CODES: Benchmarking Coupled ODE Surrogates

Robin Janssen, Immanuel Sulzer and Tobias Buck
Interdisciplinary Center for Scientific Computing
University of Heidelberg

Abstract

We introduce CODES, a benchmark for comprehensive evaluation of surrogate architectures for coupled ODE systems. Besides standard metrics like mean squared error (MSE) and inference time, CODES provides insights into surrogate behaviour across multiple dimensions like interpolation, extrapolation, sparse data, uncertainty quantification, and gradient correlation. The benchmark emphasizes usability through features such as integrated parallel training, a web-based configuration generator, and pre-implemented baseline models and datasets. Extensive documentation ensures sustainability and provides the foundation for collaborative improvement. By offering a fair and multi-faceted comparison, CODES helps researchers select the most suitable surrogate for their specific dataset and application while deepening our understanding of surrogate learning behaviour.

1 Introduction and Overview

Coupled ordinary differential equations (ODEs) are ubiquitous in the natural sciences, and significant effort has gone into using machine learning models (surrogates) to replace the computationally expensive numerical methods required to solve such systems [2–5, 10]. Despite individual benchmarking efforts when presenting such new approaches, there is a lack of comprehensive and fair comparisons across different architectures. This makes it difficult to identify the best surrogate for a specific task, especially since the optimal choice can depend on both the dataset and the target application.

The CODES benchmark (**C**oupled **ODE S**urrogates) addresses this gap by providing a framework for evaluating surrogates in the context of coupled ODE systems. Beyond accuracy, training, inference time, and computational cost, other crucial factors differentiate architectures. As shown in section 3, surrogates vary in how they handle sparse data and perform in interpolation and extrapolation tasks. Some struggle more in highly dynamic regions, while others may yield unreliable uncertainty estimates, which could be problematic in hybrid approaches where numerical solvers are employed for predictions with low certainty. CODES highlights these differences, helping researchers and practitioners choose the best model for their task and deepening our understanding of whether surrogates truly capture the underlying dynamics of the data. The primary use cases include adding a new dataset and identifying the most suitable surrogate for it or adding a new surrogate to the benchmark and comparing its performance to the baseline models on existing datasets.

CODES focuses on many essential aspects of benchmarking. Several **baseline models** and **datasets** are already implemented. Extensive **documentation**, featuring detailed doc-strings and type hints, ensures ease of use and project sustainability, supported by a dedicated documentation website [12]. To guarantee **reproducibility**, all models are trained and evaluated in a fully seeded manner. Usability is crucial for broad adoption. Therefore, features such as a web-based config-file generator, parallel training support, toggleable comparison modalities, and automated evaluation through tables and plots have been integrated. CODES is intended to grow as a **collaborative** project, with clear instructions on adding new datasets and surrogate models, encouraging community contributions.

2 Structure of the Benchmark

CODES currently includes four surrogate architectures and five datasets. After configuration, the models are first trained on the training set and then evaluated on the test set. The code is available on GitHub [6].

Datasets & Surrogates. CODES is largely inspired by astrochemistry, hence the two most complex datasets come from this field: `osu2008` was adapted from Grassi et al. [4] (an exemplary plot is shown in Figure 1) while `branca24` was kindly provided by Branca & Pallottini from their recent paper [3]. The remaining three datasets (`lotka_volterra`, `simple_ode`, and `simple_reaction`) are lower-dimensional baselines created by the authors of this paper. All datasets are hosted on Zenodo and automatically downloaded as needed.

The following architectures are currently implemented:

- `FullyConnected` (FCNN). A standard fully-connected neural network.
- `MultiONet` (MON). An adaptation of DeepONet [8] for multiple outputs as proposed in [9].
- `LatentNeuralODE` (LNODE). An autoencoder with a latent space neural ODE, following [11]
- `LatentPoly` (LP). An autoencoder with a learnable latent space polynomial, also following [11].

All models receive as input the abundances at $t = 0$ and then predict abundances for some later time $t > 0$. Time is treated either as an additional input (FCNN, MON) or as a parameter that determines the length of the latent-space evolution (LNODE, LP). Detailed explanations for all architectures and datasets are available in Appendix A and Appendix B as well as on the documentation website [12].
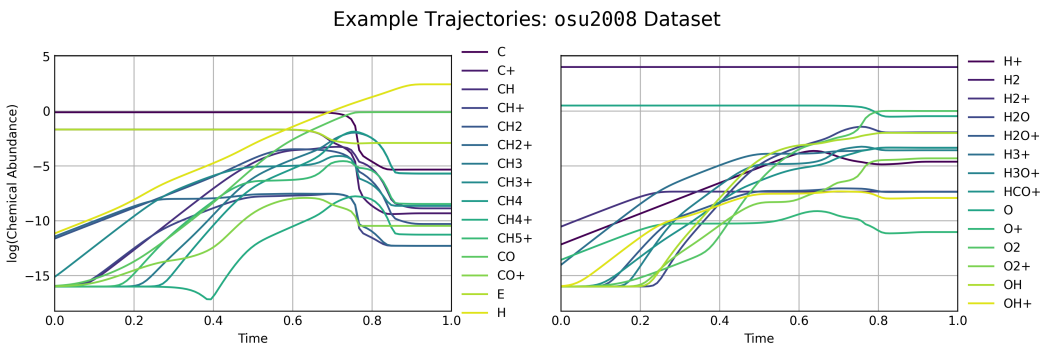

Figure 1: Example trajectories of a datapoint (29 chemicals, 100 timesteps) in the `osu2008` dataset.

