Towards Measuring Predictability: To which extent data-driven approaches can extract deterministic relations from data exemplified with time series prediction and classification

Anonymous authors
Paper under double-blind review

Abstract

Minimizing loss functions is one important ingredient for machine learning to fit parameters such that the machine learning models extract relations hidden in the data. The smaller the loss function value on various splittings of a dataset, the better the machine learning model is assumed to perform. However, datasets are usually generated by dynamics consisting of deterministic components where relations are clearly defined and consequently learnable as well as stochastic parts where outcomes are random and thus not predictable. Depending on the amplitude of the deterministic and stochastic processes, the best achievable loss function value varies and is usually not known in real data science scenarios. In this research, a statistical framework is developed that provides measures to address predictability of a target given the available input data and, after training an machine learning model, how much of the deterministic relations have been missed by the model. Consequently, the presented framework allows to differentiate model errors into unpredictable parts regarding the given input and a systematic miss of deterministic relations. The work extends the definition of model success or failure as well as convergence of a training process. Moreover, it is demonstrated how such measures can enrich the procedure of model training and guide the combination of different models. The framework is showcased with time series data on different synthetic and real world datasets. The implementation of the used models and measures for quantifying the deterministic relations are provided via the git repository .... (the repository will be published and the link will be provided in case of acceptance, but for the review process it is provided as a supplementary zip-file)

1 Introduction

Data analysis and the application of the corresponding insights work if there are reliable and stable relations between the measured quantities and their model. However, due to the fact that any measurement may not be error free or the dynamics that determine the values of the considered quantities may be inherently noisy to a certain extent, measurement values do not only purely reflect the relations to be investigated. Furthermore, the input data could miss relevant information, e.g., relevant features are not provided or even measured, to model the output data such that regarding the given input data some parts of the output are unpredictable due to the lack of information. Usually, real world datasets are not binary in terms of being predictable, meaning that there are some deterministic patterns plus patterns which can't be inferred from the provided historic data, like financial markets solely based on its history, and therefore there is no chance for any model to predict them totally accurate. Depending on the relative magnitude of these elements, the prediction error can vary even for a successful model. We extend the term for model success by its ability to extract or learn, respectively, all the available deterministic relations. Consequently, we argue that the magnitude of prediction errors alone does not unequivocally signify the model success or failure.

One key issue before any data analysis, machine learning (ML) model training or information extraction from a dataset is testing if there are such reliable relations between the quantities in the dataset of interest. Such tests deliver valuable insights to evaluate efforts for further analysis, and if reasonable at all. A second key aspect after an iteration of data analysis or information extraction, like the training of an ML model, is testing if there is information left to extract or all the reliable information is extracted in terms of deterministic relations between input and output data. If all relations are extracted or learnt, respectively, the deviations between the predictions given the input data and the target (ground truth) are supposed to be stochastically independent of the input data. Thus, given that input, additional analysis on this input-target relation might not reveal further insights or improve accuracy in terms of extracting more deterministic relations. Consequently, further training might not improve a model in terms of learning these deterministic relations.

Following the outline above, in the presented work, we provide a framework to address the following, which is illustrated in Figure 1:

- We utilize measures to evaluate the stochastic dependence and information content between random variables modeling the data to evaluate to what extent the data is interconnected and to quantify extractable information based on defined input and output (target) variables assembled from the dataset.
- After fitting a model, we use these measures to estimate if there is still information left to extract. This evaluation is done by testing the stochastic independence and information content between the input and the deviations of the model output and the ground truth. For this purpose, we investigate the relation of random variables that model corresponding quantities.

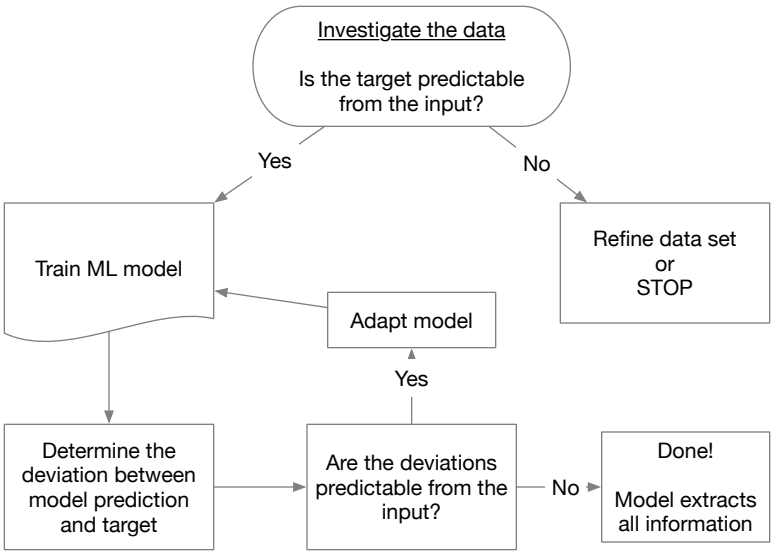

Figure 1: Graphical abstract representing the main concept of the presented work to analyze for information content and information extracted or learnt, respectively, by a model.

These bullet points provide the foundation of the framework that we endeavor to establish for characterizing failure cases in prediction modeling. A failure case is herein defined as a scenario in which a prediction model fails to capture the essential, meaning deterministic, dynamics of the target variable(s), given the input information. This definition of failure allows us to decide on whether there is potential for improvement or the prediction model has already achieved the best possible accuracy given the data. Our mathematical framework provides an additional insight for model evaluation and extends the current performance measures of ML models.

Furthermore, our framework can consequently provide an explanation in case there is a bad model accuracy by analyzing the deviations if there is information left or deviations consist only of unpredictable parts from

the perspective of the input data. The framework is not limited to a specific model architecture and thus is model-agnostic since we only inspect the dependence of input and the model errors, which does not require the knowledge of the inner function of the model.

The implementation of this general framework provided with this work is the mutual information, the chi-square test of independence and the Pearson correlation to investigate the existence of relations between random variables. One variable could be an input feature and one variable could be an output feature to be predicted by an ML method. In case of several input or output variables, the sum of all pairwise considerations between input and output variables serves as a measure for information to extract. This approach is entirely data-driven and needs no prior knowledge about the data and its distribution. Our framework is not limited to these tests and we provide pros and potential cons of the pairwise approach in the Discussion as well as other implementations of mutual information estimators that can be found in the Related Works section. Consequently, other methods that test for information content or stochastic dependence can be used and easily included into our modularized git repository that is provided with this work. One of the main intention of this work is to apply such measures to demonstrate how to and evaluate potential model success as well as to differentiate model inaccuracies into systematic failures and unpredictable parts with respect to the input data. The choice of the measures and their implementations themselves are not the focus of this work.

In analogy to the idea of using information and stochastic dependence measures for feature selection, we apply the same methods to the input and the deviations between the model prediction and the ground truth.Our approach for measuring the predictability of output based on input variables is similar to feature selection methods, like the minimize redundancy maximize relevance (mRMR) method Peng et al. (2005), which is based on mutual information. If input and these deviations take their values totally independent of each other, it means that there is nothing left to learn for an ML method, regardless of the loss value which might be high or low depending on the magnitude of the unpredictable component.

Evaluating the deviations for dependence on the input has several benefits for model training:

- Further convergence criterion: Apart from utilizing the loss function for convergence, we can stop training whenever there is no information or dependence left between the input and the deviation of a model prediction from the ground truth.
- Hyperparameter tuning: We can stop a grid search whenever we have identified a hyperparameter set with which the corresponding model has extracted all the information on the training or validation test set.
- Data efficiency: The total amount of training data can be used for fitting model parameters since no data is required to be utilized as an additional set for early stopping since overfitting can be detected directly on the training data with our framework.
- Decide if an inaccuracy measured by a loss function value is due to unpredictable parts given the input data or reflects a systematic failure of the model not to capture relevant information. Thus, we can decide if a loss function value is a corresponding lower bound for the given dataset since no further deterministic relations remain within the dataset.
- We provide a loss function agnostic framework to evaluate performance of ML accuracy, which can be used as a model selection criterion.

Firstly, we showcase our general framework with an application to time series forecasting. The basic procedure including our framework is the following. After quantifying the level of predictability by inspecting the input and ground truth target, we can perform an exactly similar test on the input and the residuals, which define the deviations as ground truth minus prediction, to see whether they are still predictable given the input. We use supervised ML techniques to predict future parts of the time series. Secondly, we showcase the application for time series classification similarly where there is nominal target data and the deviations between ground truth and model output is defined accordingly in Section 2.3 and Section 4.4.

Among different data modalities such as text, image, and timeseries, our insight is that in timeseries data, the existence of a clear relationship between input and output is often not directly noticeable. Therefore, in this work, we focus specifically on timeseries analysis. For instance, in natural language processing (NLP), we typically do not encounter a sequence of random words or tokens. Similarly, in image datasets, depending on the task (e.g., classification, segmentation), the relationship between input and output usually exists a priori, and one would not typically spend time proving such existence, instead would directly train a model to solve the task at hand. However, in timeseries analysis, the underlying dynamics of a process need to be learned from the measurements. Sometimes, the history of measurements provides little to no information about its future, leading to future samples being mainly or at least partially independent of past samples and therefore being unpredictable. For example, consider the prediction of stock prices using timeseries data. Past stock prices may not always provide a clear indication of future prices due to the complexity of market dynamics and the influence of external factors such as economic events and investor sentiment. Therefore, accurately predicting future stock prices requires understanding and modeling the underlying patterns and dynamics in the timeseries data, which may not be directly evident from historical observations alone. Without having a stopping criterion for improvement of the model in such scenarios, one could spend a huge amount of time on learning dynamics which either don't exist (such as a pure noise) or it is impossible to learn because the given history is sharing no or low information in that regards. Therefore, in such cases, we should know the upper bound of the model's performance to avoid trying to improve it while further improvement is not possible. One example is prediction of workloads, mainly network traffic in datacenters:
"measurement studies found two key results with implications for the network design. First, the traffic patterns inside a data center are highly divergent..., and they change rapidly and unpredictably." Greenberg et al. (2009). A more recent manifesto Buyya et al. (2018) re-iterated the brittleness of existing *"demand estimation and workload prediction methods"*, leaving it as an open question if *"Machine Learning (ML) and Artificial Intelligencen(AI) methods could fully address this shortcoming"*.

Therefore in this paper we would like to quantifiable answer this question: To what extent data-driven approaches can extract deterministic relations from data. Our solution is simple yet intuitive and effective. By checking the independence of residual error from the given input, we are able to judge if further improvement of models and results is still possible.

Paper outline The work is organized as follows: The main theoretical concepts that are used in the proposed framework are in detail described in Section 2. This includes a precise definition of the measure of mutual information and the chi-square test of independence, as well as an equinumeric discretization scheme for features with a continuous co-domain. Furthermore, a stacking architecture how models may be combined to extract iteratively all the information is defined.

A section of related methods follows this Methods section and addresses how our work extends related works using mutual information, ensemble learning and time series prediction.

Applications of the information extraction evaluation of a model is showcased in Section 4 with different experiments in the area of time series prediction and classification. These experiments include a basic proof of concept, an analysis of the influence of the loss function on the information extracting depending on the structure of the noise, suggest additional convergence criteria based on the presented framework, stacking of models as well as detecting distribution shifts caused by the existence of different deterministic relations.

In the Discussion, we provide assumptions and limitations of our current approach. Furthermore, we sketch further application of our framework in the area of unstructured data. Moreover, efficient model size reduction is also discussed by ranking subparts of a model with stochastic measures since the presented framework is general and can be applied to any function generating output from input data.

2 Theoretical background and Methods

In this section we provide the necessary background and mathematically establish our framework.

2.1 Basic concept for unpredictability in a nutshell

Definition of unpredictabilty
The random variable Y is deemed unpredictable with respect to an information set given by the random variable X if the conditional distribution $P(Y = y|X = x)$ aligns with the unconditional distribution $P(Y = y)$

$$P(Y = y|X = x) = P(Y = y) \tag{1}$$

for all x and y. Specifically when X comprises the past realization of Y, Equation 1 suggest that having knowledge about these past realizations does not enhance the predictive accuracy of Y. It is important to note that this form of unpredictability in Y is an inherent attribute, unrelated to any prediction algorithm Bezbochina et al. (2023). One famous example of unpredictable time series is white noise where samples are identically but independently distributed (iid), where independence of future samples from the past samples makes it essentially unpredictable and therefore training an ML model is pointless. In this case, the best predictor in terms of L^2-loss is a mean predictor, suggesting further investing on improving the prediction model is fruitless. Please note that while unpredictable data and noisy data are related, they are not equivalent. Noise represents a specific subset of unpredictability.

On the other hand, if Equation 1 doesn't hold true, it suggests that given X, it is reasonable to train an ML model to predict Y. The more the distributions of the two sides of Equation 1 deviate from each other, the more chance we have to train a model with a high accuracy to predict Y based on X. In the scope of this work, we call X the context/input variable and Y the target/output that we want to predict. In this work, we use the terms input and context interchangeably, analogously the terms output and target. In the following part, we explain how to measure and quantify the concept we have introduced so far.

Measuring predictability
Assuming Equation 1 is satisfied, we can derive the joint distribution by

$$P(Y = y, X = x) = P(Y = y)P(X = x) \tag{2}$$

for all x and y by utilizing the definition of conditional probability. However, in practical data science scenarios (2) barely holds entirely true even in case of independence of X and Y due to noise or numerical errors. Therefore it is crucial to quantify the degree to which the independence assumption is met and to introduce statistical concepts to decide based on a level of significance if the hypothesis of independence cannot be rejected.

Our approach to quantifying the independence of these random variables is grounded in Equation 2, where we measure the deviation between its left and right-hand side. Although in general any measure of deviation can be used, in this work, we mainly focus on Kullback–Leibler divergence and chi-square test of independence as a measure of this deviation. In the former case, such deviation can be calculated based on the mutual information formula given by

$$\boxed{I(X;Y) = D_{\mathrm{KL}}(P_{(X,Y)} \| P_X \otimes P_Y)}$$

where D_{KL} is the Kullback–Leibler divergence, $P_X \otimes P_Y$ denotes the outer product distribution, and $P_{(X,Y)}$ is the joint distribution (see Murphy (2022) section 2.2.4 Figure 2.3). A higher value of mutual information represents higher predictability of Y based on X which aligns with the definition of mutual information, measuring the information gained about Y through observation of X.

Quantifying measure of success
We define a successful prediction when the deviations between model output and ground truth, e.g., the residual defined as the ground truth minus the prediction, are stochastically independent of the context/input information. In case of independence, the deviations contain no information shared with the context, rendering them effectively unpredictable based on the the provided context information. Consequently, no further improvements are possible, marking the prediction as successful. It is crucial to note that as long as

some mutual information remains, there is potential for enhancements, whether through selecting a different model, loss function, or adjusting various parameters. For the sake of assessment, lower values of mutual information suggest better prediction quality.

We remark that the scope of this work does not include the development of new methods to calculate, estimate or approximate mutual information but the application of such methods to improve ML training. In the Related Work section, we provide references to such methods. For the demonstration of the application, we provide one implementation that worked for our experiments. However, our framework does not rely on a specific implementation of mutual information and thus any method to calculate mutual information can be taken. Furthermore, our provided git repository is modularized, ensuring that it can be easily extended by further functions calculating mutual information (or any other measure calculating stochastic dependence or information content).

A rigorous mathematical formulation, including details about the provided implementation of the framework, is given in the following.

2.2 Foundations and implementation details for the predictability framework

In this section, we explain our framework in detail.

Measures for stochastic independence and information content
We are given $n \in \mathbb{N}$ input features in the format $x \in \mathbb{R}^n$ and $m \in \mathbb{N}$ output features in the format $y \in \mathbb{R}^m$. Each feature is modeled as a random variable taking values upon measurement. Consequently, the set of input random variable is given by $X = (x_1, ..., x_n)$, $x_i : \mathbb{R} \to Z_{x_i}$, $t \mapsto x_i(t)$, $i \in \{1, ..., n\}$ where t is a time point of measurement and $Z_{x_i} \coloneqq \left\{z_{x_i}^k\right\}_{k=1,...,l_{x_i}}$, $l_{x_i} \in \mathbb{N}$, is a set of discrete events. Such an event can be defined by the random variable taking a value between two predefined values. Analogously, the set of output random variables is defined by $Y = \{y_1, ..., y_m\}$, $y_j : \mathbb{R} \to Z_{y_j}$, $t \mapsto y_j(t)$, $j \in \{1, ..., m\}$ and $Z_{y_j} \coloneqq \left\{z_{y_j}^k\right\}_{k=1,...,l_{y_j}}$, $l_{y_j} \in \mathbb{N}$, is a set of discrete events. Our framework holds not only for random variables with a discrete co-domain but also for random variables with a continuous co-domain. We will later present an algorithm that discretizes random variables with a continuous co-domain equinumerically. Furthermore, independent of the topology of the co-domain, we define an information or a dependence measure by $\Phi : X \times Y \to \mathbb{R}$, $(x_1, ..., x_n, y_1, ..., y_m) \mapsto \Phi(x_1, ..., x_n, y_1, ..., y_m)$ that describes how much information, resp., dependence exists between the input and the output. An example for such a measure can be the stochastic independence of multiple real valued random variables, see ,e.g., (Gallager, 2013, 1.3.4).

In this work, we focus on a specific structure of Φ, which is a pairwise test between input and output random variables providing corresponding information or stochastic dependence summing up each value of the pairwise measure. The measure Φ given by

$$\Phi(x_1, ..., x_n, y_1, ..., y_m) \coloneqq \sum_{i=1}^{n} \sum_{j=1}^{m} \phi(x_i, y_j)$$

where $\phi : X \times Y \to \mathbb{R}$, $(x_i, y_j) \mapsto \phi(x_i, y_j)$ for each $i \in \{1, ..., n\}$ and $j \in \{1, ..., m\}$.

We are aware that a pairwise test might be an approximation of the actual value of the measure for the deterministic relations, e.g., as in the case of the stochastic independence of multiple real valued random variables. However, this approximation provides the advantage of much less computational costs, in particular for large n and m as provided in the discussion section of Breitenbach et al. (2022). It is one outcome of this work that this approximation is a useful measure to estimate the deterministic connections between input and output as well as model deviations between predictions and ground truth. Moreover to estimate the learning success of a model where a consequence is the reduction of deterministic relations between the input dataset and the model deviations from the ground truth. A similar consideration holds for the mutual information where a precise calculation of the joint probability can be very costly in case of many input and output variables and where other mutual information estimators exist as well to circumvent this issue, please see the Discussion and Related works for references and further details about this issue.

In the present work, we focus on the mutual information and the chi-square test of independence between two random variables as measures, which are both explained later in detail. However, the presented work is generic and can also be executed with different measures for independence, like Pearson's correlation coefficient as defined in, e.g., Breitenbach et al. (2023) for random variables. We remark that in terms of testing a hypothesis if input and model deviations are independent of each other, it is beneficial to rely on several tests, e.g., if the requirements for the application of a test is not fulfilled.

In the following part, we explain the main ingredients of the present framework to analyze for deterministic relations. Although these concepts might be well-known, we repeat them here for the convenience of the reader since they are central for this work and a precise definition consistent with this work facilitates its understanding.

