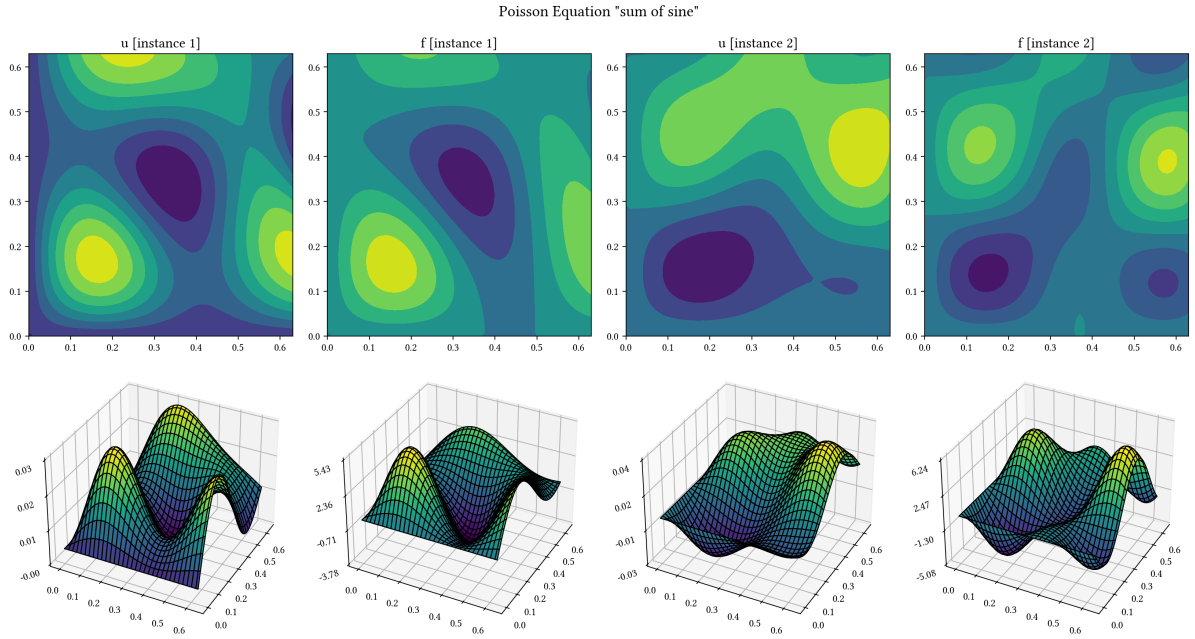


FIGURE 5.1. Visualization of two instances of the »sum-of«sines« dataset for the Poisson equation given in in Definition 10, configured as specified in Definition 7. $c_k := 4$ samples are super-positioned to generate one instance of (u, f). First instance shown in row 1 and 2, second instance in 3 and 4. First row: 2D contour-plots, second row: corresponding 3D surface-plots. Each subplot scaled to its own dynamic range.

where $\mathcal{N}^k$ denotes the probability density function (PDF) of the two-dimensional normal distribution, whose expected-value μ^k and covariance-matrix Cov^k are sampled from their respective (uniform) random distributions $\mathcal{U}^\mu := \mathrm{Unif}\left[c^\mu_{\text{low}}; c^\mu_{\text{high}}\right]$ and $\mathcal{U}^\sigma := \mathrm{Unif}\left[c^\sigma_{\text{low}}; c^\sigma_{\text{high}}\right]$ as follows:

$$\mathcal{N}^k \simeq \begin{cases} \mathbb{R}^2 \ni \quad \mu^k := \begin{bmatrix} \mu^{k,1} \simeq \mathcal{U}^\mu \\ \mu^{k,2} \simeq \mathcal{U}^\mu \end{bmatrix} \\ \mathbb{R}^{2\times 2} \ni \mathrm{Cov}^k := \begin{bmatrix} \left(\sigma^{k,1} \simeq \mathcal{U}^\sigma\right)^2 & 0 \\ 0 & \left(\sigma^{k,2} \simeq \mathcal{U}^\sigma\right)^2 \end{bmatrix} \end{cases}$$

With the conscious choice of the diagonal covariance-matrix, one could easily calculate the analytic expression for the corresponding source term f^k_i. Collecting the results, one concludes:

$$\begin{cases} u^k := \dfrac{1}{2\pi\sigma_1\sigma_2} \cdot \underline{\exp} \\ f^k := \dfrac{1}{2\pi\sigma_1\sigma_2} \cdot \left(\dfrac{1}{\sigma_1^4}\left(x_1-\mu_1\right)^2 - \dfrac{1}{\sigma_1^2} + \dfrac{1}{\sigma_2^4}\left(x_2-\mu_2\right)^2 - \dfrac{1}{\sigma_2^2}\right) \cdot \underline{\exp} \\ \underline{\exp} := \exp\left(-\dfrac{1}{2}\left(\dfrac{1}{\sigma_1^2}\left(x_1-\mu_1\right)^2 + \dfrac{1}{\sigma_2^2}\left(x_2-\mu_2\right)^2\right)\right) \end{cases}$$

Note that the sub-index i is omitted above for notational convenience.

Example 8. Choice of constants

- similar to the »sum-of-sines« dataset as introduced in Definition 10, $c_k := 4$ samples of u^k_i are used to generate one instance of the solution u_i;
- the sample-weights R^k are drawn from a uniform distribution: $\mathsf{R}^k_i \simeq \mathcal{U}\left[0.3; 0.7\right]$;

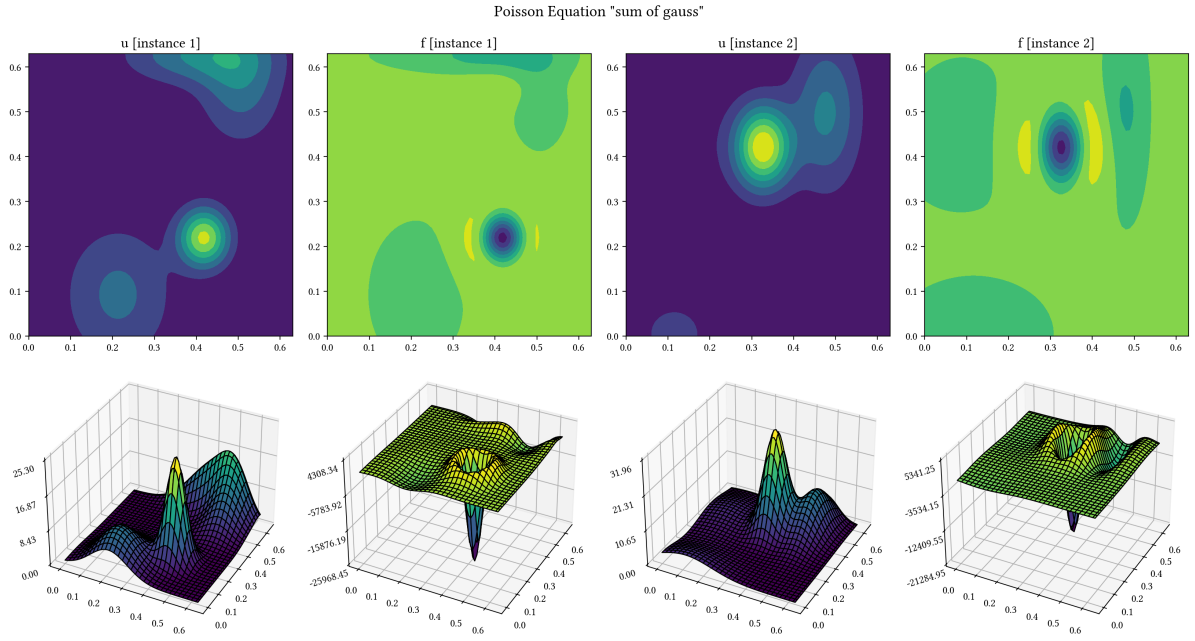


FIGURE 5.2. Visualization of the »sum-of-Gaussians« dataset, each constructed from $c_k := 4$ samples as specified in Definition 11. Plotting convention follows that of Figure 5.1 for the »sum-of-sines« dataset. Each subplot scaled to its own dynamic range.

- to construct the Gaussian PDFs $\mathcal{N}^k$, expected-values μ^k are sampled (uniformly) from the entire grid, and values of $\sigma^{k,\{1,2\}}$ of the diagonal covariance matrix Cov^k are spawned from $\mathcal{U}^\sigma\,[0.04; 0.13]$.

In similar fashion to the »sum-of-sines« dataset, two instances of the »sum-of-Gaussians« dataset are plotted in Figure 5.2.

5.2.3. *Heuristic comparisons.* By inspecting the plots for the sum-of-sines and the sum-of-Gaussians datasets in Figure 5.1 and Figure 5.2 respectively, one draws the following conclusion: the sum-of-sines dataset appears »smoother« in general, where instances of the sum-of-Gaussians dataset are more jagged and display highly pronounced extrema. Also, the sum-of-Gaussians dataset contains large areas of 0, whereas instances of the sum-of-sines dataset are generally non-vanishing.

For the purpose of this thesis, the sum-0f-sines dataset will be the primary representative of the Poisson equation from this point forward. The sum-of-Gaussians variant provides an alternative for the interested reader if further experimentation is desired.

5.3. **Heat Equation.**

Definition 12. Developed by Fourier in 1822, the parabolic heat equation describes the diffusion process of physics quantities such as its namesake through a certain region:

$$\begin{aligned} \frac{\partial}{\partial t} u &= \frac{\partial^2}{\partial x_1^2} u + \frac{\partial^2}{\partial x_2^2} u \\ \implies \quad \dot{u} &= \nabla^2 u \end{aligned}$$

Definition 13. Dataset for heat equation

The dataset is constructed with the same super-positioning technique as seen previously:

$$u_i := \sum_{k \in [1; c_k]} \left\{ \mathsf{R}^k \cdot u_i^k \right\}, \quad \mathsf{R} \in \mathbb{R}^{c_k}$$

where each instance-sample u_i^k is defined as:

$$u_i^k := \exp\left(-2k^2\pi^2 t\right) \underbrace{\sin\left(k_1\pi x_1\right)\sin\left(k_2\pi x_2\right)}_{=:\,\underline{\sin}}$$

The validity of the construction is easily checked with the property of the squared trig-terms Laplacian stated in Fact 1, previously seen when defining the sum-of-sines dataset for the Poisson equation in Definition 10.

Due to the double sin-product term, this dataset is internally named »sum-of-sines« as well, similar to the dataset for the Poisson equation seen in Definition 10.

Example 9. Configuration of heat equation dataset

The dataset above is defined on the following (two-dimensional) grid:

$$x_{1,2} \in [-1.0, +1.0]^2$$

with the usual 64 grid-points resolution. The end-time is defined as $t_T := 0.0050$. One collects the details of this dataset with the boundary conditions as follows:

$$\begin{cases} u_i^k := \exp\left(-2k^2\pi^2 t\right) \cdot \underline{\sin} \\ u_i^k\left(0, \forall x_1, \forall x_2\right) := \underline{\sin} \quad \text{[Temporal Bound.]} \\ u_i^k\left(\forall t, \partial_{x_1}, \partial_{x_2}\right) \equiv 0 \quad \text{[Spatial Bound.]} \\ \underline{\sin} := \sin\left(k\pi x_1\right)\sin\left(k\pi x_2\right) \end{cases}$$

The super-positioning procedure is configured in a similar manner as for the Poisson equation datasets in Definition 7. Notably, $c_k := 4$ samples are included in each instance, and sample weights R^k are drawn from the usual uniform distribution Unif $[-1.0; +1.0]$:

$$\mathsf{R}_k \simeq \mathcal{U}\left[-1.0; +1.0\right]$$

Note 17. Visualization of dataset

Visualization of the dataset as specified above is given in Figure 5.3. Subplots in the three columns depict, from left to right, snapshots of the solution u at the initial time $u_{t_0 \equiv 0.0}$, the middle $u_{t=0.0025}$ and the final time $u_{t_T := 0.0050}$respectively.

Note that to visualize the diffusion-process of the solution u, all subplots are plotted against the *same* dynamic range. In particular, one observes the common z-axis value range among all three 3D surface plots.

The plotting routine is provided as a method of the dataset-provider class directly. An extra utility function is provided to plot the evolution of u as an animated film.

```
# src/pde/heat/dataset.py

class DatasetHeat(DatasetPDE2d):
    def plot_snapshots(self) -> None:
        ... # implementation detail not shown

    def plot_animation(self) -> None:
        ... # implementation detail not shown
```

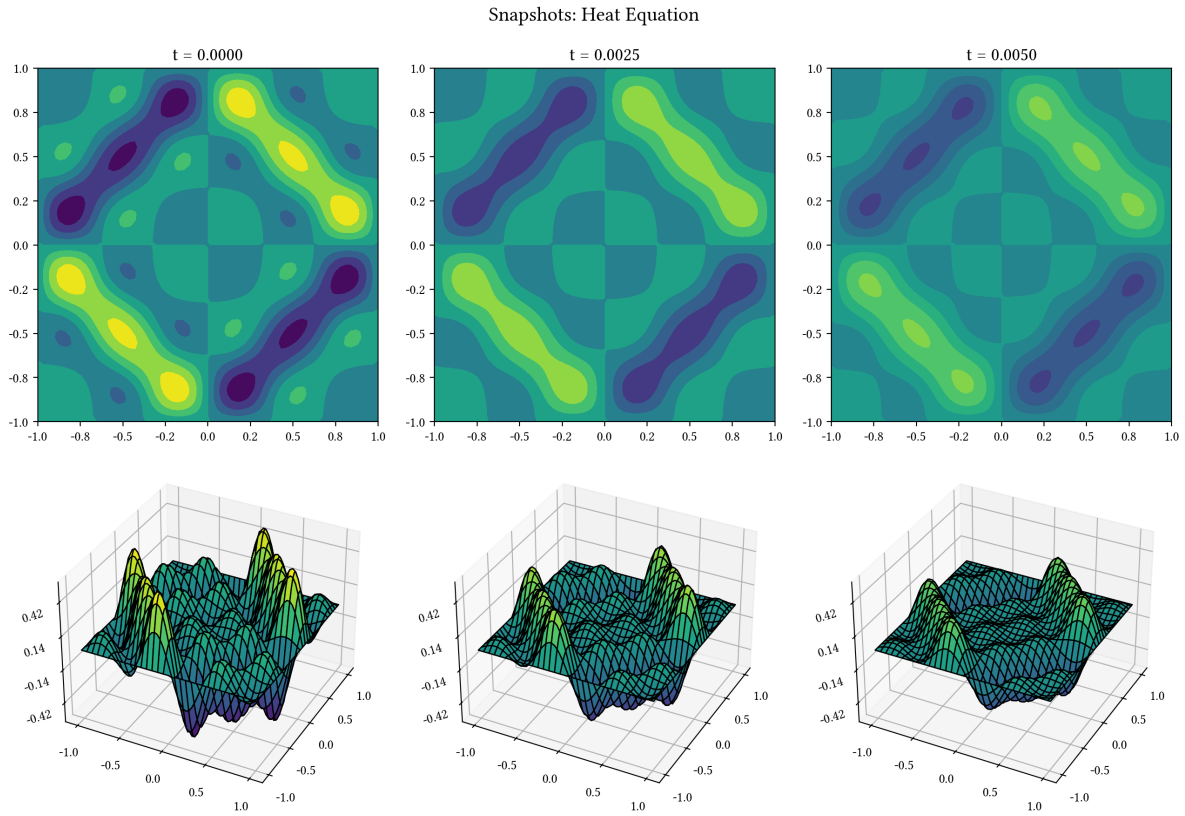


FIGURE 5.3. Visualization of the heat equation dataset, defined in Definition 13 and configured as specified in Definition 9. $c_k := 4$ samples are super-positioned to generate the instance u as shown. All figures are plotted using the same dynamic range.

5.4. **Wave Equation.** A fundamental PDE model to describe wave propagation through mediums, the wave equation is formulated as:

$$\begin{aligned} \frac{\partial}{\partial t^2} u &= c_c^2 \left(\frac{\partial^2}{\partial x_1^2} u + \frac{\partial^2}{\partial x_2^2} u \right) \\ \implies \qquad \ddot{u} &= c_c^2 \, \nabla^2 u \end{aligned}$$

The particular dataset for the wave equation as deployed in this thesis is defined as:

Definition 14. Dataset wave equation

Each instance of the solution u_i is calculated as:

$$u_i := \sum \sum \left\{ \left(\mathsf{R}_{k_1,k_2} \left(k_1^2 + k_2^2 \right)^{-c_r} \right) \cdot \cos \left(c_c \sqrt{k_1^2 + k_2^2} \pi t \right) \cdot \underbrace{\sin \left(k_1 \pi x_1 \right) \sin \left(k_2 \pi x_2 \right)}_{=: \underline{\sin}} \right\}$$

which is easily verifiable using the Trig-Laplacian property in Fact 1. Plugging in the initial time $t_o \equiv 0.0$, one obtains the following formulation of the dataset with the temporal boundary condition:

$$\begin{cases} u_i := \sum \sum \left\{ \left(\mathsf{R}_{k_1,k_2} \left(k_1^2 + k_2^2 \right)^{-c_r} \right) \cdot \cos \left(c_c \sqrt{k_1^2 + k_2^2} \pi t \right) \cdot \underline{\sin} \right\} \\ u_i^k \left(0, \forall x_1, \forall x_2 \right) := \sum \sum \left\{ \left(\mathsf{R}_{k_1,k_2} \left(k_1^2 + k_2^2 \right)^{-c_r} \right) \cdot \underline{\sin} \right\} \\ \underline{\sin} := \sin \left(k_1 \pi x_1 \right) \sin \left(k_2 \pi x_2 \right) \end{cases}$$

Configuration	Value
Space grid	$x_{1,2} \in [0.0, +1.0]^2$
End time	$t_T := 5.0$
Constant c^r	0.85
Constant c^c	0.1

TABLE 5.2. Default configuration of the dataset for the wave equation, seen in Definition 14.

Note that, once again, due to the presence of the double sin-product term, this dataset is codenamed »sum-of-sines«, just as the previous datasets for the Poisson and the heat equation.

The constants of the above dataset are configured as follows:

Example 10. Configuration of wave equation dataset

Table 5.2 enlists the values of the configuration options of the wave equation dataset above, leading to the following spatial boundary condition:

$$u_i^k \left(\forall t, \partial_{x_1}, \partial_{x_2} \right) := 0 \qquad \text{[Spatial Bound.]}$$

Similar to the construction of the heat equation dataset seen previously, the super-positioning routine is performed on $c_k := 4$ samples per instance. The weights of each sample are sampled from the usual uniform distribution spanning $[-1.0; +1.0]$:

$$\mathsf{R}_{k_1,k_2} \simeq \mathcal{U}\left[-1.0; +1.0\right]$$

Note 18. Visualization of dataset

Visualization of the dataset as specified as above is given in Figure 5.4. Following the plotting convention as established for the heat equation, the three columns depict snapshots of the solution u at the initial time $u_{t_0 \equiv 0.0}$, the middle $u_{t=2.5}$ and the final time $u_{t_T := 5.0}$ respectively. Note that to aide the visualization of the wave-like character of the solution u, all subplots are plotted against the *same* dynamic range. In particular, one observes the common z-axis value range among all three 3D surface plots.