Modalities. In addition to training a main model on the full dataset, the following modalities can be toggled to train additional models:

- `Interpolation` skips timesteps (or generally the independent variable) in a specified interval.
- `Extrapolation` trains only on timesteps below a set threshold.
- `Sparse` thins out the training data by a specified factor, using only every n-th sample.
- `Batch` trains the models with a user-defined batch size.
- `Uncertainty` trains N additional models with different random initializations and shuffled datasets to create a DeepEnsemble [7] for uncertainty estimation.

By setting multiple values (i.e., multiple intervals, thresholds, factors, or batch sizes), users can gain insights into and compare surrogate behaviour, answering questions such as: Which model handles sparse data best? Which performs the best interpolation? For evaluation, users can configure additional output such as inference time, memory usage, loss curves, and different correlations. A configuration tool on the website [12] helps users generate the `config.yaml`.
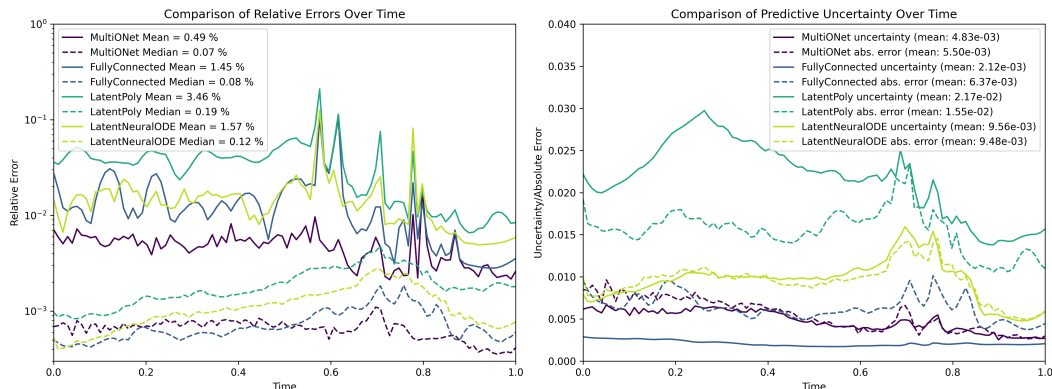
2

Figure 2: Quantities over time per model, averaged across the test set. **Left**: Mean and median relative error. **Right**: Predictive uncertainty (DeepEnsemble, $n = 5, 1\sigma$) and mean absolute error.

Training & Benchmarking. Training is the most time-intensive step in the benchmark, as for each setting an additional model is trained on the corresponding subset of training data (e.g., 96 models were trained for the evaluations presented in section 3). To accelerate this process, training is parallelized across all specified devices using a task-list approach. Progress bars show the status of active jobs. Custom hyperparameters can be specified for each surrogate, and predefined optimal hyperparameters are available for the provided datasets. After the training run finishes, evaluation results are generated in the form of metrics and plots. These include individual (per-surrogate) and comparative results. Currently, up to 18 individual plots per surrogate and 12 comparative plots are provided. Metrics include standard measures like mean-squared error (MSE), mean absolute error (MAE), mean relative error (MRE), and inference time. Additionally, there are more advanced metrics, such as the Pearson correlation coefficient (PCC) between predictive uncertainty (via DeepEnsemble) and prediction error (PE), or between data gradients and prediction error. See Table 1 for a selection of the available metrics.

3 Results

The following results represent an exemplary evaluation of the benchmark. Models were trained on the `osu2008` dataset, generated from a chemical reaction network with 224 reactions among 29 chemical species, simulated at constant temperature over 100 time steps. The dataset contains 1000 data points, split into 75% training, 5% validation, and 20% testing. Models were trained on NVIDIA TITAN Xp GPUs and evaluated on an NVIDIA GeForce RTX 2080 Ti. Optimal configurations for each model were determined using Optuna [1] and are detailed in Appendix A. While the results are specific to this dataset, broader patterns can be observed. Given the benchmark's comprehensive nature, the following results are just a subset of the analyses and insights that can be drawn. Additional plots can be found in Appendix C or on the documentation website [12].

Performance. The main benchmark metrics, along with additional model parameters, are shown in Table 1. Despite significant differences between models, all show impressive performance — even the slowest model, `LNODE`, achieves speedups of at least three orders of magnitude compared to numerical methods (as discussed by Sulzer & Buck in [11]) and all models exhibit mean relative errors of below 4%. `MON` performs best on accuracy metrics such as MSE, MAE and MRE (followed closely by `FCNN`) , while `LP` has the fastest inference time, likely due to the computational simplicity of evaluating polynomials with fixed coefficients. Figure 2 displays the mean and median error of each surrogate over time. The spikes in the second half correspond to more dynamic sections of the data, with which some surrogates struggle more than others (cf. the section on UQ and correlations).