Mutual information
The probability $P\left(x_i = z_{x_i}^{k_1}\right)$ describes the likeliness that the outcome of x_i equals the event $z_{x_i}^{k_1}$ for any $i \in \{1, ..., n\}$ and any $k_1 \in \{1, ..., l_{x_i}\}$. Analogously for $P\left(y_j = z_{y_j}^{k_2}\right)$ for any $j \in \{1, ..., m\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$. The probability $P\left(x_i = z_{x_i}^{k_1} \wedge y_j = z_{y_j}^{k_2}\right)$ describes the likeliness that the outcome of x_i equals the event $z_{x_i}^{k_1}$ and the outcome of y_j equals the event $z_{y_j}^{k_2}$ for any $i \in \{1, ..., n\}$, $j \in \{1, ..., m\}$, $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$. Based on this definition, we can define the mutual information for a pair of random variables x_i and y_j as one example for ϕ as follows

$$I\left(x_i, y_j\right) \coloneqq \sum_{k_1=1}^{l_{x_i}} \sum_{k_2=1}^{l_{y_j}} P\left(x_i = z_{x_i}^{k_1} \wedge y_j = z_{y_j}^{k_2}\right) \log_a \left(\frac{P\left(x_i = z_{x_i}^{k_1} \wedge y_j = z_{y_j}^{k_2}\right)}{P\left(x_i = z_{x_i}^{k_1}\right) P\left(y_j = z_{y_j}^{k_2}\right)} \right)$$

for any $i \in \{1, ..., n\}$ and $j \in \{1, ..., m\}$ with the basis $a \in \mathbb{N} \setminus \{1\}$ of the logarithm. The mutual information describes how much information about the outcome of y_j, we gain given the outcome of x_i. If the outcome of x_i is independent of y_j, namely

$$P\left(x_i = z_{x_i}^{k_1}\right) = P\left(x_i = z_{x_i}^{k_1} | y_j = z_j^{k_2}\right) \coloneqq \frac{P\left(x_i = z_{x_i}^{k_1} \wedge y_j = z_{y_j}^{k_2}\right)}{P\left(y_j = z_{y_j}^{k_2}\right)},$$

for all $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$, where $P\left(x_i = z_{x_i}^{k_1} | y_j = z_j^{k_2}\right)$ is the conditional probability that $x_i = z_{x_i}^{k_1}$ under the condition that $y_j = z_j^{k_2}$, we expect zero mutual information since $\log_a 1 = 0$.

The mutual information is bounded from below by 0. The upper bound depends on the number of events of y_j. In order to normalize the mutual information such that it is bounded from above by 1 for any y_j, we define the corresponding log base $a = l_{y_j}$. We remark that in case where y_j is replaced by the corresponding model deviation, the basis is defined accordingly to the number of different events of the model deviation regarding the ground truth. The normalization is in particular important to compare the information content between x_i and y_j with the left information content between x_i and the corresponding deviation between model output and ground truth after the training of the ML model to estimate the information extraction of the model from the dataset.

As a next example, we introduce the chi-square test of independence of two random variables.

Chi-square test of independence
In case the chi-square test of independence is taken as the measure for stochastic independence, then ϕ returns 1 if the corresponding random variables of the pair are not independent of each other, else 0. Let us have $N \in \mathbb{N}$ measurements where at each measurement the values of all random variables are determined. For any fixed $i \in \{1, ..., n\}$ and $j \in \{1, ..., m\}$, we define

$$P\left(x_i = z_{x_i}^{k_1} \wedge y_j = z_{y_j}^{k_2}\right) \coloneqq \frac{O_{k_1 k_2}}{N}$$

where $O_{k_1k_2} \in \mathbb{N}$ is the number of observed measurements where $x_i = z_{x_i}^{k_1}$ and $y_j = z_{y_j}^{k_2}$ for the corresponding $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$. Then, we have

$$P\left(x_i = z_{x_i}^{k_1}\right) = \sum_{k_2=1}^{l_{y_j}} \frac{O_{k_1k_2}}{N} \text{ and } P\left(y_j = z_{y_j}^{k_2}\right) = \sum_{k_1=1}^{l_{x_i}} \frac{O_{k_1k_2}}{N}$$

with $N = \sum_{k_2=1}^{l_{y_j}} \sum_{k_1=1}^{l_{x_i}} O_{k_1k_2}$. Under the hypothesis that the random variables x_i and y_j, $i \in \{1, ..., l_{x_i}\}$ and $j \in \left\{1, ..., l_{y_j}\right\}$, are stochastically independent, the number of expected measurements $E_{k_1k_2} \in \mathbb{R}$ where $x_i = z_{x_i}^{k_1}$ and $y_j = z_{y_j}^{k_2}$ is given by

$$E_{k_1k_2} \coloneqq P\left(x_i = z_{x_i}^{k_1}\right) P\left(y_j = z_{y_j}^{k_2}\right) N$$

for the corresponding $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$. Consequently, if x_i and y_j are independent, it is necessary that observed and expected number of measurements for all $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$ equal each other. The chi-square statistic given by

$$\chi^2 \coloneqq \sum_{k_1=1}^{l_{x_i}} \sum_{k_2=1}^{l_{y_j}} \frac{\left(O_{k_1k_2} - E_{k_1k_2}\right)^2}{E_{k_1k_1}}, \tag{3}$$

equals zero if $O_{k_1k_2}$ and $E_{k_1k_2}$ equal each other for all $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$ and quantifies the deviation from not being equal. However, due to the presence of noise, even under independence of x_i and y_j, it might hold that $\left(O_{k_1k_2} - E_{k_1k_2}\right)^2 > 0$ for some $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$. Consequently, we need to estimate from a distribution how likely the observed chi-square value under the hypothesis of independence is. If the observed value is too unlikely, we rather assume that the opposite of our hypothesis of independence is the case, meaning the variables depend on each other and there exists a dependence between the random variables in taking values. It can be proven that χ^2 is chi-square distributed with $\left(l_{x_i} - 1\right)\left(l_{y_j} - 1\right)$ degrees of freedom, see, e.g., (Rao, 1973, 6d.2) or (Georgii, 2015, 11.3). One important assumption is that the term $\frac{O_{k_1k_2} - E_{k_1k_2}}{\sqrt{E_{k_1k_1}}}$ is approximately normally distributed, which is usually sufficiently the case if $E_{k_1k_2} \geq 5$ for all $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$, see, e.g., McHugh (2013) or (Greenwood & Nikulin, 1996, page 21). Based on the distribution, we can calculate a p-value for the observed χ^2 value. The p-value is the probability to get a higher chi-square value than the observed one. If the p-value is too small, e.g., for this work we use lower than the level of significance of 0.01, we reject the hypothesis and assume that the random variables take their values not independent of each other. Our measure of dependence is the number of chi-square tests that indicate dependence of the tested pair while testing each pair of input and output variables (x_i, y_j) for each $i \in \{1, ..., n\}$ and $j \in \{1, ..., m\}$. However, since the number of chi-square tests is given by nm, which can scale to large numbers, we use an adapted p-value to lower the risk of wrongly rejected hypothesis, which would result in assuming too often dependence. We use the Bonferroni-correction dividing our level of significance 0.01 by the number of chi-square tests mn.

Discretization scheme for co-domains or random variables

In the next part, we describe how to discretize real valued continuous random variables, meaning where the co-domain is continuous. We take Algorithm 4 from Breitenbach et al. (2022). We remark that our algorithm also works for real valued discrete random variables without any change. Consequently, no separation between discretized and continuous random variables is necessary. The algorithm provides an adaptive discretization scheme for each random variable, meaning that boundaries of the bins, in which the co-domain of a random variable is divided and define the events like $z_{x_i}^{k_1}$ or $z_{y_j}^{k_2}$, are not set equidistant but rather equinumeric. With equinumeric, we mean that each bin has - if possible - the same number of data points which balances the likeliness of each event.

Algorithm 1 Discretize the co-domain of random variables

1. Set $\rho \in \mathbb{N}$ number of minimum data points per bin
2. For any random variable v
   (a) Determine the minimum value m and the maximum value M of all measured values of v
   (b) If $M \leq m$: Skip v
   (c) If $M > m$:
      i. Sort the measured data points of v in ascending order.
      ii. Go through the data points in ascending order. Determine the range of a bin such that there are at least ρ data points within the current bin and that the value of the last data point of the current bin is smaller than the first one of the next bin.
      iii. If there are less than ρ data points left: Join these data points with the last bin that has at least ρ data points.

Algorithm 1 works as follows. The parameter $\rho \in \mathbb{N}$ determines the minimum number of data points per bin in which the co-domain of a random variable is divided. The parameter ρ influences the margin probability $P\left(x_i = z_{x_i}^{k_1}\right)$ and $P\left(y_j = z_{y_j}^{k_2}\right)$ of the joint distribution $P\left(x_i = z_{x_i}^{k_1} \wedge y_j = z_{y_j}^{k_2}\right)$ and consequently the corresponding expected frequency $E_{k_1 k_2}$ as well as the number of bins for each random variable. Increasing ρ will increase the quantity $E_{k_1 k_1}$, which might be useful if $E_{k_1 k_1} < 5$ for one $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$. Consequently, this discretization scheme is beneficial for the our implemented chi-square test and can be used without any restriction for the calculation of the mutual information as well. We need to keep in mind that a too big ρ might lead to a too coarse discretization deleting information from the continuous random variable. In case of the chi-square test, one strategy might be to start with a small ρ where $E_{k_1 k_1} < 5$ for one $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$ and increase ρ until $E_{k_1 k_1} \geq 5$ for all $k_1 \in \{1, ..., l_{x_i}\}$ and $k_2 \in \left\{1, ..., l_{y_j}\right\}$. A further advantage of that scheme is that the by ensuring always a minimum number of data points in each bin of each marginal distribution, the denominator in the mutual information formula is never zero which could happen with a equidistant discretization strategy.

In step 2 a), we determine the minimum and the maximum of the available data points of the corresponding random variable v to filter out constant functions in 2 b). Without any variation, such random variables do not provide any information for predicting something for what at least two different kind of events are necessary. For non-constant random variables, we sort the data points of the random variable in ascending order. Once this is done, we can go through the points in ascending order and determine the boundaries of the bins such that at least ρ data points are included in a bin. If there are several data points with equal values, we include all these points into the current bin such that the first data point in the next bin is larger than all data points in the bin before. If the remaining data points, which are not yet associated to a bin, are less than ρ data points, we include them into the bin with the largest upper bound.

The binning generated by Algorithm 1 can be used for both the chi-square test to calculate corresponding marginal and joint probabilities and mutual information as we do in our implementation provided with this work.

Stochastic independence as a test of hypothesis

We conclude this section with a remark about the distribution of test statistics. Even under the hypothesis of independent data/random variables, there are some variations by coincident that lead to a distribution of the test statistic. Consequently, we need a probability how likely it is under the assumption of independent random variables (no deterministic relation between input and output) to get a test statistic value bigger than the observed one, and thus exclude that the relation in the measured data is just by coincident. Then, we can decide if a certain observed test statistic is too unlikely under the assumption of independent random variables and we should rather assume that that the opposite is true, meaning that there are some deterministic relations by whose action random variables do not take their values independent of each other. Based on such a statistical view, all models that cannot be rejected to have uncorrelated deviations from the ground truth with input are equally good in terms of extracting the deterministic relations.

While the chi-square value χ^2 is chi-square distributed with corresponding degrees of freedom under certain assumptions with which we can evaluate the observed chi-square value regarding the likeliness of being generated by two independent random variables, we are not aware of such a result for the mutual information. However, to get a threshold, such that we can decide based on a level of significance if an observed mutual information value is too unlikely under the assumption of independent data (input-output-relation), we can determine a distribution of mutual information numerically with the following permutation procedure. The idea is to shuffle the association of value pairs between input and output based on the available data according to $(x_i(t), y_j(\pi(t)))$ for all $i \in \{1, ..., n\}$ and $j \in \{1, ..., , m\}$ where $\pi : T \to T$, $t \mapsto \pi(t)$ is a bijective map, called permutation, and T is the set of all time points of measurements of the data points. The random association of pairs from different measurements is assumed to provide us a distribution of mutual information of independent input-output data to evaluate how likely the observed mutual information is assuming random input-output association. Even in case there is a strong deterministic relation, the shuffling is supposed to ensure that the corresponding dependence in taking the values is randomized. If the observed mutual information value is unlikely according to the distribution of mutual information values numerically determined based on the dataset, we should rather assume that the reason for the mutual information value is the non-random mechanisms relating input and output values or corresponding random variables, respectively, in case the output is replaced by the model deviation. The observed mutual information value is determined as unlikely under the hypothesis of independent data if less than a predefined percentage of mutual information values (level of significance; in this work 5%) generated with the randomly shuffled data is bigger than the observed mutual information value. The distribution is generated by calculating the mutual information value for several random permutations as described above. In this work, we calculate the mutual information 100 times with shuffled data. The same procedure holds to calculate a distribution for the chi-square value defined in (3) in case assumptions are violated to justify the application of the chi-square distribution for χ^2. With such a stochastic framework in place, we are able to make concrete decisions of further improvement is possible given the data or if all the learnable relations are already extracted.

2.3 Stacking architecture

As we see in the Results section 4, different properties of a model influence the capability of extracting deterministic relations. Consequently, we provide here an architecture to combine different models with different properties to systematically extract the deterministic relations. Roughly, models are stacked together where all models get the same input, however, try to learn only what the models in the stack of models so far have not extracted regarding deterministic relations.

We need to differentiate two cases. The first case is where the target is of ordinal character. Here the random variables modeling the deviations of the model output from the ground truth are defined by the difference between the model output and the ground truth. In this case the model deviations are termed as residuals. In case where the target is of nominal character, the random variables that model the deviations of the model are defined as follows. In case the prediction of the model is not correct, the random variable modeling the deviation of the model from the ground truth takes the value of the class label that would have been correct. In case the model is right, the corresponding random variable takes a value that does not represent a model class, e.g., a negative integer.

Next, we explain the stacking procedure in detail for both cases. We are given the data $\{(x_i, y_i), i \in I\}$ where x_i is the vector-valued input, y_i the corresponding vector-valued output and I is a finite subset of the natural numbers $\mathbb{N}$. With x and y we denote the corresponding vectors of random variables that take the corresponding values for a given i. First, we check if x and y have stochastic dependence. If yes, we train the first model/part of the model stack to best fit the prediction $\bar{y}$ to y given x. With $\bar{y}$, we denote a (vector-valued) random variable taking the values $\bar{y}_i$ for the corresponding $i \in I$.

For a regression task, i.e. with ordinal target data, the procedure looks as follows. If $\Delta y^1 := y - \bar{y}$ is not independent of x, there is still some information left that can be extracted. Consequently, we fit a model that may have properties different to the current model to learn these relations between x and the residuals Δy^1. In other words, the purpose of the second model on top of first model is to correct the prediction of the first model. The output of the two models is then the prediction of the first model $\bar{y}$ plus the correction Δy^1. In the next iteration, we test the residuals $\Delta y^2 := y - \bar{y} - \Delta y^1$ for stochastic dependence on x. This

procedure can be repeated until the corresponding residuals are stochastically independent of x. We call this procedure stacking of models and can be generalized as follows such that a new model on top of a stack learns to correct the prediction of the previous stack. The procedure is illustrated in Figure 2.

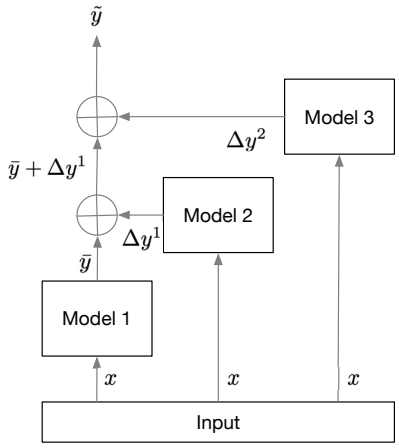

Figure 2: Stacking architecture for problems with ordinal target data.

To generalize, we define

$$\Delta y^j \coloneqq y - \sum_{k=0}^{j-1} \Delta y^k$$

where $\Delta y^0 \coloneqq \bar{y}$ for any $j \in \mathbb{N}$. In a purely deterministic scenario, we would extend the stacking until $\Delta y^j = 0$. In a real scenario where there are unpredictable parts in the data, our definition of no further improvement possible is that Δy^j is stochastically independent of x based on a statistic test or measure. The random variable representing the output of the total stack is denoted with

$$\tilde{y} \coloneqq \sum_{k=0}^{j-1} \Delta y^k,$$

taking the corresponding values $\tilde{y}_i$ upon the input x_i for $i \in I$, where j is the smallest number such that Δy^j is stochastically independent of x. Consequently, the stacking stops if $y - \tilde{y}$ is independent of x. This definition also works for discrete ordinal data, where the differences, resp., residuals take only discrete values. We remark that this architecture does not require more inference time since each model of the stack does not depend on the output of other layers but all perform their inference on the same input data and thus can be run in parallel.

For nominal target data (the difference between class labels has no meaning), the stacking architecture works analogously except the definition of the deviation of the model from the ground truth is different, please compare with Figure 3. The deviation of the model output from the ground truth is defined by a random variable θ^j, $j \in \mathbb{N}$ that take the value of the correct class in case of a wrong prediction from the model/stack below and a value that does not represent a discrete class of the ground truth in case the prediction of the model/stack below is correct. If θ^j is independent of the input x for one j, we can stop the stacking of more models. As long as θ^j is dependent on the input x, where θ^1 is based on the predictions of the first model, there is a deterministic relation that another model can learn to predict θ^j, which models a correction to the prediction of the model/stack below. In such a case the stack is extended by another model predicting θ^j. Based on the prediction $\bar{y}^j$ of the model/stack below and the prediction for the corresponding θ^j, we can decide during inference which prediction to take. In case of θ^j equalling a class label, we take the value of θ^j predicted by the corresponding model subsequent upstream in the stacking architecture as the value for $\bar{y}^{j+1}$. Otherwise, i.e. θ^j equals a number not representing a class label, the value of $\bar{y}^{j+1}$ equals the one of $\bar{y}^j$. The final prediction of the stack is denoted with $\bar{y}$ where all the variables represent the vector-valued

case as in the case above as well. We remark that the co-domain of the random variable θ^j is not one-hot encoded but the models output $\bar{y}^j$ should be due to the nominal character. However, in terms of the random variable θ, since there is a bijection between the co-domain of this random variable and the corresponding one-hot encoding, the information content or statistical independence between the random variable θ^j with a one-dimensional co-domain consisting of integer numbers and the input also exists.

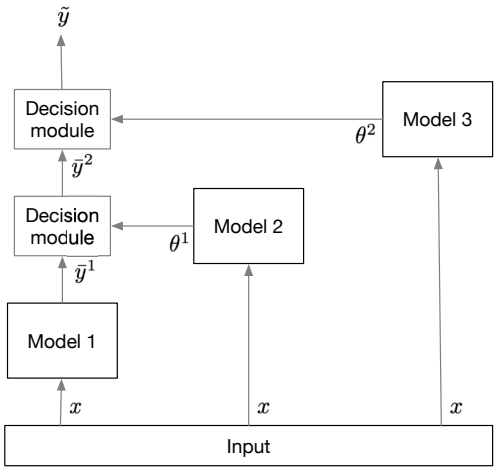

Figure 3: Stacking architecture for problems with nominal target data.

As a greedy implementation of both stacking concept above is to combine models randomly and check that after each new model Δy^j has less information with the input than Δy^{j-1} has with the input to ensure benefits of the new model on top of the stack. This is a test that the new model successfully extracts remaining information and doesn't do guessing rather than really filling in what is missing in terms of deterministic relations.