Similar to the visualization mechanism of the heat equation dataset in Note 17, the plotting routine is embedded directly in the dataset class. However, an extra option is provided for the snapshot-plots: the reader is encouraged to run the function with non-uniform scaling for the snapshot subplots to be scaled separately.

```
# src/pde/wave/dataset.py

class DatasetWave(DatasetPDE2d):
    def plot_snapshots(self, uniform_scale: bool = True) -> None:
        ... # implementation detail not shown
```

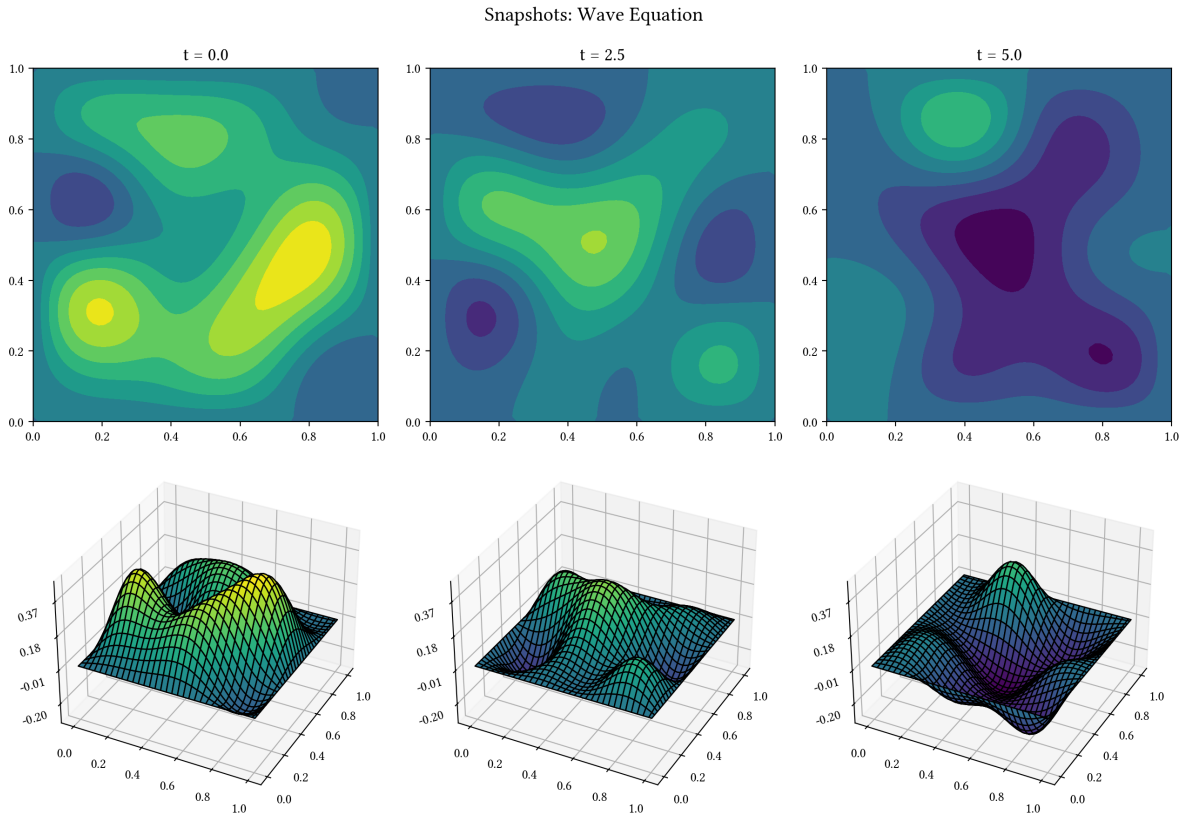


FIGURE 5.4. Visualization of the wave equation dataset, defined in Definition 14 and configured as specified in Table 5.2. All figures shown are plotted using the same dynamic range.

Equation	super-pos.	grid	t_T	key channels
Poisson (Sine) Poisson (Gauss)	$c_k := 4$	$[0.0, +0.63]^2$	$\varnothing$	solution u; source f
Heat		$[-1.0, +1.0]^2$	0.005	init $u_{t=0}$; end $u_{t=T}$
Wave		$[0.0, +1.0]^2$	5.0	

TABLE 5.4. PDE prototypes quick reference

5.5. **Summary.** To summarize this section on the PDE models used in this thesis, Table 5.4 enumerates the three equations discussed above, along with a quick overview of their key properties.

Thus concludes this introductory chapter on essential concepts used in this thesis: key concepts including spatial and temporal discretization is are revisited and formalized, masking as a concept is defined, and the dataset generation routine is laid out. In the upcoming chapter, three concretes PDE models are introduced in the context of masked dataset generation, before moving along towards the actual learning procedure and result evaluation.

Part 3. Models and Datasets

6. Network candidates

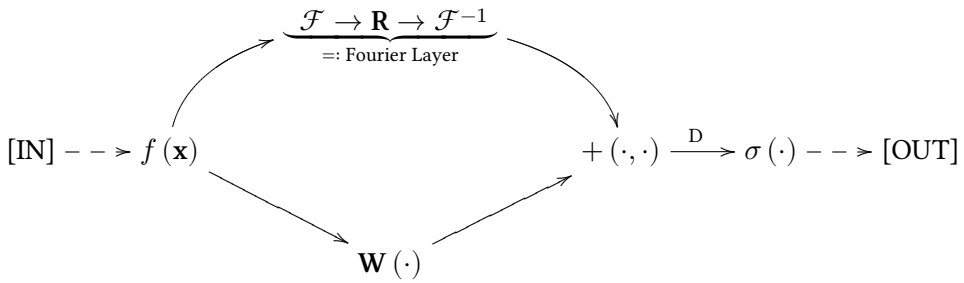
A critical component of any machine-learning pipeline is the underlying network model. This thesis deploys four networks, each with its own distinct design motivations and characteristics. This section enumerates these four networks and provides a quick overview of their respective properties and configurations as used in this project.

It should be noted that a thorough run-down of the individual algorithms is not the goal here; the interested reader shall consult the referenced papers for deeper investigation.

6.1. FNO.

Definition 15. Structural definition of FNO

Designed with the philosophy to model PDE operators directly in the continuous space, the FNO network [Li+20]; [Kov+21] approaches operator-learning with an attempt to parameterize and learn directly in function space. These goals are realized with an exceedingly simple core design, as conceptually captured by the following diagram:



FNO has since established itself with its wide application potential in the field, in particular, as a benchmarking reference for similar competing algorithms, as seen in publications such as [Rao+23]; [Xio+23a]; [Xio+23b]. Providing an execution acceleration up to three orders of magnitude compared to solver methods from applied mathematics, FNO boasts of other striking characteristics including resolution-invariance and mesh-independency, all the while escaping the curse of dimensionality [Li+20]; [Kov+21]; [BT23] due to the very nature of its Fourier operations.

Example 11. FNO Configuration

For the purpose of concept demonstration, this project provides a custom implementation of FNO in addition to including the official implementation by its authors [Li+20]; [Kov+21]. One striking characteristics of this personal implementation is the requirement of non-channel major dataset orientation, where the channels form the last axis of the dataset, serving the purpose of demonstration of this particular dataset structure and contrasting to the discussion in Section 5.

The official FNO implementation by its author(s) [Li+20]; [Kov+21] is shipped as a dependency of this project and used for systematic training for reference purposes, where:

(1) 3 Fourier layers are used, each with 64 channels;
(2) for both x_1 and x_2, 16 convolution modes are kept in the Fourier layers.

The unmentioned parameters inherit the default value as recommended by the author. The number of channels for input and output are naturally inferred from the concrete problem. For the single-masked Poisson Equation as introduced in Example 14, the number of channels are 8 and 1 respectively.

6.2. UNet.

Definition 16. UNet

Option	Value
#hidden-channels	16
#layers	4
kernel-size	3
depth	3
activation-fn	`relu`

TABLE 6.2. Configuration for the UNet network, introduced in Definition 16

Option	Value
#hidden-channels	16
#layers	4
kernel-size	3
depth	3
computation mode	`JIT`

TABLE 6.4. Configuration for the custom implementation of the CNO network, introduced in Definition 17.

Originally conceived for the purpose of efficient image segmentation, UNet [RFB15] is a convolutional neural network architecture particularly suitable for tasks such as biomedical segmentation with localization requirement for the underlying classification problem.

The UNet network consists of a contracting encoder and a expanding decoder, where the encoder follows the typical CNN architecture to repeatedly apply convolution with activation and max-pooling to achieve the downsampling effect. Afterwards, the expansive path performs upsampling coupled with convolution and cropping, before mapping to the desired output size with the final layer.

Note 19. UNet configuration

This thesis deploys a custom implementation of the UNet algorithm based on the original implementation of its authors. The default configuration of this implementation is provided in Table 6.2.

6.3. CNO.

Definition 17. CNO

A fairly involved network designed specifically for the purpose of PDE learning, the CNO network as introduced in [Rao+23] adapts traditional CNNs to preserve properties of the underlying PDE solution operators in continuous space to achieve equivalence between the discrete and the continuous spaces of the resulting model.

Based on the UNet structure as introduced previously in Definition 16, the CNO network structure begins with a lifting layer to expand the channel-count, followed by a series of consecutive upsampling and downsampling with activation. The final output is obtained via an iterative procedure with a projection operator.

Note 20. CNO configuration

This thesis deploys a custom implementation of the CNO algorithm adapted from the author's official repository[4] in the source module `src/deepl/cno.py`. Table 6.4 enumerates the default configuration of this implementation as used in the project.

[4] GitHub repository by CNO's authors

Option	Value
hidden-channels	64
layers	6
modes	4

TABLE 6.6. Configuration for the KNO network. Implementation details available in class `KNO2d` in source module `src/deepl/kno.py`.

6.4. **KNO.**

Definition 18. KNO

Designed to simulate and capture the behavior non-linear and possibly non-autonomous dynamic systems, the KNO [Xio+23a]; [Xio+23b], acronym for Koopman Neural Operator, aims to models these latent dynamics with a theoretically infinite-dimensional linear Koopman operator. Specifically, after linearizing the dynamics in some appropriate observation space of choice, a mapping shall be created to represent all possible observations of the underlying dynamic system, which is then iteratively updated in a linear fashion to capture the evolution behavior. KNO is verified by its authors in domains such as fluid dynamics and earth physics, providing a probabilistic approach with fast run-time.

Example 12. Implementation and Configuration

This thesis provides a custom implementation of the KNO algorithm as described above. The encoding and decoding steps are configured as concatenations of `pytorch`'s builtin convolution and `Tanh` operators. Further values chosen for network construction are listed in Table 6.6. The reader is encouraged to study the implementation of the `KNO2d` class in module `src/deepl/kno.py`.

6.5. **Network Factory.** The four algorithms as introduced above constitute the network pool of this thesis. A separate »factory« is written to provide a concise construction interface for all networks, configuring each to default settings reasonable for this project. The call-site is relieved of extra configuration details of the each individual network, enabling a clean abstraction for network construction: simply calling `Network.all()` returns a generator for the entire network pool of the current `LHS` and `RHS` channel widths requirement.

The network factory also contains utilities for automatic save-load management. The interested reader shall study the source for more details.

```
# /src/deepl/factory.py

class Network:
   @classmethod
   def all(
      cls, dim_lhs: int = 8, dim_rhs: int = 1
   ) -> Generator["Network", None, None]:
      yield cls.fno(dim_lhs, dim_rhs)
      yield cls.cno(dim_lhs, dim_rhs)
      yield cls.kno(dim_lhs, dim_rhs)
      yield cls.unet(dim_lhs, dim_rhs)

   def save(self, path: pathlib.Path) -> None:
      ... # implementation detail not shown
```

```
    def load(self, path: pathlib.Path) -> None:
        ... # implementation detail not shown
```

7. Learning Pipeline

This section builds upon the previous chapters on network models and PDE prototypes paired with the masking procedure to build the full learning pipeline from dataset generation to model evaluation. To commence, the masked dataset is introduced:

7.1. Masked Dataset.

Definition 19. Masked dataset structure

A masked dataset of size i contains i *instances*, where each instance z_i is composed of two channel-groups: the LHS_i and RHS_i group, acronyms for *left* and *right* hand-side respectively:

$$\text{Dataset} := \left\{ \begin{array}{rcl} \overbrace{}^{\text{instance}} & & \overbrace{\phantom{\mathsf{LHS}_i}}^{\text{ch. group}} \\ z_{i-1} & := & \cdots \\ z_i & := & \left\langle \begin{array}{l} \mathsf{LHS}_i \\ \mathsf{RHS}_i \end{array} \right. \\ z_{i+1} & := & \cdots \end{array} \right.$$

such that the prediction of some network NN as introduced in the previous chapter can be formulated as follows:

$$\begin{aligned} \mathrm{NN}\left(\mathsf{LHS}\right) &\longrightarrow \quad \widehat{\mathsf{RHS}} \\ \implies \qquad \text{error} &\simeq \left\| \widehat{\mathsf{RHS}} - \mathsf{RHS} \right\|_\star \end{aligned}$$

Note that the $\star$-formulation of the error-term is deliberately vague: this is studied more rigorously later in Section 7.2.2.

Both LHS and RHS channel-groups are a concatenation of multiple channels, which typically fall into two categories:

- those specific to the particular PDE model at hand, aka the »PDE-channels« $\mathbf{C}^{\mathrm{PDE}}$, including the solution u_i of the PDE itself,
- those common to the underlying space grid, the »grid-channels« $\mathbf{C}^{\mathrm{x}}$ such as the grid coordinates $x_{1,2\ldots}$.

The following subsections explain the constituents of these grid and PDE channels, before introducing the construction of the masked dataset for learning and evaluation.

7.1.1. *Grid channels.* One shall note that the coordinates matrices x_1 and x_2 in the proposition above bear no subscript to indicate index i. This is intentional, as the coordinates are constant across all instances of the dataset and require thus no re-computation during the generation procedure.

Definition 20. Members of grid-channels $\mathbf{C}^{\mathrm{x}}$

Channels related to the grid-coordinates $\mathbf{C}^{\mathrm{x}}$ as used in this project are composed of the following six components:

$$\mathbf{C}^{\mathrm{x}} := \underbrace{x_1, x_2}_{\text{coord}} \;,\; \underbrace{\sin\left(x_1\right), \sin\left(x_2\right), \cos\left(x_1\right), \cos\left(x_2\right)}_{4\times\ \text{trig(coord)}}$$

where the first two channels correspond to the raw coordinates of the (two-dimensional) spatial grid, and the ensuing four represent these coordinates after applying value-wise trigonometry operations $\sin(\star)$ and $\cos(\star)$.

As observed above, these grid-channels are constant for any specific spatial grid and can be saved as such via pre-computation.

7.1.2. *PDE channels.*

Definition 21. PDE-channels $\mathbf{C}^{\text{PDE}}$

Unsurprisingly, the PDE-channels are chiefly provided by the underlying PDE model. Table 5.4 in the previous chapter introducing the three PDE prototypes of this thesis provides a quick overview of their respective channels.

Recalling Listing 3, one recognizes that that the method

```
@abc.abstractmethod
DatasetPDE.solve_instance() -> Sequence[torch.Tensor]
```

serves as the key implementation detail of a PDE prototype to provide the PDE-channels relevant to the PDE at hand.

Definition 22. Masked and unmasked PDE-channels

All available PDE-channels $\mathbf{C}^{\text{PDE}}$ generated by the underlying PDE prototype are divided into two categories: those targeted for masking, and those kept »as-is« without masking.

This leads to two mutually exclusive sub channel-groups: the masking target-group $\mathbf{C}^{\text{PDE-m}}$ containing the channels $\{m_1, m_2, \cdots\}$ and the *r*aw group $\mathbf{C}^{\text{PDE-r}} := \{r_1, r_2, \cdots\}$:

$$\begin{cases} \mathbf{C}^{\text{PDE-m}} \cup \mathbf{C}^{\text{PDE-r}} := \mathbf{C}^{\text{PDE}} \\ \mathbf{C}^{\text{PDE-m}} \cap \mathbf{C}^{\text{PDE-r}} \equiv \varnothing \end{cases}$$

7.1.3. *Construction.* With the PDE and grid channels defined above, the masked dataset is constructed as follows:

Definition 23. Masked Dataset construction

Per the structure of the masked dataset in Definition 19, every instance of the dataset z_i consists of the $\texttt{LHS}_i$ and $\texttt{RHS}_i$ groups, defined as follows respectively:

The LHS channel-group is constructed as a three-fold concatenation:

- the target-group $\mathbf{C}^{\text{PDE-m}}$ after being masked with masks $\mathbf{M}_1, \mathbf{M}_2, \cdots$ for each of the channel $m_1, m_2, \cdots$ contained therein, denoted as $\mathbf{C}^{\text{PDE-m},\mathbf{M}}$;
- the raw group $\mathbf{C}^{\text{PDE-r}}$ (unmodifed);
- the grid-group $\mathbf{C}^{\text{x}}$ (unmodifed):

$$\texttt{LHS} := \left(\overbrace{m_1^{\mathbf{M}_1}, m_2^{\mathbf{M}_2} \cdots}^{\mathbf{C}^{\text{PDE-m},\mathbf{M}}}, \overbrace{r_1, \cdots}^{\mathbf{C}^{\text{PDE-r}}}, \underbrace{x_1, x_2, \cdots}_{\mathbf{C}^{\text{x}}} \right)$$

On the other hand, the RHS channel-group contains the *unmasked* target-group as »truth« for the network to learn:

$$\texttt{RHS} := \left(\overbrace{m_1, m_2 \cdots}^{\mathbf{C}^{\text{PDE-m}}} \right)$$

Definition 24. Single and double masks

A dataset constructed per Definition 23 with one channel in the target-group $\mathbf{C}^{\text{PDE-m}}$ is referred to as »single-masked«, and the target-group of a »double-masked« dataset contains two channels.

PDE	1st target channel	2nd target channel
Poisson eq.	u	f
Heat eq. Wave eq.	$u_{t=T}$	$u_{t=0}$

TABLE 7.2. Masking targets of the three PDE prototypes of interest

In this thesis, the mask used for the two channels of double-masked datasets are always configured the same, but the masked results are calculated via two separate calls of the masking routine introduced in Definition 8. Listing 4 demonstrates the syntax for performing such double masking:

```
# ...init some mask-object with strategy, intensity and value
chan_1_masked = my_mask.mask(chan_1)
chan_2_masked = my_mask.mask(chan_2)
```

LISTING 4. API calls for double masking

In particular, for the random-style mask, this ensures that the masking candidates are *recalculated* for both channels via sampling, while guaranteeing the *same* masking intensity (range) and value. On the other hand, for the island and the ring-style masks, this recalculation step equates to resampling the masking intensity, before calculating the masking candidates and setting them to the masking value. Note that no resampling is necessary if the »intensity-spread« is chosen as 0. See Section 4.3 for details.

Example 13. Masking target-group

It is up to the individual PDE provider to define an *ordered* sequence of channels as candidates for the masking target-group. For the three PDE prototypes studied in this thesis, the respective masking target channels are listed in Table 7.2.

Example 14. Masked dataset in action

Due to Table 7.2 and Table 5.4, an instance of the single-masked dataset for the Poisson equation on the 64^2 spatial grid thus follows:

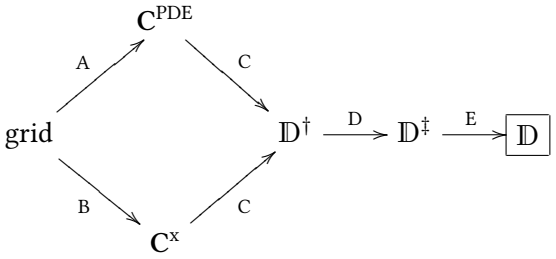
$$z_i^{\text{Poisson, single}} := \begin{cases} \mathsf{LHS}_i := \left[\overbrace{u_i^{\mathbf{M}}, f_i}^{\mathbf{C}^{\text{PDE}}} \;,\; \underbrace{x_{1,2},\, \sin_{1,2},\, \cos_{1,2}}_{\mathbf{C}^{\text{x}}} \right] & \simeq \Big(8, 64, 64\Big) \\ \mathsf{RHS}_i \simeq \left[u_i \right] & \simeq \Big(1, 64, 64\Big) \end{cases}$$

where the solution and source pair (u_i, f_i) is provided by the (raw) Poisson solver in Section 5.2. As another example, the double-masked Wave equation dataset is structured as:

$$z_i^{\text{wave, double}} := \begin{cases} \mathsf{LHS}_i := \left[\overbrace{u_{i,t=T}^{\mathbf{M}}, u_{i,t=0}^{\mathbf{M}}}^{\mathbf{C}^{\text{PDE}}} \;,\; \underbrace{x \cdots}_{\mathbf{C}^{\text{x}}} \right] & \simeq \Big(8, 64, 64\Big) \\ \mathsf{RHS}_i \simeq \left[u_{i,t=T}, u_{i,t=0} \right] & \simeq \Big(2, 64, 64\Big) \end{cases}$$

Definition 25. Masked dataset generation

The following diagram captures the dataset generation procedure:



In step A, the (raw) dataset generation routine is called upon to generate i instances of PDE-channels on the chosen spatial and optionally temporal grid, and the grid-related channels are fetched trivially in step B. These channels are then assembled in step C to form the first fully-shaped dataset $\mathbb{D}^\dagger$ in the sense of `pytorch` with the structure defined above. Note that at this point, the masking target channels in the `LHS` group are *not* masked.

In step D, the dataset is normalized. As previously hinted at in Section 4.4, this thesis applies the linear $[0, 1]$ normalization [Dev24]; [HTF09]; [GBC16] to obtain to obtain $\mathbb{D}^\ddagger$.

With all channels scaled within the desired $[0, 1]$ range, the masking procedure is applied to the masking target channels, in order to produce the final $\mathbb{D}$ primed for the learning pipeline. By applying the masking value v^{mask} from dataset-mean *post*-normalization:

$$v^{\text{mask}} := \text{Dataset}_\mu^{\text{norm}} \equiv 0.5$$

the masking procedure is guaranteed to be (mean-)neutral with respect to the output dataset $\mathbb{D}$.

7.2. **Prediction Model.** Given a dataset as prepared with the procedure in Section 7.1.3 and one of the networks as enumerated in Section 6, the prediction model of masked learning can finally be presented.