Table 1: Benchmark Results for the Surrogate Models

| Metric	MultiONet	FullyConnected	LatentPoly	LatentNeuralODE
MSE	**5.89e-05**	8.57e-05	4.50e-04	1.86e-04
MAE	**5.50e-03**	6.37e-03	1.55e-02	9.48e-03
MRE	**0.49%**	1.45%	3.46%	1.57%
Inference Time (ms)	23.37 ± 0.10	3.38 ± 0.10	**0.55 ± 0.06**	37.76 ± 0.84
Mean uncertainty	4.83e-03	2.12e-03	2.17e-02	9.56e-03
PCC UQ	0.3551	0.1909	**0.4178**	0.4166
PCC Gradient	0.2606	0.3054	0.3076	**0.5006**
Epochs	1000	1000	15000	10000
Train Time (hh:mm:ss)	02:51:36	01:51:45	**00:54:09**	01:49:30
# Trainable Params	558970	184429	62864	72368
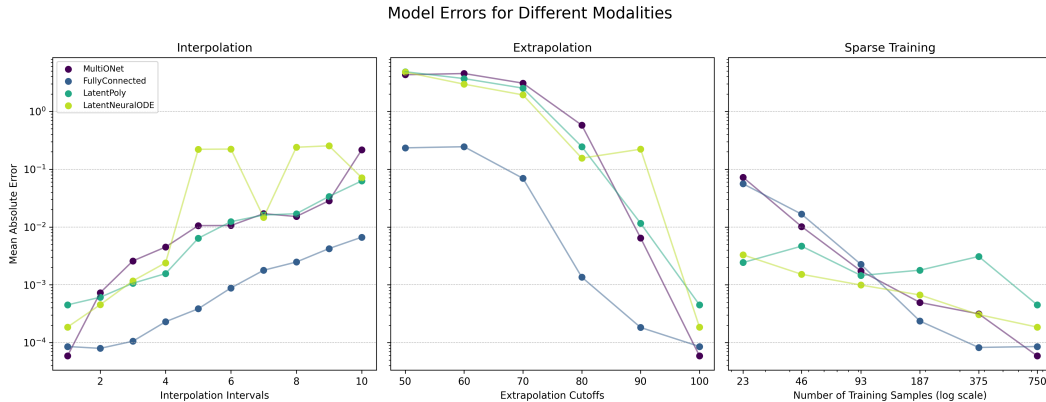
Figure 3: Model errors for the modalities `interpolation`, `extrapolation` and `sparse`.

Interpolation, Extrapolation, and Sparsity. The differences in inductive bias between surrogates become more apparent under challenging conditions. Figure 3 shows the model MAE across three different benchmark modalities. In `interpolation` and `extrapolation`, FCNN outperforms all other models. The spikes observed in LNODE may result from intervals where the remaining timesteps are not representative of the full dataset, a challenge the other models seem to handle more effectively. In the `sparse` data setting, the autoencoder-type models, LNODE and LP, outperform the fully-connected models, FCNN and MON, for low numbers of training samples (≤ 93), suggesting greater robustness to settings with reduced information. This might be explained by the fact that the former involve a temporal evolution, which more closely mirrors the problem structure, while the latter treat time merely as an additional input.

Uncertainty Quantification & Correlations. Comparing the predictive uncertainty (the 1σ interval from DeepEnsemble predictions) with the mean absolute error over time in Figure 2 allows for insights into how well each surrogate architecture estimates its own prediction errors. DeepEnsemble seems to provide a generally reliable uncertainty estimate, particularly for LNODE and MON. This is also confirmed by the close coincidence of the MAE and mean uncertainty metrics in Table 1. FCNN tends to be overconfident, underestimating the magnitude of its errors, while LP slightly tends towards the opposite behavior. It could be argued that the magnitude of an uncertainty estimate is less important than its correlation to the errors, since the estimate could just be rescaled appropriately. Following this argument, we added the Pearson correlation coefficient (PCC) between the predicted uncertainty and the prediction error (calculated across all predictions in the test set) to the benchmark. This metric is lowest for FCNN, confirming the apparent overconfidence of this architecture. Additionally, we integrated the PCC between data gradients and prediction error. This metric is highest for LNODE, evidencing a proclivity towards higher errors at points with larger gradients. This provides an additional indication to our earlier impression that this architecture struggles more with non-smooth data than other architectures.

4 Discussion

Summary. The CODES benchmark offers diverse insights into the behavior of surrogates for coupled ODE systems. Beyond identifying the fastest or most accurate architecture, its extensive plots and metrics help determine the most suitable surrogate for a specific dataset and application, which may require characteristics such as robustness to sparse data, reliable interpolation, or precise uncertainty estimation. The benchmark is modular, with a focus on usability, reproducibility, and collaboration. A comprehensive documentation website, including a web-based configuration tool, was created to support this effort [12].

Limitations. Ensuring fair and representative comparisons can be challenging. CODES addresses this by seeding and training all models consistently within the benchmark. The limitations of the benchmark can roughly be divided into two categories: the influence of architectural differences on comparability and the data-dependency of the results.

For the results in section 3, models were compared by training each until its loss trajectory stabilized, which is an effective but potentially imprecise approach. Alternatives such as training for a fixed number of epochs (implemented) or equal training time (not yet implemented) could also be argued for, and would significantly affect benchmark outcomes. Model performance heavily depends on hyperparameters, which were optimized via Optuna. This process is not yet integrated into the benchmark and must be performed manually; it may bias some models towards higher complexity, sacrificing speed. Architectural differences further complicate the comparison of the inference time. For example, LNODE is more efficient when predicting entire trajectories at once rather than generating individual predictions step-by-step, potentially skewing the comparison. We adopted the trajectory-based approach, but its representativeness depends on the target application.

In principle, training surrogates to model an ODE is an infinite-data problem because the ground truth data is generated with a numerical solver from randomly sampled initial conditions. In addition to allowing for the generation of arbitrary amounts of training data, this problem structure also invites online or active learning approaches. However, the benchmark currently only supports finite datasets provided in hdf5 format. Finally, the benchmarking results are always dataset-specific. This implies that a broad range of datasets is needed to thoroughly assess surrogate performance and necessitates further insights into each dataset to inform comparisons between them. Additionally, findings are only reliable if the distribution of the training data matches that of the target application, emphasizing the need for representative training data.

Outlook. CODES enables comprehensive comparisons of new and existing surrogate architectures and helps users select the ideal model for their specific context, ensuring optimal performance in target applications. By improving surrogate selection, CODES contributes to faster and more reliable simulation frameworks, enabling higher resolution or more complex applications. Future updates will include integrated hyperparameter tuning, dataset-specific metrics and visualizations, further exploration of inter- and extrapolation capabilities, additional baseline models, and potentially support for online training. A key focus will be adding new datasets and surrogates, for which we warmly invite community contributions that will be credited appropriately.