3 Related Work

Mutual information estimation and applications
Besides non-parametric methods to estimate mutual information such as Kraskov et al. (2004) and Gretton et al. (2005), a more recent neural network based estimation of mutual information Belghazi et al. (2018) is proposed. Such methods provide an alternative to Algorithm 1 for the calculation of mutual information. However, we remark that our implementation can be directly be used for the determination of the chi-square test as well such that we can also consider two statistical tests to decide for independent input and model deviations. DeepInfoMax (DIM) Hjelm et al. (2019) as well as contrastive predictive coding (CPC) Oord et al. (2018) maximizes mutual information between raw data and its compressed representation to find a better representation of raw images. Also Chen et al. (2016) employs mutual information to find a disentangled and interpretable representation of images. The work of Brakel & Bengio (2017) also focused on finding a disentangled/independent representation/features of images and uses mutual information as a measure of independence or better say to enforce this independence. Our framework can be directly applied to the corresponding generated representations to test if the model accuracy is only caused by unpredictable parts like noise. For more details, please see the Discussion about the case of unstructured input data.

Ensemble learning
Our proposed stacking procedure can be considered as an extension of ensemble learning methods such as Wortsman et al. (2022). However, our approach extends it in two key facets. Firstly, we refrain from the indiscriminate combination of multiple models. Our framework triggers a combination only if it confirms the presence of potential for further improvement in terms of further learnable/extractable deterministic relations. Secondly, our approach is characterized by progressiveness: We don't assign each model the task of learning the ground truth but rather focus on capturing what remains unlearned by the stack of previous

models. This progressiveness not only contributes to the efficacy of our stacking strategy but also enables each model to concentrate on specific tasks that are not covered by other models. For this purpose, we stack models in a way that the input is given to all models in a stack, and each model attempts to correct the errors left by the preceding models in the stack. In order to enable models to extract different information that the preceding models could not so far, the models should vary in their properties, like the model parameters, the loss function they are trained with or the architecture itself.

Information bottleneck
Another application of our framework is to extend the information bottleneck concept Saxe et al. (2019); Kawaguchi et al. (2023). The basic framework of the information bottleneck is to maximize the mutual information between a data representation sought and the corresponding output data and at the same time minimize the mutual information between this representation on the input data. There is a parameter to balance both contradicting requirements. With our framework, we can extend the information bottleneck method by providing a procedure how to choose this balance parameter to find a lossless compression of the input data. Starting with a configuration where the model extracts all the information between input and output data, we tune the balance parameter such that the compression is weighted higher until the model cannot extract all the information between input and output. With that procedure, we find the threshold of the balance parameter for a lossless compression.

Time series prediction
Besides the above related methods, the showcase of this work is in particular associated with time series prediction. In the past few years many of time series prediction models have been developed to improve the prediction performance, such as RNN, LSTM Memory (2010), GRU Cho et al. (2014), transformer and its variants Vaswani et al. (2017); Kitaev et al. (2019); Zhou et al. (2021); Liu et al. (2022); Zhang & Yan (2022); Wu et al. (2021); Zhang et al. (2022) and state space models such as RKN Becker et al. (2019), S4 Gu et al. (2021), MTS3 Shaj et al. (2023) as well as simple yet effective linear models such as Zeng et al. (2023). The improvement is measured in terms of corresponding loss function values of the best weight configuration of a model. We would like to extend these evaluation by our framework where we introduce another quantity that evaluates how much information is left to extract from the time series given the input length of historic data. Furthermore, our approach would like to establish a level before model training, that allows for evaluation if a time series is predictable given historic data. This is in particular important for challenging time series. Additionally, there is a discussion that a simple linear layer has a superior performance in certain time series prediction scenarios Zeng et al. (2023), notably periodic time series Li et al. (2023). Nevertheless, there is a question in all scenarios: How can we make sure that the current method has already achieved the lowest possible bound of the error for each time series or can it still be improved by fitting a better model for each case?

4 Applications|Numerical experiments

In this section, we showcase our framework to analyze time series for their predictability, measure dependence between input/context and output/target data. Furthermore, we demonstrate how to measure learning success of models with our framework and use these insights to build model stacks to extract more information from data than single models can do. Moreover, we show how we extend the performance evaluation of a model with the loss function by our framework and derive further criteria for convergence of the training of a model.

Apart from several real world datasets, we use a synthetic dataset to purely demonstrate some of the effects from above where we can control properties of the noise, like the ratio between information and noise, which we do not know in a real dataset. The synthetic dataset consists of the time series that is composed of the sinus function, $\sin : \mathbb{R} \to \mathbb{R}$, $t \mapsto \sin(t)$ and the random variable $\theta : \mathbb{R} \to \mathbb{R}$, $t \mapsto \theta(t)$, which generates noise by random values according to $y(t) \coloneqq \sin(t) + a\theta(t)$ where $a > 0$ scales the amplitude of the noise. In order to evaluate performance in an already established metric, we use the normalized root mean squared error (RMSE) defined by $\frac{1}{\sigma}\sqrt{\sum_{i=1}^{N}\left(\tilde{y}_i - y_i\right)^2}$ where $N \in \mathbb{N}$ is the number of measurement points, $\tilde{y}_i$ is the model prediction at the discrete time point $i \in \mathbb{N}$, y_i is the corresponding actual output data and σ is the standard

deviation $\sqrt{\sum_{i=1}^{N} (\bar{y} - y_i)^2}$ calculated on the training dataset where $\bar{y}$ is the mean of the values y_i of the training dataset.

The parameter ρ of Algorithm 1 is set to 5% of the number of measurement points of the corresponding dataset on which the algorithm is performed, unless otherwise stated.

We remark that the term residuals is usually used instead of model deviations to name the difference between model output and the ground truth in case of ordered target data. In case of nominal, we use further use the term model deviation. The model deviation in the nominal case is defined as as the class that would have been correct in case the model predicts the wrong class. If the model is correct, the model deviation equals a number that does not represent a class, e.g., the number -1. We will show that for both definitions the stochastic dependence and mutual information with the input decays over training epochs on training and validation datasets.

4.1 Splitting noise off from model inaccuracy

In practical scenarios where data is obscured by noise or cases that some components of the future samples are not predictable based on the history, evaluating prediction models becomes challenging. Traditional metrics such as L^2-loss on validation set may not suffice due to the uncertainty surrounding the ratio of unpredictable to predictable part. This uncertainty complicates determining the lower bound of prediction error beforehand. Therefore, solely relying on loss metrics for model assessment is insufficient, as the source of error could be either model inadequacy/failure or inherent data unpredictability. Therefore it is important to be able to distinguish between these two cases, because in the former case we have the chance to improve the prediction, however, in the latter case we cannot. For example, when the power of noise (as an unpredictable component) equals to half of the data power, the lowest possible normalized mean squared error (MSE) on validation is 0.5. However, by employing our framework that identifies when the model has learned the primary data component effectively, we can stop training and attribute the remaining residual to initial data noise, although the L^2 error is still high. Motivated by the this discussion, we start our experiments with such a data where the first component is fully predictable such as the sinusoid function plus some iid Gaussian noise that is completely unpredictable.

In the following, we demonstrate how our framework can be used to distinguish if the model inaccuracy results from noise rather than a relation between input and residuals that the current model has not learned yet. Since we need to vary the amplitude of noise compared to the deterministic relation, which is modeled by the sinus function in this case, we choose to work on our synthetic data described above.

For the numerical implementation, we consider a window of size 50 and the first 49 samples are considered as context/input to the model and the 50^{th} sample as the target/output. To generate the dataset, the window is slid over the time series that was split into training and test/validation set before the experiment according to a ratio of 0.75/0.25.

For the experiments depicted in Table 1, the random variable θ adds Gaussian noise. According to the table, chi-square test and the mutual information (at least in some cases) indicate that the MLP model, trained with L^2-loss, has extracted the sinus function. We see that all essential information are extracted because the chi-square test indicates that all residuals are independent of the input. In the case of mutual information, the p-values are greater than 0.01 indicating that based on that level of significance, we cannot reject the hypothesis that input and residuals are independent of each other and the input does not carry information about the outcome of the residuals. Therefore in these cases, even a much more sophisticated model is not able to decrease the error further and might lead to overfitting to the noise. While the RMSE increases with the amplitude of the noise, indicating that the model performance would become worse, our stochastic measures indicate that the model has extracted the sinus function as the main deterministic driver of the time series. This can be seen since the L^2-norm between model prediction and the pure sinus value, shown as Actual Err in 1, is (almost) independent of the noise amplitude. Furthermore, that the normalized test RSME is always close to the relative noise standard deviation, indicting that the deviation between model and data results from the noise and not from a systematic deviation from the sinus function. We remark that the first and third column can only be presented because we exactly know the signal and the noise

component separately. In a real world, these numbers usually cannot be computed, however, our method can provide the valuable insight if there is some relation left to extract or the deviation between model and data is rather due to numerical errors.

	Exp. Err		Chi-square Analysis		Mutual Information Analysis				
Rel. noise std	RMSE	Actual Err	Init. dep. var	Res. dep. var	Init. MI	Init. Perm	Res. MI	Res. Perm	pv
0	2.2×10^{-5}	2.2×10^{-5}	49	49*	27.497	0.7385 ± 0.0272	27.2781	0.4039 ± 0.236	0
0.2715	0.2731	0.0616	49	0	8.3351	0.7076 ± 0.0136	0.7301	0.7096 ± 0.0114	0.05
0.4927	0.4914	0.1073	49	0	4.193	0.7337 ± 0.0119	0.7643	0.7332 ± 0.0108	0
0.6465	0.6568	0.1476	47	0	2.5647	0.7024 ± 0.0104	0.7472	0.7049 ± 0.0095	0
0.7482	0.7461	0.1682	38	0	1.6659	0.7325 ± 0.0092	0.7503	0.7054 ± 0.0089	0
0.8166	0.8184	0.1789	37	0	1.2591	0.7293 ± 0.0095	0.7862	0.7296 ± 0.0085	0
0.8611	0.8595	0.1831	30	0	1.053	0.711 ± 0.0077	0.7562	0.7102 ± 0.0069	0
0.8921	0.9068	0.1893	28	1	0.9725	0.7275 ± 0.0086	0.7931	0.7278 ± 0.0075	0
0.9148	0.9134	0.1846	19	0	0.8825	0.7109 ± 0.0079	0.7605	0.7384 ± 0.0082	0.01
0.9306	0.9415	0.1934	11	0	0.8751	0.7341 ± 0.0075	0.7521	0.7070 ± 0.0073	0
0.9425	0.9410	0.1952	5	0	0.8153	0.7053 ± 0.007	0.7652	0.7042 ± 0.0085	0

Table 1: Impact of varying amplitudes of white noise on a sinusoidal signal and assessing its influence on the performance of a simple MLP. The relative power of the noise component, which is the root of the variance of noise divided by the variance of the total time series, is shown in the first column, which is the theoretical lower bound of test RMSE. In the second column, the normalized RMSE on the test data is shown. The third columns shows the L^2-loss between the prediction and actual clean sinusoid. The fourth column illustrates the dependency of the target on the history evaluated by chi-square test, which is initially complete until almost half of the power is taken by noise and decreases to small numbers when noise becomes the dominant (94 percent) part of the time series. The fifth column evaluates the dependency of residual target on the corresponding context to show how successful the model is to reduce this dependencies. The last five columns show the same concept in terms of mutual information. Similar to chi-square, we have initial and residual values as well as two more columns to report their corresponding lower bound which is obtained by random permutation tests. The last column shows the p-value of the observed mutual information under the hypothesis that residuals are independent of the input given the dataset. *Note:* * Initially might seem counter-intuitive, more plots are given in Figure 10. Due to some numerical error, some periodic patterns (only visible when multiplies by 10000 see top left plot in Figure 10) are remained in the residuals which are correctly detected as dependency by the chi-square test. Such cases can be handled by introducing a threshold indicating very close fit between model output and data in some norm. An alternative could be to impose a minimal bin width in Algorithm 1 which might impact the equinumeric property.

4.2 Information extraction influenced by loss function and noise properties

In this part, we show that depending on the noise properties, the loss function is an important key factor to extract the deterministic relations from the data. For this purpose, we use our synthetic dataset where the random variable θ is defined by $\theta(t) := \frac{1}{10}\theta_1(t) + 10\theta_2(t)$, θ_1 is Gaussian distributed with mean zero and standard deviation 1 and θ_2 is a Poisson distributed random variable over the number of peaks (high values of the time series) within a time interval. If at a time point t there is such a random peak, the variable $\theta_2(t) = 1$ and 0 otherwise. In this numerical experiment, the rate of arrivals of a peak is $\frac{1}{200}$ peaks per sample (or time between two data points), meaning that we expect one peak after 200 data points on average.

Since peaks only increase the values of the time series randomly, the mean of the noise is greater than zero, indicating asymmetry. In this case, we see in Table 2 that the initial dependence between input and output is totally reduced only by the MLP that is trained with the L^1-loss function while it does not extract the total deterministic relations when using an L^2-loss function. In this case, by saying "we extract the deterministic relations", we mean to learn/approximate the sinus-function with the model. This extraction has taken place when only noise is left that is independent of the input, as shown with Table 3 where we see that in the L^2-norm the model output trained on L^1-loss function is much closer to the corresponding sinus-function, evaluated on the test set. In other words, the results of Table 3 depict that the model trained with an L^2-loss function is prone to irregular peaks. For illustration, plots of predictions and residuals for L^1- and L^2-loss functions are given in Figure 11. We remark that also here the synthetic set is useful since we know the exact

functional formula that generates the data apart from noise. However, this example provides evidence that once a stochastic measure reports independence between residuals and input that the model has extracted the deterministic relations not having included spurious relations from noise into the learning.

-	# dep-test + L2	# dep-test+ L1	Initial MI	Residual MI + L2	Residual MI + L1
Trial 0	38	0	13.37	1.053	0.7524
Trial 1	28	1	13.567	0.9331	0.8099
Trial 2	49	0	13.394	1.217	0.7683
Trial 3	24	1	13.436	0.8886	0.7912
Trial 4	41	0	13.494	1.067	0.7451

Table 2: The effect of the choice of the loss function in mitigating asymmetric noise effects. All values are reported on the validation data. Initial number of dependent variables is 49 on the test set, i.e, the target depends on all past time series steps. The first two columns (dep-test L^2 and dep-test L^1) represent the dependence measured by the chi-square test of independence between input and residuals on the test set of a model trained with L^1- and L^2-loss function. The model trained with L^1-loss could better reduce these dependencies. Regarding mutual information (MI) all p-values are zero, however, as shown in Figure 4 further investigation shows that the p-value rises up to 10% for the L^1 model (orange curve) on validation set, although it always remains zero for the L^2 model. That means that the p-value could serve as a convergence criterion, since we cannot reject the hypothesis that the L^1 model with the corresponding weights extracts all the deterministic relations.

-	*actual*-Test-rmse-model-trained-with-L2	*actual*-Test-rmse-model-trained-with-L1
Trial 0	0.1114	0.04958
Trial 1	0.1221	0.04567
Trial 2	0.1154	0.04189
Trial 3	0.1039	0.0499
Trial 4	0.1117	0.04799

Table 3: Best actual RMSE for a model trained on L^2- and L^1-loss function. Lower bound on the error is 0 here since we compare the prediction with the pure sinus-function. It is worth nothing to remind that all models are trained on the noisy data. The term 'actual' refers to the fact that we report the errors here by comparing the prediction with the actual/pure signal.

Another conclusion that we draw from this experiment, in particular from Table 3, is that the minimization of a loss function under the constraints from the model does not necessarily coincide with extracting the real dynamic that underlies a dataset or extracting the most information from the dataset, respectively. For example, the minimum of the L^2-loss function subject to the constraints of the model is more distracted from the real dynamics (sinus-function) by the specific noise than the L^1-correspondence. However, with our framework, we provide a way to measure if there is something left to extract, e.g., since a used loss function is not suitable for the noise of a dataset.

We conclude this section with the following remark: We see the L^1-loss function is less prone to the asymmetric noise than the L^2-loss function. Since L^0-loss function weights all deviations from the real data with the same penalty, we formulate the hypothesis that L^0-loss function might perform even better than L^1-loss function. However, since the L^0-loss function is discontinuous and thus not differentiable at all, a lack of a numerical efficient optimization algorithm capable to deal with the discontinuity of the loss function might hinder its broader application. Consequently, a starting point for further research might be to analyze the performance of loss functions with regard to information extraction that consist of parts cutting off bigger deviations with, e.g., $\min(\max(\tilde{y}-y,-\tau),\tau), \tau > 0$, which can be solved with semi-smooth methods, see, e.g., Ulbrich (2011) or as shown in Breitenbach (2022) by transforming the corresponding optimization problem into a higher dimensional one to resolve the min- and max-function to differentiable functions. Such a loss function, e.g., taking the absolute value or the square of the projection above, could be a tradeoff between numerical efficiency and robustness against asymmetric noise or outliers parameterized by the parameter

τ. Starting the learning with a big τ and restarting the optimization with a smaller τ with the result from the last optimization procedure or decreasing τ within one optimization run could also accelerate the convergence speed. This procedure could make the prediction more precise with regard to extracting the deterministic relations assuming that bigger determinations come (mostly) from noise given the input data. Our framework can monitor the effect of τ with regard to extracting the deterministic relations.

4.3 Stochastic measures as convergence criteria

Next, we show how the mutual information and the chi-square test evolve over training after each epoch to demonstrate its capability to work as convergence criteria. The procedure is as follows. If input and output are not independent of each other, training of a model is started. If after an epoch, input and residuals or model deviations, respectively, are independent on the training or validation set, more detailed if we cannot reject the hypothesis that input and residuals, resp., model deviations are stochastically independent, then there is no information left to extract and we can stop the training. The value of the stochastic measure as proposed in this work is that we can evaluate if flattening of the loss function after some epochs is due to the training is done in terms of information extraction or if the convergence speed meanwhile has just slowed down. In the case of slowed down convergence, it is worth further patience since it could be that (now more slowly) further information is extracted or in later epochs the convergence speed increases again when having found the right updates for the weights. The advantage of a stochastic measure taking the relation between input and model deviations into account is that there is a lower bound of dependence known in advance that is theoretically always achievable by a model, namely when all the deterministic relations are extracted, which is by definition independent of the unpredictable parts given the input data. In contrast, for loss functions that do not take this relation into account, there is in advance no lower bound known that is achievable on the concrete dataset.

In this experiment, we use the data from Section 4.2 and plot relevant metrics in Figure 4. We see that when the loss function becomes flat, the corresponding chi-square test is zero and the p-value of the mutual information becomes non-zero in a magnitude of order such that we cannot reject the independence of input and residuals. Although we only check pairwise, this experiment shows that our framework provides valuable convergence criteria as a stagnating loss function also indicates that the model has extracted all the deterministic relations between input and output. The rationale is that if input and residuals or model deviations are independent of each other, it is necessary that also a pairwise test indicates independence. Furthermore, the turning point when stochastic measures increase marks a condition for early stopping without considering a validation set. The increasing stochastic measures, while the loss function further decreases, shows that training after the turning point adapts the model weights to relations that overfit the data defined in the sense of deterministic relations. The reason is that the loss function minimum does not coincide with the minimum regarding the stochastic measures.

