Self-Attention Between Datapoints: Going Beyond Individual Input-Output Pairs in Deep Learning

Jannik Kossen[1*] **Neil Band**[1*]

Clare Lyle[1] **Aidan Gomez**[1,3] **Tom Rainforth**[2] **Yarin Gal**[1]

[1] OATML, Department of Computer Science, University of Oxford
[2] Department of Statistics, University of Oxford
[3] Cohere

Abstract

We challenge a common assumption underlying most supervised *deep learning*: that a model makes a prediction depending only on its parameters and the features of a *single input*. To this end, we introduce a general-purpose deep learning architecture that takes as input the *entire dataset* instead of processing one datapoint at a time. Our approach uses self-attention to reason about relationships between datapoints explicitly, which can be seen as realizing non-parametric models using parametric attention mechanisms. However, unlike conventional non-parametric models, we let the model learn end-to-end from the data how to make use of other datapoints for prediction. Empirically, our models solve cross-datapoint lookup and complex reasoning tasks unsolvable by traditional deep learning models. We show highly competitive results on tabular data, early results on CIFAR-10, and give insight into how the model makes use of the interactions between points.

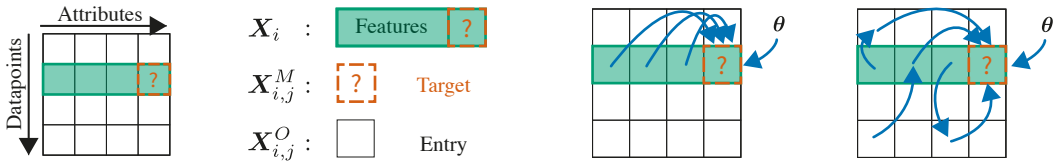
1 Introduction

From CNNs [49] to Transformers [76], most of supervised deep learning relies on *parametric* modeling: models learn parameters $\boldsymbol{\theta}$ from a set of training data $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ to maximize training likelihoods $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$ mapping from features $\boldsymbol{x} \in \mathcal{X}$ to target values $\boldsymbol{y} \in \mathcal{Y}$. At test time, they then make a prediction $p(\boldsymbol{y}^* \mid \boldsymbol{x}^*; \boldsymbol{\theta})$ that depends only on those parameters $\boldsymbol{\theta}$ and the test input $\boldsymbol{x}^*$. That is, parametric models do not consider direct dependencies between datapoints.

This paper challenges parametric modeling as the dominant paradigm in deep learning. Based on the same end-to-end learning motivations that underpin deep learning itself, we consider giving models the *additional flexibility* of using training data *directly* when making predictions $p(\boldsymbol{y}^* \mid \boldsymbol{x}^*, \mathcal{D}_{\text{train}}; \boldsymbol{\theta})$.

Concretely, we introduce **Non-Parametric Transformers (NPTs)**: a general deep learning architecture that takes the entire dataset as input and predicts by explicitly *learning* interactions between datapoints (Fig. 1). NPTs leverage both parametric and *non*-parametric predictive mechanisms, with the use of end-to-end training allowing the model to naturally learn from the data how to balance the two. Namely, instead of just learning predictive functions from the features to the targets of independent datapoints, NPTs can also learn to reason about general relationships *between* inputs. We show that these models *learn* to look up information from other datapoints and capture the causal mechanism generating the data in semi-synthetic settings. However, unlike conventional non-parametric models, NPTs are not forced to *only* make predictions in this manner: they can also use the power of conventional parametric deep learning. We use multi-head self-attention [4, 51, 76] to model relationships between datapoints and construct a training objective for NPTs with a stochastic masking mechanism inspired by recent work in natural language processing [23].

[*] **Equal Contribution.** Correspondence to {jannik.kossen, neil.band}@cs.ox.ac.uk.



(a) Input Data (b) Notation (c) Parametric Model (d) **NPT**

Figure 1: NPTs learn direct interactions between datapoints. (a) Input data: predict masked target entry **[?]** for datapoint $\boldsymbol{X}_i$. (b) Notation from §2. (c) Parametric models predict only from the features of the given input. (d) NPTs predict by modeling relationships between all points in the dataset.

A key contribution of this paper is opening the door to more general treatment of how deep learning models can make use of dependencies between datapoints for predictions. Our results demonstrate that NPTs make use of interactions between datapoints in practice, and we show highly competitive performance on several established tabular datasets as well as early image classification results. Additionally, we show that NPTs can solve complex reasoning tasks by combining representation learning and cross-datapoint lookup; something that is impossible for conventional deep learning or non-parametric models due to their inability to *learn* relations *between* datapoints.

Background. While questioning parametric modeling assumptions is unconventional in deep learning, in statistics so-called *non-parametric* models are a well-known and long-established field of study. Non-parametric models make predictions in explicit dependence of the training data $p(\boldsymbol{y}^* \mid \boldsymbol{x}^*, \mathcal{D}_{\text{train}})$. The most popular example of such models in the machine learning community are perhaps Gaussian Processes [64]. Non-parametric models typically do not require any training of parameters, and instead often directly interpolate between training points according to a fixed procedure, e.g., [64, p.17]. The interactions between inputs are fully defined by architectural choices and a small set of hyperparameters that must be carefully chosen. Conventional non-parametric models cannot *learn* – in the sense familiar to deep learning practitioners – interactions from the data, limiting the flexibility these models have in adapting to the data at hand. Approaches such as Deep Gaussian Processes [21], Deep Kernel Learning [78], and Neural Processes [32, 33, 42] have all sought to apply ideas from deep neural networks to non-parametrics. Compared to NPTs, these approaches rely heavily on motivations from stochastic processes. This leads to them being either less flexible than NPTs or requiring strong assumptions on the data, making them *inapplicable* to the practical scenarios considered in this paper (cf. §3). Unlike previous work, NPTs explicitly learn interactions between datapoints and can be applied to general supervised machine learning tasks. We refer to §3 for an overview of these and other related approaches.

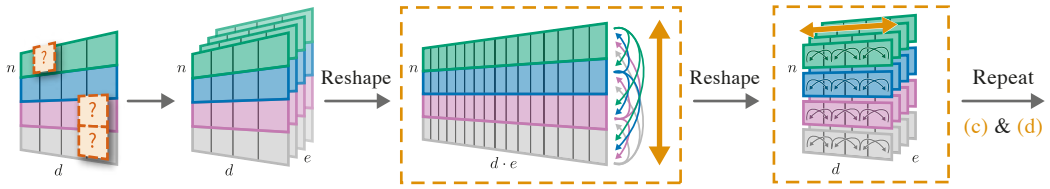
We next discuss the specifics of our model (§2), before moving on to related work (§3), empirical results (§4), and finally, limitations, future work, and conclusions (§5).

2 Non-Parametric Transformers

Non-Parametric Transformers (NPTs) explicitly *learn* relationships between datapoints to improve predictions. To accomplish this, they rely on three main ingredients: **(1)** We provide the model with the **entire dataset – all datapoints – as input**. At test time, both training and test data are input to the model; during training, the model learns to predict targets from the training data only. We approximate this where necessary for large data (§2.6). **(2)** We use **self-attention between datapoints** to explicitly model relationships between them. For example, at test time, the attention mechanism models relationships amongst training points, amongst test points, and between the two. **(3)** NPT's training objective is to reconstruct a corrupted version of the input dataset. Similar to BERT [23], we apply **stochastic masking** to both features and targets and minimize a loss on NPT's predictions at entries masked out in the input. Next, we introduce the three components in detail.

2.1 Datasets as Inputs

NPTs take as input the entire dataset $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. The datapoints are stacked as the rows of this matrix $\{\boldsymbol{X}_{i,:} \in \mathbb{R}^d \mid i \in 1 \dots n\}$, and we refer to the columns as attributes $\{\boldsymbol{X}_{:,j} \in \mathbb{R}^n \mid j \in 1 \dots d\}$. Each attribute is assumed to share a semantic meaning among all datapoints. In single-target classification and regression, we assume that the targets (labels) are the final attribute $\boldsymbol{X}_{:,d}$, and the other attributes $\{\boldsymbol{X}_{:,j} \mid j \neq d\}$ are input features, e.g., the pixels of an image. Each $\boldsymbol{X}_{i,j}$ is an entry or value. In addition to tabular data, many modalities such as images, graphs, or timeseries can be reshaped to fit



(a) Input (b) Embedding (c) **Datapoint Attention** (d) **Attribute Attention**

Figure 2: Overview of the Non-Parametric Transformer. (a) The input dataset and mask matrix are stacked and (b) linearly embedded for all datapoints independently. NPT then applies (c) Attention Between Datapoints (ABD, §2.4) across all n samples of hidden dimension $h = d \cdot e$. (d) Attention Between Attributes (ABA, §2.5) then attends between the attributes for each datapoint independently. We repeat steps (c) and (d) and obtain a final prediction from a separate linear projection (not shown).

this format. Note that this is a departure from common notation for supervised learning as introduced in §1, as the input $\boldsymbol{X}$ now includes both features and targets (collectively, attributes).

In masked language modeling [23], mask tokens denote which words in a sentence should be concealed and where model predictions will have a loss backpropagated at training time. Analogously, we use a binary matrix $\boldsymbol{M} \in \mathbb{R}^{n \times d}$ to specify which entries are *masked* in the input $\boldsymbol{X}$. This matrix is also passed to NPT as input. The task is to predict the masked values $\boldsymbol{X}^M = \{\boldsymbol{X}_{i,j} \mid \boldsymbol{M}_{i,j} = 1\}$ from the observed values $\boldsymbol{X}^O = \{\boldsymbol{X}_{i,j} \mid \boldsymbol{M}_{i,j} = 0\}$, i.e., to predict $p(\boldsymbol{X}^M \mid \boldsymbol{X}^O)$.

In summary, NPT takes as input the entire dataset and masking matrix $(\boldsymbol{X}, \boldsymbol{M})$, and makes predictions $\hat{\boldsymbol{X}} \in \mathbb{R}^{n \times d}$ for values masked at input. This general setup accommodates many machine learning settings simply by adjusting the placement of the binary masks in $\boldsymbol{M}$. We focus on single-target classification and regression – corresponding to a masking matrix $\boldsymbol{M}$ with 1s at all entries of the label column $\boldsymbol{X}_{:,d}$ – but outline multi-target settings, imputation, self-supervision using input features, and semi-supervision in Appendix C.6. Next, we describe the NPT architecture.

2.2 NPT Architecture

An overview of the Non-Parametric Transformer (NPT) is depicted in Fig. 2. NPT receives the dataset and masking matrix $(\boldsymbol{X}, \boldsymbol{M})$ as input (Fig. 2a). We stack these and apply an identical linear embedding to each of n datapoints, obtaining an input representation $\boldsymbol{H}^{(0)} \in \mathbb{R}^{n \times d \times e}$ (Fig. 2b). Next, we apply a sequence of multi-head self-attention layers [4, 23, 76]. Crucially, we alternatingly apply attention between *datapoints*, and attention between *attributes* of individual datapoints (Figs. 2c-d).

These operations allow our model to learn both relationships between datapoints as well as transformations of individual datapoints. Finally, an output embedding gives the prediction $\hat{\boldsymbol{X}} \in \mathbb{R}^{n \times d}$, which now has predicted values at entries that were masked at input. We refer to Appendix C.3 for details, such as treatment of categorical and continuous variables. Importantly:

Property 1. *NPTs are equivariant to a permutation of the datapoints. (cf. Appendix A for proof.)*

In other words, if the set of input datapoints are shuffled, NPTs produce the same predictions but shuffled in an analogous manner. This explicitly encodes the assumption that the learned relations between datapoints should not depend on their ordering. At a high level, permutation-equivariance (PE) holds because all components of NPT are PE, and the composition of PE functions is PE. We now briefly recap multi-head self-attention, an important operation in the NPT architecture.

2.3 Multi-Head Self-Attention

Multi-head self-attention (MHSA) is a powerful mechanism for learning complex interactions between elements in an input sequence. Popularized in natural language processing [4, 23, 76], MHSA-based models have since been successfully applied to many areas of machine learning (cf. §3).

Dot-product attention computes attention weights by comparing queries $\{\boldsymbol{Q}_i \in \mathbb{R}^{1 \times h_k} \mid i \in 1 \dots n\}$ with keys $\{\boldsymbol{K}_i \in \mathbb{R}^{1 \times h_k} \mid i \in 1 \dots m\}$, ultimately updating the representation of the queries by aggregating over values $\{\boldsymbol{V}_i \in \mathbb{R}^{1 \times h_v} \mid i \in 1 \dots m\}$ via the attention weights. We stack the queries, keys, and values into matrices $\boldsymbol{Q} \in \mathbb{R}^{n \times h_k}$, $\boldsymbol{K} \in \mathbb{R}^{m \times h_k}$, and $\boldsymbol{V} \in \mathbb{R}^{m \times h_v}$ and, as is commonly done for convenience, assume $h_k = h_v = h$. Then, we compute dot-product attention as

$$\text{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^T / \sqrt{h})\boldsymbol{V}. \tag{1}$$

Multi-head dot-product attention concatenates a series of k independent *attention heads*

$$\text{MHAtt}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \underset{\text{axis}=h}{\text{concat}}(\boldsymbol{O}_1, \ldots, \boldsymbol{O}_k)\boldsymbol{W}^O, \text{ where } \boldsymbol{O}_j = \text{Att}(\boldsymbol{Q}\boldsymbol{W}_j^Q, \boldsymbol{K}\boldsymbol{W}_j^K, \boldsymbol{V}\boldsymbol{W}_j^V). \quad (2)$$

We learn embedding matrices $\boldsymbol{W}_j^Q, \boldsymbol{W}_j^K, \boldsymbol{W}_j^V \in \mathbb{R}^{h \times h/k}, j \in \{1, \ldots, k\}$ for each head j, and $\boldsymbol{W}^O \in \mathbb{R}^{h \times h}$ mixes outputs from different heads. Here, we focus on multi-head *self*-attention, $\text{MHSelfAtt}(\boldsymbol{H}) = \text{MHAtt}(\boldsymbol{Q} = \boldsymbol{H}, \boldsymbol{K} = \boldsymbol{H}, \boldsymbol{V} = \boldsymbol{H})$, which uses the *same* inputs for queries, keys, and values. Following Transformer best practices to improve performance [15, 23, 51, 56, 76], we first add a residual branch and apply Layer Normalization (LN) [3] followed by $\text{MHSelfAtt}(\cdot)$,

$$\text{Res}(\boldsymbol{H}) = \boldsymbol{H}\boldsymbol{W}^{\text{res}} + \text{MHSelfAtt}(\text{LN}(\boldsymbol{H})), \quad (3)$$

with learnable weight matrix $\boldsymbol{W}^{\text{res}} \in \mathbb{R}^{h \times h}$. Then, we add another residual branch with LN and a row-wise feed-forward network (rFF), finally giving the full multi-head self-attention layer as

$$\text{MHSA}(\boldsymbol{H}) = \text{Res}(\boldsymbol{H}) + \text{rFF}(\text{LN}(\text{Res}(\boldsymbol{H})) \in \mathbb{R}^{n \times h}. \quad (4)$$

2.4 Attention Between Datapoints (ABD)

The **Attention Between Datapoints (ABD)** layer is a key operation for NPT. It explicitly transforms data by reasoning about pairwise relationships between all datapoints, see Fig. 2c. As input to ABD, we flatten the output of the previous layer $\boldsymbol{H}^{(\ell)}$ from $\mathbb{R}^{n \times d \times e}$ to $\mathbb{R}^{n \times h}$ with $h = d \cdot e$. Then, we perform multi-head self-attention between the datapoints $\{\boldsymbol{H}_i^{(\ell)} \in \mathbb{R}^{1 \times h} \mid i \in 1 \ldots n\}$ as

$$\text{ABD}(\boldsymbol{H}^{(\ell)}) = \text{MHSA}(\boldsymbol{H}^{(\ell)}) = \boldsymbol{H}^{(\ell+1)} \in \mathbb{R}^{n \times h}. \quad (5)$$

At the first ABD layer, we input $\boldsymbol{H}^{(0)} \in \mathbb{R}^{n \times d \times e}$, the linearly embedded input data. After applying ABD, we reshape the output again, from $\mathbb{R}^{n \times h}$ to $\mathbb{R}^{n \times d \times e}$.