Acknowledgments and Disclosure of Funding

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] L Branca and A Pallottini. Neural networks: solving the chemistry of the interstellar medium. *Monthly Notices of the Royal Astronomical Society*, 518(4):5718–5733, 12 2022. ISSN 0035-8711. doi: 10.1093/mnras/stac3512. URL `https://doi.org/10.1093/mnras/stac3512`.

[3] Lorenzo Branca and Andrea Pallottini. Emulating the interstellar medium chemistry with neural operators. *Astronomy & Astrophysics*, 684:A203, April 2024. ISSN 0004-6361, 1432-0746. doi: 10.1051/0004-6361/202449193. URL `https://www.aanda.org/10.1051/0004-6361/202449193`.

[4] T. Grassi, F. Nauman, J. P. Ramsey, S. Bovino, G. Picogna, and B. Ercolano. Reducing the complexity of chemical networks via interpretable autoencoders. *Astronomy & Astrophysics*, 668:A139, December 2022. ISSN 0004-6361, 1432-0746. doi: 10.1051/0004-6361/202039956. URL `http://arxiv.org/abs/2104.09516`. arXiv:2104.09516 [astro-ph, physics:physics].

[5] J. Holdship, S. Viti, T. J. Haworth, and J. D. Ilee. Chemulator: Fast, accurate thermochemistry for dynamical models through emulation. *Astronomy & Astrophysics*, 653:A76, September 2021. ISSN 1432-0746. doi: 10.1051/0004-6361/202140357. URL `http://dx.doi.org/10.1051/0004-6361/202140357`.

[6] Robin Janssen and Immanuel Sulzer. CODES, September 2024. URL `https://github.com/robin-janssen/CODES-Benchmark`. License: GNU GPL-3.0.

[7] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles, 2017. URL `https://arxiv.org/abs/1612.01474`.

[8] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *CoRR*, abs/1910.03193, 2019. URL `http://arxiv.org/abs/1910.03193`.

[9] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, April 2022. ISSN 0045-7825. doi: 10.1016/j.cma.2022.114778. URL `http://dx.doi.org/10.1016/j.cma.2022.114778`.

[10] S. Maes, F. De Ceuster, M. Van de Sande, and L. Decin. Mace: A machine learning approach to chemistry emulation, 2024. URL `https://arxiv.org/abs/2405.03274`.

[11] Immanuel Sulzer and Tobias Buck. Speeding up astrochemical reaction networks with autoencoders and neural ODEs, December 2023. URL `http://arxiv.org/abs/2312.06015`. arXiv:2312.06015 [astro-ph].

[12] Immanuel Sulzer and Robin Janssen. CODES Documentation, October 2024. URL `https://codes-docs.web.app`.

A Surrogate Architectures

Architectures.   Four architectures are currently implemented in the benchmark. Let N_q denote the number of quantities in a dataset (e.g., $N_q = 29$ for the `osu2008` dataset).

- `FullyConnected` is a standard fully-connected neural network. It has $N_q + 1$ inputs since the time t of prediction is given as extra input and N_q outputs.

- `MultiONet` follows the DeepONet architecture first described in [8]. It features two fully-connected neural networks, the branch and trunk networks. The branch network receives as inputs the N_q initial conditions, the trunk network receives as a single input the time t of prediction. In the original architecture, the network has only a single output which is generated by taking the scalar product of the latent (output) vectors of branch and trunk. As proposed in [9], we adapt this architecture to produce multiple outputs in a single forward pass by splitting the latent vectors of both networks into N_q parts and taking the scalar product of corresponding parts of branch and trunk network.

- `LatentNeuralODE` features an autoencoder which is essentially a fully-connected neural network with a smaller output size L_q than its input size N_q which encodes an initial set of abundances into a latent space. There, the encoded initial condition is used as the initial state for a numerical integration of a second neural network which generates latent space trajectories through time (Neural ODE). Finally, the entire latent space trajectory is fed to a third neural network (the decoder) with input size L_q and output size N_q which outputs the predicted time series.

- `LatentPoly` uses a similar encoder/decoder setup as `LatentNeuralODE`, but instead of a Neural ODE the initial condition in latent space is added to a learnable polynomial which is evaluated at the time points of interest. The resulting latent space trajectory is again decoded back to the original dimensionality N_q.

Hyperparameters.   All architectures used the Adam optimiser, the number of epochs are given in Table 1. The results in section 3 were achieved using the following hyperparameters:

- `FullyConnected`: The architecture consists of two hidden layers with 400 neurons per layer, uses `tanh` activation functions and was trained with a learning rate of $1.5 \cdot 10^{-5}$.

- `MultiONet`: The branch net features four hidden layers, the trunk net seven hidden layers, both with 150 neurons per layer. The output layer features 40 neurons per quantity for a total of 1160 outputs per network (which are then multiplied in the split scalar product described in Appendix A). The training was performed with `LeakyReLU` activation and a learning rate of $5 \cdot 10^{-4}$.

- `LatentNeuralODE`: The encoder features hidden layers of the sizes 184, 92 and 46, the final latent space layer has nine outputs. The decoder has the same architecture, but inverted in order. The training was performed with a learning rate of $5 \cdot 10^{-3}$, which was scheduled to decay to 10^{-5} over the course of the run. The activaton function of the neural ODE is `Softplus`, the encoder uses `ReLU`. The numerical integration scheme used is `Tsit5`.