7.2.1. *Setup.*

Definition 26. Dataset splits

Following common etiquette of machine-learning [HTF09]; [Mur22]; [Was13], the datasets for training and testing are mutually exclusive. More specifically, a 6-to-1-to-1 split is used to generate the splits:

$$\begin{cases} i_{\text{train}} := 1800 \\ i_{\text{valid}} := 300 \\ i_{\text{eval}} := 300 \end{cases}$$

The testing dataset is unseen by the model during training and serves exclusively the purpose of evaluation to examine the adaptability of the trained models to generalize on data unseen during training without overfitting the training dataset.

A batch-size of 20 samples is used throughout all three aforementioned phases.

Note 21. A word on batch-size

It should be noted that the choice of batch-size has been a topic of major contention in the general milieu of machine learning, with researches including [ML18] recommending smaller values up to 32, and traditional textbooks such as [GBC16]; [Mur22] putting forward values up to hundreds.

The particular batch-size of 20 as chosen in this thesis is purely decided upon via empirical observation and intuition. The interested reader is encourage to rerun the project code with other values for experimentation.

7.2.2. *Single VS double-masked learning.* The single and double-masked datasets in Definition 24 warrant separate treatments of their respective prediction models. The case with the single-mask variant is fairly straight-forward:

Definition 27. Single-masked prediction

For the single-masked dataset, the loss is calculated directly as the error of the one channel in the RHS channel group. In this thesis, the L^2 error-norm is used.

Continuing with the single-masked Poisson equation dataset constructed in Example 14, the model prediction of some network NN on this dataset is formulated as:

$$\left[\overbrace{u_i^{\mathbf{M}}, f_i}^{\mathrm{C^{PDE}}} \;,\; \underbrace{x_{1,2},\, \sin_{1,2},\, \cos_{1,2}}_{\mathrm{C^x}} \right] \xrightarrow{\mathrm{NN}} \underbrace{\widehat{u}_i}_{\widehat{\mathsf{RHS}}_i}$$

Thus follows the loss function:

$$\begin{aligned} \text{Loss} &= \sum_i \left\{ \left\| \widehat{\mathsf{RHS}}_i - \mathsf{RHS}_i \right\| \right\} \\ &\overset{L^2}{\equiv} \sum_i \left\{ \left| \widehat{u}_i - u_i \right|_2 \right\} \end{aligned}$$

Definition 28. Double-mask prediction

Due to the presence of two channels in the RHS channel group for a double-masked dataset, the formulation of the error term must be adjusted accordingly. To commence, the dual-channel prediction of the double-masked wave equation dataset seen in Example 14 is formulated as:

$$\left[\overbrace{u_{i,t=T}^{\mathbf{M}}, u_{i,t=0}^{\mathbf{M}}}^{\mathrm{C^{PDE}}} \;,\; \underbrace{x \cdots}_{\mathrm{C^x}} \right] \xrightarrow{\mathrm{NN}} \underbrace{\widehat{u}_{i,t=T},\, \widehat{u}_{i,t=0}}_{\widehat{\mathsf{RHS}}_i}$$

The loss function as introduced above for the single-masked dataset is now enriched with the hyper-parameter λ to balance the two masked channels:

$$\begin{aligned} \text{Loss} &= \sum_i \left\{ \left\| \widehat{\mathsf{RHS}}_i - \mathsf{RHS}_i \right\| \right\} \\ &\overset{L^2}{\equiv} \sum_i \left\{ \left| \widehat{u}_{i,t=T} - u_{i,t=T} \right|_2 + \lambda \left| \widehat{u}_{i,t=0} - u_{i,t=0} \right|_2 \right\} \end{aligned}$$

In this thesis, the λ is chosen as:

$$\lambda := 0.7$$

Regime (intensity type)	$\mathbb{E}$	spread		min	max
Full (generic)	50%	50%	$\widehat{=}$	0%	100%
Low (specialized)	30%	10%	$\widehat{=}$	20%	40%
Middle (specialized)	50%	10%	$\widehat{=}$	40%	60%
High (specialized)	70%	10%	$\widehat{=}$	60%	80%

TABLE 8.2. Four configuration modes of masking for training

Part 4. Evaluation and Comparison

8. Evaluation methodology

8.1. **Dataset Construction.** The following proposition describes the evaluation procedure of the models in both the in-distribution and the out-of-distribution manner. First, the datasets for training are introduced:

Definition 29. Construction of training datasets

Every underlying raw dataset is masked in four modes:

- the full range, i.e., where the intensity is sampled from the entire intensity range: $[0\%; 100\%]$;
- three »specialized« modes, where the intensity spread is uniformly 10%, and the center (expected-value) of intensities are set at 30%, 50% and 70%. These three masking modes are designed to target the low, middle and high masking regime respectively.
- single and double-masked datasets are then created separately for each of these four masking modes, as prescribed in Definition 23.

A summary of the training masks are found in Table 8.2, and Listing 5 shows the creation of those masks as a generic `python list` object.

```python
# src/problem.py

class Problem:
   def __init__(...):
      ... # other init work

      self._masks_train = [
         MaskerRandom.from_min_max(0.0, 1.0),
         MaskerRandom.from_min_max(0.4, 0.6),
         MaskerRandom.from_min_max(0.2, 0.4),
         MaskerRandom.from_min_max(0.6, 0.8),
      ]
```

LISTING 5. Creation of training masks

Next, the datasets for evaluation are formed to allow both in-distribution and out-of-distribution testing:

Definition 30. Construction of evaluation datasets

- Masking strategies: Despite training on the random-style masks only, the models are evaluated on all three masking styles of this project introduced in Chapter 4.3: the random-style, the island-style and the ring-style.

- Masking intensities: for each masking strategy, the intensities are sampled from ranges spanning 10% increasing at a 10% step-size: $[0\%; 10\%]$, $[10\%; 20\%]$ until $[90\%; 100\%]$, translating to 10 masking ranges for each of the three masking strategies.

Construction of the individual datasets follows the same routine as described previously for the training datasets. In conclusion, of the three masking styles, »random«, »island« and »ring«, a total of 30 evaluation masks are created. These masks are then applied separately in both the single and the double-masked configuration seen in Definition 23.

8.2. **In-distribution VS Out-of-distribution.** A powerful measure to gauge the generalization capability of machine-learning models relies on *out-of-distribution* evaluation. While projects such as [Rao+23] adjust the raw (unmasked) datasets directly by changing the value(s) of the model parameters(s), this thesis exploits the tunable nature of masking itself. Recalling the masking procedure as introduced in Section 4, one observes that out-of-distribution evaluation can be achieved by a combination of the following two key configurable components of masking:

- the masking strategy,
- the masking intensities.

Note that the effect of tuning the masking value is studied separately later in Section 11.4. As an example, a model trained on 50% random-style mask could be evaluated with an out-of-distribution 90% island-mask. This is formalized as follows:

Definition 31. Out of distribution testing

In the case of masked PDE learning studied in this thesis, there exist two opportunities for performing out-of-distribution performance evaluation:

On the one hand, the masking strategy can be varied. For this thesis, since training is performed uniformly with random masks, evaluation performed on island and ring-style masks is thus out-of-distribution.

On the other hand, the intensity of the mask, i.e., its expected value and/or its spread, can be modified for the purpose of out-of-distribution evaluation. Recalling the training and evaluation datasets, one readily realizes that *all* evaluation masking intensities differ from those used during training, as the spread of the masks never fully agree. However, in the case of the three »specialized« masks of 10% spreads, two evaluation masks *overlap* completely with each of these training masks:

$$\begin{cases} \overbrace{}^{\text{train}} & \overbrace{}^{\text{eval}} \\ [20\%; 40\%] & \supseteq \begin{cases} [20\%; 30\%] \\ [30\%; 40\%] \end{cases} \\ [40\%; 60\%] & \supseteq \begin{cases} [40\%; 50\%] \\ [50\%; 60\%] \end{cases} \\ [60\%; 80\%] & \supseteq \begin{cases} [60\%; 70\%] \\ [70\%; 80\%] \end{cases} \end{cases}$$

This suggests that while the masking configurations during training and evaluation never fully agree, certain pairings of training and evaluation datasets are more »similar« to each other. In light of this, a natural follow-up definition of out-of-distribution evaluation requires:

- either a different masking strategy,
- or a non-overlapping masking intensity range

during training and evaluation to qualify as out-of-distribution. Specifically, the differentiation between in-distribution and out-of-distribution evaluation is now understood as follows:

- For the full-range intensity mask $[0\%; 100\%]$, evaluation is out-of-distribution if and only if the masking strategy is different, i.e., on island and ring style masks;
- For any of specialized intensity masks, evaluation is out-of-distribution if the masking strategy is different *or* if the masking intensity range during evaluation is not contained within that during training. As an example, for models trained on $[40\%; 60\%]$ range, evaluation using the $[40\%; 50\%]$ random mask is in-distribution, whereas an intensity range of $[70\%; 80\%]$ (of random mask) is deemed out-of-distribution.

Thus concludes the setup of model evaluation. In the following, analysis results from all possible training-evaluation pairings ordered by masking types and channels are presented. To commence, the results of the single mask from the full $[0\%; 100\%]$ intensity-range are studied:

9. Full intensity-range

Definition 32. Plotting convention

The errors graphs are presented in three horizontal splits, all plotting the L^2 relative error against the masking intensity. The three splits from left to right depict errors of the random-style, the island-style and the ring-style respectively.

Recalling that for the full intensity range studied in this section, the island and the ring style masks correspond to out-of-distribution evaluation. Thus:

$$\begin{cases} \text{plot}^{\text{left}} \mathrel{\widehat{=}} \text{in-distribution} \\ \text{plot}^{\text{middle} \cup \text{right}} \mathrel{\widehat{=}} \text{out-of-distribution} \end{cases}$$

As the lower baseline, a dashed line is drawn at 5% to indicate excellent performance; a hand-wavy higher baseline is chosen at 15%. The error plots depict error values up to 17.5%, and those beyond this threshold are truncated.

9.1. **Single masks.**

Example 15. Poisson equation

Figure 9.1 showcases the Poisson equation. One immediately observes the general trend that for every network, the errors steadily increase with higher masking intensity. This applies to both in-distribution and out-of-distribution evaluation modes.

Observing the in-distribution subplots on the left, one concludes that FNO, CNO and UNet all perform comparably: with errors starting at slightly above 5% at lower masking intensities, steadily increasing to approximately 10% at higher intensities. KNO's performance lags behind across all intensities, starting at 10% errors at lower intensities and closing at approximately 15% for intensities close to 90%.

For the out-of-distribution plots in the middle and on the right, one observes a visible bump of the error-curve of CNO at 40% island-masks (plot in the middle), as well as of UNet and FNO at high intensities for ring-style masks (plot on the right).

Crucially, by comparing the »cross-plot« performance among all three masking strategies, one recognizes that at similar intensities, the out-of-distribution errors are only moderately higher than the in-distribution error values: no drastic performance degrade is seen when evaluating the models on island-style or ring-style masks, although training was executed with the random masks. The more prominent exceptions here include FNO and UNet on the ring-style masks, which performed visibly worse in ring-style masks compared to the case with random masks.

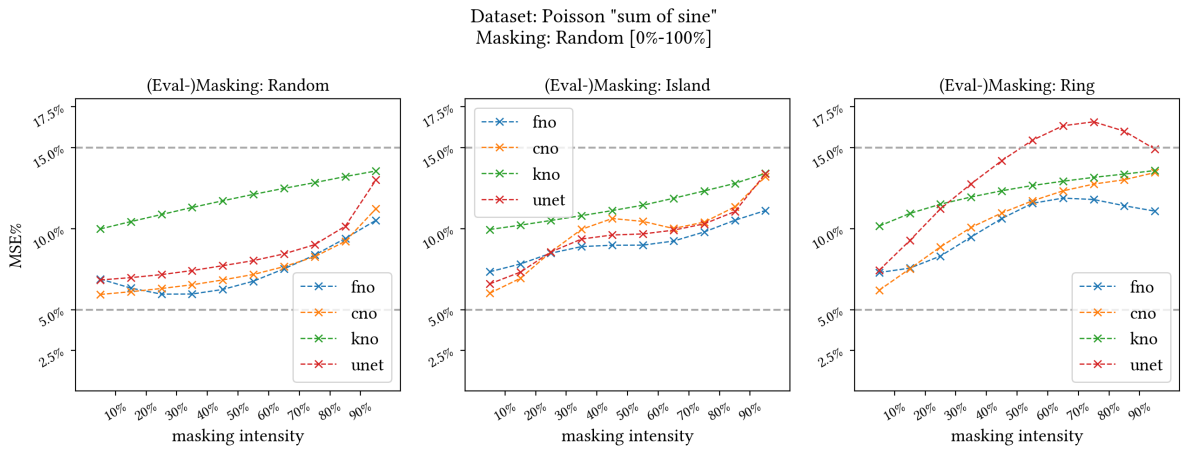


FIGURE 9.1. L^2 error-plot: Poisson equation; single mask, full range

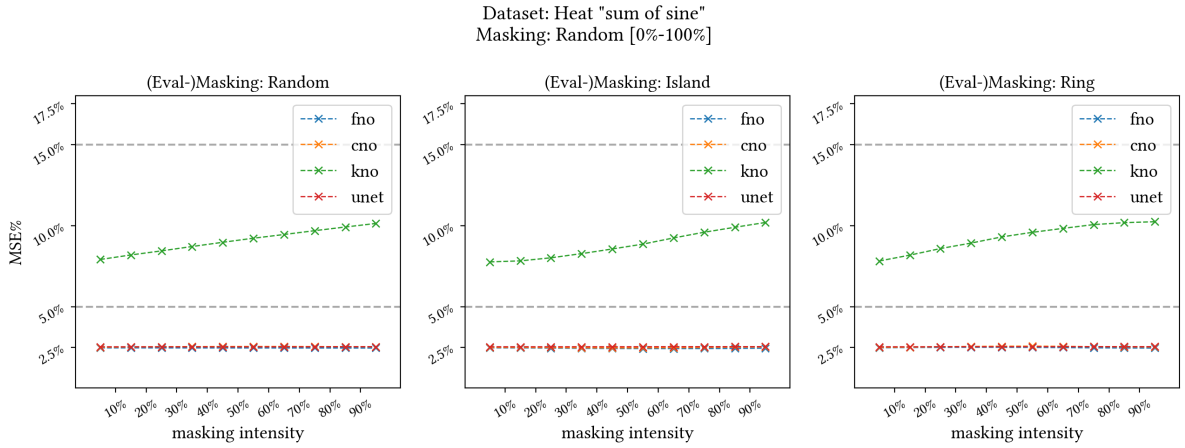


FIGURE 9.2. L^2 error-plot: heat equation; single mask, full range

Example 16. Heat equation

Figure 9.2 showcases the L^2 errors of the heat equation with single masks. Here, the performance is excellent for all models except KNO in *both* in-distribution and out-of-distribution evaluation, achieving approximately 2.5% error in all evaluation settings.

In fact, the error curves of the FNO, CNO and Unet models practically overlap and hold steady despite varying masking intensities or strategies. Although KNO performs visibly worse than the other networks, the errors still hardly ever surpassed 10%.

Example 17. Wave equation

The errors plots of the wave equation prototype are found in Figure 9.3. For the in-distribution plot on the left, all models except KNO perform very well, with errors lower than 2.5%. KNOs error values again surpass the 15% threshold at higher masking intensity levels.

Studying the out-of-distribution plots in the middle and on the right, one first observes the fantastic performance of FNO, holding at lower than 2.5% errors under all settings. For CNO and UNet, visible humps are displayed for high-intensity island-masks and for low-intensity ring-masks. However, the performance overall was still fairly decent at 7.5% for worst-case performance.

KNOs errors appear to always increase steadily with increasing masking intensity, exceeding the 15% mark with high-intensity masks with island and ring-style masks as well.

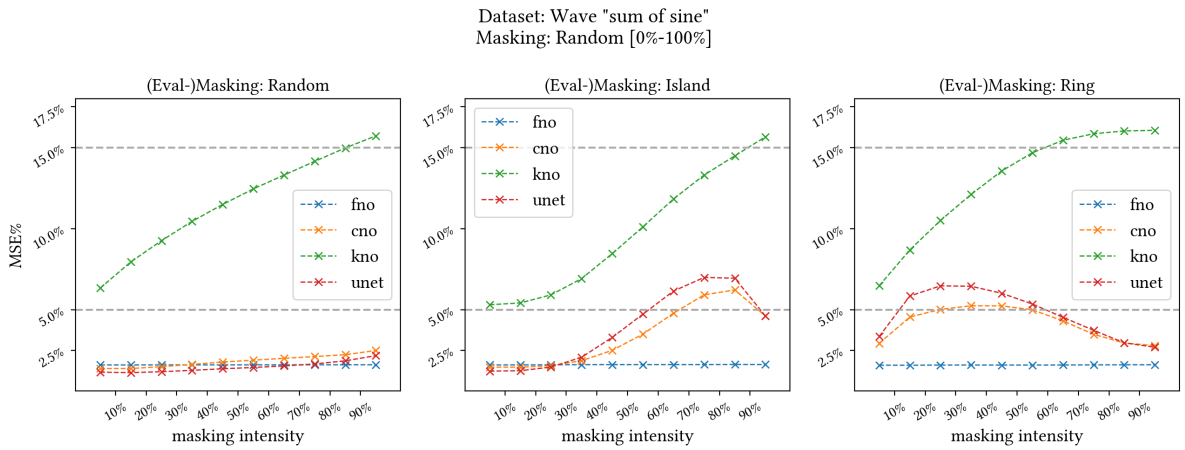


FIGURE 9.3. L^2 error-plot: wave equation; single mask, full range

Conclusion 2. Performance on full range masks

For the in-distribution evaluation mode, where both training and evaluation are performed with random masking, it is readily apparent that the errors increase steadily with higher masking intensities.

While the progression of errors evaluated on island and ring-style is generally unpredictable and indicates worse performance compared to random masking, the difference is not drastic in most cases. In fact, there exist cases, where models perform equally well in both in and out-of-distribution settings.

9.2. **Double masks.** With the preliminary result plots of the single-masks seen in the previous section, performance with double-masks is now studied. With an extra channel to analyze, the plotting convention requires some adaptations:

Definition 33. Plotting convention for double masks

Building upon the plotting convention for single-masking, plots for double mask contain two rows, with the top and the bottom row depicting the first and the second masking channel respectively.

It should be pointed out that, although masks of the same intensity and variant are used for both channels, the actual masking indexes are sampled separately for each channel, see Definition 24. This procedure applies to both the training and the evaluation dataset.

Additionally, the higher baseline is raised from 15% to 25% to cater for potentially more volatile model performance.

Now, the individual plots of the double masks models are shown. As usual, the analysis commences with the Poisson equation:

Definition 34. Poisson Equation

Figure 9.4 shows performance of the double-masked Poisson equation.

For the two in-distribution result plots on the left, one observes similar performance for both individual mask channels. FNO, CNO and UNet offer similar performance: the errors are kept under 10% for intensities values under 80%, and rise significantly for higher intensities. KNOs errors increase almost linearly with higher masking intensities, reaching almost 20% for masking intensities higher than 90%.

For both cases of the out-of-distributions scenarios, shown in the middle and on the right, one observes that the performance of the two channels are again similar, where the errors steadily increase for all models. It should also be noted that the models perform generally worse on the ring-type masks than those of island-style. Interestingly, KNO performs fairly well here, surpassing even all three other models with the ring-style masks.