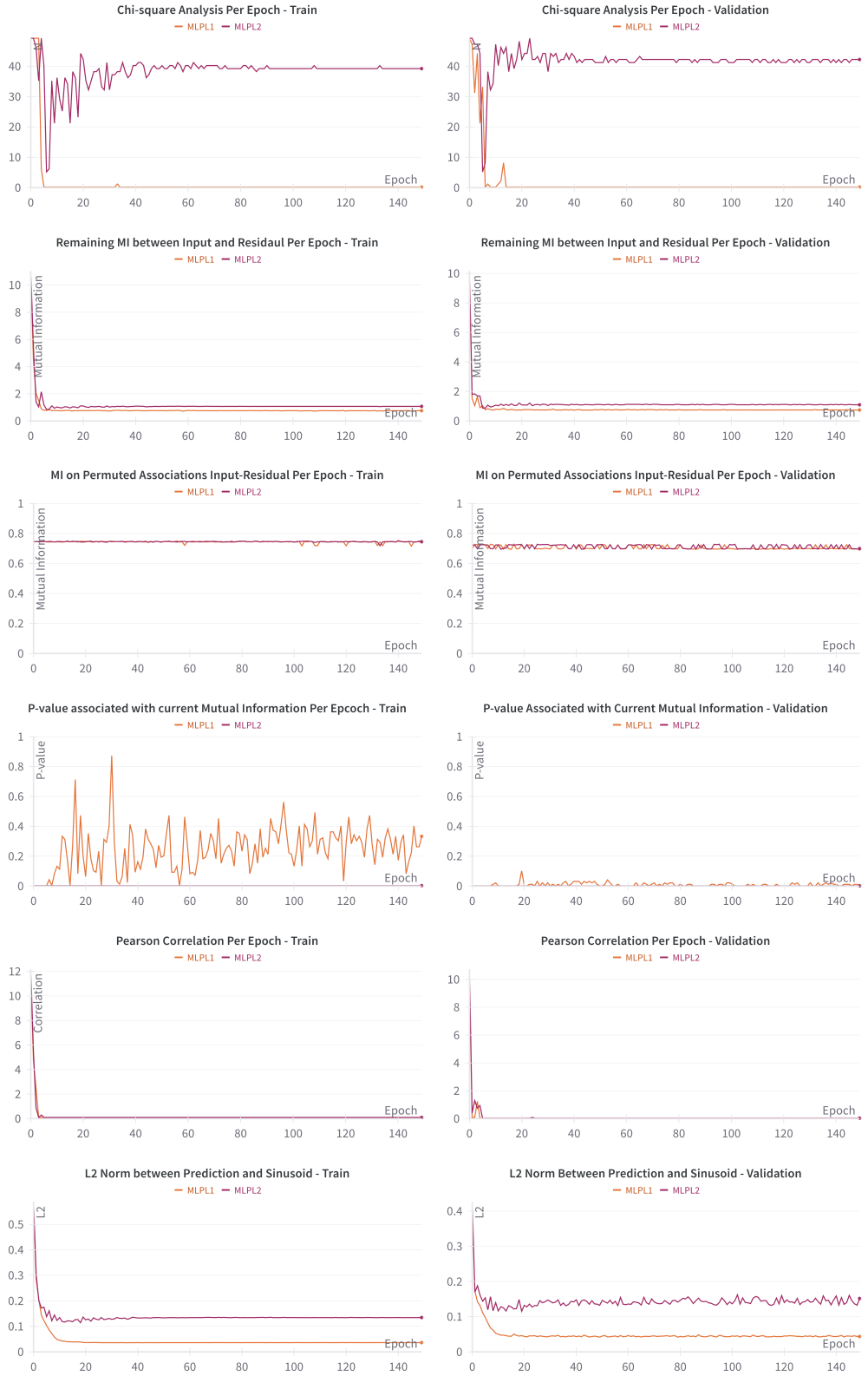

Figure 4: Images showing values of stochastic measures of independence between input and residuals as well as the loss function history.

We provide a further example based on the dataset ETTh2 Zhou et al. (2021) with a similar result depicted in Figure 5. We see based on the chi-square test that, after some epochs, there is already a set of weights based on which the residuals are clearly decoupled from the input. Similar, we see that the mutual information is close to the value generated by the permutation test, supporting the chi-square results. Furthermore, the increase of the chi-square and the mutual information on the test set for later epochs might indicate an overfitting since the minimum based on the L^1-loss function does not necessarily coincide with the maximum of extracted deterministic relations defined by a stochastic measure, see also Section 4.2.

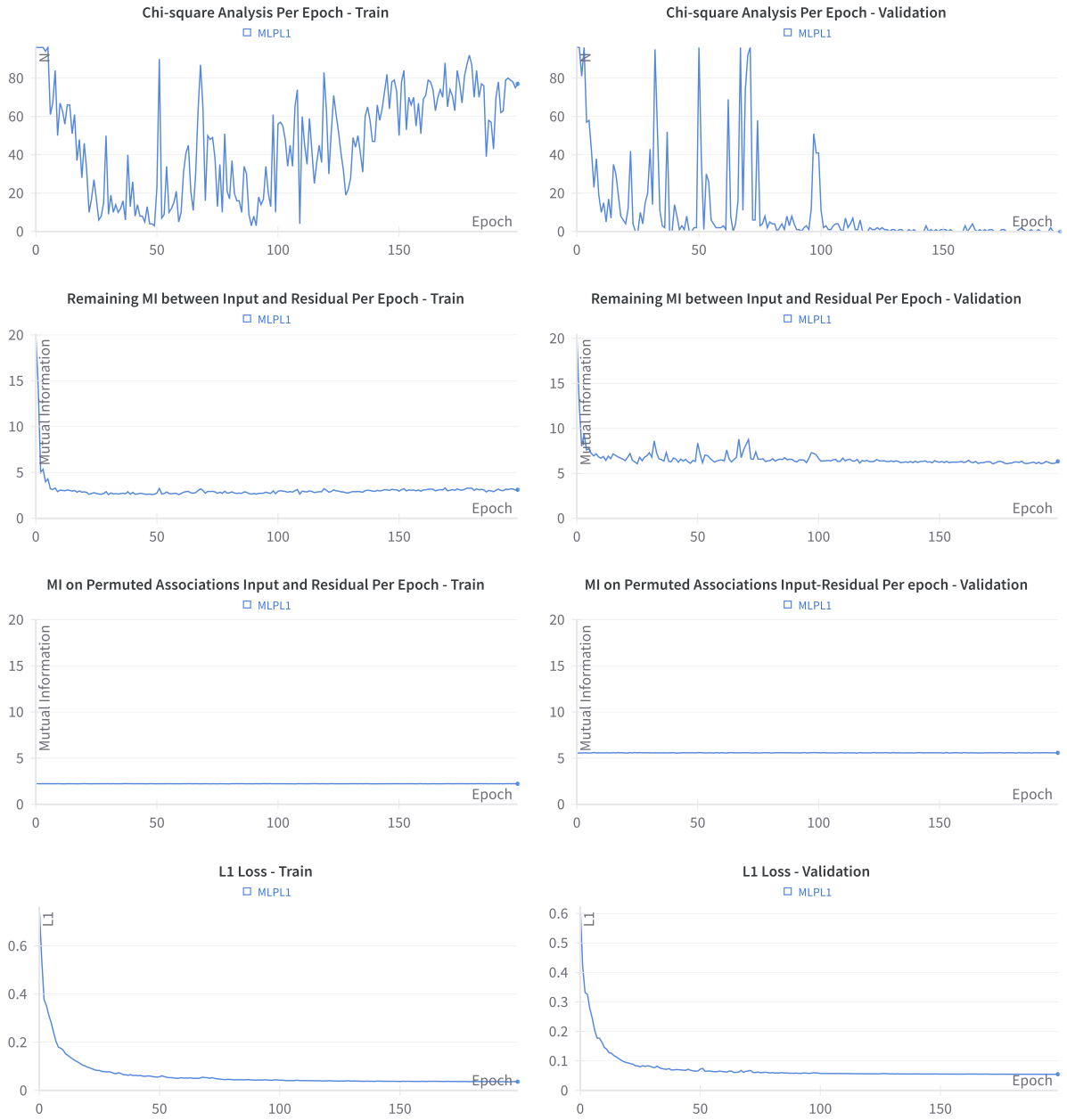

Figure 5: Images showing values of stochastic measures of independence between input and residuals as well as the loss function history based on the ETTh2 dataset.

4.4 Time series classification

In this section, we apply our framework on time series classification (TSC) problems. We remark that for ordered classes, we can apply the framework where the differences between the discrete values and the ground truth models the residuals. In order to define model deviations from the ground truth in the nominal case, we use the definition for the nominal case as described in the Methods section 2.3. In this case, the random variable θ modeling the deviation of the model classification from the ground truth takes the value of the correct class in case the output of the model is correct and -1 otherwise. We apply our framework on subset of the well-known UCR dataset Dau et al. (2019). More specifically, the dataset DistalPhalanxOutlineCorrect for Figure 6 is a binary classification problem from time series data. Furthermore, the dataset ElectricDevices for Figure 7 consists of seven classes where we relabeled the classes always starting from 0 until all classes are labeled accordingly.

In this case, the parameter ρ of Algorithm 1 is set to 10 to cope with the high imbalance of the classes and the fact that over epochs this imbalance increases since most of the cases are classified correctly and thus the class labeled with -1 for θ increases. Results including cross-entropy loss, accuracy and the mutual information per epoch are shown in Figure 6 and Figure 7. In Figure 6 and Figure 7, we see that while the accuracy is increasing, the dependence of the variable θ with the input x decreases over epochs as it is supposed to, since less and less cases are not predicted correctly. In other words, the random variable θ tends to a constant function where information gain is small/zero taking any other input variable into account. The difference in both cases is the following. From the results of Figure 6, we can say that over training epochs we approach a parameter configuration where we cannot reject the hypothesis based on a level of significance of 1% that the input and the model deviations are independent based on our mutual information measure. In this case, we do not expect a further improvement regarding the model performance since it captured potentially all deterministic relations. In contrast to the experiment depicted in Figure 7, where according to our mutual information test, we cannot reject this hypothesis and thus there are deterministic relations to extract which may improve the model performance.

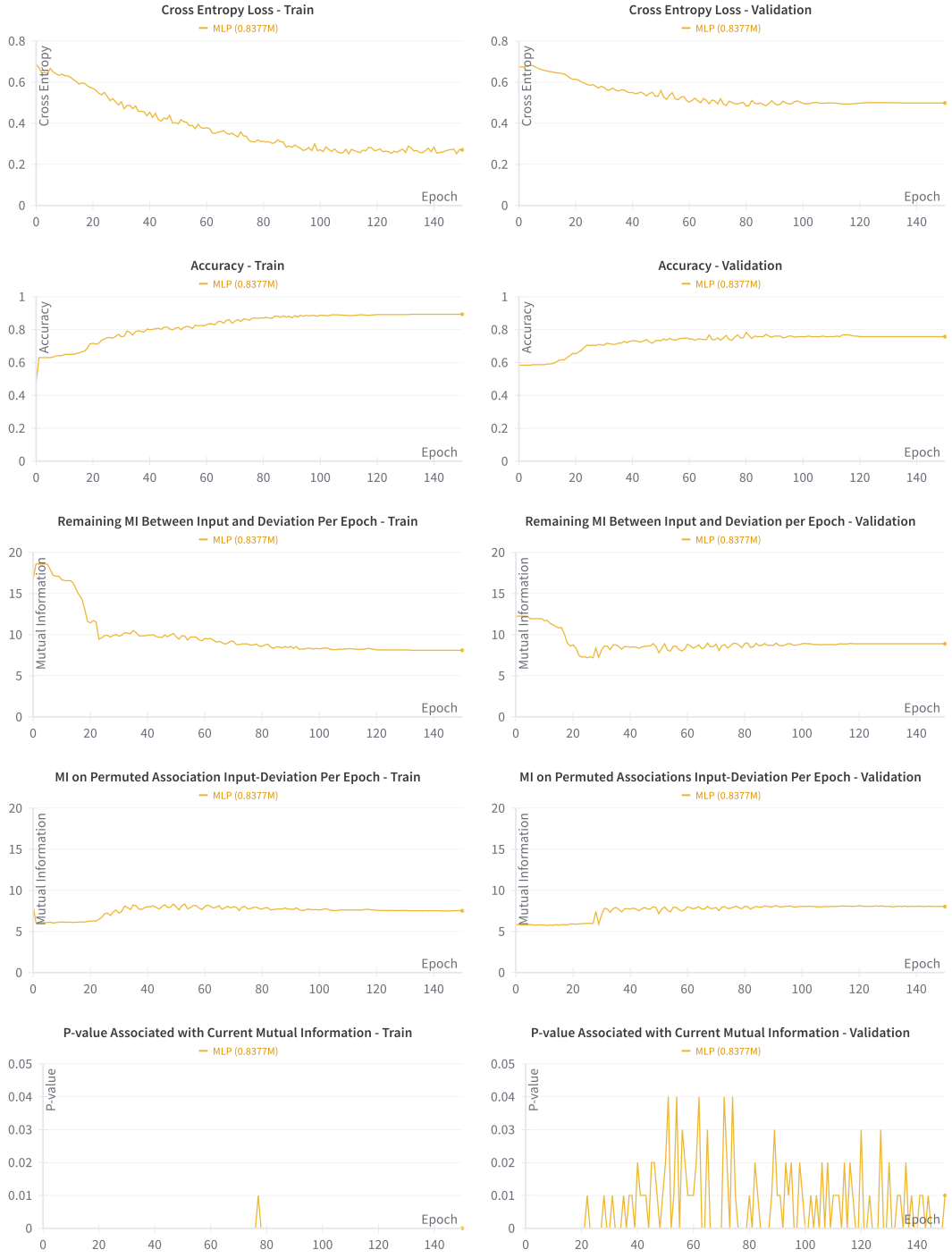

Figure 6: Results on a classification problem showcasing our framework, in particular the definition of the model deviation for nominal target data. Numbers in the parentheses shows the number of parameters of the model in million. Although validation accuracy remains below 0.8, mutual information (MI) analysis shows the model deviations are already independent of the input.

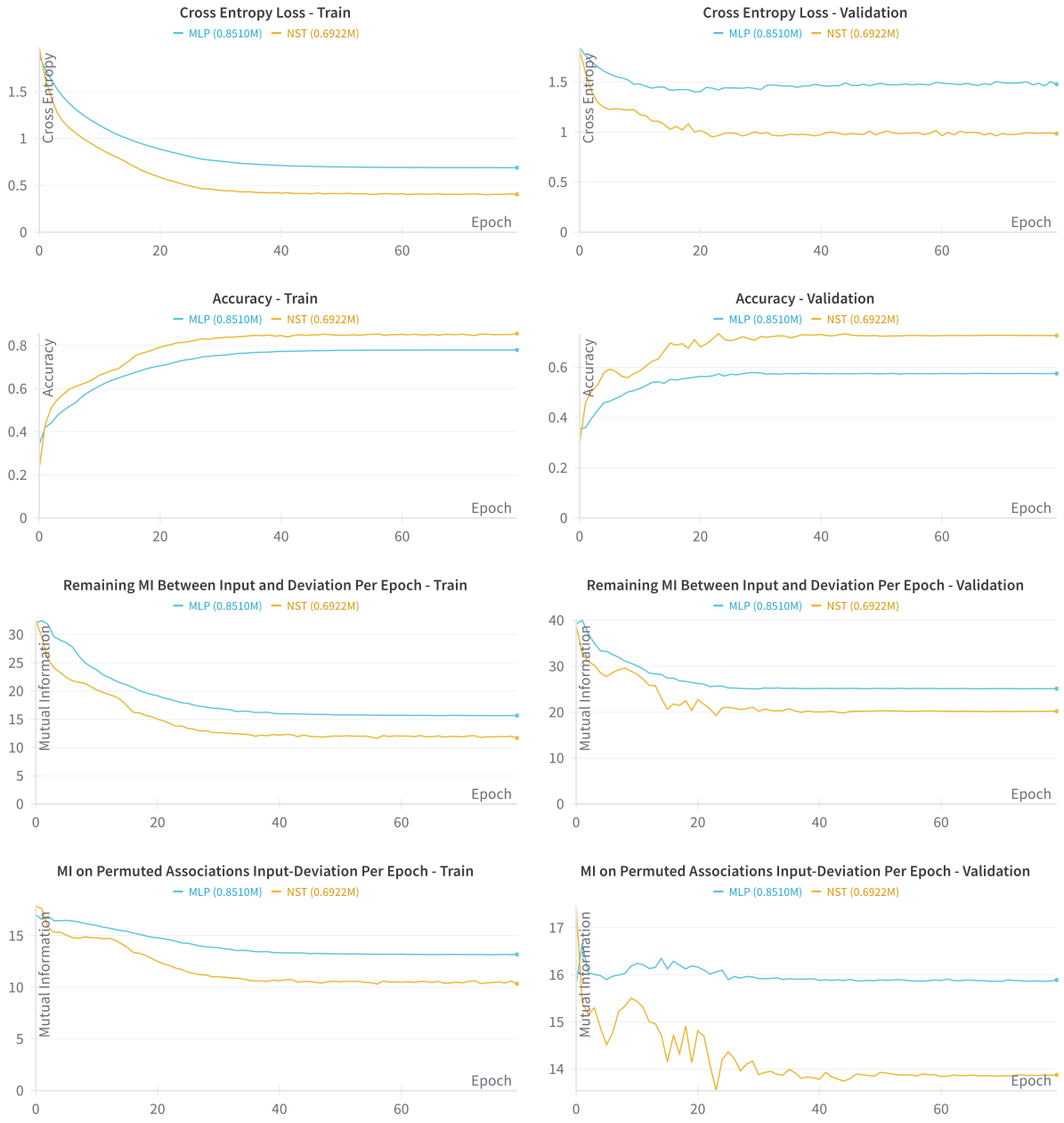

Figure 7: Results on a classification problem showcasing our framework, in particular the definition of the model deviation for nominal target data. Numbers in the parentheses shows the number of parameters of the model in million. The p-value always remains zero in this experiment for both train and validation set for the mutual information (MI).

4.5 Stacking of models systematically extracts information and improves prediction

After we have seen in the previous sections that the loss function has an influence on the information extraction depending on the noise, see in particular Section 4.2, we further investigate in this section how different model properties, like the architecture or hyperparameters, contribute to capturing different kind of information and how such differences could be systematically combined to extract all available deterministic relations in a dataset. For this purpose, we present a general architecture that not limited to only varying loss

functions and thus is supposed to combine the capabilities of different models as described in the Methods section 2.3.

We showcase the efficacy of our framework with a real-world time-series datasets. The dataset is a Nasdaq datasets taken from the UCI repository and M4 competition dataset Makridakis et al. (2020). More specifically, we take the variable DE1. We train models according to MLP and Nonstationary Transformers (NSTs) Liu et al. (2022) architecture on the dataset that is split according to the ratio of 0.7/0.3 into a training and test set. The input of the models are the past 60 time lags and the output is the next value in the time series (singlestep prediction). To this end, an MLP model trained with L^1-loss is chosen as the first model and another MLP trained with L^2-loss is chosen as a second model. The weights of the last layer of the models in the stack, except the first model, are initially set such that the output of each model is close the zero. The rationale is that if the prediction from the stack below is correct, only minor corrections are necessary building on the previous predictions. Furthermore, the last layers in models are chosen for weight rescaling since the first layers are usually intended for feature extraction. Aditionaly, the layer norm operation (dividing by standard deviation) would cancel the scaling to small values, for details about this part please see the Appendix, Section C.4.