- `LatentPoly`: The encoder features hidden layers of the sizes 200, 100 and 50, the final latent space layer has five outputs and uses `ReLU` activation. The decoder has the same architecture, but inverted in order. The latent-space polynomial has degree six, and the training was performed with a learning rate of $2 \cdot 10^{-3}$.

These settings are also stored in a surrogate configuration file for the `osu2008` dataset and will be used automatically for training unless disabled in the configuration file. To reproduce the results of the benchmark, use `seed: 42`.

B Datasets

CODES currently comprises five datasets, two from the context of astrochemistry and three lower-dimensional baseline datasets. All datasets are hosted on Zenodo and are automatically downloaded when required. The number of samples, timesteps and quantities is displayed in Table 2. They are stored as `hdf5` files containing the following data:

- Training, validation and test data stored as three separate numpy arrays. Training data is used to train models, validation data is used to compute intermediate accuracy and loss values (and for Optuna optimisation) while test data is used for the evaluation after training the models.

- The number of training, validation and test samples as well as the number of timesteps and evolving quantities (e.g. chemicals), all stored as integers.

- Optionally, timesteps and labels for the quantities can be included. The former are stored as an array of floating-point numbers, the latter as an array of strings. When provided, they will be used in plots in the benchmark.

The following datasets are currently provided in the benchmark:

- `osu2008` was generated from a chemical reaction network with 224 reactions among 29 chemical species, simulated at constant temperature over 100 time steps. Exemplary trajectories of this dataset can be found in Figure 1. For more details, refer to [11].

- `branca24` is by far the largest dataset in the benchmark. It describes the evolution of ten quantities over 16 timesteps. The dataset is a 1 % subset of the dataset used by Branca & Pallottini in [3], obtained by shuffling and slicing the original dataset.

- `lotka_volterra` A six species lotka-volterra system, featuring three predator and three prey species, integrated over 100 time steps. The corresponding ODE system has the following equations

$$\frac{dp_1}{dt} = 0.5p_1 - 0.02p_1q_1 - 0.01p_1q_2,$$

$$\frac{dp_2}{dt} = 0.6p_2 - 0.03p_2q_1 - 0.015p_2q_3,$$

$$\frac{dp_3}{dt} = 0.4p_3 - 0.01p_3q_2 - 0.025p_3q_3,$$

$$\frac{dq_1}{dt} = -0.1q_1 + 0.005p_1q_1 + 0.007p_2q_1,$$

$$\frac{dq_2}{dt} = -0.08q_2 + 0.006p_1q_2 + 0.009p_3q_2,$$

$$\frac{dq_3}{dt} = -0.12q_3 + 0.008p_2q_3 + 0.01p_3q_3.$$

- `simple_ode` This is a simple, five dimensional ODE system integrated over 100 time steps.

$$\frac{dn_0}{dt} = -0.8n_0 - 0.2n_0n_2,$$

$$\frac{dn_1}{dt} = 0.8n_0 - 0.5n_1 + 0.4n_0n_2,$$

$$\frac{dn_2}{dt} = 0.5n_1 - 0.2n_0n_2,$$

$$\frac{dn_3}{dt} = 0.2n_0 + 0.625n_1,$$

$$\frac{dn_4}{dt} = 1.6n_0n_2 - 0.5n_0n_2.$$

- simple_reaction is a six dimensional ODE system which resembles a simple chemical rate equation ODE system, integrated over 100 time steps.

$$\frac{ds_1}{dt} = -0.1s_1 + 0.1s_2,$$

$$\frac{ds_2}{dt} = 0.1s_1 - 0.15s_2 + 0.05s_3,$$

$$\frac{ds_3}{dt} = 0.15s_2 - 0.1s_3 + 0.03s_4,$$

$$\frac{ds_4}{dt} = 0.1s_3 - 0.07s_4 + 0.01s_5,$$

$$\frac{ds_5}{dt} = 0.07s_4 - 0.05s_5,$$

$$\frac{ds_6}{dt} = 0.05s_5.$$

Table 2: Dataset Size Overview

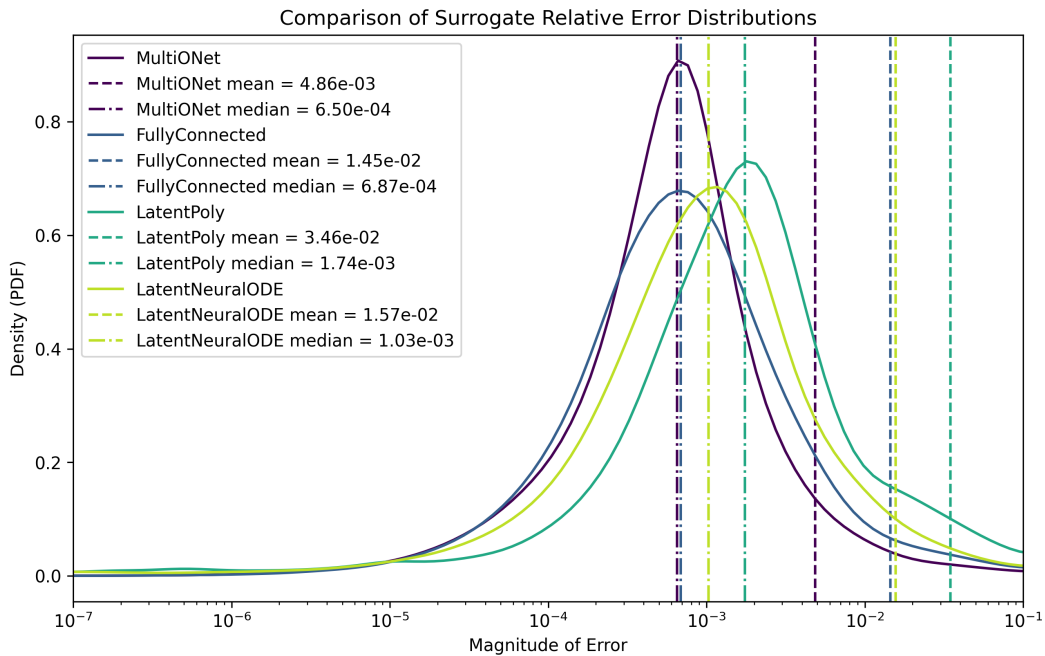| Dataset	Samples	Train/Val/Test Split	Timesteps	Quantities
osu2008	1000	750 / 50 / 200	100	29
branca24	671089	503316 / 33554 / 134219	16	10
lotka_volterra	700	500 / 50 / 150	100	6
simple_ode	700	500 / 50 / 150	100	5
simple_reaction	700	500 / 50 / 150	100	6