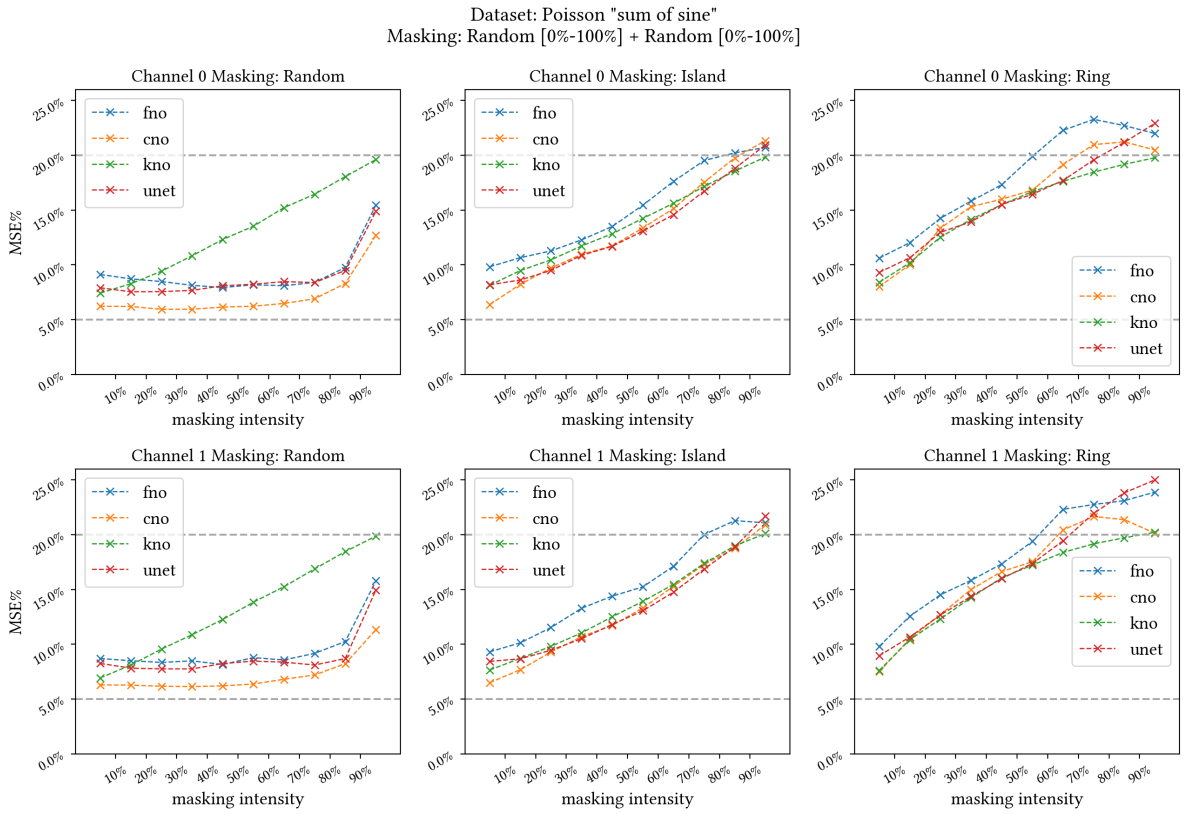


FIGURE 9.4. L^2 error-plot: Poisson equation; double mask, full range

Definition 35. Heat equation

Results for the heat equation are found in Figure 9.5.

For the in-distribution case on the left, the lower-regime errors are again very low for FNO, CNO and UNet, before rising sharpy for 90%. Again, KNO performs visibly worse than the other three models.

All four models display a steadily increasing, quasi linear error cursive for out-of-distribution evaluation scenarios, with the glaring exception of FNO: the out-of-distribution errors are maxed out for middle to high intensity masking ranges with both the island and the ring-style masks, significantly surpassing those of the other three networks.

Definition 36. Wave Equation

Figure 9.6 shows the double-masked wave equation of full intensity range.

In comparison to the performance with the heat equation perviously, one observes the significantly improved out-of-distribution errors with the island-masks. However, for the ring-style masks, the error values are still significant, overshooting the 25% mark on multiple occasions for FNO, and generally reading the 20% makr for all models in higher masking intensities ranges.

It is also noteworthy that the errors of the island-style masks are, again, slightly lower than the other out-of-distribution, ring-type masks.

Other observations, in particular though pertaining the in-distribution evaluation results shown on the left, are similar to those with Figure 9.4 for the Poisson equation, as well as Figure 9.5 for the heat equation previously.

Conclusion 3. Double masks, full range

- For any one particular masking type, performances of the two channels are fairly similar.

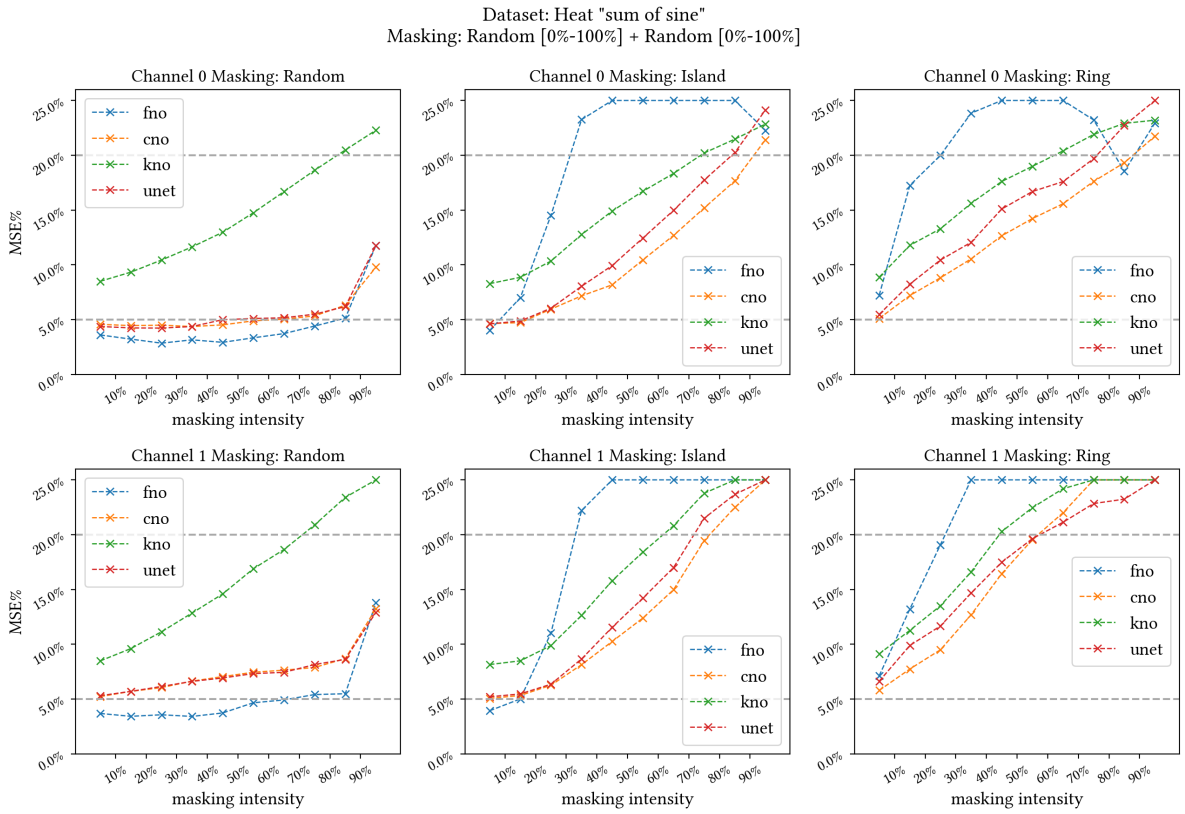


FIGURE 9.5. L^2 error-plot: heat equation; double mask, full range

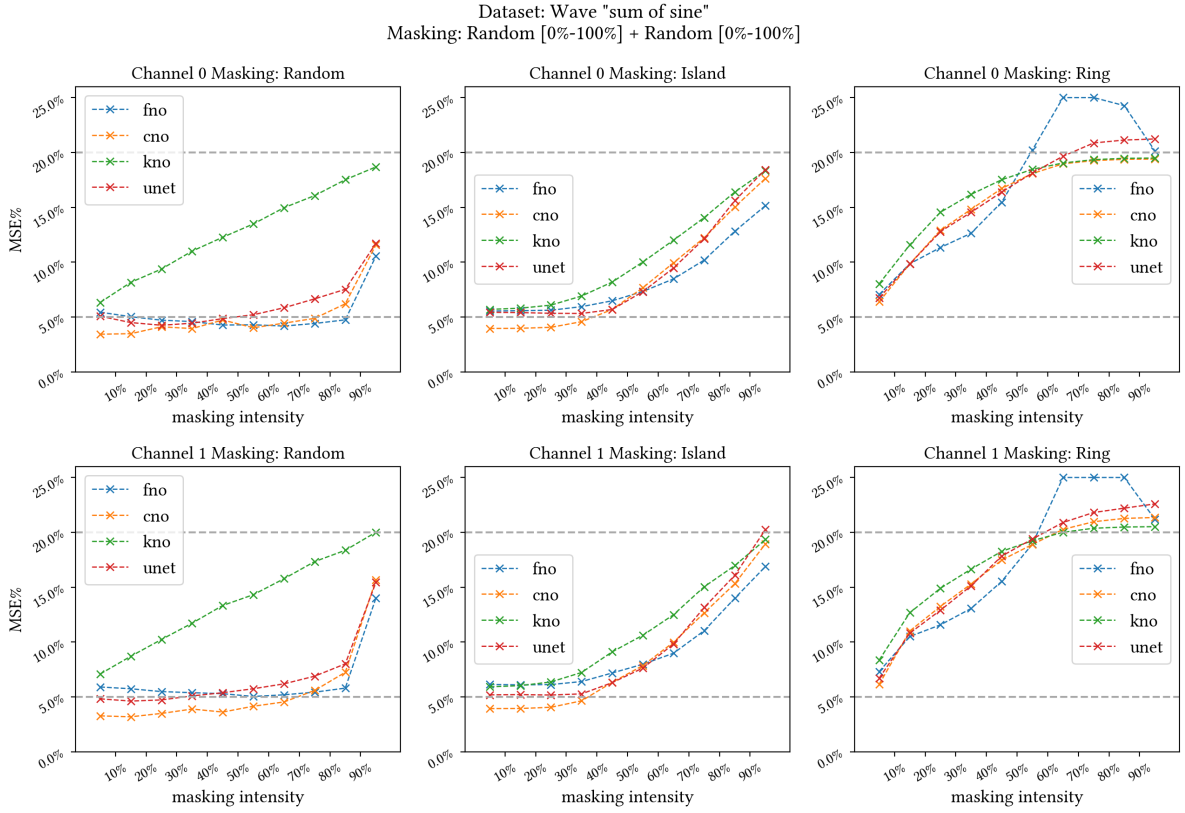


FIGURE 9.6. L^2 error-plot: wave equation; double mask, full range

- When evaluating with random-masks for in-distribution analysis, FNO, CNO and UNet peform well for masking intensities up to approximately 80%, after which a significant increase of errors is observed.
- While the errors with random masks are visibly smaller than those with island and ring style masks (out-of-distribution), the difference is more subtle among these two out-of-distribution masks. However, upon closer inspection, performance is generally better on the island-masks than the ring-style ones.
- While KNO performs visibly worse than other models in-distribution, its behavior is comparable to, sometimes even surpassing that of the other three models when evaluated out-of-distribution.

10. Specialized Ranges

After concluding the review of the models trained on the full-range $[0\%; 100\%]$, this section continues the performance analysis with the three specialized intensity-ranges enlisted in Table 8.2. One quickly recalls them as:

- low regime: intensity range $i \in [20\%; 40\%]$;
- middle regime: intensity range $i \in [40\%; 60\%]$;
- middle regime: intensity range $i \in [60\%; 80\%]$.

Continuing with the plotting conventions established previously, one observes the following additions to facilitate error analysis:

Definition 37. Visual aids for specialized intensity ranges

For the three specialized training ranges nominated above, a vertical shaded bar is added to indicate the masking intensity range used during training.

One recalls that, unlike the full-range masks studied in the previous section, it is now possible to achieve out-of-distribution even when evaluating with the random-style mask: as pointed out in Section 8.2, if the (evaluation) intensity lies outside the shaded portion of the error graph, the evaluation is out-of-distribution.

10.1. **The Poisson Equation.**

10.1.1. *Single Masking.*

Example 18. Poisson equation, specialized masks

To begin the analysis, results of the single-mask Poisson equation are shown in Figure 10.1.

One begins by observing the three plots on the *left-most* column, which depict the model performance using the same (random) style for evaluation as for training. With the usual exception of KNO, the striking phenomenon is the »shifting U shape«, with the low point perfectly matching the horizontal position of the shaded bars. This leads to the key property of »locality« of specialized masking ranges: if training and evaluation are both performed using the random masks, the closer the evaluation intensity range is to that of training, the lower the errors. This »locality« property, captured in Definition 38 with more details, contrasts sharply with the previous observation for the full intensity mask: there, the evaluation errors generally increase with *higher* intensity. Now, the locality effect leads to an increase of errors when evaluated with intensities both higher and *lower* than the training intensity.

In this particular example, this effect is magnified for the plot on the lower-left corner, where the model is trained on the high intensity regime of $i \in [60\%; 80\%]$: here, the errors are almost at the *lowest* for higher masking intensities, flipping the error curve almost completely to the steadily increasing error curve seen so far in full-range masks. such as in Figure 9.4. Also, for the middle intensity regime $i \in [40\%; 60\%]$, an almost perfect U-shaped error curve is shown for FNO, CNO and UNet, as seen in the middle plot on the left column.

Results for the out-of-distribution plots in the middle and on the right are a lot more sporadic, however, for the low-intensity regime on the first row, the errors increase linearly for all models with higher masking intensities. For the middle intensity regime, the general trend of the errors is still increasing with larger intensity values, albeit with a less inclined slope. Again, while KNO performs significantly worse than other models when evaluated on the random masks, its performance aligns fairly well with the other three for island and ring-style masks.

Definition 38. Locality of masking

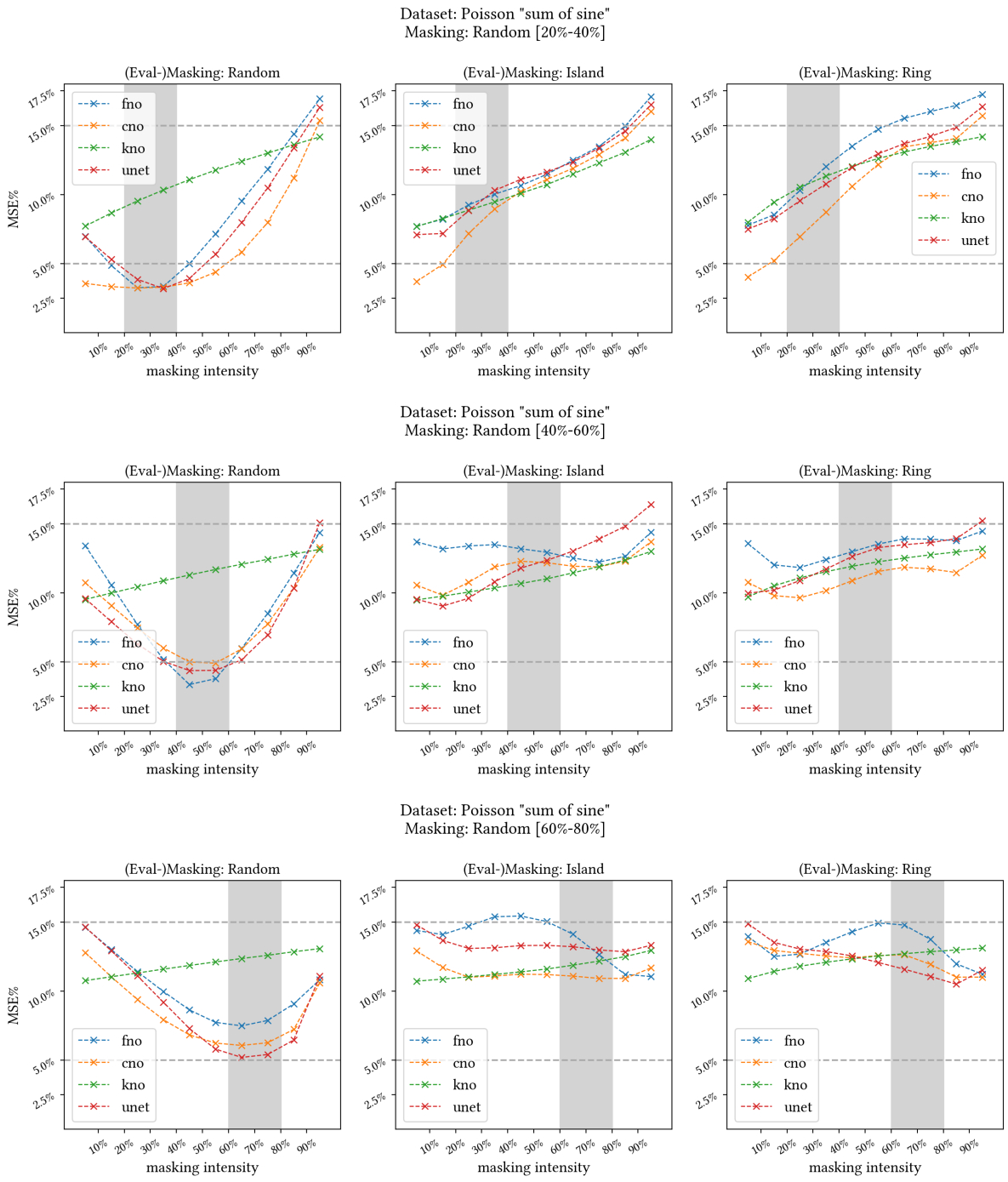


FIGURE 10.1. L^2 error-plot: Poisson equation; single mask, »specialized« intensity-ranges

When training and evaluation are both performed using the random-style masks as defined in Section 4.3.1, evaluation errors are generally lower if the masking intensity (or the intensity range) is closer to that used for training, increasing steadily with larger discrepancies between evaluation and training intensity. This leads to an instantly recognizable U-shape error curve, with the lowest error-position around the training intensity range.

This observation conforms well to the proposed differentiation between in and out-of-distribution discussed previously in Section 8.2: with both training and evaluation using random-masks, the in-distribution error is thus consistently lower than the out-of-distribution counterpart.

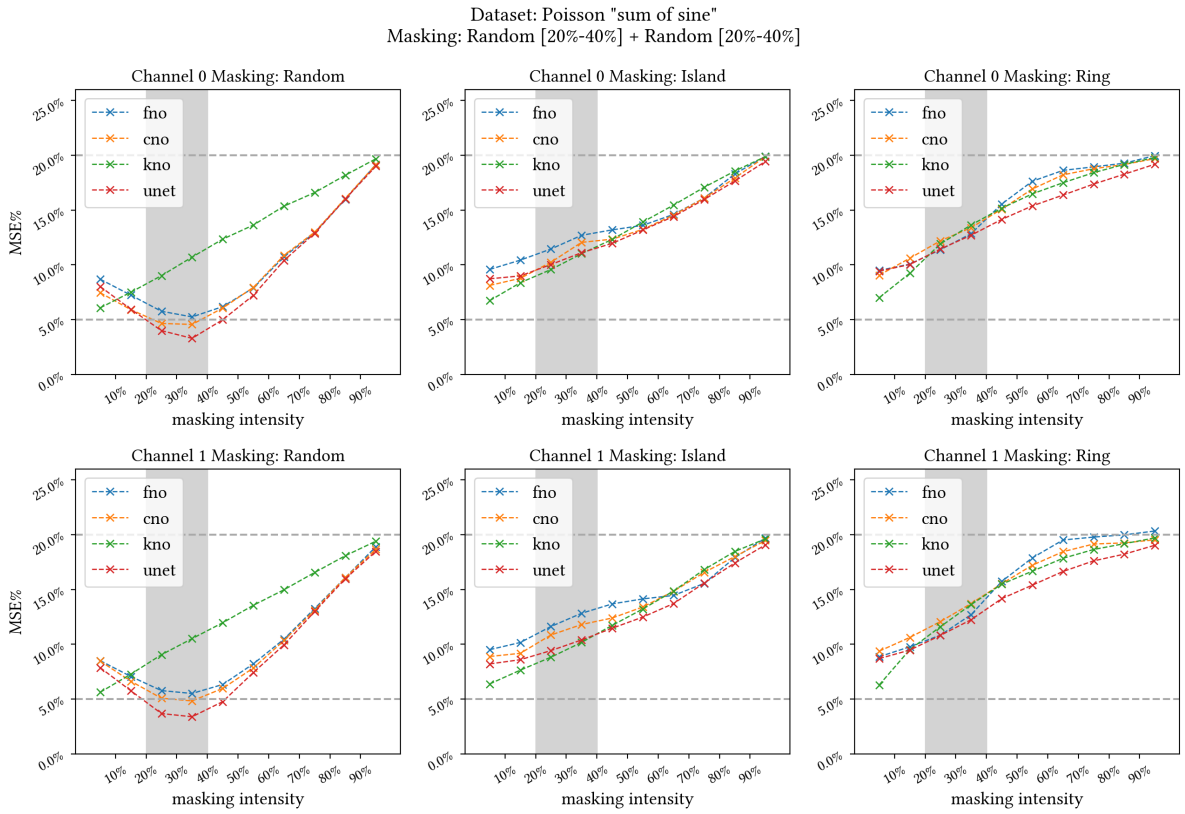


FIGURE 10.2. L^2 error-plot: Poisson equation; double mask, 20%–40%

With the first results of the models trained on specialized ranges seen in the previous section with the single-mask Poisson dataset, the performance of its double-masked counter-part is now studied as the natural extension.

10.1.2. *Double Masking.*

Example 19. Poisson equation, double-masking, low-intensity

Figure 10.2 shows the result of the double mask at $[20\%; 40\%]$.

Similar to previous observations of the full-ranged double-masked datasets in Conclusion 3, performance is fairly comparable among both dataset channels for each of the three masking styles, irrespective of the in or out-of-distribution evaluation nature.