Note that this is distinct from how $\text{MHSA}(\cdot)$ is usually applied in the literature, as we compute attention between *different datapoints* and not between the *features of a single datapoint* [23, 24, 39, 76]. For example, in natural language processing, attention is usually applied between the tokens (features) of a sentence (datapoint) but not between different sentences. For example, NPT could learn to attend between two datapoints with indices i and i' by embedding $\boldsymbol{Q}_i$ and $\boldsymbol{K}_{i'}$ in close proximity. Following (1), datapoint i will then attend more closely to i' because $\boldsymbol{Q}_i \boldsymbol{K}_{i'}^T$ will be large. By stacking many ABD layers, NPT can learn higher-order interactions between datapoints [23, 76].

2.5 Attention Between Attributes (ABA)

We now introduce **Attention Between Attributes (ABA)**, which is always performed following ABD. ABA layers can help the model learn better per-datapoint representations for the between-datapoint interactions, see Fig. 2d. In ABA, we apply MHSA independently to each row (corresponding to a single datapoint) in the input $\boldsymbol{H}_i^{(\ell)} \in \mathbb{R}^{d \times e}, i \in \{1, \ldots, n\}$, giving

$$\text{ABA}(\boldsymbol{H}^{(\ell)}) = \underset{\text{axis}=n}{\text{stack}}(\text{MHSA}(\boldsymbol{H}_1^{(\ell)}), \ldots, \text{MHSA}(\boldsymbol{H}_n^{(\ell)})) = \boldsymbol{H}^{(\ell+1)} \in \mathbb{R}^{n \times d \times e}. \quad (6)$$

Just like in standard Transformers [23, 24, 39, 76], ABA is used to transform attribute representations of single datapoints independently. We batch over the n dimension to compute ABA efficiently. By alternating between attention over datapoints (ABD) and attributes (ABA), NPTs can model both complex dependencies between points as well as learn suitable transformations of datapoints individually. Next, we describe the use of masking mechanisms during NPT training and evaluation.

2.6 Masking and Optimization

Masking. Much like in masked language modeling [23], we use masks to indicate which values NPT is expected to predict, and to prevent the model from accessing ground truth values. Recall that NPT needs to predict $p(\boldsymbol{X}^M \mid \boldsymbol{X}^O)$, with masked values $\boldsymbol{X}^M = \{\boldsymbol{X}_{i,j} \mid \boldsymbol{M}_{i,j} = 1\}$ and observed values $\boldsymbol{X}^O = \{\boldsymbol{X}_{i,j} \mid \boldsymbol{M}_{i,j} = 0\}$. Masked values can be either features or targets. Canonically, masked language modeling is used to perform self-supervised learning on a sequence of tokens in a sentence [23]. We use such *stochastic feature masking* to mask a feature value $\boldsymbol{X}_{i,j}, j \neq d$ with probability p_{feature} during training. *Stochastic target masking* is done in the same manner on the targets of the training set $\boldsymbol{X}_{:,d}$ with p_{target}. Note that we take great care to avoid test set leakage, and *never* reveal targets of the test set to NPT. Appendix C.6 gives full details on the masking procedure.

NPT Objective. During training, we compute the negative log-likelihood loss at training targets $\mathcal{L}^{\text{Targets}}$ as well as the auxiliary loss from masked-out features $\mathcal{L}^{\text{Features}}$. We write the NPT training objective as $\mathcal{L}^{\text{NPT}} = (1-\lambda)\mathcal{L}^{\text{Targets}} + \lambda\mathcal{L}^{\text{Features}}$, where λ is a hyperparameter. At test time, we only mask and compute a loss over the targets of test points. See Appendix C.7 for optimization details.

This objective has a few notable elements. Feature masking requires NPTs to make predictions over all attributes, encouraging the models to learn a representation of the entire dataset. This increases the difficulty of the task and adds more supervision, which we find tends to have a beneficial regularizing effect. Interestingly, stochastic target masking means that many training targets are *unmasked* to the model at training time. This allows NPTs to learn to predict, at each epoch, the masked targets of certain training datapoints using the *targets of other training datapoints* in addition to all training data features.[2] NPTs no longer have to memorize a mapping between training inputs and outputs in their parameters $\boldsymbol{\theta}$, and can instead use their representational capacity to learn functions using other *training features and targets as input*. For example, NPTs could learn to assign test datapoints to clusters of training datapoints, and predict on those points using interpolation of the training targets in their respective cluster. We explore the ability of NPTs to solve such complex reasoning tasks in §4.2.

Handling Large Datasets. Avoiding the poor $\mathcal{O}(n^2)$ time and space complexity of naïve self-attention, we resort to approximations once the data grows too large. For example, we reach 24 GB of GPU memory for standard NPT model sizes at about 8000 datapoints. We find that processing the data in random subsets for model training and prediction, i.e., *minibatching*, is a simple and effective solution. We construct minibatches such that, at test time, training and test data are both present in the same batch, to allow NPTs to attend to training datapoints. In §4.3, we show that NPTs make use of attention between datapoints with minibatching enabled. See §5 for further discussion and ideas for future work.

3 Related Work

Deep Non-Parametric Models. Deep Gaussian Processes [21] and Deep Kernel Learning (DKL) [78] extend ideas from Gaussian Processes [64] to representation learning. Deep GPs stack standard GPs with the aim to learn more expressive relationships between input points, sharing motivation with NPTs. However, unlike NPTs, deep GPs are difficult to work with in practice, requiring complex approximate inference schemes [13, 20, 66]. DKL applies a neural network to each datapoint *independently* before passing points on to a standard Gaussian Process, making predictions based directly on similarity in embedding space instead of *learning* the interactions themselves.

Neural Processes. Similar to GPs, Neural Processes (NPs) [32, 33] define a distribution over functions. They use a latent variable model parametrized by neural networks, fulfilling specific architectural constraints to approximately preserve consistency of finite-dimensional marginals. Attentive Neural Processes (ANPs) [42] extend Neural Processes to allow for direct attention between a context set and targets. However, as the authors themselves stress, "NPs and GPs have different training regimes" [42]. While a GP can be trained on a single dataset, *(A)NPs require multiple realizations of the dataset*. The authors further note that *"a direct comparison between the two is usually not plausible"* [42], which is why we cannot compare (A)NPs to NPT on our standard tasks.

Attention. NPTs are part of a line of recent work that explores the use of Transformer-based architectures outside of natural language processing, e.g., Transformers in computer vision [24, 39, 57] or architectures exploiting desirable invariances or equivariances [30, 37, 51, 53]. Like NPTs, Set Transformer [51] attends to a set of input points. However, unlike NPTs, Set Transformer relies on the existence of multiple independent sets for training and makes only a single prediction for each set. Like NPTs, Axial Transformers [35] and MSA Transformers [63] attend to multiple dimensions of matrix-shaped input. However, Axial Transformers process single images as input, i.e., no attention across datapoints is performed. MSA Transformers use attention within individual protein sequences and across an aligned protein family for contact prediction, but do not consider a more general setting. Recent works have improved neural network performance on tabular data using attention. AutoInt [68] is a direct application of multi-head attention to tabular data, and TabNet [2] sequentially attends to sparse subsets of the features inspired by tree-based models. Both approaches do not reason about interactions between datapoints, a key contribution that we introduce with NPT in this work.

[2] A potential concern is that the model will memorize training targets and fail to generalize. In practice, we do not observe generalization issues, likely because (i) a loss is never backpropagated on an unmasked value, and (ii) BERT-style masking [23] uses token randomization to prevent memorization. See Appendix C.6.

Table 1: Average rank order of various methods ($\pm$ standard error) on UCI benchmarks, across binary classification, multi-class classification, and regression tasks. We determine rank using the test area under the receiver operating characteristic (AUROC) curve on binary classification (4 of 10 datasets), accuracy on multi-class classification (2 of 10), and root mean squared error (RMSE) on regression (4 of 10), and sort methods by ascending rank for each metric. See Appendix B.5 for full results.

Method	AUROC
NPT	**2.50 $\pm$ 0.87**
CatBoost	2.75 $\pm$ 0.85
LightGBM	3.50 $\pm$ 1.55
XGBoost	4.75 $\pm$ 1.25
Gradient Boosting	5.00 $\pm$ 0.71
MLP	5.75 $\pm$ 1.49
Random Forest	6.00 $\pm$ 0.71
TabNet	6.50 $\pm$ 1.32
k-NN	8.25 $\pm$ 0.48

Method	Accuracy
NPT	**2.50 $\pm$ 0.50**
XGBoost	**2.50 $\pm$ 1.50**
MLP	3.00 $\pm$ 2.00
CatBoost	3.50 $\pm$ 0.50
Gradient Boosting	3.50 $\pm$ 1.50
Random Forest	6.50 $\pm$ 0.50
TabNet	7.50 $\pm$ 0.50
LightGBM	7.50 $\pm$ 1.50
k-NN	8.50 $\pm$ 0.50

Method	RMSE
CatBoost	**3.00 $\pm$ 0.91**
XGBoost	3.25 $\pm$ 0.63
NPT	3.25 $\pm$ 1.31
Gradient Boosting	4.00 $\pm$ 1.08
Random Forest	4.50 $\pm$ 0.87
MLP	5.00 $\pm$ 1.22
LightGBM	6.50 $\pm$ 1.55
TabNet	6.75 $\pm$ 0.95
k-NN	8.75 $\pm$ 0.25

Few-Shot Learning, Meta-Learning, and Prompting. In §4.2, we apply NPTs to tasks that require learning of relational structure between datapoints on training data to achieve good generalization performance on novel test inputs. This setup shares motivations with meta-learning [6, 8, 26, 48], in which a model is pre-trained on a variety of tasks, such that it can then learn new tasks using only a small number of additional training points from the new task. However, we consider evaluation without any additional gradient updates, unlike recent meta-learning methods [26, 80] which are therefore inapplicable to this setting. Recent works on few-shot learning with text prompting [12, 62] provide a trained Transformer-based language model with a few examples of a novel relationship in a prompt at prediction time, and observe strong generalization on the task. Similarly, we consider attention between a "context" of datapoints. While ground-truth input-output pairs are provided for prompting, we consider settings in which no ground-truth is given at prediction time (cf. Appendix B.1.2), but the model can solve the task if it has learned the underlying relational structure.

Due to the unique properties of NPTs, we believe that there are many other exciting connections to be drawn. We discuss a selection of possible areas of application including semi-supervised learning, graph neural networks, and relational learning in Appendix D, and leave other areas such as prediction on missing data, semi-supervised learning, and continual learning to future research. In this initial study, we instead concentrate on questions at the core of NPTs.

4 Experiments

We seek to answer the following set of questions in our evaluation[3] of NPTs: (**Q1**) How do NPTs perform on standard benchmarks for supervised machine learning? (**Q2**) Can NPTs successfully model interactions between datapoints in idealized settings? (**Q3**) Do NPTs actually learn to rely on interactions between datapoints for prediction on real-world datasets? (**Q4**) If so, what is the nature of these interactions, e.g., which other datapoints are relevant for prediction?