C   Benchmark Plots

Figure 4: Smoothed histogram plots of the distribution of the relative errors per model alongside their mean and median relative errors.

This section contains a selection of additional plots generated during a benchmark run. Note that this is not an exhaustive list, but rather an overview of some additional capabilities of the benchmark. All plots below relate to results obtained on the osu2008 dataset during the same benchmark run detailed in section 3 with hyperparameters described in Appendix A.
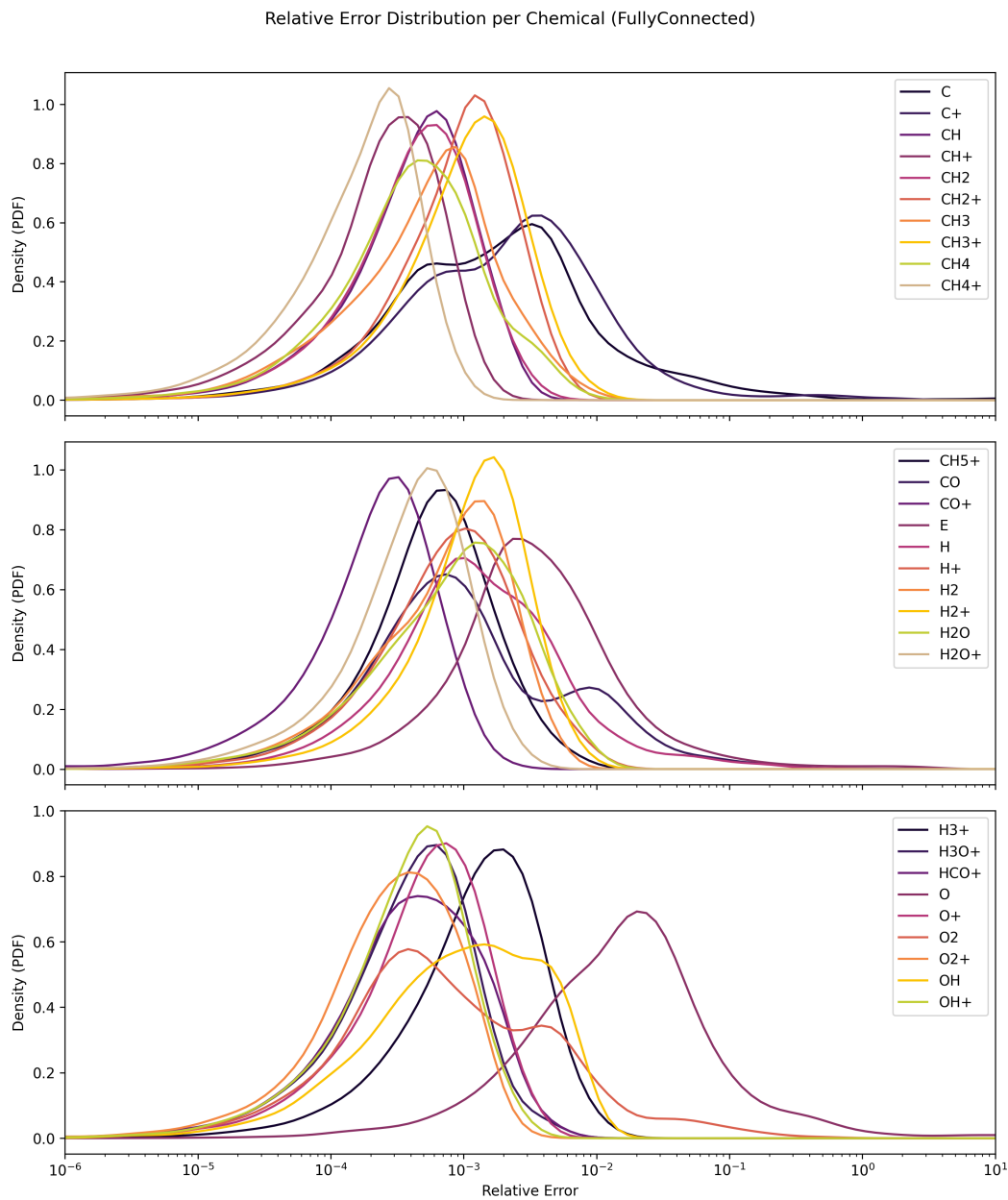
Figure 5: Smoothed histogram plots of the relative error distribution of `FullyConnected` for each quantity (chemical) in the `osu2008` dataset.

Figure 4 displays smoothed histogram plots of the relative error distribution of each model on the entire test set. The vertical lines indicate the mean and median error of the respective model. The superior accuracy of `FullyConnected` and `MultiONet` described in section 3 is confirmed here, the peak of the distribution as well as the mean and median lines are lower than those of `LatentNeuralODE` and `LatentPoly`.