As illustrated in Table 4, none of the individual models successfully rendered the residuals independent of the input. Remarkably, it was only through the combination in stacked models that an increase of p-values was observed, enhancing the overall performance. To provide a more comprehensive comparison, results for NST are also included. We see that in this case, the MLP stack not only extracts more information as the NST but are computationally even cheaper. In order to exclude that the effect is a result of more free parameters, we have included MLPs with about the half of free parameters each indicated by the "0.5". Moreover, we see that only the stacked models provide a non-zero p-value such that only in this case, we cannot reject the hypothesis that input and residuals are independent based on a level of significance of 1%. Beyond mutual information and chi-square test, considerations such as L^2-loss and learning curves in Figure 8 further support the empirical evidence that stacking models outperforms their individual counterparts by having a smaller L^2-loss function (comparison only valid if the last layer of the stack is trained with the same loss functions, which is in this case L^2-loss function, as the corresponding single model). This evidence showcases the capacity of multiple models to learn diverse aspects. However, we remark that a comparison in terms of loss functions and information extraction is tricky as shown in Section 4.2.

Metrics	Init MI	Init Perm	**Init diff**	Res MI	Res Perm	pv	**Res diff**	Init Chi-square	Res Chi-sauare
MLP L1-0.5	23.038	4.167	18.871	5.256	4.617	0	0.6392	60	2
MLP L2-0.5	23.038	4.167	18.871	5.347	4.579	0	0.7679	60	8
stacked-0.5	23.038	4.167	18.871	4.843	4.564	0.06	0.2798	60	0
2nd stacked model-0.5	5.256	4.617	0.6392	4.483	4.564	0.06	0.2798	2	0
MLP L1	23.038	4.167	18.871	5.262	4.586	0	0.6758	60	4
MLP L2	23.038	4.167	18.871	5.414	4.533	0	0.8814	60	5
Stacked MLP	23.038	4.167	18.871	4.955	4.579	0.01	0.3765	60	0
2nd stacked model	5.262	4.543	0.7182	4.955	4.579	0.01	0.3765	4	0
NST L1	23.038	4.167	18.871	5.267	4.577	0	0.6901	60	1
NST L2	23.038	4.167	18.871	5.468	4.564	0	0.9038	60	8

Table 4: Comparison of performance metrics for various standalone and stacked models. Initial mutual information (MI), permutation analysis values, and their differences are presented, providing insights into the starting states. Residual metrics, including mutual information, permutation values, and the corresponding p-values in the mutual information framework assessing the independence of input and residual output, are also reported. Additionally, chi-square values for both initial and residual states are included indicating the number of correlated input lags of the time series.

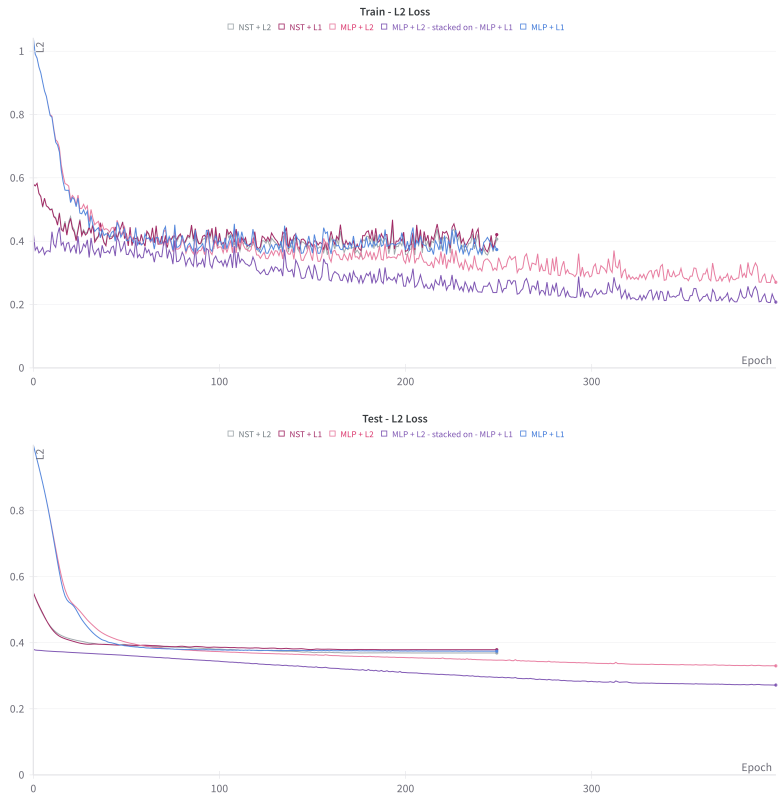

Figure 8: The L^2 loss comparison for stacked models and single models.

The last example in this section is provided in Table 5 on Weather dataset in 1-step ahead prediction setting. The split setting is matched with Nie et al. (2022). Except the prediction length, we use the same training parameters and architecture for PatchTST as used in Nie et al. (2022).

Metrics	Init MI	Init Perm	**Init diff**	Res MI	Res Perm	**Res diff**
PatchTSTL2 (0.41M)	3213	153	3060	659	177	482
MLPL1 (0.59M)	3213	153	3060	514	174	340
NSTL1 (3.88M)	3213	153	3060	608	182	426
NSTL1(1.05M)	3213	153	3060	641	183	458
NSTL2(0.426M)	3213	153	3060	700	182	518
NSTL1(0.86M)+PatchTSTL2(0.41M)	3213(659)	153(177)	3060(482)	506	181	325
NSTL1(1.13M)+PatchTSTL2(0.41M)	3213(659)	153(177)	3060(482)	516	183	333
NSTL2(0.53M)+MLPL1(0.59M)	3213(514)	153(171)	3060(343)	400	181	219
NSTL1(0.86M)+MLPL1(0.59M)	3213(514)	153(171)	3060(343)	441	182	259
NSTL1(0.53M)+MLPL1(0.59M)	3213(514)	153(171)	3060(343)	442	182	260
NSTL1(2.05M)+MLPL1(0.59M)	3213(514)	153(171)	3060(343)	448	182	266
MLPL1(0.59)+NSTL1(1.13M) + PatchTSTL2 (0.41M)	3213(516)	183(153)	3060(363)	499	182	317

Table 5: Comparison of performance metrics for various standalone and stacked models on weather dataset. Initial mutual information (MI), permutation analysis values, and their differences are presented, providing insights into the starting states. Residual metrics, including mutual information, values from the permutation test of mutual information are also reported.The corresponding p-values assessing the independence of input and residual output is always 0 in all experiments. Additionally, In the first column, number of parameters for the models is shown in parentheses in millions. In the other columns the values of the metrics only for the last model of the stack is depicted in the parentheses. In the first column the model on the left is the first one and the one on the right is the last model in the stack.

We remark that a linear combination of loss functions according to $\lambda_1 \|\cdot\|_{L^2}^2 + \lambda_2 \|\cdot\|_{L^1}$ with the hyperparameters $\lambda_1, \lambda_2 > 0$ could be an alternative approach to stacking in terms of loss functions. However, the success of that formulation depends on the right choice of the hyperparameters λ_1 and λ_2. Our framework provides an option to choose the hyperparameters accordingly such that most information is extracted from the data, e.g., the mutual information is minimized between input and the residuals. Please note that finding the best combination or any hyperparameter optimization is not the focus of this paper. In the present work, the focus is on showcasing our framework in terms of its potential for applications in various ML use cases.

To conclude this section, we show that our stacking framework also works for classification problems. For architectural details, please see the Methods section 2.3 and Section 4.4. In Table 6, we provide numbers based on the ElectricDevices dataset. To further improve the effect of stacking, we see potential when including a corresponding loss function into the training process such that parameters can be optimized such that the corresponding ML architecture extracts most deterministic relations possible. Regarding this topic, please see the corresponding Discussion part "Alternative for nominal data" and the Conclusion and Future Work section.

Metrics	Init MI	Init Perm	**Init diff**	Res MI	Res Perm	**Res diff**
SVM-rbf kernel	41.65	16.00	25.65	23.72	15.50	8.22
SVM-sigmoid kernel	41.65	16.00	25.65	34.13	16.62	17.51
SVM-sigmoid + SVM-rbf	41.65 (23.72)	16.00 (15.50)	25.65 (8.22)	23.23	14.41	7.82

Table 6: Numbers in parentheses shows the starting point of the last stacked model. The abbreviation SVM refers to the standard Python sklearn implementation of a support vector machines. The p-values assessing the independence of the input and the model deviation from ground truth alsways remains zero in all experiments. For a description of the meaning of the columns, please refer to Table 5.

4.6 Applying the stacking of models to multiple output variables exemplified by multistep prediction

In this experiments, we demonstrate our framework for a multistep prediction. We take the NASDAQ dataset from Kim et al. (2021) analogously to Section 4.5. We choose the input of length 60 to predict a target length of 30. The result is provided in Table 7 and shows that also in this case, with stacking of models, we can systematically extract the information and decouple input and residuals in contrast to single models. The evidence is provided by the fact that a stack of MLP and NST models provides the smallest mutual information between input and residuals subtracted the mutual information generated by coincidence based on the given data (column "Res diff" of Table 7).

Metrics	Init MI	Init Perm	**Init diff**	Res MI	Res Perm	**Res diff**	Init Chi-square	Res Chi-sauare
MLP L1(2.09M)	617.48	123.29	494.19	169.45	135.11	34.34	1800	332
MLP L2 (3.77M)	617.48	123.29	494.19	163.53	135.36	28.17	1800	182
MLP L2 (4.61M)	617.48	123.29	494.19	162.82	135.47	27.35	1800	177
NST L1(2.67M)	617.48	123.29	494.19	183.23	135.15	48.08	1800	811
NST L2(2.67M)	617.48	123.29	494.19	179.47	135.28	44.19	1800	729
MLP L2(0.67M) + MLP L1(2.09M)	617.48(169.45)	123.287(135.11)	494.19(34.35)	157.91	135.28	22.63	1800(332)	55
MLPL2(0.64M)+NSTL1(2.67M)	617.48(183.23)	123.287(135.15)	494.19(48.08)	156.84	135.22	21.62	1800(811)	35
MLP L2(0.64M)+NSTL2(2.67M)	617.48(179.47)	123.287(135.28)	494.19(44.19)	153.15	135.25	17.90	1800(729)	30
MLPL2(0.09M)+MLPL2(0.64M)+NSTL2(2.67M)	617.48(153.17)	123.287(135.28)	494.19(17.89)	151.80	135.27	16.53	1800(30)	20
Avg Ensemble (3.4M)	617.48	123.29	494.19	170.95	135.36	35.59	1800	387

Table 7: Comparison of performance metrics for various standalone and stacked models. Initial mutual information (MI), permutation analysis values, and their differences are presented, providing insights into the starting states. Residual metrics, including mutual information, values from the permutation test of mutual information. Additionally, the number of dependent input lags tested by the chi-square test of independence for both initial and residual states are included. In the first column, number of parameters for the models is shown in parentheses in millions. In the other columns, the values of the metrics only for the last model of the stack is depicted in the parentheses. In the first column the model on the left is the first one and the one on the right is the last model in the stack. In the last row, we take the average prediction of the three models in the penultimate row when each of those models is separately trained to predict the original ground truth. The p-values assessing the independence of input and residual output based on mutual information remains always zero in all experiments in this table.

4.7 Detecting distribution shifts

One prevalent issue hindering the advancement of machine learning models towards higher accuracies is distribution shift, meaning that relations that hold within the training set do not hold on the validation set. Especially several existing works such as (Zeng et al., 2023, Figure 5) and Kim et al. (2021), in particular (Kim et al., 2021, Figure 3) have confirmed this phenomenon, e.g., on ETT1 and ETT2 data sets. This section presents a novel insight into this phenomenon, enlightening how our proposed framework can detect and distinguish such cases from mere overfitting to noise. In a typical training scenario, after some epochs while training loss continues to decrease, validation loss may gradually start to increase. Without prior assurance of the absence of distribution shift, a pure loss function based approach without including stochastic measures struggles to differentiate between overfitting to the noise in training data and (partial) distribution shift due to different deterministic relations between the training and validation set, as both can lead to similar observations of an increasing loss function on the validation set.

Our framework provides a concise solution. Instead of solely monitoring the loss function, tracking mutual information enables us to determine the types of relationships the model is learning. A decrease in mutual information across epochs indicates successful extraction of information, suggesting that the model is learning deterministic relationships within the training set, and not already overfitting to noise. If it does so as well on the validation set until input and deviations between model and ground truth are independent, then we can stop the training process since the model might have learned all deterministic relations on the training and validation dataset. In that case further training might cause an overfitting to noise. Similarly, if there is no significant reduction in mutual information despite decreasing training loss, it may indicate overfitting to the noise in the training data, as the model is fitting to the unpredictable elements of the ground truth which shares no mutual information with the input.

On our synthetic dataset, the deterministic relations are identical on training and training set by construction. In Figure 4, we see for the training based on L^2-loss function that the mutual information increases on training and validation set after reaching their minimum upon a few epochs simultaneously. Here the model fits noise rather than the actual sinus function since the norm of the difference between model prediction and sinus function increases simultaneously.

In case the deterministic relations in the training data may (partially) not hold true for the validation set, it may lead to an increase in loss and potentially mutual information over epochs on the validation set, while mutual information on the training set decreases, as we see in Figure 9. In this figure, the deterministic

relations learned on the training data do not cause a fitting output of the model on the validation set. In contrast, please see Figure 5 where mutual information between input and residuals on training and validation set from ETTh2 simultaneously decreases over epochs. Results in Figure 9 show the learning curves when fitting an MLP with L1 as a loss function on the residuals of PatchTST Nie et al. (2022) with the best setting they proposed.

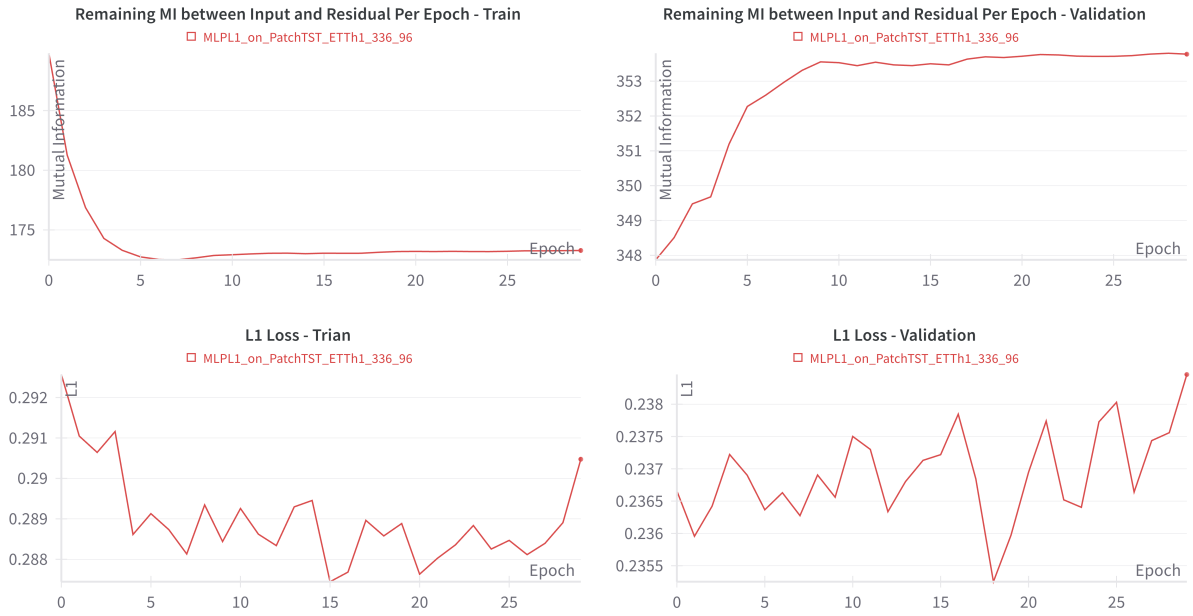

Figure 9: Distribution shift experiment on ETTh1 Dataset.

5 Discussion

In this section, we discuss our assumptions and limitations of our implemented approach before we sketch further potential applications of our approach.

Assumptions and Limitations: The implemented approach tests pairwise the relation between input and output features. We are aware that there is a difference, e.g., in pairwise stochastic independence and (mutual) stochastic independence in case of more than two random variables (Gallager, 2013, Section 1.3.4). That means that there might be more information considering, e.g., two input features at once instead of testing for pairwise relations with an output feature. However, the full consideration, instead of a pairwise testing, scales exponentially in terms of the computational costs. Consequently, we are aware that the current pairwise approach, which is computationally cheap compared to the full approach, cannot provide in general a full statement like, there is no information left to learn. In this regard, the current approach can only be used as an additional metric to evaluate if training is done and further iterations might not provide a further improvement. This could be the case if a low pairwise measure of the relation between input and output coincides with a small/no loss function improvement. However, since pairwise test is a specific case of a full consideration, a pairwise test indicating deterministic relations between input and output implies that there is information left a model can learn. An analogous framework is the gradient within an optimization framework where the gradient provides necessary conditions for convergence to a global minimum, which are only under some conditions sufficient to characterize a global optimum. However, even in this case, using only necessary conditions, provides useful optimization results while keeping the computational costs manageable, which might be analogous to our pairwise definition of our used stochastic measures.

It is left to investigate under which conditions a pairwise consideration is sufficient to test for a total stochastic independence of input and output. Furthermore, apart from considering pairwise testing as an approximation

for the mutual independence, there might be further approximations to the mutual independence that might be considered as well to decide for stopping an ML training. Please also see the Related Work section about mutual information estimation for further examples of approximations of full mutual information instead of a pairwise consideration. Furthermore, we remark that our framework is not limited to a specific choice of stochastic dependence/information measures and also our git repository is designed that new measures can be included quickly in a modularized manner.

In particular for time series, we would like to remark that, even under a method that considers (full) mutual independence between the input and output, a result of total independence of input and output does not imply that the time series is not predictable. It just says with the given input, the time series is not predictable autoregressively. Maybe with other features that are related to the quantity measured as a time series, there is a deterministic relation that can be used for predicting the time series.

Alternative for nominal data: One further option to model deviations of a model from the ground truth in the case of nominal target data could be a multi-dimensional random variable that models the difference between the actual probability distribution (e.g., 1 for the correct class, 0 otherwise) and the predicted distribution from the model for the classification. If the input variables and the difference distribution, which serves as the correction to the prediction to get the correct distribution, are deterministically related, then the correction (difference distribution) could be learned by another model and added to the distribution from the previous stage. After adding, which is all done in the decision module, see Figure 3, the new distribution can be processed with the softmax-function for normalization and the classification may correspond to, e.g., the most likely class.

Extension to unstructured data: For models that extract information from an input like images or text the input data needs to be transformed into a representation that shares information with the output. An example are the pixels in a figure where an object of interest moves over different pictures or tokenized text, where the position of a word can vary while not changing the meaning. We could optimize first layer(s) to have the highest mutual information with the ground truth that is to be predicted analogously to Chen et al. (2016) or Brakel & Bengio (2017) focusing on finding entangled representations, see the Related Work section about the application of mutual information for further details. This procedure could also foster the application of a pairwise test between each node in such a representation and the output feature since the entangled representation extracts potential mutual information considering several input features at once such that each node of the representation is independent of each other.