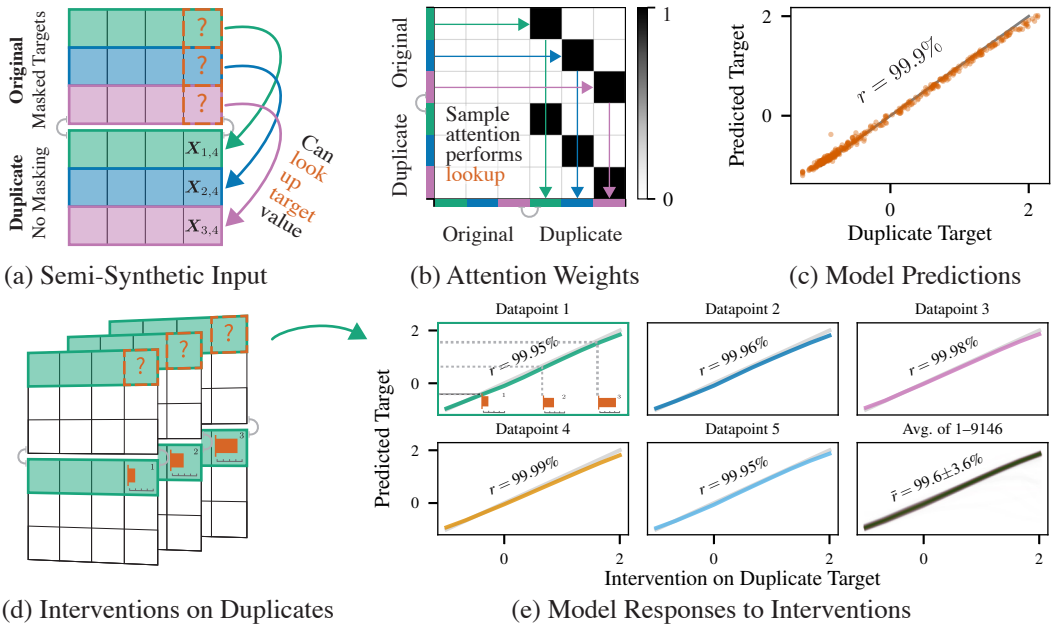
4.1 NPTs Perform Competitively on Established Benchmarks

To answer **(Q1)**, we evaluate NPTs on tabular data from the UCI Repository [25] as well as the CIFAR-10 [47] and MNIST [50] image classification datasets. Tabular data is ubiquitous in real-world machine learning [19] but notoriously challenging for general purpose deep neural networks, which consistently underperform boosting models [67] and are rarely used in practice.[4]

Tabular Datasets, Setup, and Baselines. We evaluate NPTs over 10 datasets varying across the number of datapoints, number of features, composition (categorical or continuous) of features, and task. 4 of the 10 are binary classification, 2 are multi-class classification, and 4 are regression. We compare NPT against a wide set of standard or state-of-the-art baselines: Random Forests [10], Gradient Boosting Trees [29], XGBoost [16], CatBoost [61], LightGBM [41], MLPs, k-NN [1, 27], and TabNet [2]. For additional background on tree-based models see Appendix D.2. We tune the parameters of all models on validation sets and use 10-fold cross-validation whenever

[3] We release code for NPTs at github.com/OATML/Non-Parametric-Transformers.

[4] We conduct an informal survey of all Kaggle [38] competitions using tabular data completed in 2020 with a public leaderboard. In 11 out of a total of 13 cases, the winning entries relied on some form of boosting.



(a) Semi-Synthetic Input (b) Attention Weights (c) Model Predictions

(d) Interventions on Duplicates (e) Model Responses to Interventions

Figure 3: Demonstrating NPT's ability to predict from Attention Between Datapoints (ABD). (a) We append to the original data with masked targets [?] a copy of the same data with all masked values revealed, such that perfect prediction via lookup is possible. (b) Attention weights indicate that the ideal lookup behavior is learned by NPT. Shown are actual values learned by NPT at head 0 and depth 4 for the first 3 datapoints. (c) NPT predictions closely match the ideal values. (d) Additionally, we intervene on the values of individual targets, (e) finding that NPT predictions adjust accordingly.

computationally feasible. Note that while we perform an extensive grid search for the baselines, we only search over a small set of configurations for NPTs. We refer the reader to Appendix E for further details on the datasets and baseline setups, and Appendix C.1 for NPT hyperparameters.

Tabular Data Results. We report the average rank order for NPT and various tree-based and deep learning baselines in Table 1. NPT achieves the highest average ranking on binary and multi-class classification tasks, outperforming CatBoost and XGBoost, two popular state-of-the-art boosting methods designed specifically for tabular data. On regression tasks, NPT ties in average rank with XGBoost, and is outperformed only by CatBoost. In addition to its strong rank-wise performance, NPT achieves best performance on 4 of the 10 benchmark datasets – more than any other method. We find that these are remarkable results for a general purpose model that does not include tabular-specific design, supporting our hypothesis that attention between datapoints is a useful architectural inductive bias for prediction. For all metrics across all datasets, i.e., NLL for classification, AUROC/accuracy for binary/multi-class classification, and (R)MSE for regression, we refer the reader to Appendix B.5.

Image Data Results. NPT achieves 68.2% accuracy on CIFAR-10 and 98.3% accuracy on MNIST. Similar to previous work on Transformers for computer vision, we would expect (pre)-training on millions of images to significantly boost NPT's performance [22, 39, 65, 71, 74]. We perform no pre-training, and therefore a direct comparison of our results to this line of work is inappropriate. Crucially, we show in §4.3 that NPTs learn to make use of interactions between images, indicating that attention between datapoints is valuable for image classification. Appendix B.6 contains further discussion.

4.2 NPTs Can Learn to Predict Using Attention Between Datapoints

To determine if NPTs can successfully learn to exploit interactions between datapoints **(Q2)**, we introduce a task with strong input correlations for which we know ground-truth interactions. Concretely, we take the UCI Protein regression dataset (cf. §4.1), to construct the following semi-synthetic task: for each batch, we input the original data with masked target values as well as a *copy* of the original data where all target values have been revealed, i.e., no masking is applied (Fig. 3a). NPTs can use attention between datapoints to achieve arbitrarily good performance by *learning* to look up the target values in the matching duplicate row. At test time, we input novel semi-synthetic test data to ensure that NPT has learned the correct relational mechanism and not just memorized target values.

Table 2: Drop in NPT performance after destroying information from other datapoints. Shown are changes in test set performance, where negative values indicate worse performance after corruption.

Δ *Accuracy*	CIFAR-10	Poker	Income	Higgs	MNIST	Forest	Kick	Breast Cancer
	–5.1	–1.1	–1.1	–0.5	–0.4	–0.1	–0.1	0.0
$\Delta_{RMSE}/_{RMSE}$ (%)	Yacht	Protein	Boston	Concrete				
	–52%	–21%	–20%	–7%				

NPTs successfully learn to perform this lookup between original and duplicate datapoints. The ABD attention weights, visualized for the first three datapoints in Fig. 3b, clearly show the model correctly attending to the duplicates. As a result, NPT predictions are Pearson-correlated with the duplicate targets at $r = 99.9\%$ (Fig. 3c). This equals an RMSE of only 0.44, about a magnitude lower than the error on the original Protein dataset (Table 8). We conclude that NPTs learn to predict by looking up the target values from matching points. Further discussion and attention maps are in Appendix B.1.1.

Purely parametric models cannot exploit information from other datapoints, limiting their performance. For example, MLPs achieve an RMSE of 3.62 on this task. Non-parametric approaches also cannot solve this task in its original form, because unlike NPTs they must be told which datapoints are the originals (training data) and which the duplicates (test data) as well as which columns contain features and which target values. We demonstrate in Appendix B.1.2 that even when we make these concessions, we can easily adapt the task such that both k-Nearest Neighbors and Deep Kernel Learning fail to solve it. In fact, we are not aware of any other model that can solve the adapted task.

Additionally, we perform an *interventional* experiment to investigate the extent to which NPTs have actually learned the causal mechanism underlying the lookup task. As illustrated in Fig. 3d, we now intervene on individual duplicate datapoints at test time by varying their target value across a wide range. We stress that we perform these experiments without retraining the model, using exactly the same NPT from Figs. 3a-c. The model is now confronted with target values associated with features that are highly unlikely under the training data. This label distribution shift [31] is a challenging setting for neural networks. However, NPT predictions follow the intervened target values with near-perfect correlation, Fig. 3e, continuing to predict by correctly looking up targets.

We now confidently conclude that NPTs robustly learn the causal data-generating mechanism underlying the semi-synthetic dataset. This requires NPTs to *learn* a non-trivial sequence of compuational steps. They must learn to match rows based on similarity of relevant features; to look up the target value of the duplicated datapoint; and, to copy that value into the target of the masked datapoint.

4.3 NPTs Learn to Use Attention Between Datapoints on Real Data

We next consider **(Q3)**: do NPTs actually learn to use attention between datapoints for prediction on real data? We design a test that allows us to quantify the extent to which NPT predictions depend on relationships between datapoints at test time. Concretely, for each target value in the input we randomize the data for all *other* datapoints by independently shuffling each of their attributes across the rows. We then evaluate the loss on the prediction at the target entry and repeat this procedure for all test datapoints. This completely corrupts the information from all datapoints except the one for which we evaluate. Hence, a model that relies meaningfully on attention between datapoints will show deteriorating performance. We give an algorithm for the corruption procedure in Appendix B.2.1.

We report the resulting change in performance after corruption in Table 2 for all datasets from §4.1. We find that for most datasets, the corruption of other rows at test time significantly decreases the performance of the trained NPT models. This indicates that the NPTs have successfully learned to make predictions supported by attention between datapoints. For some datasets, the corruption experiment deteriorates performance completely. For example, for the Protein regression dataset NPT achieves state-of-the-art performance, but corrupting the input leads to NPT performing worse than all of the baselines considered in §4.1. We note that minor differences in performance are often still significant, as differences between competing models in §4.1 are often likewise small.

Interestingly, on certain datasets such as Forest Cover, Kick, and Breast Cancer, corrupted inputs do not significantly affect performance. It appears that when NPTs do not find it advantageous to rely on attention between datapoints during training, they can learn to completely ignore other inputs, essentially collapsing into a standard parametric model. This supports our earlier claims that NPTs can learn end-to-end from data the extent to which they rely on other datapoints for prediction. We

think this is extremely interesting behavior and are unaware of prior work reporting similar results. However, we stress that these results reflect inductive biases of the NPT architecture and do not lend themselves to general statements about the performance of parametric versus non-parametric models.

4.4 NPTs Rely on Similar Datapoints for Predictions on Real Data

So far, we have presented convincing evidence that NPTs (sometimes strongly) depend on attention between datapoints. However, we do not know what kind of interactions are learned in practice on real data (**Q4**). As an initial step towards understanding this, we now present two experiments investigating *to which* other datapoints NPT attends.

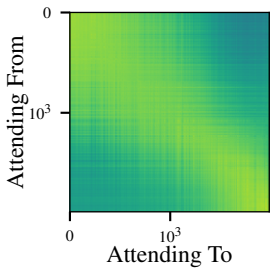


Fig. 4: Attention weights.

Qualitative Evidence. Figure 4 shows an attention map for attention between datapoints (ABD) of NPT on a batch of the Protein regression dataset. We sort the input data with respect to their feature space distance such that similar datapoints are now close to each other. The diagonal pattern in Fig. 4 indicates that NPT attends more strongly to datapoints that are similar in feature space. Appendix B.3.1 discusses this further and gives additional attention maps.

Quantitative Evidence. Seeking a quantitative measure for this hypothesis, the *data deletion* experiment repeats the following procedure for all test set points: iteratively delete other datapoints from the input if they do not significantly affect the prediction. We stop if less than 2% of the original datapoints remain, or if the total change in prediction for the target (relative to the original prediction with all data) exceeds 10%. We investigate the average feature space distances between the test point and the *kept* datapoints, as well as the distances between the test point and the *deleted* datapoints. We find that kept datapoints have a significantly lower average feature space distance to the test point than those deleted. This indicates that two datapoints i, i' that are similar in feature space, such that $\sum_{j<d}(X_{i,j} - X_{i',j})^2$ is low, have a larger effect on the predictions of one another. A Wilcoxon signed-rank test is significant at $p \approx 8.77 \cdot 10^{-130}$. We give full details on this in Appendix B.3.2.

Both experiments support the hypothesis that NPTs rely on similar datapoints for prediction in real data settings. One possible explanation is that similar datapoints might have different realizations of observation noise which NPTs could learn to average out. Altogether, we conclude that NPTs can and do learn representations which rely on interactions between datapoints for prediction.

5 Limitations, Future Work, and Conclusions

Limitations. NPTs share scaling limitations with all naïvely non-parametric approaches [64] and Graph Convolutional Networks [44]. While we have seen success with minibatching (§2.6), NPT justifies future work in principled attention approximations, such as learning representative input points [51], kernelization [18, 40], or other sparsity-inducing methods [5, 17, 72].

Future Work. We believe that the unique predictive mechanism of NPTs makes them an interesting object of study for other tasks including continual learning, multi-task learning, few-shot generalization, and domain adaptation. For example, when predicting under distribution shift, general relations between datapoints and attributes may remain valid and allow NPTs to accommodate such scenarios better. Additionally, future work could explore the connections to stochastic processes, e.g., extending NPTs to be approximately consistent, similar to Neural Processes [32, 33, 42].

Conclusions. We have introduced Non-Parametric Transformers (NPTs), a novel deep learning architecture that takes the entire dataset as input and uses self-attention to model complex relationships *between* datapoints. NPTs challenge and naturally extend parametric modeling as the dominant paradigm of deep learning. They have the additional flexibility to learn to predict by directly attending to other datapoints. Notably, NPTs learn this end-to-end from the data at hand. Empirically, NPTs achieve highly competitive performance on a variety of benchmarks, and additional experiments demonstrate their ability to solve complex reasoning tasks over datapoints. Further, we show that on real data, NPTs learn to rely on attention between datapoints for prediction. We believe that the characteristics of NPTs will make them an exciting object of further study.

References

[1] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46, 1992.

[2] Sercan O Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *arXiv:1908.07442*, 2019.

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

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[5] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

[6] Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. In *International Joint Conference on Neural Networks*, volume 2, 1991.

[7] J. L. Bentley. Multidimensional binary search trees used for associative searching. In *Communications of the ACM*, volume 18, 1975.

[8] John B Biggs. The role of metalearning in study processes. *British journal of educational psychology*, 55, 1985.

[9] Leo Breiman. Bagging predictors. *Machine learning*, 24, 1996.

[10] Leo Breiman. Random forests. *Machine learning*, 45, 2001.

[11] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[12] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv:2005.14165*, 2020.

[13] Thang Bui, Daniel Hernández-Lobato, Jose Hernandez-Lobato, Yingzhen Li, and Richard Turner. Deep gaussian processes for regression using approximate expectation propagation. In *International Conference on Machine Learning*, 2016.

[14] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. *arXiv:2012.07805*, 2020.

[15] Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. The best of both worlds: Combining recent advances in neural machine translation. In *Annual Meeting of the Association for Computational Linguistics*, volume 56, 2018.

[16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Knowledge Discovery and Data Mining*, volume 22, 2016.

[17] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv:1904.10509*, 2019.

[18] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

[19] Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra. Notes from the AI frontier: Insights from hundreds of use cases, 2018.

[20] Zhenwen Dai, Andreas Damianou, Javier González, and Neil Lawrence. Variational auto-encoded deep gaussian processes. In *International Conference on Learning Representations*, 2016.

[21] Andreas Damianou and Neil D Lawrence. Deep gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, volume 16, 2013.

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

[24] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[25] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[26] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, volume 34, 2017.

[27] Evelyn Fix. *Discriminatory analysis: nonparametric discrimination, consistency properties*, volume 1. USAF school of Aviation Medicine, 1985.

[28] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55, 1997.

[29] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001.