The locality effect of the low-intensity masks used during training is clearly visible with the low errors in the shaded region and the clearly increasing errors for masking ranges higher than 40%.

For the out-of-distribution evaluation, the errors increase steadily with higher masking intensity. Also, the error appears slightly larger for the ring-style masks than the island-style, which also aligns with previous observations. Note that all models perform comparably in the out-of-distribution settings.

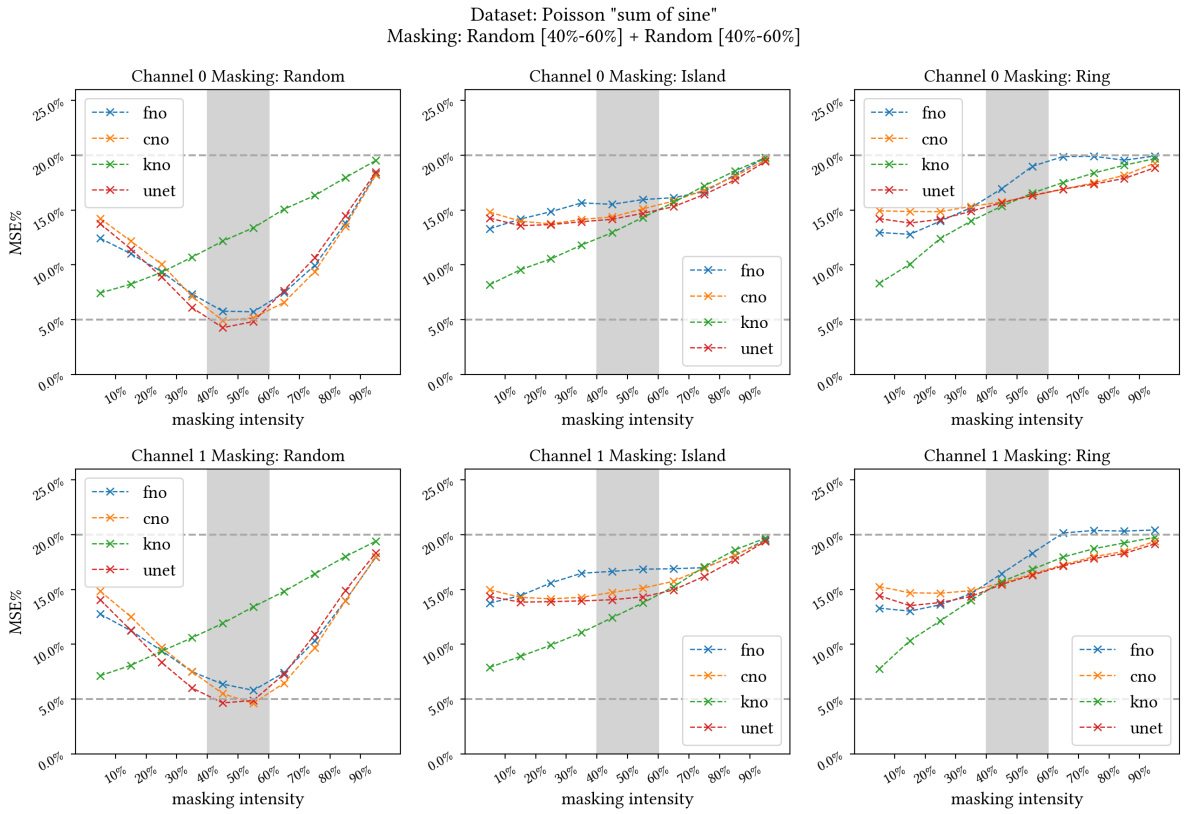


FIGURE 10.3. L^2 error: Poisson equation, double mask, masking intensity 40%–60%. Top row: first channel of double mask; bottom row: second channel of double mask.

Example 20. Poisson equation, double-masking, middle-intensity

Results of the four models trained with the double-masked, middle intensity range $[40\%; 60\%]$ are shown in Figure 10.3.

For the same masking type, i.e., same column, evaluation performance is similar for both channels, corresponding to previous observations.

The signature U-shaped error curve in once again visible for FNO, CNO and UNet in the two subplots on the left, showing errors close to 5% in the shaded region, corresponding to the in-distribution evaluation mode with the same masking type and masking intensity range as used during training. Interestingly, the unmistakable linear error curve of KNO leads to its outperforming the rest at lower intensity regimes.

Out-of-distribution errors generally increase with higher masking intensities, however, CNO and UNet once display some minor signs of locality awareness for the island-style masks shown in the middle: their errors only begin to increase when the evaluation masking intensities are higher than the $[40\%; 60\%]$ range used during training, indicated by the shaded area.

It should also be noted that in the out-of-distribution settings, KNO constantly out-performs all three other models, the difference is particularly noticeable in the lower intensity regimes.

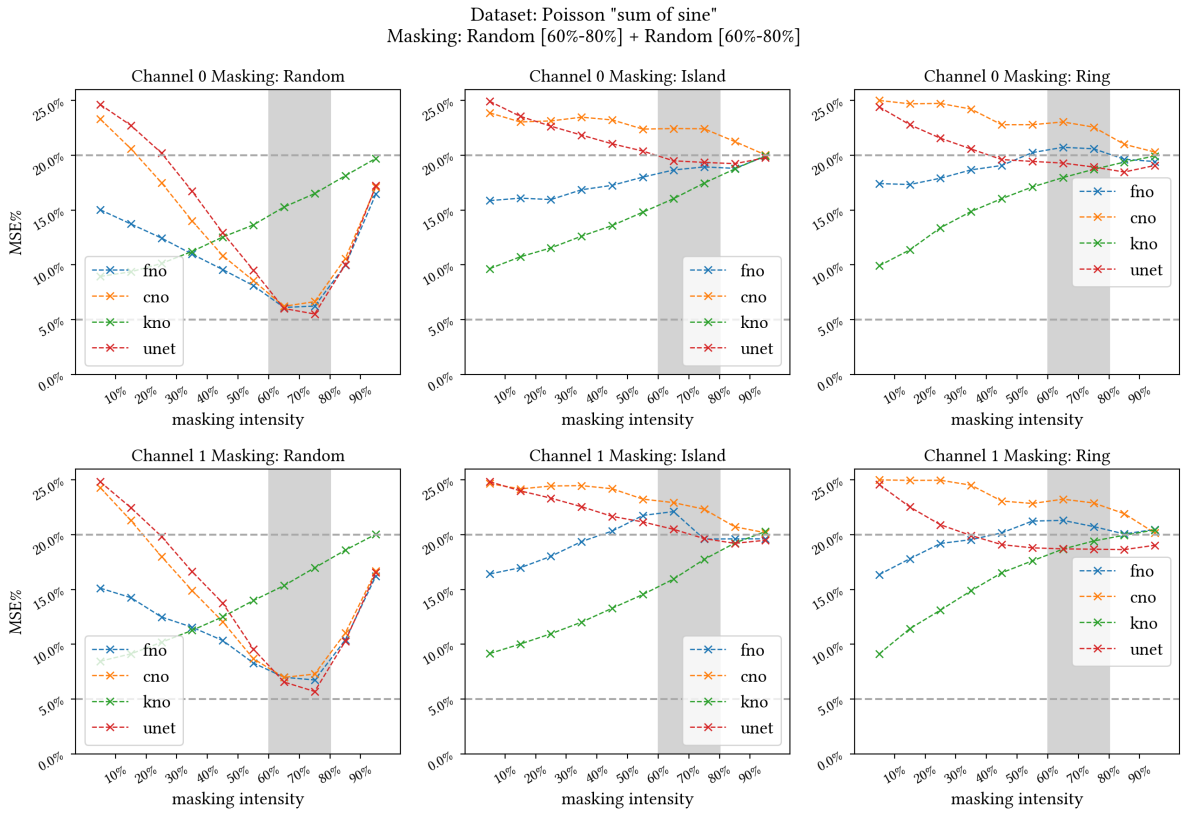


FIGURE 10.4. L^2 error: Poisson equation, double mask, masking intensity 60%–80%. Top row: first channel of double mask; bottom row: second channel of double mask.

Example 21. Poisson equation, double-masking, high-intensity

Figure 10.4 shows the result of the double mask at $[60\%; 80\%]$.

For in-distribution evaluation plotted on the left column, the skewed U-shaped error curves of FNO, CNO and UNet are again testament to the locality effect. Inside the shaded region, the three models perform comparably, with errors hovering slightly over 5%. However, outside portion, FNO performs significantly better than CNO and UNet, which is particularly apparent in lower intensity ranges. However, similar to the previous analysis, the best performer there is KNO, thanks to its linearly increasing error values.

The out-of-distribution errors curves in the middle and on the right are now more sporadic than the single-mask case. One notes that, in particular, UNet and CNO again appears to display locality awareness, as the errors decrease in the higher intensity ranges, as indicated by the shaded areas. Also, in these out-of-distribution settings, KNO manages to consistently out-perform all the other three models by a comfortable margin.

One shall note that the out-of-distribution performance is, in general, uninspired: with the rare exception of KNO on lower intensity regimes, the error values almost always surpass 15%. In fact, CNO and UNet both hardly managed to make the baseline of 20% for most of the evaluation scenarios.

Departing from the Poisson equation, one now observes the performance graphs of the heat equation:

10.2. **Heat Equation.** Figure 10.5 shows the L^2 performance graphs of the four networks trained on the heat equation dataset as defined in Section 5.3, post-processed with the random-style mask with intensity sampled from the range $[20\%; 40\%]$.

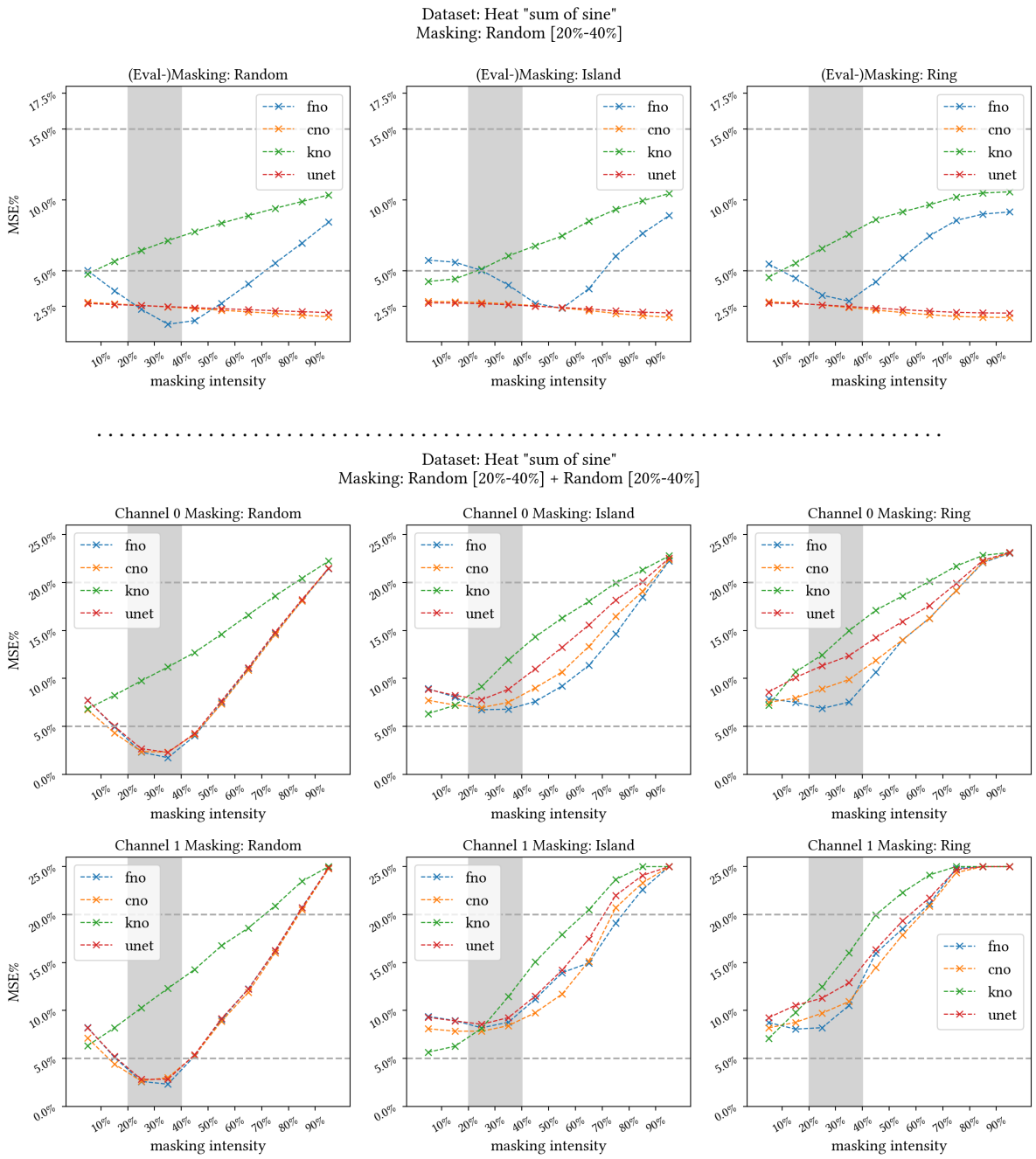


FIGURE 10.5. L^2 error-plot: heat equation, masking intensity 20%–40%. Top row: single mask; second row: first channel of double mask; third row: second channel of double mask.

The three plots in the first row depict errors of the single-mask problem. Here, the locality effect is almost completely absent, with CNO and UNet performing better in non-local, out-of-distribution settings, contrary to the locality theory. The double-masking scenario is represented on the two lower rows. In contrast, the locality effect is demonstrated perfectly there.

One also observes that the additional channel to learn (channel 1 in figure title) significantly changes the overall performance of the first channel (channel 0 as shown). In fact, even the in-distribution performance on the left bears no semblance between the single and double masking variants. Also, for evaluation using the island and the ring style with double masking, the errors increase steadily with higher masking intensities, with all four networks performing similarly to each other.

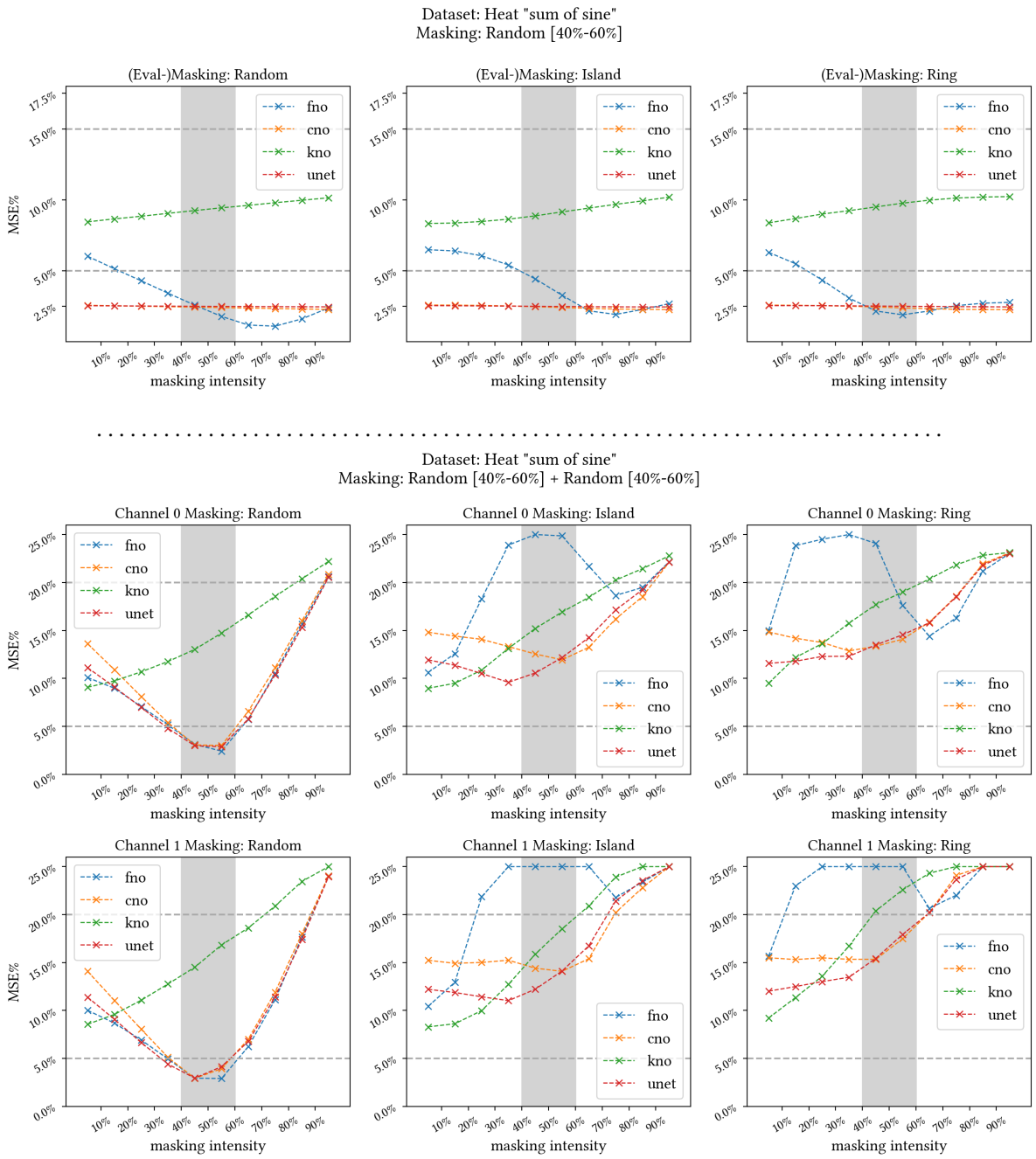


FIGURE 10.6. L^2 error-plot: heat equation; single and double mask, 40%–60%

L^2 errors of the four networks trained on the heat equation dataset as defined in Section 5.3 are presented in Figure 10.6. Random-style masking is deployed, with intensity uniformly sampled from the middle-strength range $[40\%; 60\%]$.

The typical U-shaped error curve is observed for the double-masking evaluation mode with random-style.

However, also striking is the almost unvarying error curves of all models in the single mask mode, in particular, CNO and UNet both perform admirably, with an almost constant error slightly exceeding 2.5%. In addition, error values of the double-masked, out-of-distribution settings appear to increase in general with higher intensities, howeve, FNOs error curve is still off-the-charts in multiple cases.

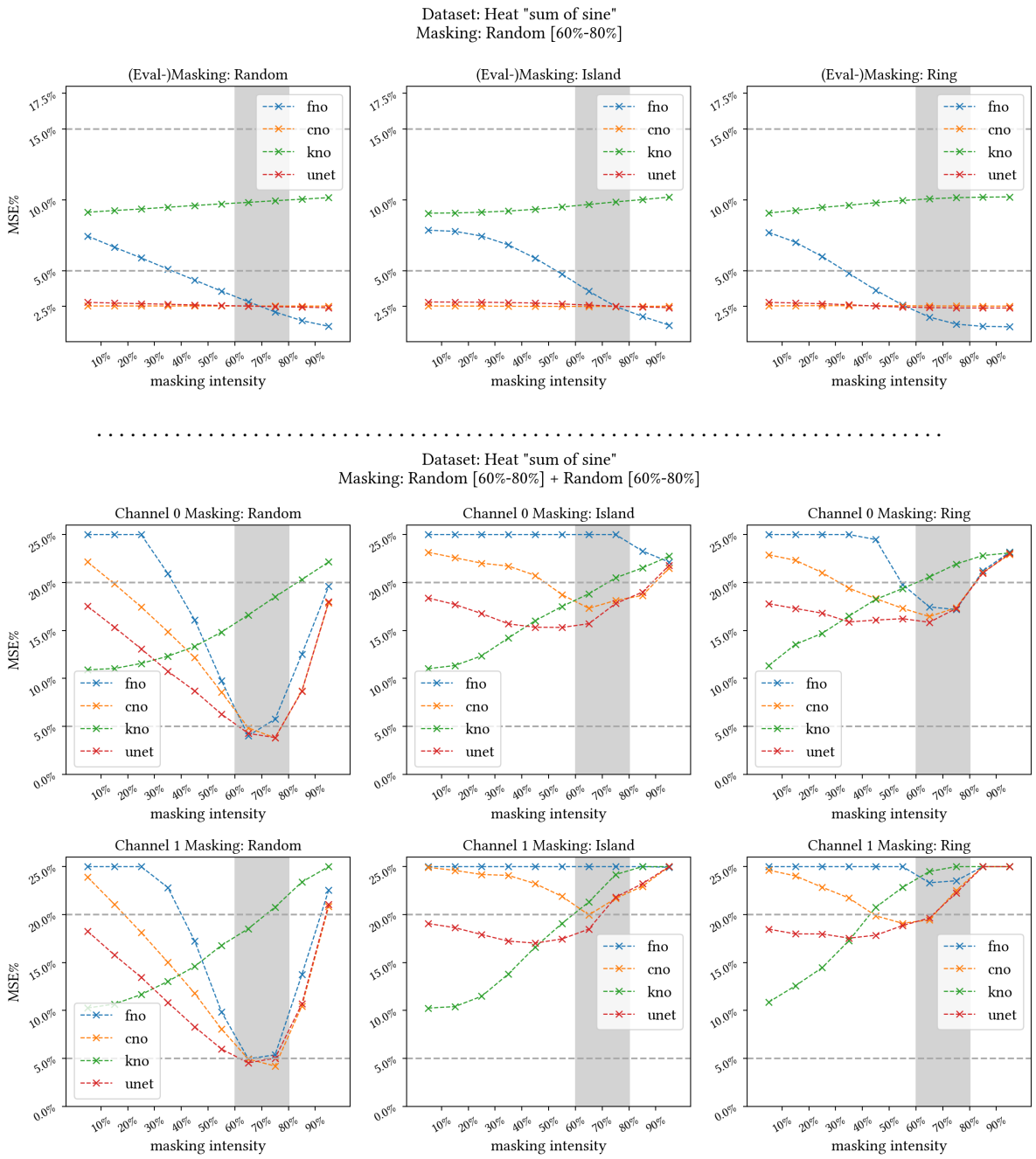


FIGURE 10.7. L^2 error-plot: heat equation; single and double mask, 60%–80%

Figure 10.7 shows the L^2 performance graphs of the four networks trained on the heat equation dataset as defined in Section 5.3, with random masking of $[60\%; 80\%]$.