However, also in the scenario where no specific optimization for mutual information between representations and target ground truth is done, it can be tested if the nodes of a layer share mutual information with the output as the nodes may compress the information over several input nodes such that a pairwise test is meaningful, like in a encoder-decoder scenario where such compressing representations are usually present. In such a scenario, such a layer could be considered as a purposeful approximation for the mutual independence or full test as we mentioned in the discussion about the limitations above.

A specific issue with the output of large language models, and in general for unstructured output data, is that there are sometimes more than one correct class (e.g., tokens associated with synonyms), but only one word is taken by a large language model. In such a case, a solution might be another large language model to tell synonyms to evaluate if a model is right. Thus this definition of the model correctness transforms the unstructured output to a structured one that can be investigated for dependence with other layers/representations. One possible implementation can be like the corresponding random variable taking the value for correct which could be several classes defined by the other model. If the model is not correct, this variable could take a number of a correct class, where in terms of synonyms one correct option, according to the evaluating model, is enough. However, also in this unstructured data case, the stacking concept works as the random variable θ^j in Figure 3 is defined with the help of a second large language model instead of a simple rule based definition from the known tabular data. Such an investigation can help to answer the question how small a model can be and where most information is learned/extracted to make large language or even multimodal models more efficient. For more details, see the next paragraph about cutting down models.

Another advantage of calculating stochastic measures between representations of a (pre-)trained model and the ground truth belonging to a specific task, like a classification, is to find out which representations of

which model have the highest dependence to the corresponding task, which could efficiently enrich the way how (pre-)trained models are selected from the pool of available models.

Cutting down models to structures extracting most information: If we perform a training with usual loss functions without any additional optimization for mutual information in specific layers, we could also identify the best structures within a trained model that extract most information and which subparts only contribute minor to, e.g., an embedding. While in this work so far the idea of stacking models has been discussed, zooming more in a model's architecture would help us analyzing the model itself regarding its subunits (e.g. layers, embeddings, attention mechanisms, etc.). The approach is to apply mutual information like in the scenario of multidimensional input and output (see, see Section 4.6). The output can be filled with the data to predict or the output of other subunits, like an embedding, upstream towards the output of the total model. If there is no contribution or only a minor one, we could cut the corresponding subunit out.

By ranking subunits with such measures, we have a clear procedure what subunits to exclude, instead of randomly selecting some for cutting out before retraining the model. The training could start from the current model parameters to just fine-tune the remaining layers. Training from the scratch is also possible but could delete a lot of parameter values that are still valid. The procedure can be iteratively repeated and even to that point where model size is balanced against a (small) drop of accuracy. Our framework can in additional help to find a lossless pruning similar to the Related Work section about the information bottleneck by testing if the stochastic measures between input or any representation of it and the model deviations get worse after a cutoff. Such investigations might facilitate an understanding about the problem-specific relevant structures and keeping model sizes efficient without lowering their accuracy. As an example, we could test how large, e.g., large language or multimodal models need to be and which structures extract the most information, similar to Liu et al. (2024). Further we remark, that with a measure for self-similarity, like the Pearson correlation, we could probably identify identities, resp. structures that behave similarly, e.g., in a sequential arrangement of layers that have the same output behavior, which should be lowly ranked since they only direct information through. A similar work is done in Gromov et al. (2024) to identify layers with a similar behavior that can be cut off. However, with mutual information, we can also investigate structures that do not behave similarly, but how much single parts of a branches structure contribute to the part where these branches come together.

If a training fails, we could also use such methods to test which structures fail. An example is a cascade of layers with one layer where input and output do not share any information anymore. The disruption of routing information could mean that this layer is like a constant function, which may delete important information.

Stacking software pipeline: Apart from finding relevant properties of a model to vary, like size, deepness, loss function etc., another aspect is that we assume the stack to be constant once trained and parameters are kept constant while only the new model on top of the stack is trained. It is to investigate in future research if, e.g., training all the parameters of the whole stack after training the new model on top of the stack might benefit the accuracy before testing if another model for the top of the stack is needed. One important application of a pipeline is to facilitate a precise time series prediction related to a concrete single time series and provide this capabilities to a broad audience even outside the ML community that apply the predictions of time series, like weather forecasts. Another use case is improving therapies where the effect depends on time varying patient-specific parameters. Thus, by taking, e.g., the daily rhythm of gene expression of humans into account, as argued in the concept of chronotherapy Zhang et al. (2014), a precise prediction of the expression levels may improve the effect of therapies and can be one brick for personalized medicine.

6 Conclusion and Future Work

In this work, a framework for measuring predictability of input-output relation was developed. Furthermore, it was shown how the information extraction of an ML model from this input-output-relation can be measured. Based on this framework, a stacking architecture was presented, which is able to extract information systematically in case a single model fails to do so. Moreover, it was demonstrated how the corresponding

stochastic measures for predictability can be used to extend the current definition of model convergence and training success. The total framework was showcased with time series prediction and classification on synthetic and real world datasets.

The presented framework provides measures to evaluate the existence of deterministic relations that a model can extract and how successful a model have been with extracting them. Promising further research might be the development of a loss function to fit the model to the deterministic relations directly, sorting out unpredictable parts like noise, in contrast to currently common loss functions, like L^1, L^2, cross entropy or KL-Divergence that fit the model's output distribution to the data distribution without differentiating if a data point is influenced mainly by, e.g., noise or the deterministic relations.

According to the presented framework the mutual information between input and the model deviations might serve as such a loss functions sought. However, for the implementation there are some challenges left that need for research to over come them. One challenge might be that the current implementation is not differentiable since minor changes regarding the position of a residual can cause a change in the belonging to the corresponding bin that are generated during the calculation of the mutual information. A suitable smoothing might facilitate the application of a corresponding loss function within a numerically efficient algorithm that requires a smooth loss function. These challenge of the non-smoothness is already described in (Oord et al., 2018, Section 2.1). There are smooth approximations of mutual information, like Belghazi et al. (2018) or Franzese et al. (2023), however, these approximations could become computationally too costly since an ML model needs to be trained in any epoch approximating the mutual information between input and model deviations. A further challenge could be that there might be instabilities in estimating the mutual information, as reported in Choi & Lee (2022). Such an inaccuracy in the estimation of the mutual information for a loss function could cause divergence of the optimization procedure and thus not improving the model's capability to extract more deterministic relations differentiating them from unpredictable parts like noise given the input data. Further research for a smooth and computational cheap approximation of mutual information is promising to focus the ML training on the deterministic relations encoded in the data.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Philipp Becker, Harit Pandya, Gregor Gebhardt, Cheng Zhao, C. James Taylor, and Gerhard Neumann. Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 544–552. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/becker19a.html`.

Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *International conference on machine learning*, pp. 531–540. PMLR, 2018.

Alexandra Bezbochina, Elizaveta Stavinova, Anton Kovantsev, and Petr Chunaev. Enhancing predictability assessment: An overview and analysis of predictability measures for time series and network links. *Entropy*, 25(11):1542, 2023.

Philemon Brakel and Yoshua Bengio. Learning independent features with adversarial nets for non-linear ica. *arXiv preprint arXiv:1710.05050*, 2017.

Tim Breitenbach. On the SQH method for solving optimal control problems with non-smooth state cost functionals or constraints. *Journal of Computational and Applied Mathematics*, 415:114515, 2022.

Tim Breitenbach, Lauritz Rasbach, Chunguang Liang, and Patrick Jahnke. A principal feature analysis. *Journal of Computational Science*, 58:101502, 2022.

Tim Breitenbach, Bartosz Wilkusz, Lauritz Rasbach, and Patrick Jahnke. On a method for detecting periods and repeating patterns in time series data with autocorrelation and function approximation. *Pattern Recognition*, 138:109355, 2023.

Rajkumar Buyya, Satish Narayana Srirama, Giuliano Casale, Rodrigo Calheiros, Yogesh Simmhan, Blesson Varghese, Erol Gelenbe, Bahman Javadi, Luis Miguel Vaquero, Marco A. S. Netto, Adel Nadjaran Toosi, Maria Alejandra Rodriguez, Ignacio M. Llorente, Sabrina De Capitani Di Vimercati, Pierangela Samarati, Dejan Milojicic, Carlos Varela, Rami Bahsoon, Marcos Dias De Assuncao, Omer Rana, Wanlei Zhou, Hai Jin, Wolfgang Gentzsch, Albert Y. Zomaya, and Haiying Shen. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Comput. Surv.*, 51(5):105:1–105:38, November 2018. ISSN 0360-0300. doi: 10.1145/3241737. URL `http://doi.acm.org/10.1145/3241737`.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29, 2016.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724. Association for Computational Linguistics, 2014.

Kwanghee Choi and Siyeong Lee. Combating the instability of mutual information-based losses via regularization. In *Uncertainty in Artificial Intelligence*, pp. 411–421. PMLR, 2022.

Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

Giulio Franzese, Mustapha Bounoua, and Pietro Michiardi. Minde: Mutual information neural diffusion estimation. *arXiv preprint arXiv:2310.09031*, 2023.

Robert G. Gallager. *Stochastic Processes: Theory for Applications*. Cambridge University Press, 2013. doi: 10.1017/CBO9781139626514.

Hans-Otto Georgii. *Stochastik: Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Walter de Gruyter GmbH & Co KG, 2015.

Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pp. 51–62, 2009.

Priscilla E Greenwood and Michael S Nikulin. *A guide to chi-squared testing*, volume 280. John Wiley & Sons, 1996.

Arthur Gretton, Ralf Herbrich, Alexander Smola, Olivier Bousquet, Bernhard Schölkopf, et al. Kernel methods for measuring independence. 2005.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*, 2024.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bklr3j0cKX`.

Kenji Kawaguchi, Zhun Deng, Xu Ji, and Jiaoyang Huang. How does information bottleneck help deep learning? In *International Conference on Machine Learning*, pp. 16049–16096. PMLR, 2023.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2019.

Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

Zhe Li, Shiyi Qi, Yiduo Li, and Zenglin Xu. Revisiting long-term time series forecasting: An investigation on linear mapping. *arXiv preprint arXiv:2305.10721*, 2023.

Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Non-stationary transformers: Exploring the stationarity in time series forecasting. *Advances in Neural Information Processing Systems*, 35:9881–9893, 2022.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905*, 2024.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020.

Mary L McHugh. The chi-square test of independence. *Biochemia medica*, 23(2):143–149, 2013.

Long Short-Term Memory. Long short-term memory. *Neural computation*, 9(8):1735–1780, 2010.

Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT Press, 2022. Available at `https://github.com/probml/pml-book/releases/latest/download/book1.pdf`.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *The Eleventh International Conference on Learning Representations*, 2022.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.

Calyampudi Radhakrishna Rao. *Linear statistical inference and its applications*, volume 2. Wiley New York, 1973.

Andrew M Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D Tracey, and David D Cox. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124020, 2019.

Vaisakh Shaj, Saleh GHOLAM ZADEH, Ozan Demir, Luiz Ricardo Douat, and Gerhard Neumann. Multi time scale world models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Michael Ulbrich. *Semismooth Newton methods for variational inequalities and constrained optimization problems in function spaces*. SIAM, 2011.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pp. 23965–23998. PMLR, 2022.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34: 22419–22430, 2021.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 11121–11128, 2023.

Ray Zhang, Nicholas F Lahens, Heather I Ballance, Michael E Hughes, and John B Hogenesch. A circadian gene expression atlas in mammals: implications for biology and medicine. *Proceedings of the National Academy of Sciences*, 111(45):16219–16224, 2014.

Xiyuan Zhang, Xiaoyong Jin, Karthick Gopalswamy, Gaurav Gupta, Youngsuk Park, Xingjian Shi, Hao Wang, Danielle C. Maddix, and Bernie Wang. First de-trend then attend: Rethinking attention for time-series forecasting. In *NeurIPS '22 Workshop on All Things Attention: Bridging Different Perspectives on Attention*, 2022. URL `https://openreview.net/forum?id=GLc8Rhney0e`.

Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations*, 2022.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 11106–11115, 2021.

A Models' settings for numerical experiments

This section of the appendix is allocated to the architecture of the utilized neural networks (NNs) in the experiments. Through this appendix we show the architecture of the MLPs with the number of nodes per each layer inside a list. The number of layers is the same as the length of the list. Unless specified differently, all activation functions are Relu and initial learning rates are 1e-4.

Section 4.1:

All NNs are MLPs with Relu activation functions.
MLP Layers: [49,490,980,1]
Activation functions: Relu
Initial learning rate: 1e-4
0.506M parameters

Section 4.2:

All NNs are MLPs with Relu activation functions. Number of nodes in each layer is written in the list.
MLP Layers: [49,490,700,490,1]
Activation function: Relu
Initial learning rate: 1e-4
0.712M parameters

Figure 6 : Timesreise Classification
MLP CrossEntropy (0.84M)
MLP Layers: [80, 720, 720, 360, 2]

Figure 7 : Timesreise Classification
NST CrossEntropy (0.69M)
Number of encoder layers: 2
Number of decoder layers: 1
Number of heads: 8
d_model: 128
Dropout: 0.1

MLP CrossEntropy (0.85M)
MLP Layers: [96, 720, 720, 360, 7]

Table 4:
MLPL1-0.5 (0.025M) & MLPL2-0.5 (0.025M):
Layers ob both models: [60,80,120,80,1]
Activation function: Relu
InitiaL learning rate: 1e-4

MLPL1(0.05M) & MLPL2(0.05M) :
Layers for both models: [120,80,180,1]

NSTL1 (0.05M) & NSTL2 (0.05M)
Number of encoder layers: 1
Number of decoder layers: 1
Number of heads: 2

d_model: 40
Dropout: 0.1

Table 5:
Here is the details of the pool of the used models -MLPs and Transformers- in one step ahead prediction experiment on weather dataset. For all non-stationary transformers (NSTs) dropout is set to 0.1.
The initial learning_rate for all models as the first stack is 1e-4 and for the second and the third stack is 1e-5.
The dropout for NSTs is 0.1 and for PatchTsT is 0.2, and there is no dropout for MLPs.
The size of subsequent hidden layer after the attention head (d_ff) in transformers are set to the provided default numbers, i.e. 4*d_model for NSTs and for 2*d_model for PatchTST.
Please note that PatchTST uses the vanilla Transformer encoder as its core architecture Nie et al. (2022) and therefore the number of decoder layer is zero.

PatchTST L2(0.41M)
PatchTST L2 architecture
Number of encoder layers: 3
Number of decoder layers: 0
Number of heads: 16
d_model: 128
Dropout: 0.2

MLP L1(0.59M)
MLP Layers: [96, 720, 720, 1]

NST L1(3.88M)
NST L1 architecture
Number of encoder layers: 2
Number of decoder layers: 2
Number of heads: 8
d_model: 256
Dropout: 0.1

NST L1(1.05M)
NST architecture
Number of encoder layers: 2
Number of decoder layers: 2
Number of heads: 8
d_model: 128
Dropout: 0.1

NST L1(0.86M) on PatchTST L2(0.41M)
NST architecture
Number of encoder layers: 2
Number of decoder layers: 1
Number of heads: 8
d_model: 128
Dropout: 0.1
PatchTST architecture
Number of encoder layers: 3
Number of decoder layers: 0
Number of heads: 16
d_model: 128

Dropout: 0.2

NST L1(1.13M) on PatchTST L2(0.41M)
NST architecture
Number of encoder layers: 2
Number of decoder layers: 2
Number of heads: 8
d_model: 128
Dropout: 0.1
PatchTST architecture
Number of encoder layers: 3
Number of decoder layers: 0
Number of heads: 16
d_model: 128
Dropout: 0.2

NST L2(0.53M) on MLP L1(0.59M)
MLP Layers: [96, 720, 720, 1]
NST architecture:
Number of encoder layers: 2
Number of decoder layers: 1
Number of heads: 6
d_model: 96
Dropout: 0.1

NST L1(0.86M) on MLP L1(0.59M)
MLP Layers: [96, 720, 720, 1]
NST architecture:
Number of encoder layers: 2
Number of decoder layers: 1
Number of heads: 8
d_model: 128
Dropout: 0.1

NST L1(2.05M) on MLP L1(0.59M)
MLP Layers: [96, 720, 720, 1]
NST architecture:
Number of encoder layers: 4
Number of decoder layers: 4
Number of heads: 8
d_model: 128
Dropout: 0.1

MLP L1(0.59M) on NST L1(1.13M) on PatchTST L2(0.41M)
MLP Layers: [96, 720, 720, 1]
NST architecture
Number of encoder layers: 2
Number of decoder layers: 2
Number of heads: 8
d_model: 128
Dropout: 0.1

PatchTST architecture
Number of encoder layers: 3
Number of decoder layers: 0
Number of heads: 16
d_model: 128
Dropout: 0.2

Table 6:
In this experiment, two simple SVM models are used:
SVM with rbf kernel
SVM with with sigmoid kernel

Table 7:

Here is the details of the pool of the used models -MLPs and Transformers- in multistep ahead prediction experiment on NASDAQ-DE1 dataset.

MLP L1 (2.09M):
MLP Layers: [60,360,3440,240,30]

MLP L2 (3.77M):
MLP Layers: [60,720,3440,360,30]

MLP L2 (4.61M):
MLP Layers: [60,900,3440,420,30]

NST L1 & NST L2 (2.67M):
Number of encoder layers: 2
Number of decoder layers: 1
Number of heads: 8
d_model: 256
Dropout: 0.1

MLP L2 (0.67M) on MLPL1 (2.09M)
MLP L2 Layers: [60,360,1080,240,30])
MLP L1 Layers: [60,360,3440,240,30]

MLP L2 (0.64M) on NSTL1 (2.67M):
MLP L2 layers: [60,360,1020,240,30]
NST L1 architecture:
Number of encoder layers: 2
Number of decoder layers: 1
Number of heads: 8
d_model: 256
Dropout: 0.1

MLP L2 (0.09M) on MLP L2 (0.64M) on NST L2 (2.67M)
NST L2 architecture:
Number of encoder layers: 2
Number of decoder layers: 1

Number of heads: 8
d_model: 256
Dropout: 0.1
MLP L2 (0.64M) layers: [60,360,1020,240,30]
MLP L2 (0.09M) layers: [60,720,60,30]

A.1 Data description

Datasets Here is a description of the datasets used in our experiments:

(1) *ETT* Zhou et al. (2021) contains seven features including the oil temperature and six power load feature. ETTh indicates the ETT data with a granularity of 1-hour-level and ETTm indicates the ETT data with a granularity of 15-minutes-level.

(2) *Weather*[1] is recorded every 10 minutes for 202 whole year, and contains 21 meteorological indicators such as humidity and air temperature.

(3) *Nasdaq* dataset consists of 82 variables, including important indices of markets around the world, the price of major companies in the U.S. market, treasury bill rates, etc. It is measured daily, having a total of 1984 data samples for each variable. We set the corresponding input length as 60 similar to Kim et al. (2021). In this work, we conduct our experiments on DE1 variable.

(4) *ElectricDevice* is a subset of the UCR time series classification dataset. It consists of seven different classes, each representing a specific type of electric device which is to be predicted from the time series.

(5) *DistalPhalanxOutlineCorrect* is a subset of the UCR time series classification dataset. This dataset focuses on the classification of outlines of distal phalanx bones from time series.