[30] Fabian Fuchs, Daniel Worrall, Volker Fischer, and Max Welling. Se(3)-transformers: 3d roto-translation equivariant attention networks. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[31] Saurabh Garg, Yifan Wu, Sivaraman Balakrishnan, and Zachary Lipton. A unified view of label shift estimation. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[32] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, volume 35, 2018.

[33] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv:1807.01622*, 2018.

[34] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585, 2020.

[35] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv:1912.12180*, 2019.

[36] James Honaker and Gary King. What to do about missing values in time series cross-section data. *American Journal of Political Science*, 2010.

[37] Michael Hutchinson, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, and Hyunjik Kim. Lietransformer: Equivariant self-attention for lie groups. *arXiv:2012.10885*, 2020.

[38] Google Inc. Kaggle. *https://www.kaggle.com/*, 2021.

[39] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. *arXiv:2103.03206*, 2021.

[40] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, volume 37, 2020.

[41] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*, volume 30, 2017.

[42] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

[43] Gary King, James Honaker, Anne Joseph, and Kenneth Scheve. Analyzing incomplete political science data: An alternative algorithm for multiple imputation. *American Political Science Review*, 2001.

[44] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[45] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, volume 35, 2018.

[46] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European Conference on Computer Vision*, 2020.

[47] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[48] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350, 2015.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86, 1998.

[50] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]*, 2, 2010.

[51] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, volume 36, 2019.

[52] T. Liu, A. Moore, and A. Gray. New algorithms for efficient high-dimensional nonparametric classification. In *Journal of Machine Learning Research*, volume 7, 2006.

[53] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[54] Wei-Yin Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82, 2014.

[55] James N Morgan and John A Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American statistical association*, 58, 1963.

[56] Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Févry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, Yanqi Zhou, Wei Li, Nan Ding, Jake Marcus, Adam Roberts, and Colin Raffel. Do transformer modifications transfer across implementations and applications? *arXiv:2102.11972*, 2021.

[57] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, volume 35, 2018.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, 2019.

[59] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

[60] Google Cloud AI Platform. Getting started with the built-in tabnet algorithm, 2021. URL cloud.google.com/ai-platform/training/docs/algorithms/tab-net-start.

[61] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018.

[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[63] Roshan Rao, Jason Liu, Robert Verkuil, Joshua Meier, John F Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives. Msa transformer. *bioRxiv*, 2021.

[64] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, 2003.

[65] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. *arXiv:2104.10972*, 2021.

[66] Hugh Salimbeni and Marc Deisenroth. Doubly stochastic variational inference for deep gaussian processes. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, 2017.

[67] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5, 1990.

[68] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[69] D.J. Stekhoven and P. Buehlmann. Missforest - nonparametric missing value imputation for mixed-type data. *Bioinformatics*, 2012.

[70] Yu-Sung Su, Andrew E. Gelman, Jennifer Hill, and Masanao Yajima. Multiple imputation with diagnostics (mi) in R: Opening windows into the black box. *Journal of Statistical Software*, 2012.

[71] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[72] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv:2009.06732*, 2020.

[73] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Herve Jegou. Fixing the train-test resolution discrepancy. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[74] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv:2012.12877*, 2020.

[75] Stef van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 2011.

[76] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[77] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[78] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In *International Conference on Artificial Intelligence and Statistics*, volume 19, 2016.

[79] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[80] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018.

[81] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020.

[82] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See §5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In Appendix F, we give a lengthy discussion of potential negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Appendix A gives proof that NPTs are equivariant with respect to a permutation of the datapoints.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Footnote in §4 gives link to anonymized code that should allow for result replication.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Appendix E discusses data details and baseline hyperparameters, and Appendix C.1 gives NPT hyperparameters.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We perform 10-fold cross-validation whenever computationally feasible.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] This is addressed in Appendix G.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite the creators of the UCI Repository, MNIST, and CIFAR. We also cite some of the most important libraries used in this work in Appendix G.
   (b) Did you mention the license of the assets? [Yes] Also in Appendix G.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include a link to code for NPT in a footnote in §4.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We are using standard machine learning datasets and cite licenses in Appendix G.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We are using standard machine learning datasets with a long history of academic usage.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] Does not apply – no crowdsourcing or research including human subjects.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] Does not apply – no crowdsourcing or research including human subjects.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] Does not apply – no crowdsourcing or research including human subjects.

A Proof – NPT Is Equivariant over Datapoints

We here provide proof that NPT is equivariant to a permutation of the datapoints. This requires, among other things, showing that multi-head self-attention is equivariant. We were unable to find this proof in the existing literature, e.g., Set Transformer [51] relies heavily on equivariance of self-attention but does not provide proof. In the following, we will refer to datapoints as the *rows* of our input, see e.g., Fig. 1.

Definition 1. *A function $f : \mathcal{X}^n \to \mathcal{X}^n$ is row-equivariant if for any permutation $\sigma : [1, \dots, n] \to [1, \dots, n]$ applied to the dimensions of $\mathcal{X}^n$, we have for all i, $f(X_1, \dots, X_n)[i] = f(X_{\sigma^{-1}(1)}, \dots, X_{\sigma^{-1}(n)})[\sigma(i)]$.*

Lemma 1. *Any function of the form $f(X_1, \dots, X_n) = (g(X_1), \dots, g(X_n))$ for some g is row-equivariant. These functions are denoted as 'row-wise operations', as they consist of the same function applied to each of the rows of the input.*

Proof. Follows immediately from the structure of f. □

Lemma 2. *The composition of row-equivariant functions is row-equivariant.*

Proof. This result is widely known, but a proof here is included for completeness. Let f and g be row-equivariant.

$$f \circ g(\sigma X) = f(g(\sigma X)) = f(\sigma g(X)) = \sigma f(g(X)). \tag{7}$$

□

Lemma 3. *Let $W \in \mathbb{R}^{n \times m_1}$ and $X \in \mathbb{R}^{m_2 \times n}$. The function $X \mapsto XW$ is row-equivariant.*

Proof. Let σX be a permutation of the rows of X. Then we have

$$(\sigma X)W[i, j] = \sum \sigma X[i, k] W[k, j] \tag{8}$$

$$= \sum X[\sigma^{-1}(i), k] W[k, j] = XW[\sigma^{-1}(i), j] = \sigma(XW)[i, j]. \tag{9}$$

□

Lemma 4. *The function $X \mapsto Att(XW^Q, XW^K, XW^V)$ is row-equivariant.*

Proof. Let the row-wise softmax function be denoted $\omega(\cdot)$. Then we have

$$Att(XW^Q, XW^K, XW^V) = \omega(XW^Q(XW^K)^\top/\sqrt{h})XW^V, \tag{10}$$

where

$$\sigma XW^Q(\sigma XW^K)^\top[i, j] = \sigma(XW^Q)\sigma(XW^K)^\top[i, j] \tag{11}$$

$$= \sum \sigma(XW^Q)[i, k]\sigma(XW^K)[j, k] \tag{12}$$

$$= \sum XW^Q[\sigma^{-1}(i), k] XW^K[\sigma^{-1}(j), k] \tag{13}$$

$$= XW^Q(XW^K)^\top[\sigma^{-1}(i), \sigma^{-1}(j)] \tag{14}$$

$$=: A. \tag{15}$$

Note that the above result states that the function $XW^Q(XW^K)^\top$ is *not* row-equivariant because of the additional permutation of the columns. Let σ denote a permutation operator on matrices. Then straightforwardly we have the following:

$$\omega(\sigma A/\sqrt{h})) = \sigma\omega(A/\sqrt{h})\,. \tag{16}$$

Finally, it remains to show that the final matrix multiplication step restores the row-equivariance property we seek.

$$\sigma \underbrace{\omega(XW^Q(XW^K)^\top/\sqrt{h})}_{=:M}(\sigma XW^V)[i,j] = \sigma(M)(\sigma XW^V)[i,j] \tag{17}$$

$$= \sigma(M)\sigma(XW^V)[i,j] \tag{18}$$

$$= \sum M[\sigma^{-1}(i), \sigma^{-1}(k)](XW^V)[\sigma^{-1}(k), j] \tag{19}$$

$$= M(XW^V)[\sigma^{-1}(i), j]. \tag{20}$$

Which shows that self-attention is row-equivariant. □

Lemma 5. *The following hold:*

1. *Multihead self-attention is equivariant.*
2. *If f and g are row-equivariant, then the function $x \mapsto g(x) + f(x)$ is also row-equivariant.*
3. *Res(H) is row-equivariant.*
4. *MHSA(H) is row-equivariant.*
5. *ABD is row-equivariant.*
6. *ABA is row-equivariant.*

Proof. We show each item.

1. We know that $X \mapsto O_i$ is equivariant from the previous lemma, and this trivially implies that $X \mapsto \text{concat}(O_1, \ldots, O_k)$ will also be row-equivariant. Finally, because $\sigma AB = \sigma(AB)$, get that MHSelfAtt(H) is row-equivariant.
2. Straightforward.
3. Because LayerNorm is row-equivariant (being a function applied row-wise to the matrix), Res(H) is a sum of two row-equivariant functions and so by a previous result will also be row-equivariant.
4. Because rFF is again a row-wise operation and so trivially row-equivariant, the previous results on sums and compositions of row-equivariant functions directly yield row-equivariance of MHSA.
5. ABD is by definition an application of MHSA(H), and therefore is row-equivariant by the above result.
6. ABA is a row-wise operation and is therefore trivially row-equivariant.

□

Property A.0.1. *NPT is row-equivariant.*

Proof. Each layer of NPT has been shown to be row-equivariant. Because NPT is a composition of such row-equivariant functions, it is therefore row-equivariant. □

B Additional Results

B.1 Semi-Synthetic Experiments

B.1.1 Attention Maps for the Semi-Synthetic Experiments

We here display additional results for the semi-synthetic experiments of Section 4.2. In Fig. B.1, we display attention weights for Attention Between Datapoints (ABD) for all depths and a subset of heads of the architecture. We see that some, but not all, attention heads display the desired diagonal lookup pattern. Note that, in this case, one head would suffice to implement lookup and perfectly solve the task.

A brief comment on the attention maps with the "double diagonal" structure (e.g., depth 4, head 0): we see that (a) original datapoints attend to the duplicate points and (b) duplicates also attend to duplicate datapoints. Behavior (a) makes sense: NPT needs to attend to the duplicates from the originals to look up the target values. This behavior in turn minimizes loss. Behavior (b) is irrelevant to loss, because NPT does not need to predict anything for the duplicates, and no loss is computed. However, (b) suggests that the query embeddings learned by the self-attention *ignore* the masked out label column in the input. Hence, the resulting queries for the originals and the duplicates would

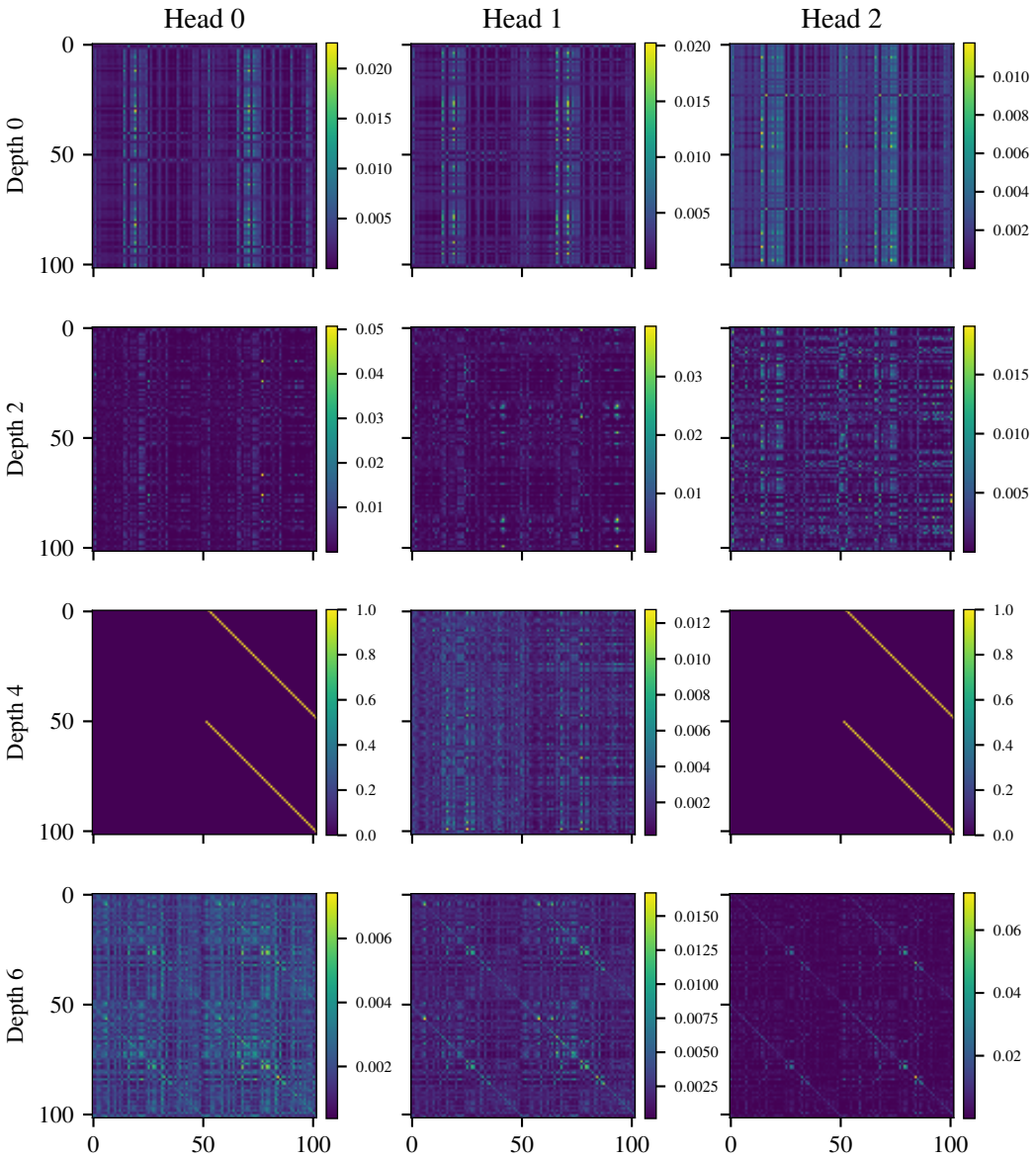


Figure B.1: Visualizations of NPT attention maps for Attention Between Datapoints (ABD) for the semi-synthetic experiment at all model depths, a selection of heads, and a single batch of input data. Evidently, not all attention maps need to perform a "lookup" for the model to solve the task. In fact, some heads appear to learn almost query-independent behavior (e.g., heads 0, 1, and 2 at depth 0).

be identical – both leading to high attention values for the keys of the duplicates – and ultimately resulting in the double diagonals in Fig. B.1.