Figure 5 shows an exemplary relative error distribution per quantity for `FullyConnected`. The benchmark generates a corresponding plot for each surrogate. These plots are useful for getting a feeling for how the surrogate performs on each quantity in the dataset. Notably, some quantities skew quite far towards large relative errors (O), small relative errors (CH4+) or exhibit two distinct peaks (CO).
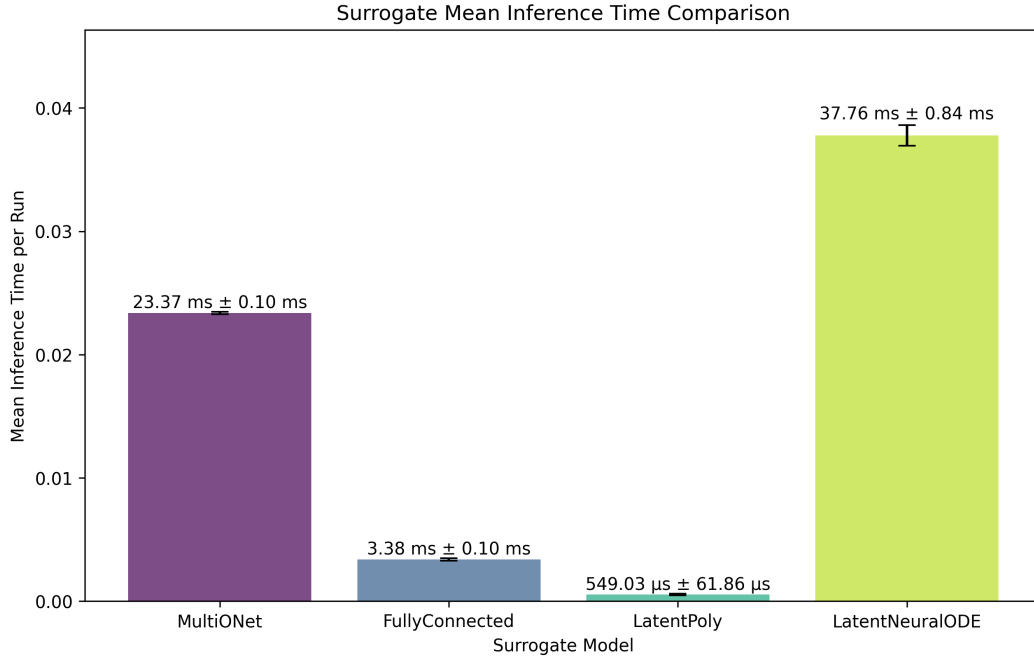
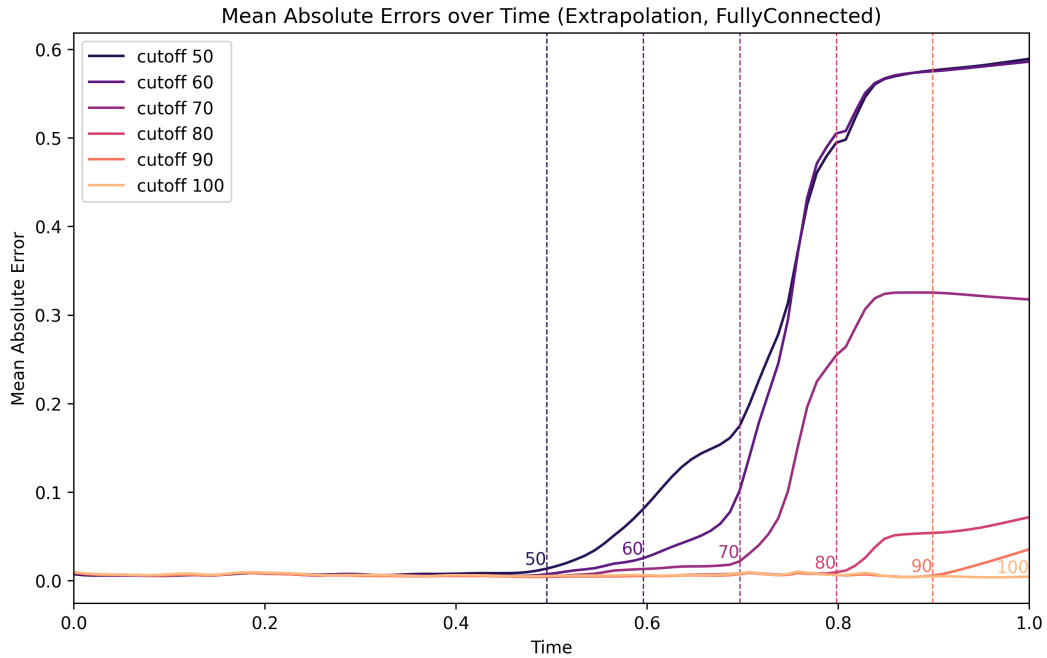Figure 6: Bar plot of the inference times of all models for predicting the entire test set once.

Figure 7: Mean absolute errors over time of `FullyConnected` for different `extrapolation` cutoffs.

Figure 6 displays a bar plot of the inference times, which are measured as the time required for one full prediction of the test set (200 datapoints, 100 timesteps each). The errors were calculated by timing this duration five times to obtain a mean and standard deviation. Fluctuations in the inference times are very low and should result mostly from slight differences in the computational load of the processors or GPU during inference.

Figure 7 is another model-specific plot, showing the mean absolute error over time for `FullyConnected` in the `extrapolation` modality for the different time cutoffs the model was trained with. One can clearly discern how error increases over time once the model predicts timesteps that were not included in the training set. Similar plots are also generated for the modalities `interpolation`, `sparse` and `UQ` for all surrogatess.