[1] `https://www.bgc-jena.mpg.de/wetter/`

B Further tables and figures

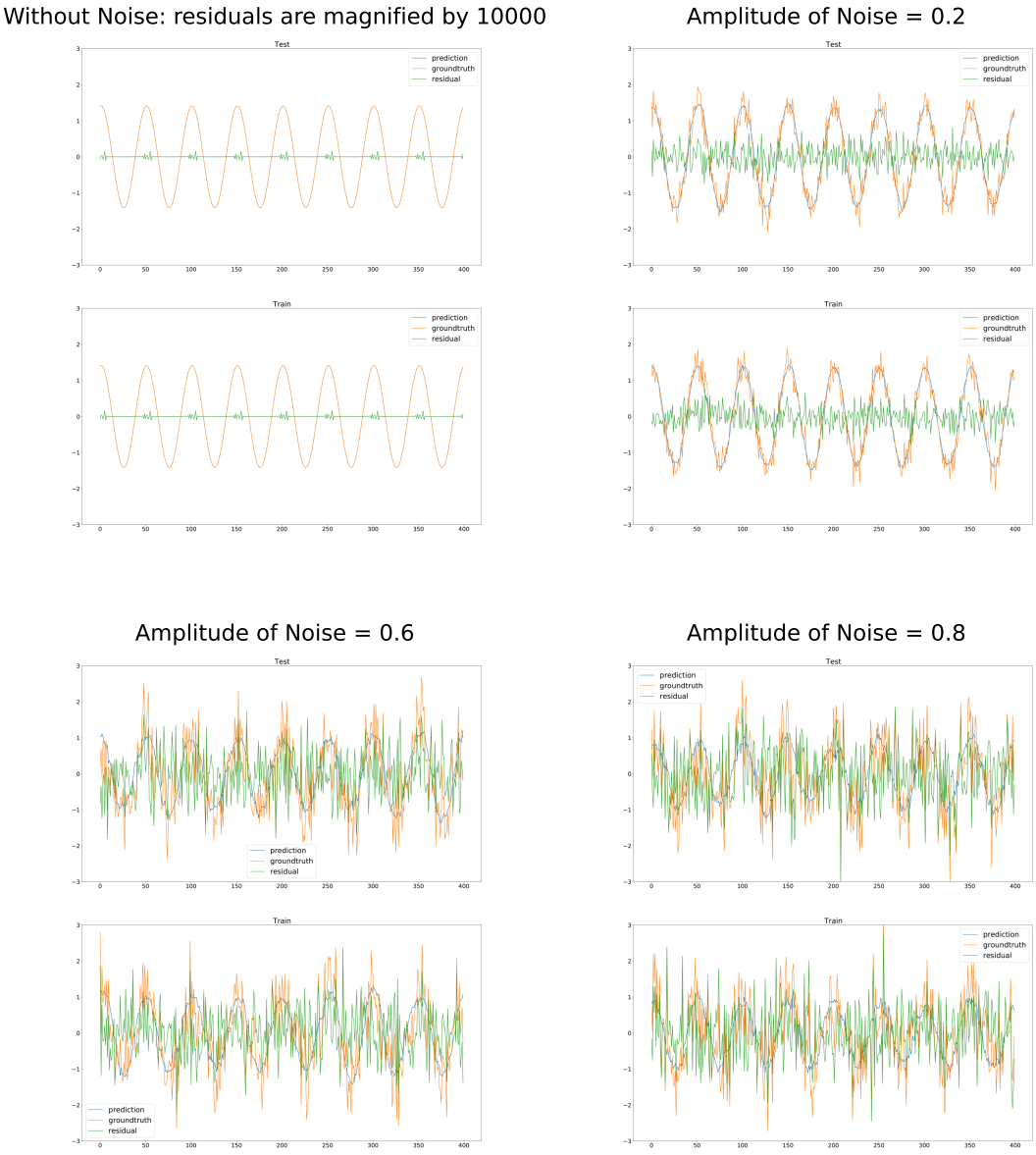

Figure 10: Plots of model outputs (prediction), residuals and the data (ground truth) of the experiments of Section 4.1.

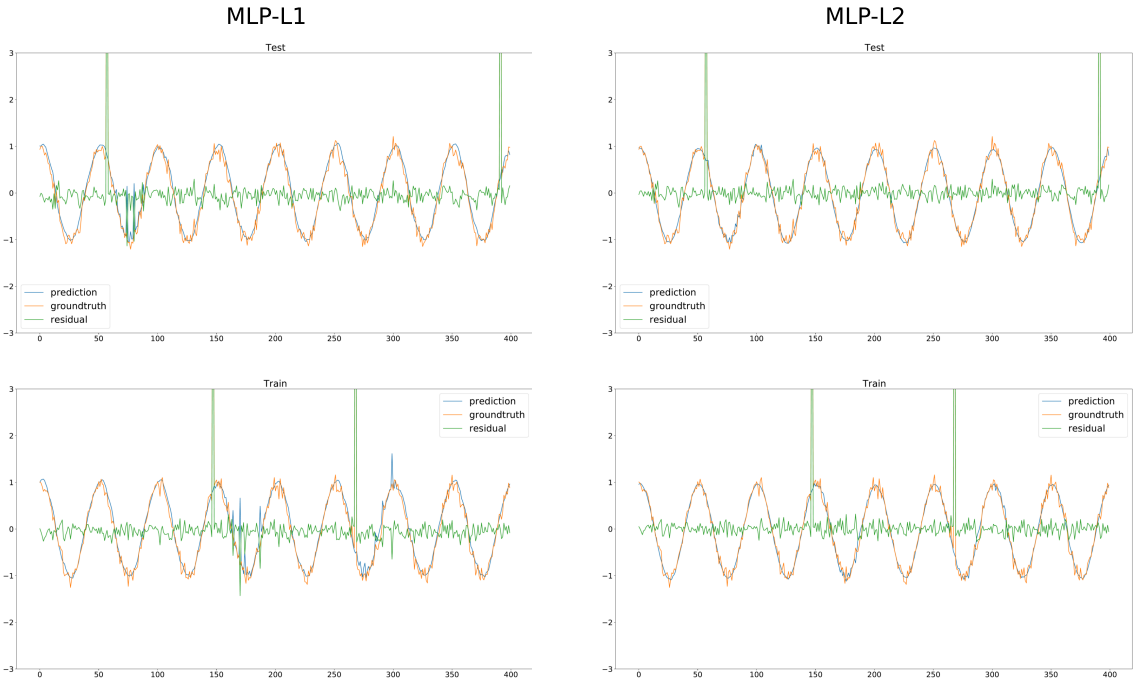

Figure 11: Plots of model outputs (prediction), residuals and the data (ground truth) of the experiments of Section 4.2.

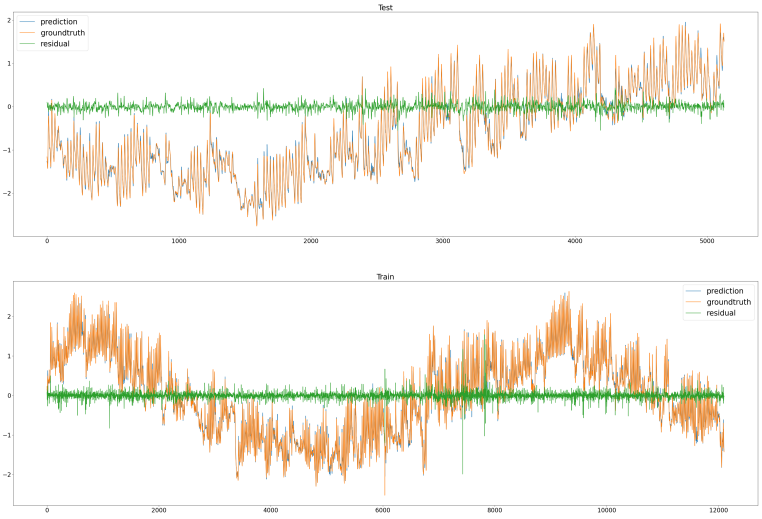

Figure 12: Plots of model outputs (prediction), residuals and the data (ground truth) of the ETTh2 dataset.

Theoretical bound	Experimental Error	Pearsonr Analysis	
Relative noise std	Normalized Test RMSE	Initial R	Residual R
0	0.000023	30.764	0
0.2715	0.2737	28.538	0
0.4919	0.5043	23.129	0
0.6465	0.6687	17.494	0.9446
0.7499	0.7683	13.197	0
0.8161	0.8287	10.455	0
0.8614	0.8718	7.868	0
0.8923	0.901	5.561	1
0.9149	0.9206	4.664	0
0.9308	0.9427	3.723	0
0.9429	0.9373	2.882	0

Table 8: Pearson correlation results for Section 4.1. The test consists of the sum of the absolute value of the Pearson correlation between each input and output features. However, the correlation measure is only considered if the p-value is smaller than 0.01. Otherwise, the corresponding input and output pair is considered as uncorrelated. In the first column, there is the relative noise, which is the root of the variance of the noise divided by the total signal. The second column provides the normalized RMSE as defined in Section 4. The third column is the sum of the absolute value of the Pearson correlation between input and the single step forecast as output and the forth column is analogous to the third column where the single step forecast is replaced by the corresponding residual (model prediction minus data). Since the residuals are less correlated or even decorrelated with the input, the model extracted the deterministic relations measured in the Pearson correlation test.

-	Initial pearsonr	Residual Pearsonr + L2	Residual Pearsonr + L1
Trial 0	15.33	0	0
Trial 1	16.925	0	0
Trial 2	15.25	0	0
Trial 3	17.104	0	0
Trial 4	15.449	0	0

Table 9: The effect of the choice of the loss function in mitigating asymmetric noise effect in terms of distracting a model from extracting/learning the deterministic relations. The experiments are conducted in Section 4.2. The Pearson test is described in Table 8. While the chi-square test and the mutual information test depicted in Table 2 reflect the better fitting of the model trained with L^1-loss function to the real data, see Table 3, the Pearson test does not, which could be traced back to the limitation of testing only linear correlations where chi-square and mutual information are generalizations in terms of measuring stochastic independence.

C Transformer architecture explanation

In this section, we explain the transformer architecture in more detail. Specifically, we explain the building blocks as depicted in (Vaswani et al., 2017, Figure 1).

C.1 Token embedding

In the transformer model, each token of the input sequence is first represented as a dense vector called a token embedding. This embedding captures the semantic meaning of the token in the context of the task being performed.

Each token is encoded by a vector $z \in \mathbb{R}^{d_z}$ one-hot encoding the corresponding token where $d_z \in \mathbb{N}$ is the number of different tokens generated from the vocabulary. The token embedding is obtained by applying a linear transformation $A \in \mathbb{R}^{C \times d_z}$ to the one-hot encoded representation of tokens z where $C \in \mathbb{N}$ is the number of dimensions of the transformer's internal representation of the embeddings (C sometimes also denoted with *embedding_size*). The entries of A are learnable weights and optimized during the training process. The mapping $A : \mathbb{R}^{d_z} \to \mathbb{R}^C$, $z \mapsto Az$ can be implemented as:

$$\text{token\_embedding} = \text{nn.Linear}(\text{config.vocab\_size}, \text{config.n\_embed})$$

where config.vocab_size $= d_z$ is the size of the vocabulary (number of unique tokens), and config.n_embed is the desired embedding dimension C. Consequently, this mapping turns the input shape (B, T, d_z) to the output shape of the token embedding (B, T, C), where B represents the batch size and T represents the sequence length of the input (number of tokens). The transformation A is applied to each token and element of a batch by $z(b, :, :) \mapsto z(b, :, :)A$ for each $b \in \{1, ..., B\}$ where $z \in \mathbb{R}^{B \times T \times d_z}$, $z(b, :, :) \coloneqq (z_{bik})_{i \in \{1,...,T\}, k \in \{1,...,d_z\}}$ provides the one-hot encoded representation for each token and each element of the batch. By using batches, several inputs can be considered at once.

For the specific case of time series modeling, the token embedding is replaced by the following. A convolutional neural network (CNN) is used for generating the embedding. The input channels of the CNN equal 1 in an autoregressive scenario (only historic parts of the time series itself are use to predict future parts of the time series) but can be set to any feature number $F \in \mathbb{N}$, which is measured at each time point, in case, e.g., an output is predicted from several input time series. Each time window for each channel, which cuts out each part of the F time series of length T and which is used as the input for the prediction, is represented by a vector of size T. These vectors from the sliding window are transformed by a one-dimensional CNN into the space $\mathbb{R}^C$. The padding is set such that the input length T equals the output length of the CNN. We remark that the shift one by one time points is not necessary and can be increased such that the token number in the embedding is smaller than the number of time points used as the input for the transformer. In our case, each filter (convolution) of the CNN is applied to each output, controlled by the parameter groups=1 and a common choice for the kernel size of 3. To achieve this transformation, we employ a one-dimensional convolutional layer in our implementation. Specifically, a 1D convolutional layer with an input channel size of F and an output channel size of C (the embedding dimension of the transformer) can be utilized:

$$\text{nn.Conv1d}(In\_channels = F, out\_channels = C)$$

This convolutional layer applies a set of learnable filters across the temporal dimension T of the input data, extracting relevant patterns and features. It's important to note that the kernel size, padding, and stride parameters of the convolutional layer can be adjusted to ensure that the output length matches the input length T. For more details, see the PyTorch documentation, e.g., https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html.

In general, the transformation (token embedding by a CNN) is applied as follows $CNN : \mathbb{R}^{T \times F} \to \mathbb{R}^{T \times C}$, $z(b, :, :) \mapsto CNN(z(b, :, :))$ to each each element of a batch $z(b, :, :) \in \mathbb{R}^{B \times T \times F}$, $z(b, :, :) \coloneqq (z_{bik})_{i \in \{1,...,T\}, k \in \{1,...,F\}}$, numerated by $b \in \{1, ..., B\}$ with the batch size $B \in \mathbb{N}$, (in parallel) generating a tensor of dimension (B, T, C).

For more details about the general framework of token embedding, see, e.g., Zhou et al. (2021).

C.2 Positional embedding

The purpose of a positional embedding is to include information about the position of a token relative to other tokens from the input into the total embedding of each token. The positional embedding is a vector of dimension C and is added to the token embeddings to provide information about the relative positions of tokens.

There are several methods available to code for positional information, see, e.g., Vaswani et al. (2017).

C.3 Attention head operation

The attention head is the essential building block of a transformer. Each attention head forms one layer where the output of one layer is processed by a subsequent layer until the output of the final layer is processed by a linear layer to obtain a corresponding output, see, e.g., (Vaswani et al., 2017, Figure 1).

Let's consider a single attention head operation in a transformer model. In this operation, we have the input tensor $x \in \mathbb{R}^{B\times T\times C}$. The tensor x can be the one after the token embedding (inclusive positional encoding) or the output of a previous layer. We remark that each layer in this presentation preserves the format (B, T, C).

The input tensor x is next transformed into different representations via linear transformation matrices. These matrices contain the learnable weights and are given by $W^K \in \mathbb{R}^{C\times C}$, $W^Q \in \mathbb{R}^{C\times C}$ and $W^V \in \mathbb{R}^{C\times C}$. We define a tensor for key (K), query (Q), and value (V) by the linear mappings

$$K(b,\cdot,\cdot) \coloneqq x(b,\cdot,\cdot)W^K \in \mathbb{R}^{T\times C}, \quad Q(b,\cdot,\cdot) \coloneqq x(b,\cdot,\cdot)W^Q \in \mathbb{R}^{T\times C}, \quad V(b,\cdot,\cdot) \coloneqq x(b,\cdot,\cdot)W^V \in \mathbb{R}^{T\times C}$$

for each $b \in \{1,...,B\}$ where $x(b,\cdot,\cdot) \in \mathbb{R}^{T\times C}$ such that $x(b,i,k) = x_{bik}$ for all $b \in \{1,...,B\}$, $i \in \{1,...,T\}$ and $k \in \{1,...,C\}$. Each of the matrices W^K, W^Q and W^V is an instance of a linear layer and can be implemented with PyTorch as follows:

$$nn.Linear(C, C, bias = False).$$

We remark that C may be called *embedding_size*.

In order to quantify the attention of token $i \in \{1,...,T\}$ represented in its key representation (K) with regard to token $j \in \{1,...,T\}$ of the input represented in its query representation (Q), the dot product is calculated for each $b \in \{1,...,B\}$ between the query (Q) and key (K) tensors over the vector embedding for each token pair $i, j \in \{1,...,T\}$. This operation can be represented as

$$\Theta : \{1,...,B\} \times \{1,...,T\} \times \{1,...,T\} \to \mathbb{R}, \quad (b,i,j) \mapsto \Theta(b,i,j) \coloneqq \frac{1}{\sqrt{C}} \sum_{k=1}^{C} Q_{b,i,k} \cdot K_{b,j,k} \tag{4}$$

where b represents the batch index, i and j represent the positions in the sequence (input), and k represents the embedding dimension. The sum is scaled by $C^{-0.5}$. One reason behind dividing the sum by the square root of the embedding dimension is given in the section about the Softmax function below, see Section C.3.1.

We implement the mapping Θ using the key (K) and query (Q) tensors and the Einstein summation convention as follows:

$$\Theta(b,i,j) = \frac{1}{\sqrt{C}} \text{torch.einsum}(bij, bkj- > bik, Q, K)$$

C.3.1 Softmax

After calculating the similarities between keys and queries, the purpose of the Softmax function is to normalize the scores of similarity. A high similarity between the key of token i and the query of token j is a proxy for a high association or attention the token i has to token j, meaning the connection is important for predicting the corresponding output. Due to the monotonicity of the Softmax function, a bigger similarity score between the corresponding key and query will result in a bigger value, called attention between the corresponding tokens, compared to smaller ones.

The Softmax function in our case is defined by

$$\text{Softmax} : \{1,...,B\} \times \{1,...,T\} \times \{1,...,T\} \to \mathbb{R}, \quad (b,i,j) \mapsto \text{Softmax}(b,i,j) \coloneqq \frac{\mathrm{e}^{\Theta(b,i,j)}}{\sum_{l=1}^{T} \mathrm{e}^{\Theta(b,i,l)}}.$$

Here, the Softmax function is applied along the last dimension, ensuring that the attention weights sum up to 1 along this dimension. This normalization means, fixing a batch number b and a token number i of the input provides us a normalized attention score about all the other token numbers $j \in \{1, ..., T\}$. The implementation is done by applying the corresponding Softmax function along dim$=-1$ to the tensor $\Theta_{b,i,j} \coloneqq \Theta(b, i, j)$ for all $b \in \{1, ..., B\}$ and $i, j \in \{1, ..., T\}$.

Next, we explain the normalization by $C^{-0.5}$ of (4). For large numbers, the Softmax function is approximately a constant function and changes in the weights of the transformer model do not result in a significant change of the attention. Depending on the embedding dimension C, the corresponding sum scales. For an illustration, see, e.g., (Vaswani et al., 2017, footnote 4). The scaling of the sum by $C^{-0.5}$ ensures to stay in a range where the Softmax is in an area of larger steepness and changes in the weights of the transformer result in significant changes of the attention values. Similarly, see (Vaswani et al., 2017, Section 3.2.1).