B.1.2 Modified Semi-Synthetic Experiments

Setup. In Section 4.2, we mention that with some concessions the original lookup task can also be solved by standard non-parametric models. However, we also mention that simple modifications to the task make it, again, unsolvable for any model of which we are aware other than NPT. We here demonstrate these hypotheses for two non-parametric models: k-Nearest Neigbors (k-NN) and Deep Kernel Learning (DKL).

First, we apply k-NN and DKL to the original duplication tasks. As mentioned in the main text, this already requires us to make some concessions: we now need to explicitly split the input data into a global training set (all duplicated datapoints) as well as a test set (all original datapoints). That is,

Table 3: Variations of the semi-synthetic dataset that require learning of between-datapoint interactions more complex than simple lookups. While NPTs can learn complex interactions between datapoints, conventional non-parametric approaches lack flexibility and fail.

Test RMSE ↓	Original Synthetic	Random Feats.	Add One	Random Feats. + Add One
1-NN	**0.00**	7.19	6.11	7.80
k-NN	**0.00**	5.42	5.18	5.64
DKL	**0.00**	5.94	6.31	6.36
NPT	0.34	**0.24**	**0.46**	**0.75**

if all duplicate datapoints make up the training set, then non-parametric models are able to predict perfectly on the original datapoints, because most non-parametric models rely on distances in some manner, and here, distances in input feature space are sufficient to successfully match entries. This is trivially true for k-NN but also for DKL, where the RBF kernel of the GP will lead to the desired "matching behavior" as long as the learned neural network embedding does not collapse distances.

In other words, NPTs would ideally learn a k-NN-style prediction for the semi-synthetic dataset. Crucially, while non-parametric models predict based on distances because of fixed design choices, NPTs *learn* this behavior and can just as well learn other more complicated relations between datapoints.

We now present two modifications to the semi-synthetic dataset; NPT can accommodate them because the model learns the nature of interactions, but they significantly affect the performance of the fixed kernel methods.

- **Random Features**: A subset of the features are randomized across both original and duplicate datapoints independently. Specifically, we overwrite the entries of the last three features with noise drawn independently from a Gaussian distribution $\mathcal{N}(1, 1)$. To solve the task, matches between datapoints must now be computed using the subset of non-randomized features only.
- **Add One**: We add 1 to all target regression values *only* for the duplicate datapoints. Matches can still be made based on all features, but now a 1 must be subtracted from the lookup value to solve the task.

As in the original setting, we train the models on the modified semi-synthetic datasets and check with novel test data whether they have learnt the correct relational mechanism underlying the experiment.

Note that the Random Features and Add One settings also distinguish our setup from prompting in natural language processing literature [12, 62] because the original datapoints are no longer "correct" input-output pairs; the model must use an underlying relational structure instead of memorization to solve the task.

Results. Table 3 presents RMSE values obtained by the models when trained on the original duplication task, the two modifications separately, as well as both modifications applied.

Evidently, for NPTs, the different scenarios do not lead to a large difference in performance; in all instances, they achieve near-perfect loss because their predictions leverage attention between datapoints. Careful optimization of NPT training convergence would likely lead to a further reduction in loss. Nevertheless, the achieved losses by NPT are more than a magnitude lower than those on the original data and correspond to a near-perfect Pearson-correlation with the target values of $r > 99.9\%$. We conclude that NPTs successfully learn to attend to the correct subset of features, to subtract 1 from the lookup target values, or to do both at the same time.

Next, we consider the non-parametric models. First, we confirm in *Original Synthetic* that the non-parametric models can indeed solve the original lookup task. However, we find that neither DKL nor k-NN can accommodate any of the modifications, reverting to an RMSE that is worse than the performance of all baselines on the original Protein dataset, see Table 8.[5]

[5]In fact, the RMSEs are about equal to the standard deviations of the target values in the Protein dataset, 6.11, such that the values obtained by the models on the modified setups amount to random guessing. We further

For k-Nearest Neighbor, $k = 1$ is clearly optimal in the original semi-synthetic setup. However, k-NN cannot learn to ignore certain attributes (Random Features) and or to modify looked-up values. Setting $k > 1$ actually improves prediction because it considers other matching points in addition to the (now misleading) duplicates for prediction. However, even with $k > 1$, k-NN does not achieve much better than guessing performance on the modified tasks.

DKL also fails to accommodate any of the presented task modifications. We suspect that DKL, in theory, should be able to solve the Random Features task. That is, DKL should be able to use the neural network to learn a representation that discards any information from the randomized columns. We were unable to achieve this, but it may be possible with additional adaptations to the model. Ideally, we would condition the GP on new "test data" (the duplicates) in each minibatch during training. This was not easily possible with the GPyTorch codebase.[6] At test time however, we did directly reconstruct an exact GP using embedded inputs and RBF scale parameters learned during training.

In any case, DKL can never solve the Add One scenario because, after independently transforming features with a neural network, DKL simply applies a GP in embedding space. This means that it will always naively interpolate target values between training data (duplicates) and test data (features) in embedding space, and cannot *learn* interactions between points, such as subtracting 1 from all duplicate targets.

Even further, there is another easy option of how to construct this experiment such that only NPT will be able to solve it: we could *randomly sample the attribute* for which we mask out the entry, i.e., all columns can now be target columns. All non-parametric models presented here rely on a fixed set of features as input to predict for a fixed target column. They are not compatible with this style of "imputation" problem, i.e., there is no way to even take as input data like this in such models. NPTs, however, take both features and targets as input, only using the masking mechanism to distinguish between features and targets as well as train and test data. Hence, they can easily adapt to this scenario.

The bad results for the non-parametric models also highlight that these models must predict non-parametrically, unlike NPT, which could always fall back to parametric prediction if it cannot learn the interactions required for a task.

(k)-NN Hyperparameter details. We use the scikit-learn [59] implementation of (k)-Nearest Neighbors, where we exhaustively search for neighbors by setting `algorithm=brute` and otherwise use default parameters. For 1-NN, we set $k = 1$, for k-NN we sweep over $k \in [1, \dots, 10]$ and report results for the k that achieved the best performance.

DKL Hyperparameter details. We use the GPyTorch implementation of Deep Kernel Learning. We perform a non-exhaustive random sweep over a selection of hyperparameters and select those with best validation performance. This results in the following changes from the default hyperparameter values: for the Original Synthetic and Add One scenario we disable dropout, use hidden layers $[100, 100]$, a learning rate of 0.0001, train for a maximum of 30000 epochs, with 256 inducing points, 8 features, batch size of 128, and early stopping patience on the validation loss of 20 epochs. For the Random Features and the Random Features + Add One scenarios, we arrive at the same configuration, except that we train with 64 inducing points.

B.2 Attention Between Datapoints on Real Data

B.2.1 Corruption Experiments

In our Data Corruption experiments in Section 4.3, we make use of Algorithm 1 below. When predicting for a datapoint k, this algorithm completely destroys information from all other datapoints $i \neq k$ in the batch b by randomly permuting attribute values across all other datapoints. Therefore, if

note that we apply all modifications to the standardized input data, such that the Add One setting adds a full standard deviation for the final evaluation in Table 3.

[6] Gardner, Jacob R., et al. "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration." NeurIPS 2018.

NPT's loss increases after corruption, it must meaningfully rely on attention between datapoints for prediction.

Algorithm 1: Data Corruption

Input: list of masked minibatches $\mathcal{B} = [\boldsymbol{X}^{(b)} \in \mathbb{R}^{K \times d} \mid b \in 1 \dots B]$, unmasked label column $\boldsymbol{X}_{:,d}$, trained model $f : \boldsymbol{X}^{(b)} \rightarrow \boldsymbol{X}^{(b)}$, batch size K, loss function $\mathcal{L}$, number of attributes (including features and target) d

Returns: test loss under data corruption $\mathcal{L}^{\text{corr}}$

$\mathcal{L}^{\text{corr}} \leftarrow 0$
for $\boldsymbol{X}^{(b)}$ *in* $\mathcal{B}$ **do**
  for k *in* $1 \dots K$ **do**
    $\boldsymbol{X}^{(b,k)} \leftarrow \boldsymbol{X}^{(b)}$ `// initialize batch to be corrupted`
    for j *in* $1 \dots d$ **do**
      $\boldsymbol{X}^{(b,k)}_{i \neq k,j} \leftarrow \texttt{permute}_{\text{axis}=i}(\boldsymbol{X}^{(b,k)}_{i \neq k,j})$ `// permute each attr. column indep.`
    end
    $\mathcal{L}^{\text{corr}} \mathrel{+}= \mathcal{L}(f(\boldsymbol{X}^{(b,k)})_{k,d}, \boldsymbol{X}_{k,d})$ `// compute loss w/ unmasked label column`
  end
end
return $\mathcal{L}^{\text{corr}}$

Alternatively, we could also input datapoints *individually*, i.e., decrease the minibatch size to 1, to test if NPT depends on attention between datapoints. Indeed, we find that performance also deteriorates in this scenario. However, we believe that the Data Corruption experiment provides stronger evidence because it preserves batch statistics across attributes. This makes sure that performance deterioration is not caused by spurious factors, such as a decreased batch size that was not encountered in training. While NPT is generally compatible with varying batch sizes, we leave a thorough investigation of this for future work.

B.3 Real Data – *To Which* Other Points Does NPT Attend?

B.3.1 Attention Maps on Real Data

In Fig. B.2, we display ABD attention maps of NPT for the Protein regression dataset in addition to the one shown in Section 4.4. For visualization purposes, we sort the input datapoints with respect to their feature space distance to an arbitrary test datapoint. This is to ensure that the global structure of the attention maps in Fig. B.2 has meaning. Specifically, nearby entries in the attention maps belong to input datapoints that are close in input space. With this transformation, the diagonal patterns appearing in Fig. B.2 clearly suggest that our model is attending more strongly between datapoints that are similar in input space. Similar to the semi-synthetic experiments, some but not all attention heads display this pattern of interest.

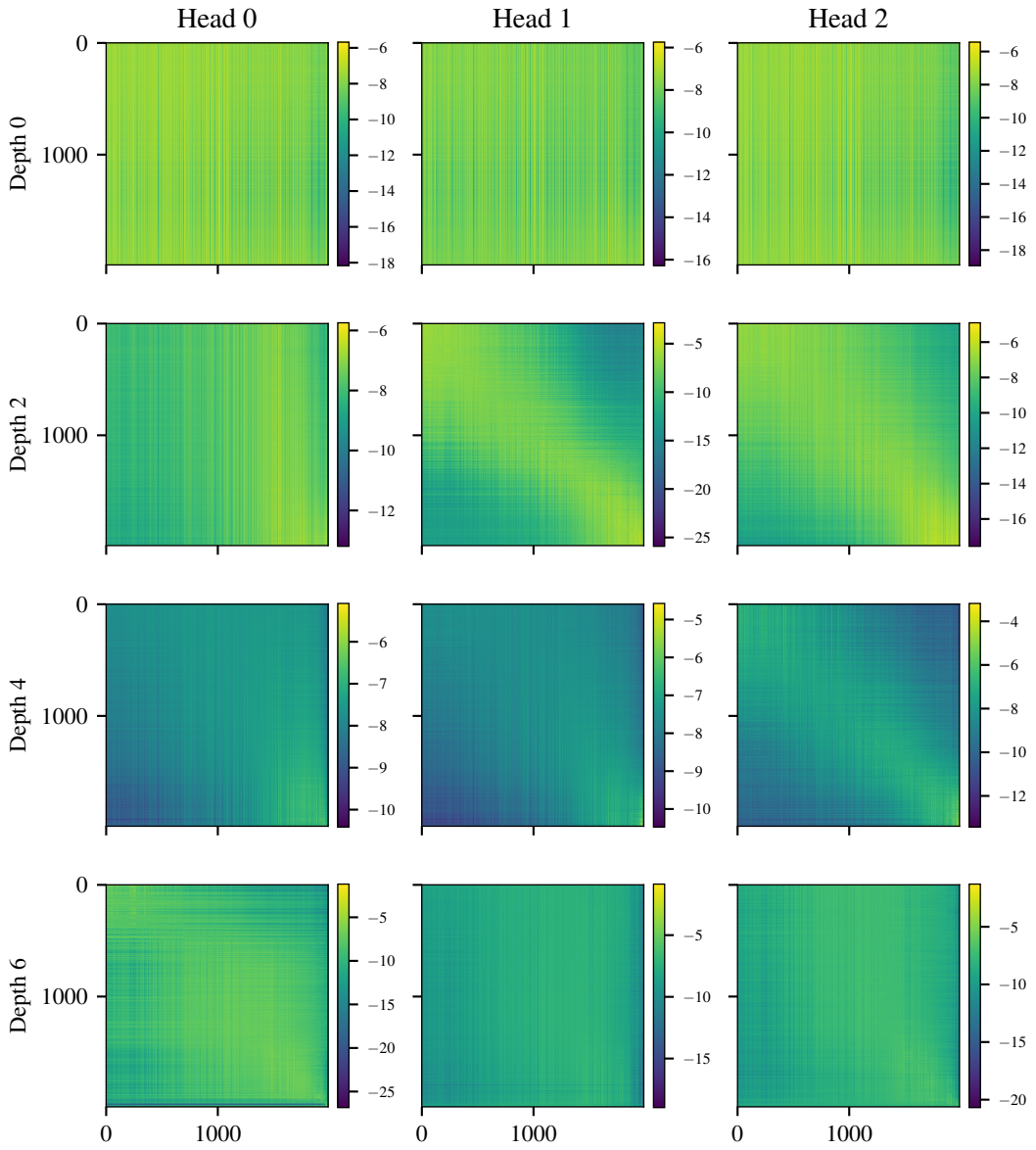


Figure B.2: Visualizations of the Attention Between Datapoints (ABD) attention maps for real data – here, the Protein regression dataset – for all depths and a selection of heads. Input to the model is sorted such that datapoints that are similar in input space have nearby indices. The diagonal pattern (e.g., depth 2 and head 1) indicates that the model attends to similar inputs more strongly. For illustration purposes, we here plot the log of the attention values.