While the locality effect of masking is visible for the two lower plots on the left, it appears to have little effect for the single mask problem shown in the upper left corner, corresponding to the single-masked settings. Interestingly, also for the single-mask problem, KNO's common trend of steadily increasing error appears have been »counter-acted« by the high masking intensity during training, leading to an almost horizontal error curve.

Similar to previous observation as seen with Figure 10.6, the single-mask problem sees almost constant performance for all models respectively, with excellent performance of CNO and UNet of 2.5% error throughout the entire intensity spectrum.

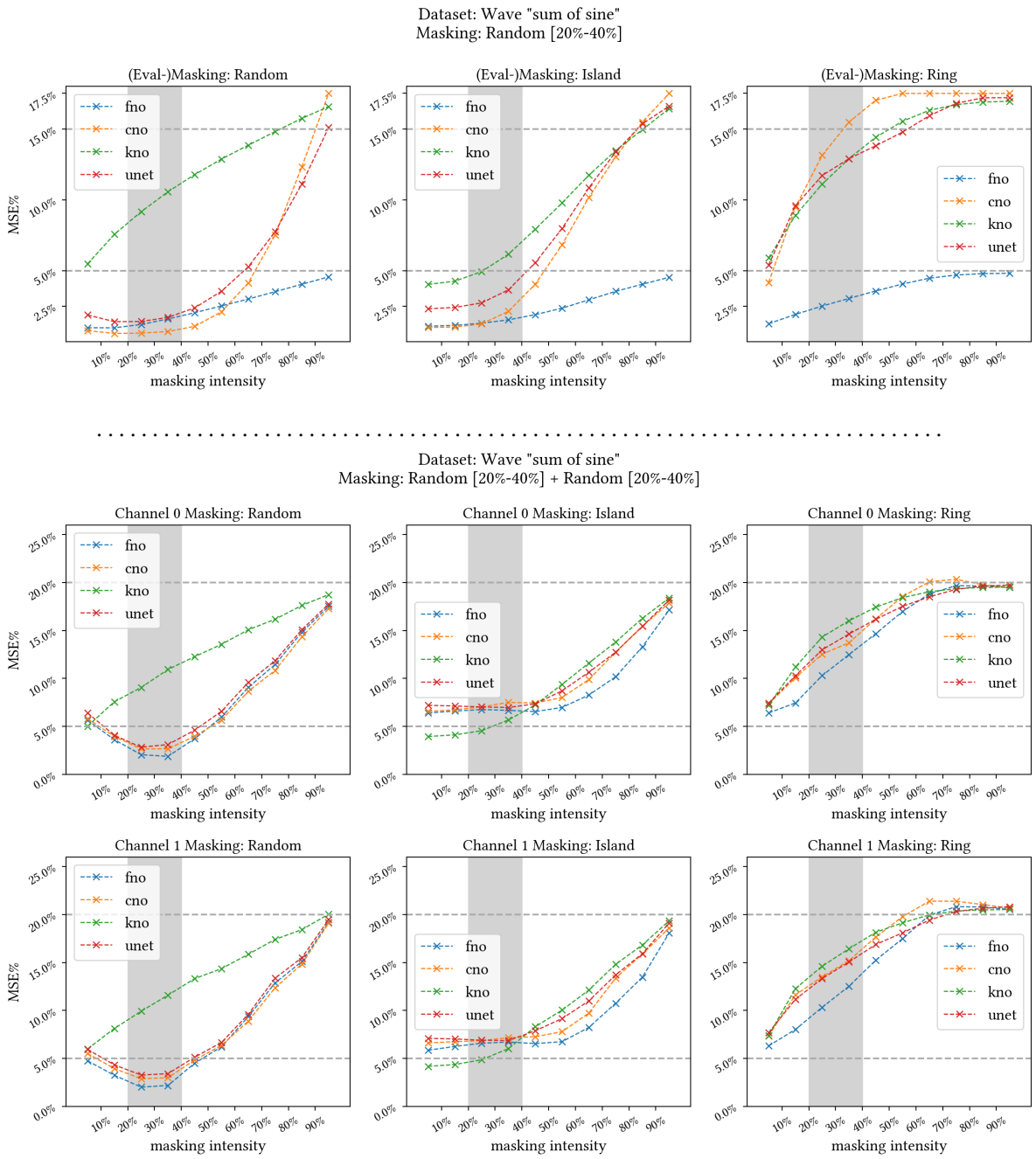


FIGURE 10.8. L^2 error-plot: wave equation; low intensity-regime, 20%-40%

10.3. **Wave Equation.** With the conclusion of the heat equation analysis in the previous chapter, one now studies the performance graphs of the wave equation.

Figure 10.8 shows the L^2 performance graphs of the four networks trained on the heat equation dataset as defined in Section 5.4, post-processed with the random-style mask of intensity sampled from the range $[20\%; 40\%]$.

The results follow the patterns observed previously: for the plots on the left, i.e., random-masks for both evaluation and training, the errors are lowest in the shaded masking range, conforming to the locality argument.

It should be pointed out that FNO performs exceptionally well for the single mask regime, with evaluation errors always below the lower baseline of 5% for both in and out-of-distribution evaluation.

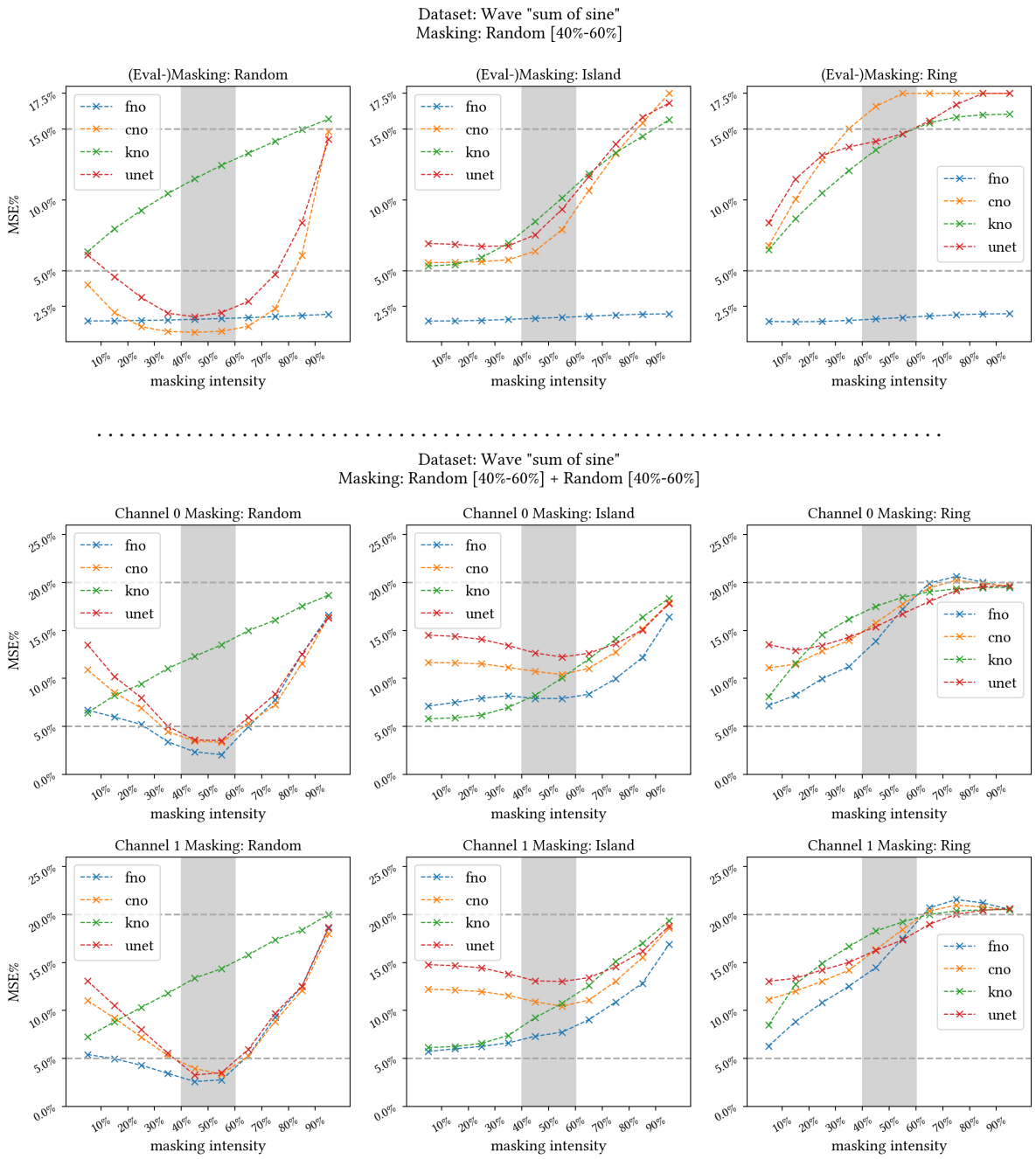


FIGURE 10.9. L^2 error-plot: wave equation; middle intensity-regime, 40%-60%

L^2 performance graphs of the four networks trained with the wave equation dataset are seen in Figure 10.9, with masks of intensity sampled from the middle intensity-range.

For the three plots on the left, the locality effect of masking is again visible with the unmistakable U-shaped error curve centered around the shaded region of $[40\%; 60\%]$ masking intensity.

For evaluation with the island and the ring-style masks, the errors generally increase with higher masking intensities. Interestingly, for the double-mask problem with the island-masks, both CNO and UNet perform best around a 50% masking intensities, indicating some semblance of locality-awareness even the masking type is different!

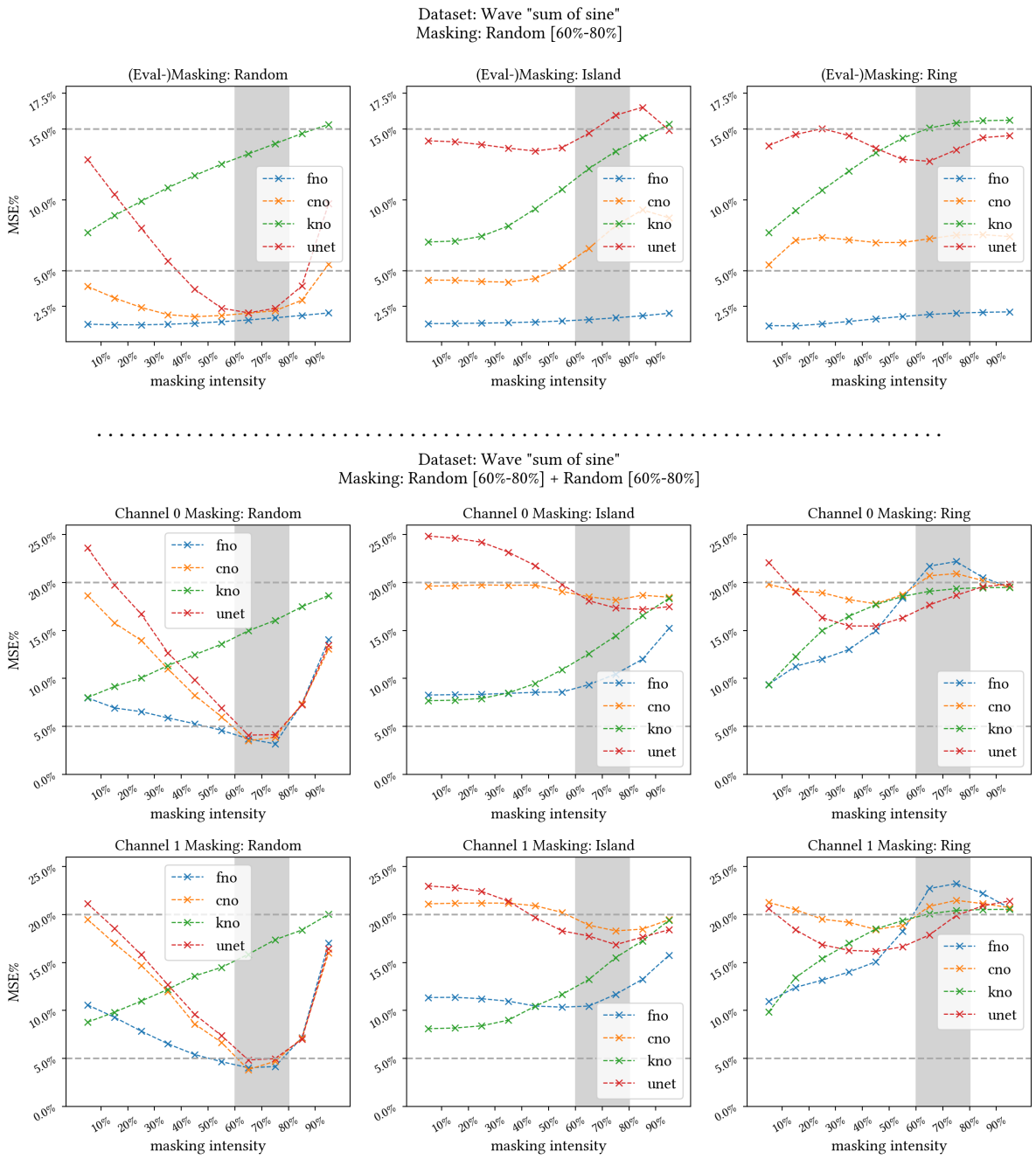


FIGURE 10.10. L^2 error-plot: wave equation; high intensity-regime, 60%-80%

Figure 10.10 shows the L^2 performance graphs of the four networks trained on the heat equation dataset as defined in Section 5.4, processed with the random-style mask of intensity sampled from the range $[60\%; 80\%]$.

Masking locality is seen with the considerable error decrease in the shaded area on the three plots on the left.

Again, FNO performed very well on the single-mask problem, delivering almost constant sub-2.5% errors in both in and out-of-distribution evaluation modes. In addition to this, progressions of the out-of-distribution errors with island and ring style are fairly sporadic. However, once again, CNO and UNet display locality-awareness for the double-masking problem using island-style masks (lower two plots in the middle), where the error dips around 70% to 80%.

11. Other Evaluations

11.1. **Dynamic remasking.** A special tactic designed to leverage properties of the masking PDE learning is used to regulate the training process. This technique, for reasons soon to become apparent, is referred to as »dynamic remasking«. This chapter explains the dynamic remasking procedure and provides evidence for its efficacy.

Critical to dynamic remasking is the state of staleness during the training phase:

Definition 39. Staleness during training

- A »stale« *epoch* is defined as a training epoch, after which the model's performance on the validation dataset did not improve. Conversely, the model's performance will have improved after a non-stale epoch.
- A *mask* is said to be »stale«, if a certain number of *consecutive* stale epochs have been accumulated. This number is a hyper-parameter of the training process and is set to 30 by default in this project. If any epoch yields improvement of model, the current number of consecutive stale epochs is reset per definition.

This leads to the definition of the dynamic remasking procedure:

Definition 40. Dynamic remasking procedure

During training, the dataset is *remasked* after every stale mask, i.e., 30 stale epochs by default. This value is also referred to as »remasking-frequency«. The remasking procedure applies freshly sampled masking candidates per Definition 1 and 25to all instances of the dataset. Note that the underlying, unmasked dataset is *not*changed during this process.

If no improvement is made after 3 consecutive remasking, the training process is terminated in advance. Otherwise, training is terminated at the expiration of the maximal number of training epochs, set at 1000 for this project.

The remainder of this chapter showcases the effect of applying dynamic remasking during training.

Definition 41. Plotting convention

Similar to the error-plots analyzed in previous chapters, the high and low baselines are indicated with dashed lines. The x axis of the plot now indicates the training epoch number during the training process. Vertical dotted lines indicate the epoch where a remask is triggered.

Example 22. Discovery of improvement

Figure 11.1 demonstrates the error-plot during the training process of the UNet network using the full intensity range $[0\%; 100\%]$ on the single-mask Poisson equation.

While the red triangle at epoch 22 marks the best model without remasking, the green stars mark all subsequent model improvements discovered after remasking is performed.

One observes that the network is improved on seven (7) separate occasions after executing the first remask. In particular, immediately after the first remask at epoch 53, a better model is discovered, which is subsequently improved upon in epoch 54. This is significant, as no improvement has been made during 30 consecutive training epochs prior to remasking.

Table 11.2 enumerates the MSE in percentage of all model improvements after remasking, alongside the respective improvement relative to the initial best model found without remasking. One observes the 25% decrease of MSE of the final best model discovered at epoch 91 compared to the original one found at epoch 22, a testament to the positive impact of dynamic remasking.

Note 22. Restrictions of dynamic remasking

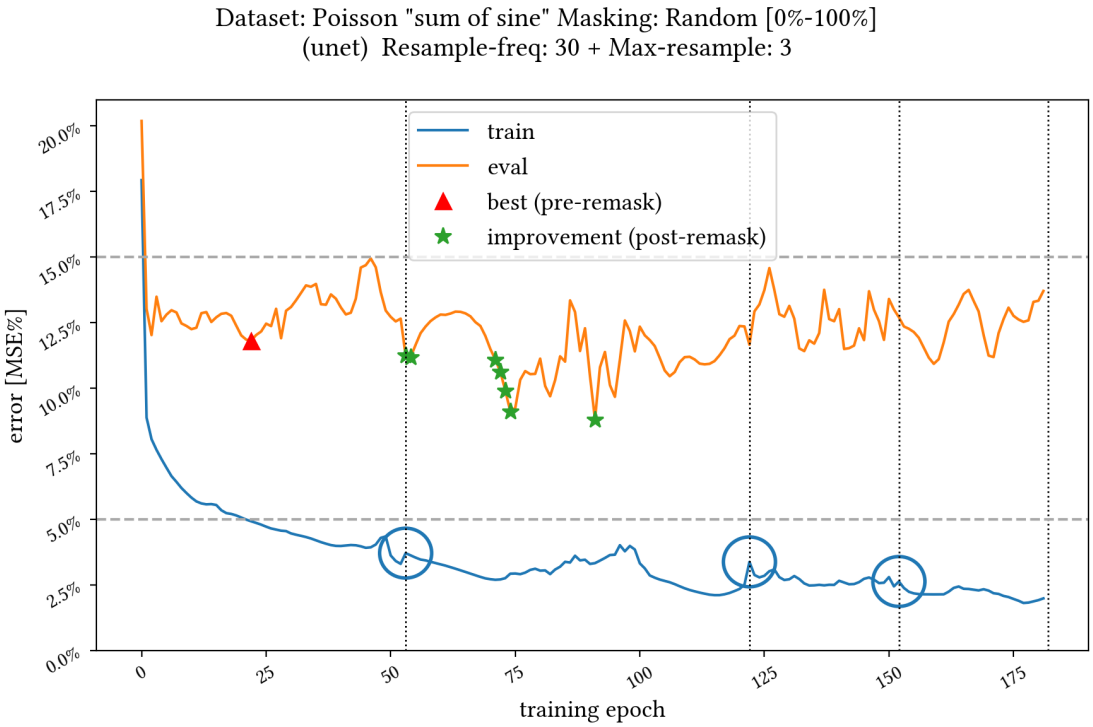


FIGURE 11.1. L^2 error-log of UNet during training. Remask epochs with vertical dotted lines. Observe the model improvements (green stars, cf. Table 11.2) discovered due to dynamic remasking, as well as the encircled blue bumps of training-error (cf. Table 11.4) immediately after every remask.