C.3.2 Weighted aggregation

We apply the attention scores to the value tensor (V) to obtain a weighted sum. The attention tensor is defined by

$$\mathcal{A}_{b,i,j} \coloneqq \text{Softmax}(b, i, j)$$

for all $b \in \{1, ..., B\}$ and $i, j \in \{1, ..., T\}$ such that $\mathcal{A} \in \mathbb{R}^{B \times T \times T}$. The weighted sum of attention scores is given by

$$\mathcal{A}(b, \cdot, \cdot)V(b, \cdot, \cdot) \in \mathbb{R}^{T \times C}$$

for each $b \in \{1, ..., B\}$ and turns the output of an attention head into the tensor format (B, T, C). The output tensor of attention head $H \in \mathbb{R}^{B \times T \times C}$ is defined by

$$H_{b,i,k} \coloneqq \sum_{l=1}^{T} \mathcal{A}_{b,i,l} V_{b,l,k}$$

for all $b \in \{1, ..., B\}$, $i \in \{1, ..., T\}$ and $k \in \{1, ..., C\}$.

C.3.3 Multi-head attention

Optionally, in order to calculate attention on different subspaces of keys and queries for the same input in each layer, there is a multi-head attention taking only projected parts of keys and queries.

In the multi-head attention formalism, the output of each head is concatenated along the last dimension, which is the embedding dimension

$$Concat((B, T, C/n\_heads), ..., (B, T, C/n\_heads)) = (B, T, C)$$

where n_heads is the number of attention heads. Specifically, the output of each head is calculated with the following weight matrices $W_h^K \in \mathbb{R}^{C \times C/n\_heads}$, $W_h^Q \in \mathbb{R}^{C \times C/n\_heads}$, $W_h^V \in \mathbb{R}^{C \times C/n\_heads}$ and $W^O \in \mathbb{R}^{C \times C}$ where $h \in \{1, ..., n\_heads\}$. Furthermore, n_heads and C are chosen such that the quotient C/n_heads is an integer. The matrices calculate the corresponding projections of keys, queries and values as follows $K^h(b, \cdot, \cdot) \coloneqq K(b, \cdot, \cdot)W_h^K \in \mathbb{R}^{T \times C/n\_heads}, Q^h(b, \cdot, \cdot) \coloneqq Q(b, \cdot, \cdot)W_h^Q \in \mathbb{R}^{T \times C/n\_heads}, V^h(b, \cdot, \cdot) \coloneqq V(b, \cdot, \cdot)W_h^V \in \mathbb{R}^{T \times C/n\_heads}$ for all $b \in \{1, ..., B\}$.

The weight matrices W_h^K, W_h^Q and W_h^V are each implemented with a linear layer according to

$$nn.Linear(C, head\_size, bias = False)$$

and $head\_size$ is calculated as

$$head\_size = \left\lfloor \frac{embedding\_size}{n\_heads} \right\rfloor$$

where for W^O the number $head\_size$ is replaced by C.

Applying the procedure for the single-head attention for each $h \in \{1, ..., n\_heads\}$ by replacing each K by the corresponding K^h, each Q by the corresponding Q^h and each V by the corresponding V^h provides us the tensor of each attention head $H^h \in \mathbb{R}^{B,T,C/n\_heads}$ where in (4) the index in the sum is only over 1 to C/n_heads each.

After processing all attention heads, the outputs are concatenated along the last dimension, resulting in a tensor of shape (B, T, C). The output of the multi-head attention is given by

$$H(b, \cdot, \cdot) \coloneqq \text{Concat}(H^1(b, \cdot, \cdot), ..., H^{n\_heads}(b, \cdot, \cdot))W^O \in \mathbb{R}^{T \times C}$$

for each $b \in \{1, ..., B\}$.

C.4 Adding and layer normalization

After the attention head operation, residual connections and layer normalization are applied.

The adding of the input and the output of a layer (residual learning He et al. (2016)) is crucial for maintaining an information flow and is easing the training of deep networks. The residual connection involves adding the input tensor x to the output of the multi-head attention operation H according to $x + H$ where $+$ denotes an element-wise addition. The addition helps to loop through the original information from the input to all the layers in a sequential layer architecture while also incorporating the information learned by the attention mechanism. A prerequisite is that the attention head preserves the input format.

Following the residual learning operation, layer normalization is applied to stabilize the learning process Ba et al. (2016) according to

$$\text{LayerNorm} \coloneqq \frac{z - E(z)}{\sqrt{Var(z) + \epsilon}}$$

where z is the output of the previous layer, $E(z)$ is the mean value of all the values of the output of the previous layer (mean over the elements of z) and $Var(z)$ is the corresponding variance. Since the square root is not differentiable at 0, a small constant $\epsilon > 0$ keeps the numerical implementation stable in case of a small variance of z. Moreover, the constant ϵ avoids division by zero errors. More details about the implementation can be found under `https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html`. Due to its construction, the normalization is done for each element of the batch.

We remark that for our implementation, the normalization is applied over the embedding dimension (C) separately for each token of the input (T dimension), meaning that $z \in \mathbb{R}^C$. This is reasonable since the feed forward networks (explained in Subsection C.5) are applied over the d_model (C) on each element of the batch (B) and input length (T).

C.5 Feed forward network (FFN)

After the normalization step of the attention head's residual learning, each token's representation is passed through a feed-forward neural network (FFN). Such an FFN consists of two linear transformations separated by a non-linear activation function $g : \mathbb{R}^{m \times n} \to \mathbb{R}^{m \times n}$, $m, n \in \mathbb{N}$, $z \mapsto g(z)$, such as ReLU (Rectified Linear Unit; $g = \max$) according to

$$\text{FFN}\left(x(b, t, \cdot)\right) = g\left(x(b, t, \cdot)W_1 + b_1\right) W_2 + b_2 \in \mathbb{R}^C$$

for each $b \in \{1, ..., B\}$ and $t \in \{1, ..., T\}$ where $x \in \mathbb{R}^{B \times T \times C}$ is the output from the previous operation in the architecture, $W_1 \in \mathbb{R}^{C \times d_{ff}}$, $d_{ff} \in \mathbb{N}$, in our implementation $d_{ff} = 4C$, $W_2 \in \mathbb{R}^{d_{ff} \times C}$ and $b_1 \in \mathbb{R}^{d_{ff}}$. Furthermore, $x(b, t, \cdot)W_1 \in \mathbb{R}^{d_{ff}}$, which is the same bias for each t in contrast to $b_1 \in \mathbb{R}^{T \times d_{ff}}$ where

for each t there is another bias, and $b_2 \in \mathbb{R}^C$ are learnable parameters. These operations are applied by Pytorch's linear layer `https://pytorch.org/docs/stable/generated/torch.nn.Linear.html`. The activation function g introduces non-linearity to the model, enabling it to learn non-linear patterns from the data. In the transformer architecture, an FFN is also followed by residual learning and layer normalization as described above in Subsection C.4.

C.6 Masking

Looking at (Vaswani et al., 2017, Figure 1), the masking is one of the essential building blocks located within the decoder (explained in Subsection C.7).

The masking of values of the attention weights Θ has the purposes of forcing attention between tokens to zero, meaning not allowing them to interact or to extract information from the interaction. Due to the iterative application of the transformer for text generation, we would like to force the attention mechanism that a token only considers tokens backwards in time (that come earlier in a sentence). This backward orientation helps to generate representations of tokens that collect information from tokens that are already there and prevents generating representations that make use of tokens that come after that token in a sentence. By the procedure of masking, the representations of tokens are more unified independent of the input length. As an example: "I am hungry and thus I go to a restaurant." Although probably "hungry" should get a lot of attention with "restaurant" without masking, in an iterative application of the transformer, the word "hungry" in "I am hungry and thus I go" would not be useful since its most attention was on restaurant that is not there yet. However, with masking we force the transformer to find embeddings and representations such that the word "hungry" gets a useful representation to predict the next token during learning, independent if the input is "I am hungry and thus I go to a restaurant." or "I am hungry and thus I go".

A different interpretation of the masking can be causality in a use case where the sequence of events is of importance forcing attention only to historic events.

We implement the masking effecting only backwards interaction by a lower triangular mask $M \in \mathbb{R}^{T\times T}$. This matrix is applied to the attention weights Θ generating masked attention weights. The tokens of the input are counted in the second dimension of the tensor Θ where the third dimension accounts for the dimension of the current vector representation (T or C depending on the current representation). Consequently, the lower triangular matrix (1 on the diagonal and below and 0 above), allows token 1 to have a non-zero attention only with token 1, token 2 can interact with token 2 and token 1 and so on until token T.

Subsequently, for each $b \in \{1, ..., B\}$, the entries of the attention weight tensor $\Theta(b, :, :)$ are replaced by $-\infty$ where M equals zero. The attention tensor Θ is thus transformed into the masked attention weight tensor Θ^M. The $-\infty$ forces the corresponding Softmax calculation to zero in the corresponding positions, meaning that the corresponding token does not pay attention to the corresponding other tokens.

C.7 Encoder and decoder

In a transformer architecture, there are typically two main components. This is the encoder and the decoder. Next, we explain both components according to (Vaswani et al., 2017, Figure 1).

Usually each layer in the encoder consists of an attention head followed by a residual learning and normalization, which is the input into a two linear transformation separated by a non-linear activation function, also followed by a residual learning and normalization. For the decoder, a layer consists of a masked attention unit (self-attention), followed by residual learning and layer normalization, followed by an attention unit where key and values are calculated from the corresponding encoder layer (cross-attention). The rest of the layer is according to an encoder layer. The repetition of layers of encoder and decoder is the main building block for the transformer.

The encoder processes the input sequence. These are the text input or for time series prediction the historic time series (the time series itself or other time series of features) generating a vector representation that

captures the contextual information of each token, which is meant also for the time series case as discussed in Subsection C.1.

The decoder, on the other hand, takes the encoded representations and generates an output sequence. It also consists of multiple layers, each containing self-attention mechanisms and cross-attention mechanisms. The self-attention mechanisms help the decoder focus on different parts of its input sequence, while the cross-attention mechanisms allow it to incorporate information from the encoder's output.

For the case of generating iteratively the next token for text generation, the prediction target is typically the next token in a sequence. Since for the text generation, several predicted tokens are required, the input of the decoder grows by the predicted token after each iteration. For inference the next token is predicted. Also during the training, the model is trained to predict the next token given the previous tokens in the input sequence. The number of input tokens for the decoder is given by $L \in \mathbb{N}$. The input to the encoder can be passed to the input of the decoder. If tokens are iteratively generated, the number L is supposed to be bigger than T in such cases, where T is the input length of the encoder. If the iterative output of the decoder becomes longer than a maximum size $\tilde{L} \in \mathbb{N}$ for the decoder's input, which can exist due to limitation on the hardware to calculate attention for such an input length between each token, then only the latest $\tilde{L}$ tokens are used as an input for the decoder. If the sequence is shorter, corresponding positions are masked out as explained in Subsection C.6. For translating, input language of the encoder's input can be different to the language of the decoder's output/input. In such a case, the encoder's input may not be passed to the decoder's input and the input of the decoder is iteratively generated by several applications of the transformer.

In any case, the output of the decoder $x \in \mathbb{R}^{B\times L\times C}$ is transformed by a linear map $\bar{W} : C \to \mathbb{R}^{d_z}$ with bias $\bar{b} \in \mathbb{R}^{d_z}$ such that the last dimension fits the number of available tokens from a dictionary according to

$$\bar{x}(b, i, :) \coloneqq x(b, i, :)\bar{W} + \bar{b} \in \mathbb{R}^{d_z}$$

for each $b \in \{1, ..., B\}$ and $i \in \{1, ..., L\}$. Then, the Softmax function is applied to the last slice L of the tensor $\bar{x}$ according to

$$\text{Softmax} : \{1, ..., B\} \times \{1, ..., d_z\} \to \mathbb{R}, \quad (b, s) \mapsto \text{Softmax}(b, s) \coloneqq \frac{\mathrm{e}^{\bar{x}_{b,L,s}}}{\sum_{l=1}^{d_z} \mathrm{e}^{\bar{x}_{b,L,l}}}$$

to obtain a probability over all possible tokens to choose the most likely token as the following token for each $b \in \{1, ..., B\}$. To include some variety on choosing the next token, we can disturb this distribution for the next token (e.g., introducing a temperature parameter) a bit such that also tokens become the most likely one that are close to the most likely token according to the undisturbed distribution over the tokens.

For time series forecasting tasks, the prediction target may vary depending on the application. It could be the next value in the time series sequence, multiple future values, or even a binary classification indicating whether certain conditions will be met in the future. The basic concept is that a linear transformation $\tilde{W} : C \to E$, $E \in \mathbb{N}$, with a bias $\tilde{b} \in \mathbb{R}^E$ transforms the output of the decoder to the output format that corresponds to what is to predict, like the number of features or the numbers of classes that is then turned into a probability over classes by a corresponding Softmax function.

In this work, we focus on time series prediction. As discussed in Zhou et al. (2021), it is advantageous to generate a multistep prediction (which includes a singlestep prediction) not by an iterative application of the transformer, like explained above, but provide the prediction at once, meaning to provide the prediction of length $L \in \mathbb{N}$ with a single application of the transformer. As a consequence, the training is done with a direct multistep loss. The rationale behind generating the prediction at once is to avoid error accumulation within the multistep ahead time series prediction task. Considering (Liu et al., 2022, Algorithm 4), we define the input for the decoder by $x \in \mathbb{R}^{B\times \frac{T}{2}+L\times F}$ where $L \in \mathbb{N}$ is the number of steps within the multistep prediction or prediction length, respectively, and $F \in \mathbb{N}$ the number of features, analogously to the token embedding for time series prediction tasks described in Subsection C.1. While for the encoder the initialization is the input sequence, the initialization values for the decoder are as follows. The first $\frac{T}{2}$ slices of the decoder input x are filled with the last $\frac{T}{2}$ slices of the input of the encoder $\tilde{x} \in \mathbb{R}^{B\times T\times F}$ according to $x_{b,i,f} = \tilde{x}_{b,\frac{T}{2}+i,f}$ for all

$b \in \{1, ..., B\}$, $i \in \{1, ..., \frac{T}{2}\}$, $f \in \{1, ..., F\}$. The last slices of the decoder are initialized with zeros according to $x_{b,i,f} = 0$ for all $b \in \{1, ..., B\}$, $i \in \{\frac{T}{2} + 1, ..., \frac{T}{2} + L\}$, $f \in \{1, ..., F\}$. This representation is embedded, see Subsection C.1, and processed as shown in (Vaswani et al., 2017, Figure 1) by a number of layers within the transformer. The output of the decoder, again denoted with $x \in \mathbb{R}^{B \times \frac{T}{2} + L \times C}$, is transformed by a linear mapping according to

$$P(b, i, :) \coloneqq x(b, i, :)\tilde{W} + \tilde{b} \in \mathbb{R}^E$$

for each $b \in \{1, ..., B\}$ and $i \in \{1, ..., \frac{T}{2} + L\}$ where $\tilde{W} \in \mathbb{R}^{C \times E}$, $\tilde{b} \in \mathbb{R}^E$ and $P \in \mathbb{R}^{B \times \frac{T}{2} + L \times E}$. In our application, where we predict the time series from its history, $F = E = 1$. The output after the linear transformation represents the L-step prediction and is given by

$$P(b, i, e) \text{ for all } i \in \{\frac{T}{2} + 1, ..., \frac{T}{2} + L\}$$

for each element of the batch $b \in \{1, ..., B\}$ and dimension $e \in \{1, ..., E\}$. Based on the output, loss functions are calculated with respect to the corresponding ground truth.

D Architecture for multilayer perceptrons for time series prediction

In this section, we describe the multilayer perceptron (MLP) architecture that we use for the time series prediction in this work. There is evidence that also MLPs are a very powerful model to predict time series Zeng et al. (2023).

Iteratively, an input tensor $x \in \mathbb{R}^{B \times F \times T}$ with $B \in \mathbb{N}$ as the batch size, $T \in \mathbb{N}$ as the length of the historic input of the time series for the prediction and $F \in \mathbb{N}$ as the number of features is transformed to the output tensor $y \in \mathbb{R}^{B \times F \times L}$ where $L \in \mathbb{N}$ is the length of the multistep prediction, which includes singlestep prediction where $L = 1$. In between there can be several hidden layers. All layers have the following structure taking an input tensor $z_{d-1} \in \mathbb{R}^{B \times F \times N_d}$ with a certain number of nodes ("neurons") $N_d \in \mathbb{N}$ where $d \in \{1, ..., n\}$, $n \in \mathbb{N}$ is the number of layers, $N_1 = T$, $N_{n+1} = L$ and $z_0 \coloneqq x$. The layer d is defined by the function given as follows

$$M_d : \mathbb{R}^{N_d} \to \mathbb{R}^{N_{d+1}}, \quad z_{d-1}(b, f, :) \mapsto M_d\left(z_{d-1}(b, f, :)\right) \coloneqq g_d\left(z_{d-1}(b, f, :)W_d + b_d\right)$$

for each $b \in \{1, ..., B\}$ and $f \in \{1, ..., F\}$ where $g_d : \mathbb{R}^{F \times N_{d+1}}$ is a pointwise applied non-linear activation function for each $d \in \{1, ..., n\}$, like the ReLu function where $g_d = \max$, $z_{d-1} \in \mathbb{R}^{B \times F \times N_d}$ is the output from layer $d - 1$ and the input for layer d, $W_d \in \mathbb{R}^{N_d \times N_{d+1}}$ and $b_d \in \mathbb{R}^{N_{d+1}}$. The operation $z_{d-1}(b, f, :)W_d$ is the common matrix-vector multiplication for any $d \in \{1, ..., n\}$, $b \in \{1, ..., B\}$, $f \in \{1, ..., F\}$. We remark that g_d can be but does not have to be a different function for each layer. For each $b \in \{1, ..., B\}$ and $f \in \{1, ..., F\}$, we have that $y(b, f, :) \coloneqq M_n(z_{n-1}(b, f, :)$. In this formulation, all weights in each layer d are the same for all features. This is the implementation we provide and is used in Zeng et al. (2023). However, in the examples within the present work, we have $F = 1$.

To implement a version that has different weights for each feature in each layer d, we just need to reformulate the input of the layers by $z_{d-1,f}(b, :) = z_{d-1}(b, f, :) \in \mathbb{R}^{N_d}$ for all $f \in \{1, ..., F\}$. Accordingly, the definition of the layers looks like

$$M_{d,f} : \mathbb{R}^{N_d} \to \mathbb{R}^{N_{d+1}}, \quad z_{d-1,f}(b, :) \mapsto M_{d,f}\left(z_{d-1,f}(b, :)\right) \coloneqq g\left(z_{d-1,f}(b, :)W_{d,f} + b_{d,f}\right)$$

where applying the definitions separately to each feature leads to F different MLPs where the weight matrices and bias can differ per feature.

In order to introduce cross learning where information from one feature can influence the prediction of other features, we need to reshape the three dimensional tensor (B, F, N_d) to (B, FN_d) for some layers where a corresponding weight matrix $W_d^* : FN_d \to FN_{d+1}$ can mix information from different features.

In a multi layer architecture, we can combine cross learning and learning per feature in different layers assembling them in one model by reshaping outputs in the corresponding formats from (B, F, N_d) to (B, FN_d) or (B, FN_d) to (B, F, N_d) after a layer before the next one depending on the learning type to change.

With Pytorch such layers are implemented with

$$nn.Linear(n, m, bias = True)$$

where $n \in \mathbb{N}$ is the dimension of the input and $m \in \mathbb{N}$ is the dimension of the output. The *bias* parameter $True$ adds a bias with non-zero values and the parameter $False$ fixes the values of the bias to zero.