B.3.2 Data Deletion Experiment

We here give full details on the Data Deletion experiment presented in Section 4.4. To recap, we consider the prediction of NPT for a single test sample i^*. We then iteratively delete other datapoints from the input if they do not significantly change the prediction of NPT on i^*. Algorithm 2 describes this in detail. We are then interested in differences between the deleted and the kept datapoints. Specifically, we compare the average feature space distance in input space between the active datapoint i^* and either the kept datapoints $\mathcal{R}$ or deleted datapoints $\{1, \dots, n\} \setminus (\{i^*\} \cup \mathcal{R})$, obtaining average distances $D_{i^*,\text{kept}}$, $D_{i^*,\text{deleted}}$. We break out of the deletion algorithm if less than $\epsilon\%$ of the original points remain, to reduce variance in our estimates of the kept statistic. We repeat Algorithm 2 for all 5567 test points $i^* \in \mathcal{D}_{\text{test}}$ in the Protein regression dataset.

We perform a Wilcoxon signed-rank test on the pairs $\{D_{i^*,\text{kept}}, D_{i^*,\text{deleted}}\}_{i^* \in \mathcal{D}_{\text{test}}}$ to determine if the median of the kept datapoints is less than the median of the deleted ones. The test is highly significant at $p \approx 0$, i.e., smaller than the floating point precision of SciPy Stats allows. The raw Wilcoxon statistic is 3125889.5.

To make sure the difference is not an effect of sample size, we also construct a set of average differences to a set of randomly drawn datapoints.[7] That is, instead of using Algorithm 2 for *targeted* deletion, we *randomly* construct $\mathcal{R}$, essentially only applying lines 10 and 15 of Algorithm 2. For each active test row i^*, we randomly delete as many datapoints as were deleted in targeted fashion. A Wilcoxon signed-rank test between the distances for the random and kept subset is likewise significant at $p \approx 8.77 \cdot 10^{-130}$. This is the value we report in the main body.

We also run a computationally more demanding version of the algorithm with $\Delta_{\text{it}} \leftarrow 0.005$, $\epsilon \leftarrow 0.01$ to see how many points we can successfully delete. This version of the algorithm requires more computation which is why we limit execution to the test datapoints of a single batch. The results are statistically significant at $5.26 \cdot 10^{-49}$ for kept $<$ deleted and $8.38 \cdot 10^{-39}$ for kept $<$ random for a Wilcoxon signed-rank test. We illustrate the differences between the distances in Fig. B.3. We further note that using Algorithm 2, we are able to reduce the set of datapoints present in the input to 1% of the original n for 79.5% of active test datapoints and to 10% in 99.5% of cases. Percentages refer to $n = 2048$ datapoints in total, of which 398 were test datapoints.

All in all, these experiments strongly suggest that NPT relies on interactions between similar datapoints for prediction.

B.4 Ablation Study

We conduct an ablation study on the Protein and Boston Housing datasets (Table 4). For Protein, the same 0.7/0.1/0.2 train/validation/test split is used for all model configurations. Boston Housing uses a 0.7/0.2/0.1 train/validation/test split with 10-fold cross-validation.

Despite the significant difference in dataset sizes between Boston Housing $(n = 506)$ and Protein $(n = 45730)$, and the fact that Boston Housing includes both categorical and continuous variables, the base models used for each dataset are nearly identical.

On both datasets, we use an NPT model with 8 layers, 8 heads, per-attribute hidden dimension $e = 128$, feature and target masking with $p = 0.15$ for each, a cosine annealing schedule for the loss tradeoff λ, the LAMB [81] optimizer with Lookahead [82], a flat-then-anneal learning rate schedule with cosine decay and base learning rate 0.001, dropout with rate 0.1 on the attention weights and after linear layers, and gradient clipping at 1. This configuration is essentially the same as the `NPT-Base` configuration described in Appendix C.1, which we use with minimal per-dataset modifications for all other results in this work.

Different in our base models between the two datasets are the following settings. The Boston Housing model takes as input the full dataset (i.e., batch size $= 507$) and Protein uses minibatching with batch size $= 2048$. Boston Housing trains for $20\,000$ steps, and Protein for $400\,000$. The learning rate is constant for the first $70\,\%$ of steps for Protein, but only for the first $50\,\%$ of steps for Boston, starting

[7] There are many fewer kept than deleted datapoints. Further, there are outliers in the dataset, and these affect the deleted datapoints more often than the kept datapoints. We find that the average distance between a *random* subset and the *deleted* (not the kept!) datapoints also becomes statistically significantly smaller at large sample sizes. Hence, we compare the *deleted* datapoints to a *random* subset to control for size effects.

Algorithm 2: Data Deletion

Input: Masked data $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, active sample index i^*.

$\hat{y} \leftarrow \text{NPT}(\boldsymbol{X})_{i^*,d}$ // original NPT prediction at active datapoint
$\Delta_{\text{max}} \leftarrow 0.1$ // maximum allowed change in prediction
$\Delta_{\text{it}} \leftarrow 0.01$ // initialize maximum change per deleted datapoint
$N_{\text{max-retry}} \leftarrow 50$ // maximum number of retries before increasing Δ_{it}
$\epsilon \leftarrow 0.02$ // fraction of points remaining at which we break
$\mathcal{R} \leftarrow \{1, \dots, n\} \setminus \{i^*\}$ // initialize remaining set
$N_{\text{retry}} \leftarrow 0$ // initialize no. of retries

while *True* **do**
  $c =$`random_choice`(R) // random proposal for data deletion
  $\hat{y}_{\text{proposal}} = \text{NPT}(\boldsymbol{X}_{(\mathcal{R} \setminus \{c\}) \cup \{i^*\}})_{i^*,d}$ // predict without proposed datapoint
  $\Delta_{\text{proposal}} = \frac{|\hat{y}_{\text{proposal}} - \hat{y}|}{\hat{y}}$ // change in pred. when deleting proposal
  if $\Delta_{\text{proposal}} < \Delta_{it}$ **then**
    if $\Delta_{\text{proposal}} < \Delta_{max}$ **then**
      $\mathcal{R} \leftarrow \mathcal{R} \setminus \{c\}$ // delete datapoint from input
      $N_{\text{retry}} \leftarrow 0$
    else
      break // exceeded maximum change
  else
    $N_{\text{retry}} \leftarrow N_{\text{retry}} + 1$ // candidate change was too large, try again
  if $N_{\text{retry}} \geq N_{\text{max-retry}}$ **then**
    $\Delta_{\text{it}} \leftarrow 1.1 \cdot \Delta_{\text{it}}$ // increase allowed change per iteration
    $N_{\text{retry}} \leftarrow 0$
  if $|\mathcal{R}| < \epsilon \cdot n$ **then**
    break // less than $\epsilon\%$ of original datapoints remaining
end
return $\mathcal{R}$

the learning rate annealing earlier to defend against overfitting on the small dataset.These changes directly result from the different dataset sizes.

As Table 4 shows, the performance of NPT is robust to a variety of significant hyperparameter choices. This illustrates that practitioners will likely *not need to spend much time tuning hyperparameters* when applying NPT to novel datasets. We now give results for the ablation study on the Protein and Boston datasets separately.

Protein Dataset. See Table 4 for results and performed ablations. It is computationally too expensive for us to perform full cross-validation over all ablations for the Protein regression dataset. Instead, we report the results of a single 5-fold cross-validation for the Base NPT configuration on Protein (also varying the model random state). This results in an RMSE of 3.40 ± 0.05 (σ). The standard deviation of the 5-fold cross-validation allows us to roughly gauge which ablations have significant effect. Given the results in Table 4, we find that the majority of ablations do not lead to meaningful changes in performance. Only the somewhat dramatic changes to the optimization of NPT result in its performance falling from the top rank on the Protein Dataset (second rank CatBoost has RMSE $= 3.51$): removing stochastic feature masking ($p_{\text{feature}} = 0$), removing both stochastic feature masking ($p_{\text{feature}} = 0$) and stochastic target masking ($p_{\text{target}} = 1$, training targets are always masked out at training time and NPT therefore cannot learn to attend to training targets at test time), or changing p_{feature} to 0.5 (meaning that 50% of all input features are masked out). NPT appears to be particularly robust to changes in model complexity, e.g., depth and number of heads, although the results suggest that we could have further increased the size of Base NPT to achieve slightly higher performance.

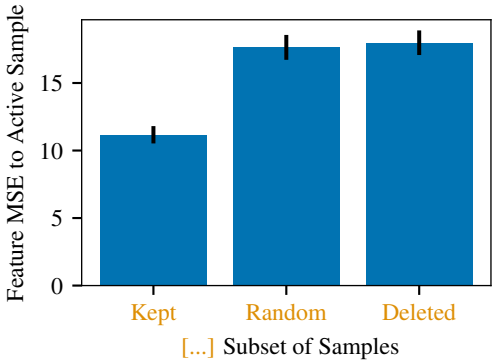


Figure B.3: When predicting for any given datapoint, NPT prefers to keep similar datapoints around. Displayed are average feature space differences and their standard errors between the active datapoint and the sets of kept, random, and deleted datapoints for a single batch.

Boston Dataset. See Table 4 for results and performed ablations. For the Boston dataset, we repeat ablations over all 10 CV splits. Similarly, ablations on the Boston dataset are largely inconsequential; none of them result in a statistically significant change in performance from the base model. The second rank performer on Boston is MLP, at RMSE $= 3.32$. Only ablation of semi-supervision or changing p_{feature} to 0.5 result in a change in the top ranking of NPT among the baselines.

Altogether, the ablation study supports the claim that NPT can be applied successfully with very little tuning to datasets of vastly different sizes and feature types. Changes in model depth and number of heads do not appear significant, but using a reasonably low feature masking probability (e.g., 15%, as has been commonly used in the literature [23]) may be important to stable training.

Supported by these ablations, we sweep over only a small selection of configurations for our main benchmark comparison in Section 4.1. And indeed, it seems that NPT is robust to hyperparameter changes, given that these configurations perform well across vastly different settings (binary and multi-class classification, datasets with millions of datapoints, etc.) than those explored in the ablations. See Appendix E for details.

We speculate that NPT's robustness stems from (a) being a relatively overparametrized architecture that is powerful enough to model a wide variety of datasets and (b) from the effective regularization introduced by the feature masking mechanism. Finally, we emphasize that the aim of this work is to introduce the NPT architecture and examine its properties, not to spend significant effort and compute resources on achieving top performance across all benchmarks.

B.5 Extended Results for Tabular Data Benchmarks

See Table 5 (Table 6) for test accuracies (negative log-likelihood scores) on the UCI classification datasets and additionally Table 7 for AUROC results on the binary classification datasets. For the regression datasets, see Table 8 for RMSE scores and Table 9 for MSE scores.

[8] Out-of-memory on the Higgs Boson dataset when attempting approximate 3-NN on an Azure D64 v3 instance with 256 GB RAM.

[9] TabNet had notably lower accuracy in our setup on the Poker Hand dataset (which has a fixed test set) than that the 99.2% reported in the original work [2]. We are in communication with the authors, attempting to improve these results. However, our results on Higgs Boson match the reported performance more closely (78.44% (theirs) vs 77.1% (ours)). Further, we note that our other baselines achieve significantly better performance on the same datasets than those reported in [2]; e.g., our MLP achieves 99.5% accuracy on Poker Hand dataset while they report 50.0%; our XGBoost achieves 97.1% on Forest Cover while they report 89.34%. However, we note that some of the datasets – such as Forest Cover – do not have fixed test sets. Therefore, we cannot exclude the possibility that the performance differences are due to differently chosen train-test splits.

[10] See above note on out-of-memory.

[11] See above note on out-of-memory.

Table 4: NPT ablation study: test root mean-squared error (RMSE) on the Protein and Boston Housing regression datasets.

Test RMSE ($\pm$ Std Err) $\downarrow$	Protein	Boston
Base NPT	3.41	3.00 ± 0.23
No Semi-Supervision	3.38	3.38 ± 0.46
No Target Masking	3.32	2.93 ± 0.18
No Feature Masking	3.56	2.95 ± 0.21
No Feature Masking, No Target Masking	3.58	3.20 ± 0.26
Feature Mask $p = 0.15 \rightarrow p = 0.5$	3.87	3.39 ± 0.23
Target Mask $p = 0.15 \rightarrow p = 0.5$	3.37	3.11 ± 0.28
$8 \rightarrow 4$ Layers	3.43	3.30 ± 0.41
$8 \rightarrow 16$ Layers	3.36	3.05 ± 0.24
$8 \rightarrow 4$ Heads	3.42	3.25 ± 0.30
$8 \rightarrow 16$ Heads	3.37	3.20 ± 0.39
Tradeoff $\lambda = 0.5$	3.50	2.96 ± 0.25

Table 5: UCI classification datasets: test accuracy. Standard error reported for datasets with multiple cross-validation splits.

Test Accuracy $\uparrow$	Higgs Boson	Poker Hand	Forest Cover	Income	Kick	Breast Cancer
Random Forest	76.2	71.5	94.8	95.4	90.1	94.20 ± 0.70
Gradient Boosting	76.5	94.1	96.7	95.8	90.2	94.03 ± 0.90
XGBoost	77.0	95.9	**97.1**	95.6	**90.3**	94.91 ± 0.68
CatBoost	76.6	99.2	95.7	**95.8**	90.1	$\mathbf{95.61 \pm 0.75}$
LightGBM	75.9	92.8	85.0	**95.8**	**90.3**	95.26 ± 0.82
MLP	78.3	**99.5**	95.2	95.4	90.0	94.73 ± 0.89
k-NN[8]	—	50.4	90.7	94.8	87.7	95.26 ± 0.79
TabNet[9]	77.1	53.3	94.2	95.5	89.5	94.91 ± 0.76
NPT	**80.7**	99.3	96.7	95.6	90.0	94.73 ± 0.69

Table 6: UCI classification datasets: negative log-likelihood (NLL). Standard error reported for datasets with multiple cross-validation splits.