Epoch	MSE% (eval)	Rel. improvement
22 (pre-remask)	11.8167%	$\times 1.000$
53	11.2453%	$\times 0.952$
54	11.1899%	$\times 0.947$
71	11.0914%	$\times 0.939$
72	10.6254%	$\times 0.899$
73	9.9109%	$\times 0.839$
74	9.1208%	$\times 0.772$
91	8.8084%	$\times 0.745$

TABLE 11.2. Epochs of model improvement. Best before remasking: epoch 22, MSE 11.81%. Best after remasking (epoch 53): epoch 91, MSE 8.80%, aka relative improvement of $\times 0.745$ with remasking.

There are, however, situations where the benefits of the dynamic remasking strategy above diminishes significantly. One such example is seen with Figure 11.2, which depicts the training progress of KNO on Wave equation using masks of the full range, i.e., $[0\%; 100\%]$.

As discussed previously, KNO behaves in an easily recognizable manner: it does not seem to demonstrate any locality attachment to masking and performs generally worse in similar conditions in comparison to its competitors such as FNO. Figure 11.2 provides yet another curious insight to KNO: extended periods of continuous (but slow) improvement. This example shows KNO terminating only at epoch 888, almost exhausting the maximal training limit of 1000 epochs. Recall the previous examples of UNet

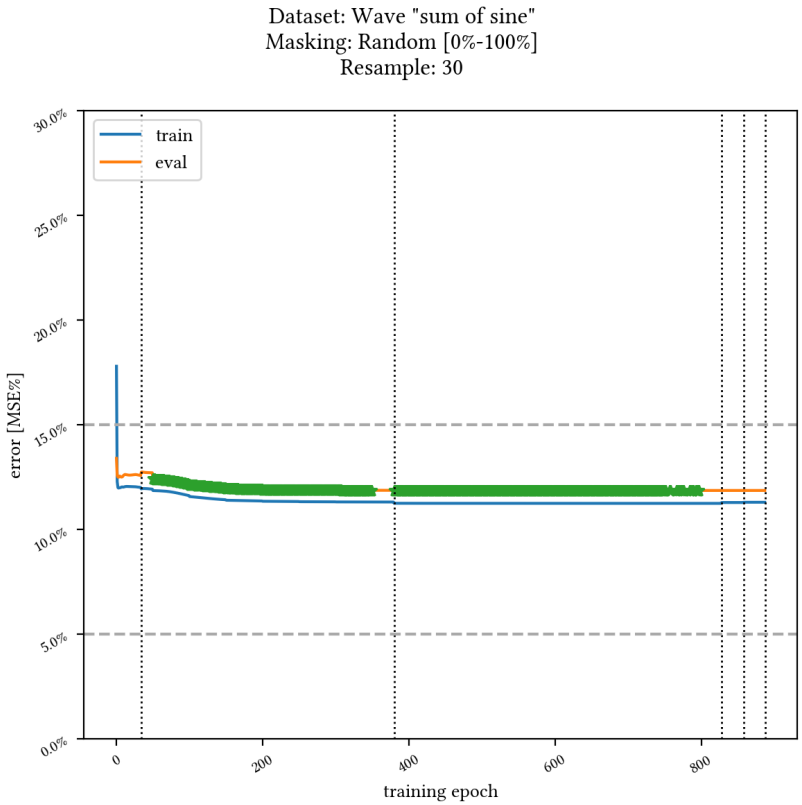


FIGURE 11.2. L^2 error-log of KNO during training. Due to the continuous (but slow) improvement of the network, the green $\star$ collapsed together to form two segments. Training performed on single-mask wave equation with intensity $[0\%; 100\%]$.

Epoch	MSE% (pre)	MSE% (post)	Rel. bump
53	3.3016%	3.7112%	$\times 1.124$
122	2.5086%	3.3767%	$\times 1.346$
152	2.4430%	2.6296%	$\times 1.076$

TABLE 11.4. Training error bump before and after every remask

finishing at less than 200 steps, one concludes that the training progress of KNO offers steady progress of improvement of model performance, although, ultimately, the final performance is comparatively weaker against its competitors.

Example 23. Post-remask error-bump

Another consequence of the dynamic remasking procedure is seen by observing the three encircled points on the training error-curve in Figure 11.1, indicating the training errors at the onset of every remask.

One observes a clear and immediate increase of the training error, following every remask. Table 11.4 enumerates these training-error »bumps« induced by dynamic remasking. This is intuitively understandable, as the remasked dataset imparts previously unseen knowledge to the model currently in training, leading to the immediate elevation of training error. However, since the underlying, unmasked dataset remains the same, the error bump remains modest.

The positive effect of remasking can now also be understood as the infusion of new information through the new masking pattern, allowing the model to expand on its current prediction capabilities to to potentially discover new optima.

Conclusion 4. Characteristics of dynamic remasking

The dynamic masking procedure as defined above positively impacts the overall learning pipeline.

Firstly, it serves as a sensible early-stopping criteria, allowing significant time and resource saving during the training step.

Additionally, it provide the model in question with a controlled amount of new information during training, allowing it to discover new improvements in continuation with the expansion of its current knowledge.

It should also be noted that the remasking routine might be considered configurable, but to determine its optimal values would potentially require highly time-consuming testing, as the inner loop would be the raw training procedure itself. Use-cases not warrantying such tuning might consider relying on the default configuration values of 30 stale epochs specified at the beginning of this chapter.

However, dynamic masking finds its greatest limitation in network models with slow, yet constant ongoing improvement. In such cases, the remasking criteria is almost never triggered and the functionality of this technique vanishes almost fully, equating approximately to running the training process without remasking at all.

11.2. **Direct visual inspection.** Directly comparing plots of the truth and the inferred prediction of the object(s) of interest plots is arguably one of the most visual and direct methods for evaluating model performance in machine learning [GBC16]; [Mur22]. In the settings of two-dimensional operator learning, such visualization is readily available, notably via the 2D contour and the 3D surface plots as used throughout this thesis.

This chapter provides inspections of the FNO network trained on the full-range singly masked Poisson equation. To start off, the in-distribution evaluation variant is given:

Example 24. In-distribution direct inspection

Figure 11.3 depicts the direct inspection options available: as seen with previous visualization-related plots in this thesis, the second row provides a 3D surface-plot to complement the 2D contour-plot of the same underlying matrix. The first three columns all depict the learning target, i.e., the solution u, specifically:

- the first column shows the raw, exact u, as obtained from calling the raw dataset generator;
- the second column shows the masked version of the solution object; here, the underlying u is remasked with a random mask of intensity in range $[50\%; 60\%]$;
- the third columns shows the model's prediction.

The forth column shows the (raw) difference between the »truth« and the model's prediction:

$$\text{Diff} := \text{Prediction} - \text{Truth}$$

The particular plot is generated with the first instance of the dataset, and the underlying FNO model performed at a relative MSE of 6.64%.

Now, the reconstruction results on the island and ring-style masks are shown:

Definition 42. Reconstruction, Island-style

Following the same plotting convention above, Figure 11.4 provides visualization of the reconstructed Poisson equation with island-style masking. It should be noted that the *same* FNO model is used to predict the *same* instance in dataset as seen in Figure 11.3 to allow direct comparison.

As shown, the model performance corresponds to a relative MSE of 10.30%. This confirms to the typical observation in Chapter 3: the »out-of-distribution« performance with the island-style mask lags slightly behind the »in-distribution« counterpart of the random-style.

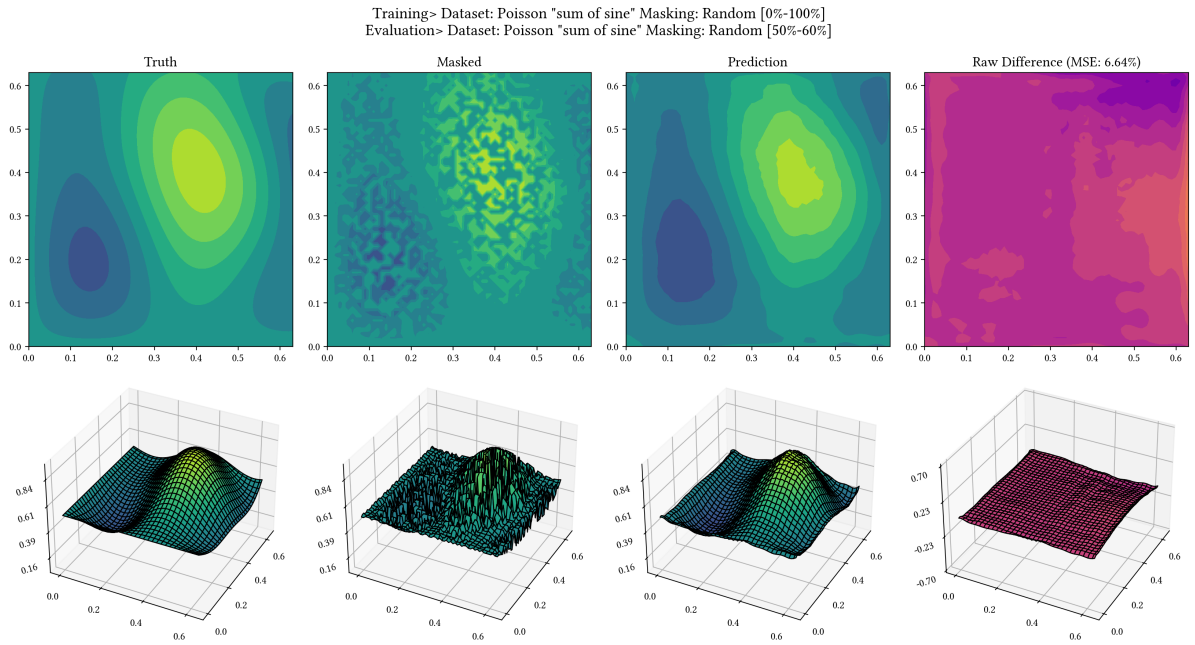


FIGURE 11.3. FNO's Reconstruction of Poisson equation with single mask of random-style. As shown: first instance of dataset; absolute MSE: 0.0013, relative MSE: 6.64%

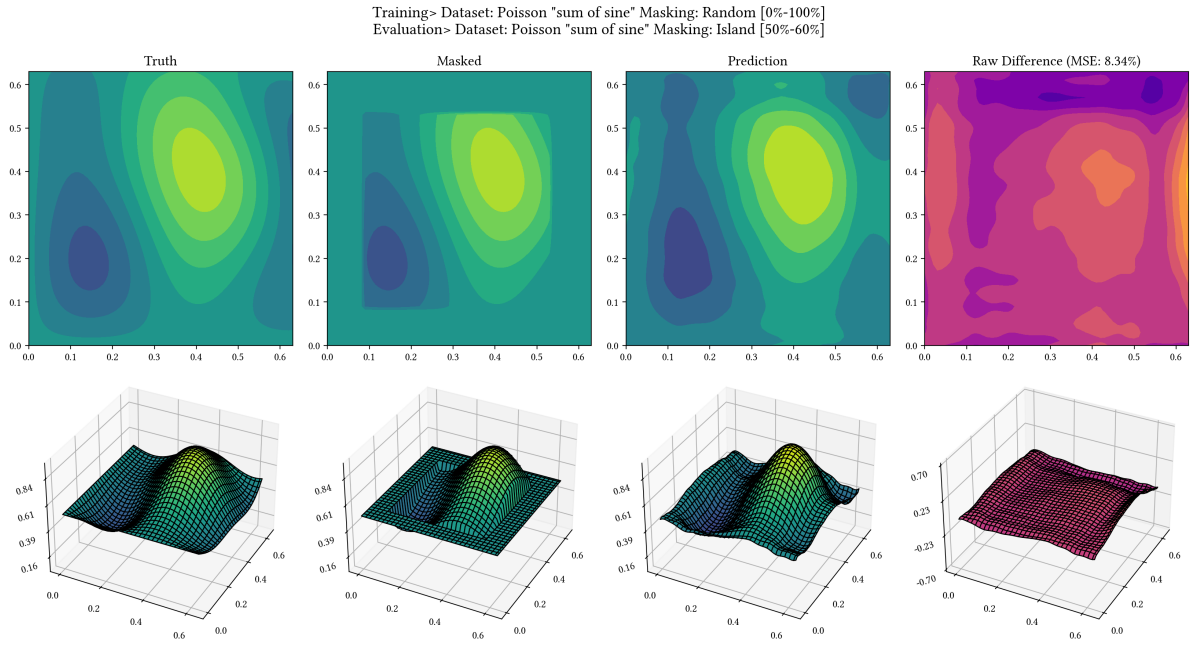


FIGURE 11.4. FNO's Reconstruction: Poisson with single mask of island-style. As shown: first instance of dataset; absolute MSE: 0.0021, relative MSE: 8.34%. Plotted with the same FNO as in Figure 11.3.

Definition 43. Reconstruction, Ring-style

Figure 11.5 visualizes the FNO reconstruction of the same Poisson equation instance now masked with the ring-style. The particular reconstruction corresponds to an absolute MSE of 0.0033 and 10.42% relative to input.

Note 23. Further reconstruction possibilities

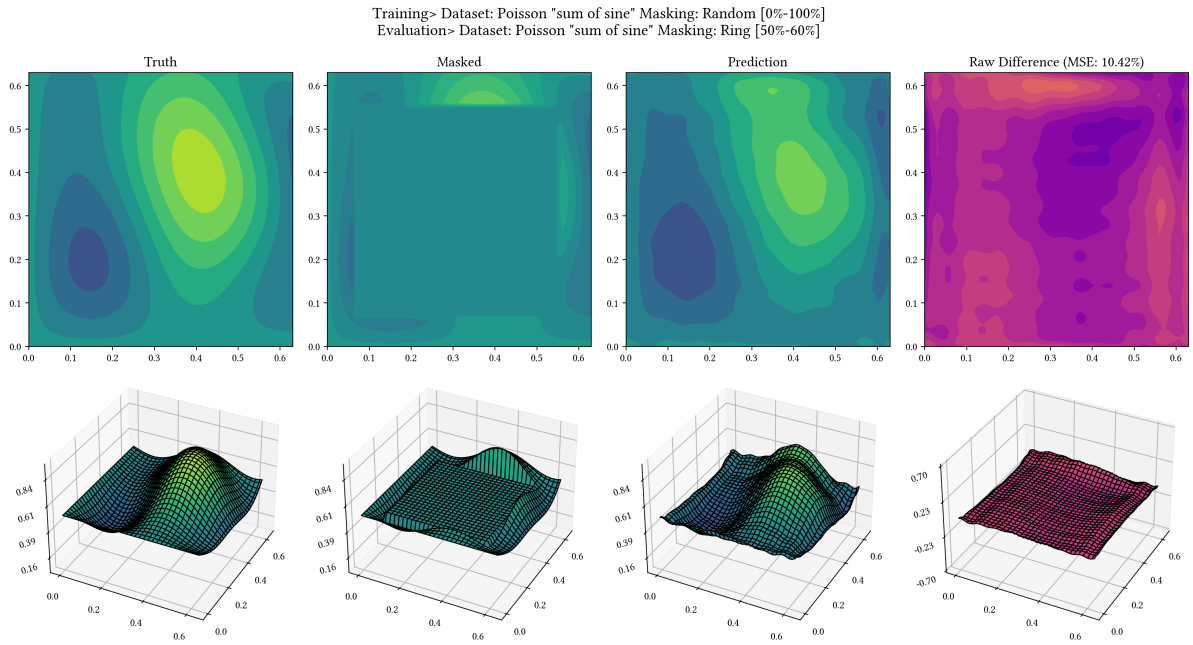


FIGURE 11.5. Reconstruction: Poisson with single mask of ring-style. Plotted with the same FNO as used in Figure 11.3 and Figure 11.4. As shown: first instance of dataset; absolute MSE: 0.0033, relative MSE: 10.42%

Apparently, the direct inspection technique above is performed on *one* particular instance of a masking dataset of *one* particular model. As such, this type of visualization is done on a »point-wise« basis and the interested reader is invited to review any trained model's performance with the plotting procedure above.

The plotting routine is implemented in the base `Problem` class:

```
# src/problem/

class Problem:
    def plot_reconstruction(self) -> None:
        ... # detail omitted
```

and is thus available to all three PDE models of this thesis for rapid visualization capability.

Intensity	island (in)	ring (out)	random (out)
$[0\%; 10\%]$	5.0452%	7.0281%	12.0592%
$[10\%; 20\%]$	4.8685%	11.8477%	17.9906%
$[20\%; 30\%]$	4.2850%	14.7517%	21.4952%
$[30\%; 40\%]$	3.3851%	17.7182%	23.3162%
$[40\%; 50\%]$	3.0814%	22.4464%	24.6729%
$[50\%; 60\%]$	3.2372%	23.3570%	25.2784%
$[60\%; 70\%]$	8.7907%	22.8414%	25.1982%
$[70\%; 80\%]$	17.9895%	21.4422%	24.8265%
$[80\%; 90\%]$	20.3517%	20.5185%	23.7969%
$[90\%; 100\%]$	19.9440%	20.5284%	22.0637%
Average	9.0978%	18.2480%	22.0698%

TABLE 11.6. Evaluation errors of CNO trained on $[40\%; 60\%]$ island-masks, evaluated with masks of island, ring and random-style masks respectively. In and out-of-distribution marked in column headers. Evaluation results using the *complementary* ring-style masks marked blue. Definition of the masks as provided in Section 4.3.

11.3. **The island-ring juxtaposition**. As hinted at initially when introducing the three fundamental masking types in Chapter 4.3, the island and the ring-style masks are defined as natural complements of each other. One recalls that, as pointed out in Note 7, the masked portion of an island-mask of intensity i corresponds exactly to the unmasked region of a ring-style mask of intensity $1 - i$.

This section studies the effect of this island-ring juxtaposition with the following task:

Definition 44. The island-and-ring problem

The model is trained on the island style, with masking range sampled from 50% as mean, with a 10% spread:

$$\text{mask-intensity} \simeq \mathcal{P} := \text{Unif}\left[c_{\text{low}} = 0.4; c_{\text{high}} = 0.6\right]$$

The evaluation datasets use all three masking styles with the usual masking intensity ranges of $[0\%; 10\%]$, $[10\%; 20\%]$ until $[90\%; 100\%]$.

Of course, of particular interest are the evaluation cases with ring-style masks of intensity $[40\%; 50\%]$ and $[50\%; 60\%]$, where the island and the ring style masks form complements of each other.

The same training procedure seen so far is applied to the four networks models in Section 6.

Example 25. Island-and-ring problem with CNO

Table 11.6 documents the evaluation error of a CNO network with the island mask (in-distribution), as well as the ring and random masks (out-of-distribution). One observes an average in-distribution error of 9.1%, and 18% and 22% out-of-distribution error rates for the ring and the random style respectively.

Although the result is slightly disappointing in terms of raw error values, the reader should consider the implication of these results above: the model trained with the island-style masks saw only the center portion of the data channel during training. The out-of-distribution evaluation with the ring-style mask means feeding the model with data on everywhere *except* the center section of the channel the model is trained on, i.e., the network has to make predictions on portions of the data, where it never saw the truth values during training. In this sense, the error rate of 22% is more understandable.