Figure 8 displays two kinds of heatmaps for each surrogate to provide further insights into their correlational behaviour. The heatmaps are to be understood as a two-dimensional histograms with 100 x 100 bins. Notably, the colormap is applied on a log-scale due to the large disparities between dense and empty cells. The y-axes in both plots display the absolute prediction error.

Figure 8a displays the correlation between gradient magnitude and prediction errors, hence the x-axis shows the normalised absolute gradient. Overall, `LatentNeuralODE` and `LatentPoly` show broader scattering than `FullyConnected` and `MultiONet`, indicative of higher prediction errors for points with higher gradients.

Figure 8b shows the correlation between the predicted uncertainty (using DeepEnsemble) and prediction errors, hence the x-axis is the predictive uncertainty. A perfect uncertainty quantification method would always predict the actual prediction error, signified here by the white diagonal line. Notably, the overconfidence of `FullyConnected` discussed in section 3 is clearly visible in the corresponding heatmap. The autoencoder-type models again exhibit broader scattering towards larger prediction errors and predictive uncertainties which is related to their lower overall accuracy. But for UQ, quantifying the PE is more relevant than achieving a low PE, and the autoencoder models seem to do this more reliably. It is worth noting that a large fraction of the total counts falls into the cell in the lower-left corner for all surrogates, as many of the PEs and corresponding uncertainties are too small to resolve.
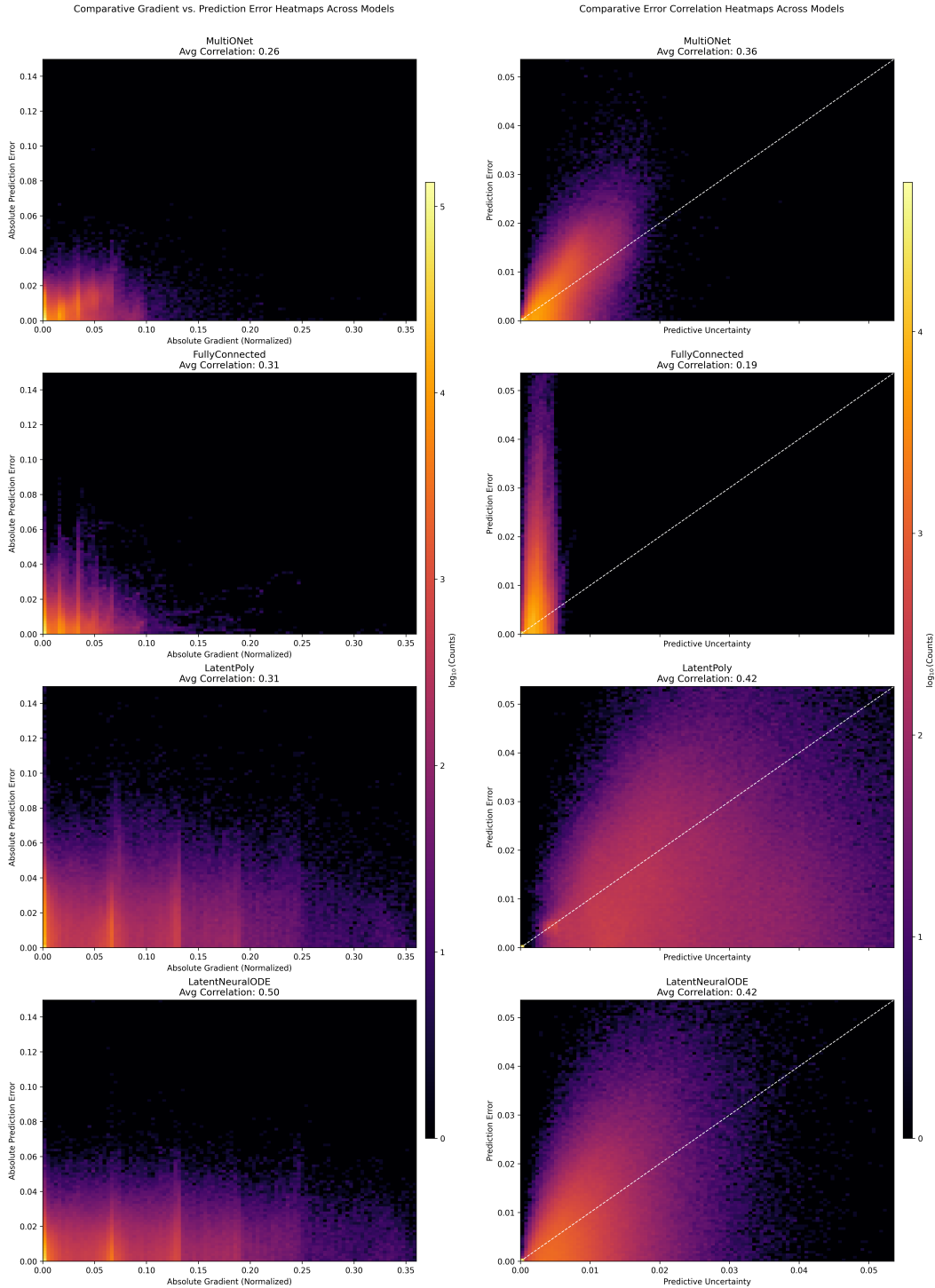
(a) Gradient-error correlation heatmaps.

(b) Uncertainty-error correlation comparison.

Figure 8: Heatmaps visualising the correlations between **a)** absolute prediction error and the (absolute and normalised) gradients of the data and **b)** the prediction error and predictive uncertainty (determined as the $1\,\sigma$ interval of DeepEnsemble predictions) of each surrogate.