Test NLL $\downarrow$	Higgs Boson	Poker Hand	Forest Cover	Income	Kick	Breast Cancer
Random Forest	0.489	0.843	0.191	0.126	0.305	0.142 ± 0.012
Gradient Boosting	0.477	0.379	0.109	0.111	0.296	0.185 ± 0.024
XGBoost	0.471	0.178	**0.080**	0.147	**0.293**	0.143 ± 0.025
CatBoost	0.476	0.065	0.120	**0.109**	0.296	$\mathbf{0.124 \pm 0.024}$
LightGBM	0.486	0.420	0.361	**0.109**	0.294	0.163 ± 0.034
MLP	0.452	**0.028**	0.131	0.118	0.333	0.545 ± 0.254
k-NN[10]	—	0.975	0.274	0.139	0.333	0.466 ± 0.167
TabNet	0.469	0.973	0.151	0.119	0.314	0.233 ± 0.036
NPT	**0.412**	0.119	0.087	0.115	0.299	0.137 ± 0.026

Table 7: UCI classification datasets: test area under the receiver operating characteristic curve (AUROC) on binary classification tasks. Standard error reported for datasets with multiple cross-validation splits.

Test AUROC ↑	Higgs Boson	Income	Kick	Breast Cancer
Random Forest	0.847	0.947	0.759	0.989 ± 0.003
Gradient Boosting	0.850	0.955	0.769	0.987 ± 0.004
XGBoost	0.854	0.946	0.775	0.989 ± 0.003
CatBoost	0.851	**0.956**	0.773	0.992 ± 0.003
LightGBM	0.843	**0.956**	**0.776**	0.992 ± 0.003
MLP	0.867	0.949	0.739	0.982 ± 0.007
k-NN[11]	—	0.932	0.747	0.980 ± 0.005
TabNet	0.857	0.948	0.745	0.978 ± 0.005
NPT	**0.892**	0.952	0.770	$\mathbf{0.997 \pm 0.001}$

Table 8: UCI regression datasets: test root mean-squared error (RMSE). Standard error reported for datasets with multiple cross-validation splits.

Test RMSE ↓	Protein	Concrete	Boston Housing	Yacht
Random Forest	3.57	5.48 ± 0.18	3.78 ± 0.33	0.91 ± 0.13
Gradient Boosting	3.61	4.70 ± 0.18	3.44 ± 0.22	$\mathbf{0.85 \pm 0.12}$
XGBoost	3.60	4.68 ± 0.15	3.39 ± 0.29	0.88 ± 0.13
CatBoost	3.51	$\mathbf{4.28 \pm 0.16}$	3.44 ± 0.34	1.05 ± 0.16
LightGBM	3.65	4.64 ± 0.18	3.86 ± 0.27	13.60 ± 0.73
MLP	3.62	5.53 ± 0.20	3.32 ± 0.39	0.91 ± 0.13
k-NN	3.77	8.51 ± 0.30	4.27 ± 0.37	12.02 ± 0.65
TabNet	3.59	5.85 ± 0.15	3.88 ± 0.34	3.41 ± 1.12
NPT	**3.41**	5.21 ± 0.20	$\mathbf{2.92 \pm 0.15}$	1.27 ± 0.15

Table 9: UCI regression datasets: test mean-squared error (MSE). Standard deviation reported for datasets with multiple cross-validation splits.

Test MSE ($\pm$ Std Dev) ↓	Protein	Concrete	Boston	Yacht
Random Forest	12.8	30.4 ± 6.4	15.4 ± 9.5	0.986 ± 0.818
Gradient Boosting	13.0	22.4 ± 5.2	12.3 ± 4.9	$\mathbf{0.867 \pm 0.779}$
XGBoost	13.0	22.1 ± 4.2	12.3 ± 7.6	0.939 ± 0.881
CatBoost	12.3	$\mathbf{18.6 \pm 4.3}$	13.0 ± 9.8	1.36 ± 1.12
LightGBM	13.3	21.9 ± 5.3	15.6 ± 7.6	190.0 ± 65.1
MLP	13.1	31.0 ± 6.9	12.6 ± 11.0	0.994 ± 0.937
k-NN	14.2	73.3 ± 16.0	19.6 ± 11.0	149.0 ± 52.6
TabNet	12.9	34.4 ± 5.8	16.2 ± 11.0	24.1 ± 54.3
NPT	**11.6**	27.6 ± 7.6	$\mathbf{8.77 \pm 2.60}$	1.80 ± 1.49

B.6 Image Classification Results

To apply NPT to high-dimensional image data, we append the mask dimension as an extra channel and apply image patching with linear embeddings as in [24]. Further following [24], we use a learned position embedding for each patch and the class token. We use $7 \times 7 = 49$ patches on MNIST and $8 \times 8 = 64$ patches on CIFAR-10.

There are a number of notable differences between our setup and other works, which report state-of-the-art results on image classification using Transformers. Most importantly, previous works [24, 39] either consider only, or pretrain on, large or huge datasets; for example, ImageNet [22, 39, 74], ImageNet-21k [65], or JFT-300M, with over 375 million labeled datapoints [24, 71]. Additionally, previous works use significantly more patches (e.g., 256 in [24]) and use higher resolutions, including during fine-tuning by upscaling from 32×32 to 224×224 resolution [24, 39, 46, 73].

The aim of this work is not to match performance of such studies. Instead, we demonstrate that NPTs can make use of between-datapoint interactions for prediction on a wide variety of data modalities and tasks, including image classification. With greater scale and the above tweaks, we believe that NPTs have strong potential in computer vision.

C Additional Details on the NPT Architecture

C.1 NPT Training and Hyperparameters

C.1.1 NPT-Base Architecture

Below, we outline the `NPT-Base` model configuration. The final configurations used for each dataset are essentially the same as `NPT-Base`, with minor alterations in parameters such as hidden dimension size, learning rate warmup, batch size, and number of training steps. Given our limited memory and compute time budget, these changes directly result from differences in number of datapoints/attributes between the datasets. We divide the `NPT-Base` configuration into architectural details and optimization details.

`NPT-Base` Architecture

- 8 layers, alternating Attention Between Datapoints and Attention Between Attributes.
- 8 heads.
- Row-wise feed-forward (rFF) networks with one hidden layer, 4x expansion factor, and GeLU activation (standard in Transformer literature [56, 76]).
- Attention weight and hidden layer dropout with $p = 0.1$ (cf. Appendix C.2.1).
- Per-attribute hidden dimension $e = 64$.

`NPT-Base` Optimization

- LAMB [81] optimizer with $\beta = (0.9, 0.999)$ and $\epsilon = 1e-6$, and a Lookahead [82] wrapper with slow update rate $\alpha = 0.5$ and $k = 6$ steps between updates.
- Stochastic feature masking probability $p_{\text{feature}} = 0.15$.
- Anneal the tradeoff λ between feature and target loss with a cosine schedule, starting at 1 (all feature loss) to 0 (all target loss) over the course of training.
- Flat-then-anneal learning rate schedule: flat at the base learning rate for 70% of steps, and then anneals following a cosine schedule to 0 by the end of training.
- Base learning rate 1e-3.
- Gradient clipping at 1.

On all datasets with minibatching, we approximately maintain relative train, validation, and test datapoint proportions in each batch. We train NPT in semi-supervised mode (cf. Appendix C.6.2) but have found that this does not consistently improve performance compared to conventional training because the amount of unlabeled test data is usually comparatively small.

C.1.2 NPT Training on Small Data

Here we describe the hyperparameter sweep details for small datasets – Breast Cancer, Boston, Concrete, and Yacht.

Base Hyperparameter Configurations. Across these small datasets, we make a few minor adjustments to the `NPT-Base` architecture and optimization to obtain the `NPT-Small` configuration: we increase the default number of hidden dimensions to $e = 128$, fix the flat-then-anneal schedule to be flat for 50% instead of 70% of steps, and train with the entire dataset as input, i.e., no minibatching. We set stochastic target masking probability to $p_{\text{target}} = 1$ by default, i.e., deterministically mask out train labels as would be done in a normal supervised setting, and then introduce modifications in our sweep.

Note that the vast majority of hyperparameters such as the number of layers and heads, optimizer, p_{feature}, tradeoff annealing schedule, learning rate schedule, and gradient clipping are exactly the same between `NPT-Base` and `NPT-Small`.

We would like to keep the base configuration for each of the small datasets exactly the same. However, we need to slightly vary the learning rate and number of epochs per dataset to optimize loss convergence across datasets. We use a base learning rate 5e-4 on Breast Cancer and 1e-3 on the other small datasets. We train for 2000 epochs on Breast Cancer and Concrete, and 10 000 epochs on Yacht and Boston. On Breast Cancer, we additionally drop $e = 32$ due to memory constraints (it has more attributes than other small datasets).

Small Data Sweep. Based on these configurations, we sweep over the following 8 configurations of the model on each dataset.[12]

- Vanilla `NPT-Small` model for given dataset.
- Increase number of layers $8 \rightarrow 16$.
- Increase number of heads $8 \rightarrow 16$.
- Increase number of layers $8 \rightarrow 16$, and number of heads $8 \rightarrow 16$.
- Stochastic target masking with probability $p_{\text{target}} = 0.1$.
- Stochastic target masking with probability $p_{\text{target}} = 0.5$.
- Increase stochastic feature masking probability from 0.15 to $p_{\text{feature}} = 0.2$.
- Use a cosine cyclic learning rate scheduler with two cycles, initial learning rate 1e-7, final learning rate 1e-7, and max learning rate given by the base model learning rate.

For the stochastic target masking variants, we proportionally increase the number of epochs (e.g., with $p_{\text{target}} = 0.5$, half as many targets are observed in a given epoch, so we double the total number of epochs).

Small Data Variant Rank Orders. We report the rank order ($\pm$ standard error) of these variants in Table 10. A notable trend is that the *target masking configurations perform particularly well*. One of the two configurations with target masking is the top performer on each of the four datasets. This could be attributed to some combination of the representational advantage of label masking (cf. Section 2.6), an additional regularization effect akin to dropout, or stabler convergence over a greater number of epochs.

Other configurations did not display similarly obvious trends in performance. This is in concordance with the ablation study (Appendix B.4) and supports the claim that NPT is robust to changes in hyperparameters.

C.1.3 NPT Training on Medium and Large Data

For the medium and large datasets, we again adopt the `NPT-Base` architecture and optimization hyperparameters, and make minor manual changes on a per-dataset basis to account for differences in number of datapoints and attributes across the datasets. No more than 3 manual iterations are performed to find these adaptations. We generally attempt to maximize batch size given a fixed memory budget. Given the rank order results on small data (cf. Table 10) we use target masking on the medium and large datasets whenever computationally feasible.[13]. These per-dataset alterations are reported below.

UCI Datasets. We report results for Protein using the Base NPT configuration in the ablation study (cf. Table 4). On Kick, we use batch size 4096, train for 250 000 steps, and use $p_{\text{target}} =$

[12] Note that we do not search a 2^8 grid over these modifications. We only try out these 8 distinct models.

[13] Training is slower as only a p_{target} proportion of training labels are used for backpropagation in each epoch. Therefore, target masking may increase training time beyond our budget.

Table 10: Average rank order of variants of `NPT-Small` ($\pm$ standard error) across 10 cross-validation splits on each small dataset. We determine rank using negative log-likelihood and sort methods by ascending rank for each metric.

Dataset	Boston
$p_{\text{target}} = 0.5$	2.50 ± 0.73
$p_{\text{target}} = 0.1$	2.50 ± 0.83
$8 \rightarrow$ 16 Layers, $8 \rightarrow$ 16 Heads	2.60 ± 0.65
Cosine Cyclic LR Schedule	3.10 ± 0.75
Base `NPT-Small`	3.70 ± 0.84
$8 \rightarrow$ 16 Layers	4.30 ± 0.67
$8 \rightarrow$ 16 Heads	4.40 ± 0.60
$p_{\text{feature}} = 0.2$	4.90 ± 0.46

Dataset	Breast Cancer
$p_{\text{target}} = 0.1$	2.60 ± 0.92
Base `NPT-Small`	2.70 ± 0.65
$8 \rightarrow$ 16 Heads	3.00 ± 0.49
$p_{\text{feature}} = 0.2$	3.20 ± 0.68
$p_{\text{target}} = 0.5$	3.50 ± 0.56
Cosine Cyclic LR Schedule	4.10 ± 0.89
$8 \rightarrow$ 16 Layers, $8 \rightarrow$ 16 Heads	4.40 ± 0.70
$8 \rightarrow$ 16 Layers	4.50 ± 0.81

Dataset	Concrete
$p_{\text{target}} = 0.5$	2.30 ± 0.76
Cosine Cyclic LR Schedule	2.50 ± 0.69
$8 \rightarrow 16$ Heads	2.60 ± 0.62
Base `NPT-Small`	2.70 ± 0.52
$p_{\text{target}} = 0.1$	3.10 ± 0.64
$8 \rightarrow 16$ Layers	3.90 ± 0.80
$8 \rightarrow 16$ Layers, $8 \rightarrow$ 16 Heads	5.10 ± 0.66
$p_{\text{feature}} = 0.2$	5.80 ± 0.39

Dataset	Yacht
$p_{\text{target}} = 0.1$	1.20 ± 0.53
$p_{\text{target}} = 0.5$	2.70 ± 0.52
$8 \rightarrow$ 16 Heads	2.80 ± 0.66
Cosine Cyclic LR Schedule	3.10 ± 0.69
Base `NPT-Small`	3.60 ± 0.54
$8 \rightarrow 16$ Layers	4.10 ± 0.74
$p_{\text{feature}} = 0.2$	5.20 ± 0.47
$8 \rightarrow 16$ Layers, $8 \rightarrow$ 16 Heads	5.30 ± 0.83

0.5. On Income, we use batch size 2048, train for 2 000 000 steps, use no feature masking (and correspondingly fix the tradeoff parameter $\lambda = 0$), and use $p_{\text{target}} = 0.15$. On Poker Hand, we use batch size 4096, train for 200 000 steps, use $p_{\text{target}} = 0.5$, and stratify by class (i.e., compose training datapoints in each minibatch proportionally to the empirical label distribution of the training set to account for significant class imbalance). On Forest Cover, we use batch size 1800, train for 800 000 steps, use a polynomial decay learning rate scheduler with warmup over the first 1% of steps, use base learning rate 0.005, $p_{\text{target}} = 0.5$, and class balancing as above. The changes to learning rate scheduling were made to speed up training and hence save compute resources. On Higgs, we use batch size 4096, train for 500 000 steps, and do not use target masking due to constraints on compute time.

Image Data (CIFAR-10 and MNIST). On CIFAR-10, we use batch size 512, train for 1000000 steps, use random crops and horizontal flips for data augmentation, use $8 \times 8 = 64$ patches of each image, and do not use target masking due to constraints on compute time. On MNIST, we use batch size 512, train for 500000 steps, use hidden dimensions $e = 16$, $p_{\text{target}} = 0.15$, and use $7 \times 7 = 49$ patches. See Appendix B.6 for details on patching.