Example 26. Island-and-ring visualization with KNO

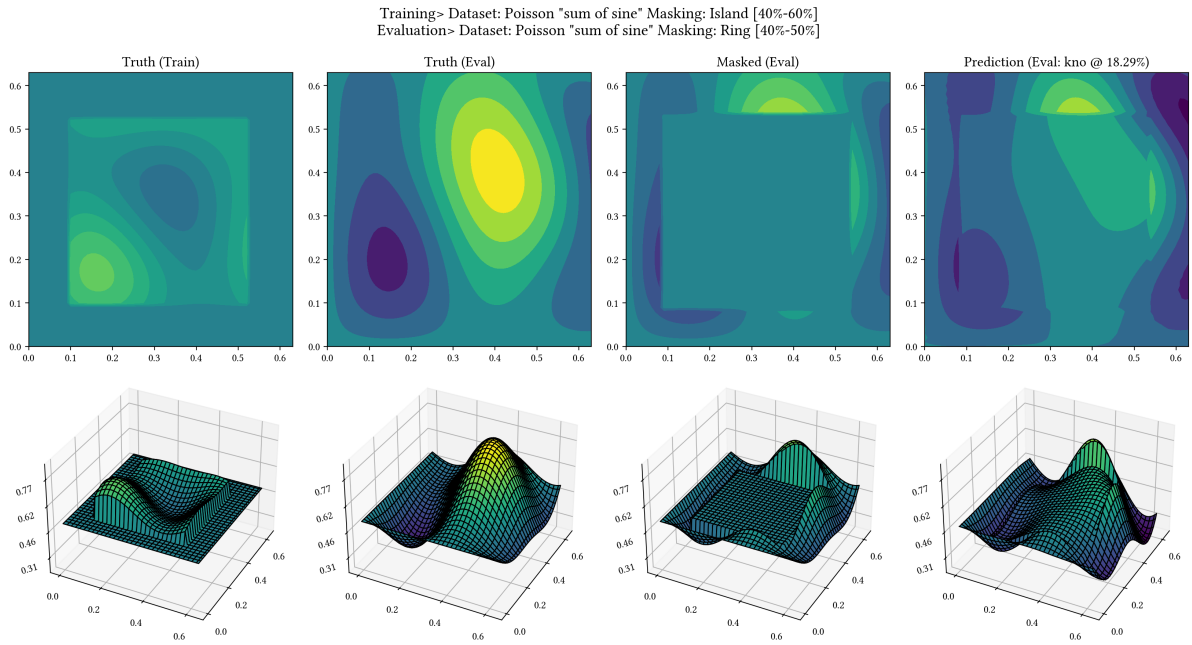


FIGURE 11.6. First column: instance during training (island mask with intensity $\in [40\%; 60\%]$). Second to fourth column: truth, masked and KNO's prediction of one eval instance, masked with ring-style, intensity $\in [40\%; 50\%]$. MSE as shown: 0.0102 absolute,18.29% relative. Observe the complementary training and evaluation portions!

Figure 11.6 showcases the complementary nature of the island and the ring-style masks. Using the direct inspection technique introduced in Section 11.2, the first column depicts one training instance masked with island $[40\%; 60\%]$.

With the evaluation mask chosen as the ring-style mask of $[40\%; 50\%]$, the second to fourth columns represent the truth, the masked result and the prediction by KNO respectively. As shown, the prediction corresponds to an absolute and relative MSE of 0.0102 and 18.29%.

11.4. **Non-standard masking.** As previously discussed in Chapter 4.4, the standard masking value used in this thesis seen so far is 0.5, corresponding exactly to the expected value of the value range of the $[0; 1]$ normalization strategy of this project.

On the other hand, frequently seen in other researches pertaining to masking is the the vanishing masking value 0.0[Ton+22a]; [Ton+22b]; [Bac+22]. This chapter investigates such non-standard masking values, in order to contrast model performance under the influence of masking value changes.

Definition 45. Experiment setup

The standard masking value used so far is calculated as the expected value, aka mean, of the dataset post-normalization in the range of $[0; 1]$:

$$v^{\text{mask}} := \text{Dataset}_{\mu} \equiv 0.5$$

Contrast to this standard »mean-masking« are the practices of »min« and »max-masking«:

$$v^{\text{mask}} := \begin{cases} \text{Dataset}_{\text{min}} \equiv 0.0 \\ \text{Dataset}_{\text{max}} \equiv 1.0 \end{cases}$$

With the min-masking regime particularly popular in the masked-learning context.

In-distribution and out-of-distribution errors are studied to compare the impact of these three masking values on model performance. In particular, the Poisson equation with the full intensity-range $[0\%; 100\%]$ is used for experimentation. Single masking is used with FNO as the underlying network.

Other configurations of the learning pipeline, such as the training-epoch count and the dataset-split numbers are kept unchanged.

Example 27. In-distribution performance

The in-distribution evaluation results are shown in Table 11.8.

One observes that the best overall average performance of 7.3962% is achieved using the standard 0.5 masking value, which corresponding to the expected value of the normalized dataset. In comparison, the average errors with masking value of 0.0 (minimum of dataset) and 1.0 (maximum of dataset) both approach 10% approximately.

At lower intensity ranges, in particular, for the two ranges of $[0\%; 10\%]$ and $[10\%; 20\%]$, the 0.0 masks and 1.0 masks do visibly out-perform the standard 0.5 masks, as shown by the error values in blue indicating best performance for one particular intensity range.

However, the errors increase steeply for the non-standard masks with higher masking intensities. At higher intensity ranges such as $[80\%; 90\%]$ and $[90\%; 100\%]$, errors of the standard 0.5 mask at approximately 10% are significantly lower than those of the 0.0 and 0.5 masks, both surpassing 15% by a fair margin.

One concludes that in comparison to the non-standard, minimum and maximum masking values of 0.0 and 1.0, the standard 0.5 masking leads to lower average errors overall with stable performance across the full spectrum of intensity ranges.

Example 28. Out-of-distribution performance

Table 11.10 showcases the out-of-distribution errors.

Again, one observes better performance using the standard masking value 0.5 in both out-of distribution evaluation cases with the island and ring-style masks. Observing the row-wise best error values marked in green and blue, one concludes that although minimum masking with value 0.0 locally out-perform mean-masking for intensity values, the overall best performance is achieved with the latter for masking of both island and ring-style.

Intensity	0.5 (standard)	0.0 ($\text{Dataset}_{\text{min}}$)	1.0 ($\text{Dataset}_{\text{max}}$)
$[0\%; 10\%]$	6.8956%	1.8869%	3.9493%
$[10\%; 20\%]$	6.3342%	2.8223%	5.4531%
$[20\%; 30\%]$	5.9610%	4.6118%	7.0412%
$[30\%; 40\%]$	5.9683%	6.3858%	8.6560%
$[40\%; 50\%]$	6.2500%	8.2742%	10.2120%
$[50\%; 60\%]$	6.7590%	10.0805%	11.6263%
$[60\%; 70\%]$	7.5202%	11.9137%	12.9951%
$[70\%; 80\%]$	8.3690%	13.6400%	14.2366%
$[80\%; 90\%]$	9.3936%	15.5404%	15.5530%
$[90\%; 100\%]$	10.5111%	17.4296%	16.8266%
Average	7.3962%	9.2585%	10.6549%

TABLE 11.8. In-distribution errors for masking values of 0.5 (standard, mean-masking), 0.0 (minimum-masking) and 1.0 (maximum-masking). Best performing mask-variant of each row marked blue. Dataset: Poisson equation, single mask, linearly normalized to $[0; 1]$. Network as shown: FNO.

Intensity	0.5 island	0.0 island	1.0 island	0.5 ring	0.0 ring	1.0 ring
$[0\%; 10\%]$	7.3387%	3.4390%	4.2509%	7.2926%	3.4224%	4.0788%
$[10\%; 20\%]$	7.8197%	5.6579%	6.2150%	7.5648%	6.4754%	6.7281%
$[20\%; 30\%]$	8.4838%	7.9534%	8.2012%	8.3061%	9.2473%	9.0674%
$[30\%; 40\%]$	8.8928%	9.4391%	9.5032%	9.4824%	11.5996%	10.8434%
$[40\%; 50\%]$	8.9740%	10.3829%	10.4518%	10.6205%	13.5792%	12.3557%
$[50\%; 60\%]$	8.9802%	11.3247%	11.4227%	11.5566%	14.9632%	13.4732%
$[60\%; 70\%]$	9.2314%	12.7234%	12.6959%	11.8797%	15.6670%	14.2111%
$[70\%; 80\%]$	9.7914%	14.2507%	13.9129%	11.7918%	16.0739%	14.8999%
$[80\%; 90\%]$	10.5130%	15.8436%	15.2514%	11.4092%	16.6917%	15.7196%
$[90\%; 100\%]$	11.1132%	17.6946%	16.8784%	11.0738%	17.6626%	16.7686%
Average	9.1138%	10.8709%	10.8783%	10.0978%	12.5382%	11.8146%

TABLE 11.10. Out-of-distribution errors (island-style and ring-style) for masking values of 0.5 (standard, mean-masking), 0.0 (minimum-masking) and 1.0 (maximum-masking). Best performing mask-variant marked blue and green for island and ring-style respectively. Dataset: Poisson equation, single mask, linearly normalized to $[0; 1]$. Network as shown: FNO.

Tangentially, one observes that the average errors using the island-masks are consistently lower than those with the ring-style masks. This aligns with the previous findings in Conclusion 3.

With this, the analysis section of this thesis is concluded. An executive summary is now given to recapitulate the key findings of this project.

12. Executive Summary

After opening with a quick recap of the discretization techniques in Chapter 3, the masking procedure is introduced in Section 4. Three separate configuration options are identified: the masking intensity to dictate the number of masked entries, the masking strategy to translate the intensity to concrete masking candidates, and, finally, the masking value used to modify those candidates.

Section 5 enumerates the three PDE prototypes of interest bundled with their respective raw datasets, formulated with the super-positioning technique. Table 5.4 summarizes these three PDE models.

The full learning pipeline is then introduced, incorporating multi-channel masking with the PDE prototypes. In particular, the construction of single and double-masked datasets are streamlined in Definition 25, along with four networks of interest with various design principles and configurations, as presented in Section 6. Exploiting the configurable nature of the masking procedure, in and out-of-distribution regimes are defined Section 8.2, guiding subsequent analyses.

Section 9 illustrates that models trained on the generic, *full* range masks generalize fairly well out-of-distribution, although errors tend to increase with higher masking intensities during evaluation. Conclusion 3 points out that out-of-distribution performance is usually better with island-style masks than with ring-style ones.

On the other hand, Section 10 demonstrates that, if models are trained using datasets with *specialized*, concentrated ranges of masking intensities, they tend to display locality attachment to those intensities ranges, in that performance degrades visibly when evaluated on masking-intensities outside the training ranges. This observation sometimes applies to both in and out-of-distribution evaluation modes. Per Conclusion 3, performances of the two target channels in double-masking mode are generally similar, but might differ significantly compared to the results seen with single masking due to the presence of the second masking channel.

Mean-masking, where the masking value is set to the mean of the unmasked dataset post normalization, is found in Section 11.4 to yield better result than min or max-masking. The dynamic remasking technique is recognized in Conclusion 4 as a nifty addition to the usual training loop, functioning both as an early termination criteria and as a source of new knowledge for models to discover new improvements when struggling with local optima. Finally, experiments in Section 11.3 showed that reasonable performance can be garnered even if training and evaluation are deliberately performed with complementary masks (island VS ring-style), which target mutually exclusive regions in space respectively.

The above analyses are realized by training 100 models (4 of island-style and 96 of random-style: 3 PDEs, 4 networks, 4 intensity regimes, each in single and double-masked mode), each evaluated on 30 corresponding inference datasets, as documented in Definition 30.

Component	Model	Notes
GPU	NVIDIA RTX 2080Ti	∅
CPU	Intel Core i7-12700K	with default overclocking
RAM	192 GB	4×48 GB DDR5

TABLE 13.2. Hardware setup

Component	Version
OS Kernel	Linux Zen, 6.6.9-6.7.9
python Interpreter	reference implementation, 3.11-3.12
Deep Learning Toolkit	pytorch, 2.2-2.3
cuda suite	12.3.1-12.5.0
C++ toolchain	clang, 16.0-17.0

TABLE 13.4. Supported software versions

Part 5. Epilogue

13. Review and Projection

13.1. Environment Setup.

13.1.1. *OS-level setup.* A typical machine-learning project must be run on a carefully orchestrated environment setup spanning both hardware and software components. This section documents the particular setup utilized by the author of this thesis to execute the full project pipeline. To start off, Table 13.2 enumerates the hardware specifications.

On the other hand, Table 13.4 enlists the software used in this project. Due to the non-trivial, six-month time-span of the project, several key pieces of software of this project have seen multiple versioned releases. As such, the respective version *ranges* of these software components are given, which are tested for successful deployment to execute the project. It shall serve as a general orientation for readers attempting to reproduce the results of the project. Note that although non-listed versions of these pieces of software will not necessarily break code, no explicit guarantee is given by the author for functionality if they are used.

13.1.2. *python-specific setup.*

Solution 1. Virtual environment

The poetry [ET] virtual-environment tool is used to manage the virtual environment, which handles the dependency management of the project.

The interested reader is invited to duplicate the environment to verify the results or run individual modules. To this end, a virtual environment shall first be created, then populated with the necessary python packages for executing the source code of the repository. This is achieved with the following command[5]:

```
$ poetry install # install package dependencies in virt. env.
```

[5] official documentation by poetry

At this point, two possibilities exist for running commands in the now available virtual environment:

```
$ poetry run python <module> # execute some (python-) module
$ poetry shell # activate interactive shell
```

Each method has its own merits, and the choice should depend on the specific coding use-case at hand.

Solution 2. Module testing

Core utilities of the project are tested with a separate test suite structured as follows:

```
tests
|-- test_dataset.py
|-- test_deepl.py
|-- test_integral.py
|-- test_multidiff.py
|-- test_sample.py
|-- test_util_math.py
|-- test_util.py
```

The established `pytest` utility [Kre+04] is used to format and manage the tests. and automatically installed with the `poetry` virtual environment setup procedure above. All tests are automatically run by issuing the following commands:

```
$ cd <project_root> # e.g., location after repo-cloning
$ pytest ./tests
```

Note 24. Reproducibility

While significant segments of machine-learning banks on randomness and seeding [HTF09]; [GBC16]; [Mur22], measures can still be taken to guarantee some semblance of reproducibility. The key is to consciously set the seed-value for random-number-generators whenever reproducibility is desired. Conforming to common practice, this project uses 42 as the manual seed value.

Listing 6 shows a utility function collecting common seeding functions:

```
# src/definition.py
# import...

class Definition:
    @staticmethod
    def seed(seed: int = 42) -> None:
        torch.manual_seed(seed)
        np.random.seed(seed)
        random.seed(seed)
```

Listing 6. Common seeding operations

13.2. **Timeline.** The administrative details of this thesis can be retrieved under the course-catalogue of ETH Zürich under the registration number `401-4990-01L`. Commencing on **04.12.2023**, the project consists of six (**6**) months of full-time work, including the drafting of the thesis and the composing of the source-code. Specifics are found in Table 13.6.

Activities	Duration (×weeks)
Initial planning; ML lab environment setup	3
Literature survey; project structuring	3
Inspection and trial run of existing libraries	3
Initial coding and testing	3
Theoretical formulation; bulk implementation work	6
Benchmarking; results collection	2
Final code review; thesis drafting	4

TABLE 13.6. Time-stamps of key activities

13.3. **Future work.** Due to the objective limitation of the framework of a Master Thesis, various aspects related to masked PDE learning remain to be explored. Some immediate prospects for further researches include, but are not limited to:

- application of more involved PDE models, including generalization of the existing three prototypes beyond the two-dimensional settings and the usage of discretization schemes beyond the generic square mesh structure as discussed in Section 1,
- implementation of more advanced masking strategies, such as varying masking value as proposed in Section 4 and larger *kernels* for the random mask, and other mask shapes and strategies in general,
- augmentation with other general machine-learning techniques for training, including the exploration of other loss functions and scheduling rules with the possible application of strategies such as bagging and graph-based learning.

The list is definitely non-exhaustive: the reader is encouraged to continue upon the findings of this thesis to explore follow-up possibilities to build upon the framework of masked PDE learning established herein as general research direction.

14. ACKNOWLEDGEMENTS

To the supervisor of this thesis, Prof. Dr. S. Mishra:
my deepest appreciation for the encouragement of pursuing this project and for the rigorous yet illuminating approach to the realm of scientific learning as a whole;

To my advisor, Bogdan:
my heart-felt thanks for the rapid and intuitive implementations of abstract concepts and for providing me with constructive suggestions during hours of auditing and reviewing;

Finally, to my family and my friends:
my sincere gratitude for the unyielding support and care throughout the course of my studies.

References

1. Amann, H. & Escher, J. *Analysis III* ISBN: 9783764374808. https://books.google.se/books?id=bofFBAAAQBAJ (Birkhäuser Basel, 2009).
2. Atkinson, K. & Han, W. *Theoretical Numerical Analysis: A Functional Analysis Framework* ISBN: 9780387287690. https://books.google.fr/books?id=6TpGAAAAQBAJ (Springer New York, 2007).
3. Bachmann, R., Mizrahi, D., Atanov, A. & Zamir, A. *MultiMAE: Multi-modal Multi-task Masked Autoencoders* 2022. arXiv: 2204.01678 [cs.CV].
4. Boullé, N. & Townsend, A. *A Mathematical Guide to Operator Learning* 2023. arXiv: 2312.14688 [math.NA].
5. Burden, R. & Faires, J. *Numerical Analysis* ISBN: 9780538733519. https://books.google.fr/books?id=zXnSxY9G2JgC (Cengage Learning, 2010).
6. Chen, S. *Default Thesis* https://github.com/shengdichen/def_tss.
7. Cheney, E. & Kincaid, D. *Numerical Mathematics and Computing* ISBN: 9781133712350. https://books.google.fr/books?id=PewJAAAAQBAJ (Cengage Learning, 2012).
8. Developers, G. *Machine Learning - Foundational courses* https://developers.google.com/machine-learning/data-prep/transform/normalization. Accessed: 2024-04-20. 2024.
9. Eustace, S. & The Poetry contributors. *Poetry: Python packaging and dependency management made easy* https://github.com/python-poetry/poetry.
10. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* ISBN: 9780262337373. https://books.google.fr/books?id=omivDQAAQBAJ (MIT Press, 2016).
11. Harris, C. R. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020).
12. Hastie, T., Tibshirani, R. & Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition* ISBN: 9780387848587. https://books.google.fr/books?id=tVIjmNS3Ob8C (Springer New York, 2009).
13. Isaacson, E. & Keller, H. *Analysis of Numerical Methods* ISBN: 9780486680293. https://books.google.fr/books?id=y77n2ySMJHUC (Dover Publications, 1994).
14. Königsberger, K. *Analysis 1* ISBN: 9783540403715. https://books.google.ch/books?id=eQRtZXuUHl4C (Springer Berlin Heidelberg, 2003).
15. Kovachki, N. B. *et al.* Neural Operator: Learning Maps Between Function Spaces. *CoRR* **abs/2108.08481** (2021).
16. Krekel, H. *et al. pytest x.y* 2004. https://github.com/pytest-dev/pytest.
17. Li, Z. *et al. Fourier Neural Operator for Parametric Partial Differential Equations* 2020. arXiv: 2010.08895 [cs.LG].
18. Lu, L., Jin, P., Pang, G., Zhang, Z. & Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence* **3**, 218–229 (2021).
19. Mao, Z., Lu, L., Marxen, O., Zaki, T. A. & Karniadakis, G. E. DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. *Journal of Computational Physics* **447**, 110698. ISSN: 0021-9991. http://dx.doi.org/10.1016/j.jcp.2021.110698 (Dec. 2021).
20. Masters, D. & Luschi, C. *Revisiting Small Batch Training for Deep Neural Networks* 2018. arXiv: 1804.07612 [cs.LG].
21. Murphy, K. *Probabilistic Machine Learning: An Introduction* ISBN: 9780262046824. https://books.google.fr/books?id=wrZNEAAAQBAJ (MIT Press, 2022).

22. Paszke, A. *et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library* in *Advances in Neural Information Processing Systems 32* (eds Wallach, H. *et al.*) (Curran Associates, Inc., 2019), 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
23. Quarteroni, A. & Valli, A. *Numerical Approximation of Partial Differential Equations* ISBN: 9783540852681. https://books.google.ch/books?id=nfdDAAAAQBAJ (Springer Berlin Heidelberg, 2009).
24. Raonić, B. *et al. Convolutional Neural Operators for robust and accurate learning of PDEs* 2023. arXiv: 2302.01178 [cs.LG].
25. Ronneberger, O., Fischer, P. & Brox, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation* 2015. arXiv: 1505.04597 [cs.CV].
26. Team, N. M. *Darcy Flow with Fourier Neural Operator*
27. Tong, Z., Song, Y., Wang, J. & Wang, L. *VideoMAE: Masked Autoencoders are Data-Efficient Learners for Self-Supervised Video Pre-Training* in *Advances in Neural Information Processing Systems* (2022).
28. Tong, Z., Song, Y., Wang, J. & Wang, L. VideoMAE: Masked Autoencoders are Data-Efficient Learners for Self-Supervised Video Pre-Training. *arXiv preprint arXiv:2203.12602* (2022).
29. Vaswani, A. *et al. Attention Is All You Need* 2023. arXiv: 1706.03762 [cs.CL].
30. Virtanen, P. *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17**, 261–272. https://doi.org/10.1038/s41592-019-0686-2 (2020).
31. Wasserman, L. *All of Statistics: A Concise Course in Statistical Inference* ISBN: 9780387217369. https://books.google.fr/books?id=qrcuBAAAQBAJ (Springer New York, 2013).
32. Willing, C. *et al. The uncompromising Python code formatter*
33. Xiong, W. *et al.* Koopman neural operator as a mesh-free solver of non-linear partial differential equations. *arXiv preprint arXiv:2301.10022* (2023).
34. Xiong, W. *et al.* Koopmanlab: machine learning for solving complex physics equations. *APL Machine Learning* **1** (2023).