Again, we stress that the vast majority of hyperparameters used on all datasets (small, medium, and large benchmarks from UCI as well as the image benchmarks) are identical; configurations follow `NPT-Base` (cf. Appendix C.1.1) very closely and changes usually affect NPT optimization rather than architecture.

C.2 Further Details on ABD and ABA Layers

C.2.1 Dropout

In practice, we apply elementwise dropout on the attention scores $\exp(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{h})$, as well as on the input/output embeddings and the output of the MHSelfAtt$(\cdot)$ function (often referred to as attention and hidden dropout).

C.3 Input and Output Embdedings

C.4 Input Embedding

At a high-level, we embed inputs by encoding categorical attributes as one-hot vectors and standardizing continuous attributes, followed by a learned linear embedding for each attribute to obtain $\text{InputEmbed}(\boldsymbol{X}) = \boldsymbol{H}^{(0)} \in \mathbb{R}^{n \times d \times e}$.

More specifically, we perform the following sequence of steps: Attributes $\boldsymbol{X}_{:,j}, j \in \{1, \dots, d\}$ of the input matrix can be either continuous or categorical. We first apply a function $\text{Encode}(\cdot)$ to each attribute $\boldsymbol{X}_{:,j}$. This "encodes" categorical attributes with a one-hot representation and standardizes continuous attributes to zero mean and unit standard deviation. Each encoded attribute j has (potentially unique) dimensions $n \times e_j$. Then, we concatenate this encoded attribute with its respective column of the masking matrix $\boldsymbol{M}_{:,j}$ along the second dimension to produce a column encoding of dimensions $n \times (e_j + 1)$. We learn separate embedding weights for each attribute $\boldsymbol{W}_j^{\text{in}} \in \mathbb{R}^{(e_j+1) \times e}$ that embed all attributes to a common hidden dimension e. Altogether, we can state the embedding of a single attribute column $\boldsymbol{X}_{:,j}$ as

$$\boldsymbol{H}_{:,j}^{(0)} = \underset{\text{axis}=e}{\text{concat}}(\text{Encode}(\boldsymbol{X}_{:,j}), \boldsymbol{M}_{:,j})\boldsymbol{W}_j^{\text{in}} + \boldsymbol{H}_{:,j}^{\text{Index}} + \boldsymbol{H}_{:,j}^{\text{Type}}, \tag{21}$$

where $\boldsymbol{H}_{:,j}^{\text{Index}} \in \mathbb{R}^{n \times e}$ is a learnt embedding for the index and $\boldsymbol{H}_{:,j}^{\text{Type}} \in \mathbb{R}^{n \times e}$ for the type (either continuous or categorical) of attribute j.

Finally, we write the full NPT input embedding layer as

$$\text{InputEmbed}(\boldsymbol{X}) = \underset{\text{axis}=d}{\text{stack}}(\boldsymbol{H}_{:,1}^{(0)}, \dots, \boldsymbol{H}_{:,d}^{(0)}) = \boldsymbol{H}^{(0)} \in \mathbb{R}^{n \times d \times e}. \tag{22}$$

The stack operation constructs $\boldsymbol{H}^{(0)} \in \mathbb{R}^{n \times d \times e}$ from d attribute embeddings $\boldsymbol{H}_{:,j}^{(0)} \in \mathbb{R}^{n \times e}, j \in \{1, \dots d\}$.

C.5 Output Embedding

For an NPT with L layers, we obtain an output prediction by applying a learnt linear output embedding (that closely mirrors the process of the input embedding) to the output of the last attention layer $\boldsymbol{H}^{(L)}$. We write the output embedding layer as

$$\text{OutputEmbed}(\boldsymbol{H}^{(L)}) = [\boldsymbol{Z}_{:,1}, \dots, \boldsymbol{Z}_{:,d}] = \boldsymbol{Z}, \tag{23}$$

$$\text{where } \boldsymbol{Z}_{:,j} = \boldsymbol{H}_{:,j}^{(L)} \boldsymbol{W}_j^{\text{out}}. \tag{24}$$

Our prediction $\boldsymbol{Z}$ is a list of d attribute predictions $\boldsymbol{Z}_j \in \mathbb{R}^{n \times e_j}$. We learn output embedding weights $\boldsymbol{W}_j^{\text{out}} \in \mathbb{R}^{e \times e_j}$ which are applied on attribute slices $\boldsymbol{H}_{:,j,:}^{(L)} \in \mathbb{R}^{n \times e}$ of the output of the Lth layer $\boldsymbol{H}^{(L)} \in \mathbb{R}^{n \times d \times e}$. Note that the second dimension of each attribute prediction $\boldsymbol{Z}_j$ is determined by the encoding size (i.e., $e_j = 1$ for continuous attributes, e_j is the number of categories for a categorical attribute) as in the input embedding. Note also that we do not predict a mask value (i.e., we do not predict to dimensions $n \times (e_j + 1)$ for each attribute). To obtain the final prediction matrix $\hat{\boldsymbol{X}} \in \mathbb{R}^{n \times d}$ we take the $\arg\max$ over the categorical predictions.

C.6 NPT Masking

C.6.1 Handling Missing Values

Real-world data – particularly tabular data – often contains *missing entries*. Many popular models for supervised prediction on tabular data cannot accommodate missing values as input. Instead they require that missing features are *imputed*, i.e., an additional model predicts a surrogate value for what the missing values could have been, such that the supervised model then receives a "clean" dataset as input which no longer overtly contains missing values.

For example, all scikit-learn [59] predictors, including Gradient Boosting and Random Forests, require an explicit imputation step before training. Often, extremely simple imputation methods are used in practice. For example, TabNet [2] drops datapoints with >10% missing entries and otherwise

applies univariate mean imputation as part of a Google AI Platform pipeline [60]; and CatBoost [61] treats a missing continuous entry as the minimum or maximum of that feature (univariate min/max imputation), or raises an error. While more complex imputation methods could in theory be applied as pre-processing [36, 43, 69, 70, 75], there will always remain a separation between the imputation step and the prediction model. Additionally, more complex imputation methods often require training and hyperparameter selection, such that the combined imputation and prediction process becomes cumbersome. Both for practical as well as performance reasons, it is desirable to have a single model that can *directly* handle missing data, learn complex internal imputation operations from the data, and at the same time learn the desired predictive function from features to target.

This is exactly what NPTs achieve. They are able to accommodate inputs with missing values gracefully without requiring any imputation pre-processing steps, therefore modeling data with missing values end-to-end. We can explicitly indicate that a value $\boldsymbol{X}_{i,j}$ is missing by simply setting the mask token $\boldsymbol{M}_{i,j} = 1$. Already in standard NPTs, the stochastic feature masking during training teaches NPTs to predict values for which $\boldsymbol{M}_{i,j} = 1$ while ignoring the value of their entry $\boldsymbol{X}_{i,j}$ at input. Further, no choice of fixed imputation algorithm has to be made with NPTs. Instead, NPTs learn directly from the data how to make predictions given missing values. Attention between datapoints might be particularly useful for learning a general mechanism of how to impute missing values by attending to other datapoints. We therefore suspect that NPTs could be a strong contender for predicting on data with missing values. Further, unlike common imputation pre-processing, NPTs do not discard the information of *which* attributes were missing. Future work could also explore the ability of NPT to model arbitrary correlations underlying the pattern of which data is missing, i.e., datasets where values are not missing at random.

C.6.2 Masking Encompasses Many Common Machine Learning Settings

The flexible masking mechanism of NPTs can be used to accommodate a variety of common machine learning settings.

Multi-Target Prediction. In *multi*-target classification or regression, more than one column of the dataset contains targets. Standard supervised models often do not support multi-output settings and must resort to training multiple models, one for each target. NPTs can accommodate multi-target prediction trivially, since they learn to make predictions at any masked input entry. For prediction in a multi-target setting, we simply apply target masking on all columns with targets.

Self-Supervision. In self-supervised learning, we are often interested in learning a generative model or useful encoding from unlabeled data. The reconstruction of corrupted input features as part of stochastic feature masking can already be seen as self-supervised learning. The stochastic masking mechanism allows NPTs to learn to predict masked out values anywhere in the input. In theory, NPTs should be able to learn a fully generative model of the dataset in this manner.

Semi-Supervision. In semi-supervised learning, we hope to use large quantities of unlabeled data to aid in learning a predictive function on a small set of labeled data. Often, this involves a two-step process, such as learning a powerful autoencoder from all data and then training a predictor using the learnt encoder and the small set of labeled data. NPTs can accommodate semi-supervised learning without changes to the architecture. Specifically, we can include large amounts of unlabeled data by simply appending those feature values to the labeled input dataset. We indicate that no labels are available for all unlabeled datapoints i' by setting their mask token at the target column $\boldsymbol{X}_{i',d} = 1$. NPTs can use attention between datapoints to make use of information from the features of the unlabeled datapoints.

Imputation. With imputation, we refer to scenarios where the main task is to predict missing values for arbitrary attributes and datapoints. Similar to self-supervision, NPTs already learn how to do this from the stochastic masking mechanism that is enabled by default. (Unlike for the self-supervision category, the imputation scenario assumes that there are actually some missing values that we would like to predict.)

C.6.3 Stochastic Masking: Details

For stochastic masking, a specified proportion of training entries (we default to 15% following [23]) are selected for masking at the start of each epoch. Among those entries chosen, we mask out the value with 90% probability and randomize it with 10% probability. "Masking out" means that the original value $\boldsymbol{X}_{i,j}$ is overwritten with zeros and the mask token is set to 1. Randomization is done

for categorical targets by sampling a new class uniformly at random. Continuous targets are sampled from a standard Normal $\mathcal{N}(0, 1)$.

This sampling scheme is applied for both stochastic feature masking and stochastic target masking, where we allow for different masking proportions between the two (p_{feature} and p_{target}). During training, a loss is backpropagated on the masked entries.

C.7 NPT Optimization

Each of the losses $\mathcal{L}^{\text{Features}}$ (feature loss) and $\mathcal{L}^{\text{Targets}}$ (target loss) is normalized by the number of entries on which it is evaluated.

As described in Appendix C.1.1: we anneal the λ parameter in the NPT objective using a cosine schedule, i.e., starting with full weight on the feature loss term at epoch 0 and annealing to full weight on the target loss term by the end of training. We use LAMB [81] with Lookahead [82] for optimization, which we find to perform well with large minibatches. We use a flat-then-anneal learning rate schedule with cosine decay, notable as Transformer works [23, 76] often report that a linear learning rate warmup is necessary for training stability. Our placement of Layer Normalization before self-attention ("pre-LayerNorm" [3, 15]) may contribute to our not needing this.

D Related Work – Continued

D.1 Semi-Supervised Learning and Graph Neural Networks

Kipf et al. [44] introduce semi-supervised classification over graphs with graph convolutional networks (GCNs). In a similar fashion, NPTs can natively perform semi-supervised learning over arbitrary data by including test datapoints (with permanently masked out targets) at training time. NPTs can be seen as a conceptual generalization of GCNs and other graph neural networks like GAT [77] and GIN [79], in which a set of dependencies (edges) between datapoints is not known a priori and instead is learned from data using self-attention. Neural Relational Inference (NRI) [45] also attempts to discover relations amongst datapoints. However, it uses message passing, which lacks scalability because it requires storing embeddings for each potential edge: an $O(en^2)$ cost at each layer, where e is the embedding size, and n the number of nodes. NPTs improve scalability using self-attention with cost $O(hn^2)$, where h is the number of heads, n the number of rows or columns, and $h \ll e$. We leave further exploration of the close connection between NPTs and graph neural networks to future work.

D.2 Tree-Based Baselines

Tree-based approaches in machine learning have been popular for over half a century [11, 54, 55]. Each node of a tree splits the data into smaller subsets, and predictions are made at each of the leaves. The splits are learned from a set of training data by minimizing some objective function. Many established methods combine predictions of multiple trees through bagging [9] and/or boosting [67]. Bagging uses an ensemble of trees, each learned by training on a random subsample of the data. This approach is most popularly used in Random Forests [10]. Boosting learns a sequence of trees, conditioning the learning of each additional model on the predictions of previous models, with the aim of reducing overall prediction error.

Popular examples of tree-based boosting models include AdaBoost [28], XGBoost [16], CatBoost [61], and LightGBM [41]. To date, boosting arguably comprises the most popular approach for tabular data prediction. These models often rely on careful tuning of a large variety of hyperparameters. However, training cost is often cheap compared to neural network architectures, and therefore, so is hyperparameter optimization. This balance is slightly offset for NPTs, which seem largely robust to hyperparameter tuning. Hence, the training of a single NPT is often competitive to a grid search over hyperparameters for a tree-based model.

E Classification and Regression Benchmark Details

E.1 General Setup

For certain datasets we use a canonical fixed test set. Otherwise, we default to 10-fold cross validation with 0.7/0.2/0.1 splits on smaller datasets and a single 0.7/0.1/0.2 split on larger datasets, where the exact split indices are always consistent across baselines. The full details on all UCI benchmark datasets are given in Tables 11 and 12. Note the variety of the datasets across number of instances, number of features, composition (categorical or continuous) of features, and task (multi-class classification, binary classification, and regression).

E.2 Hyperparameter Tuning

E.2.1 Overview

Table 13 lists the number of unique hyperparameter configurations swept over for each baseline and classification/regression dataset.

All details on the NPT hyperparameter setup are given in Appendix C.1. Note that for any given dataset, NPT is tuned over fewer configurations than the baselines: we fix a base model configuration with minimal data-dependent tuning of hyperparameters such as learning rate, scheduler, number of steps, and target masking percentage p_{feature}, and choose the largest batch size viable for our hardware. On small datasets, we then sweep over 8 variants, and on medium and large datasets (including image data) use only the fixed variant with minor modifications.

In the case of TabNet, the configurations used for Poker Hand, Forest Cover, and Higgs Boson are those reported by the original authors for these datasets [2]; for Income, we performed a sweep over configurations including one reported for that dataset in the original publication. All deep learning approaches (MLP, TabNet, and NPT) use early stopping on the validation target loss.

E.2.2 Baseline Sweep Details

We report hyperparameter sweep details for baselines below. The associated tables for each baseline give the bounds of the search space for numerical hyperparameters and all values for categorical hyperparameters. We clarify specific hyperparameters and provide context where helpful.

Random Forest (Tables 14, 15). `criterion` refers to the split criterion. `max_features` is the number of features to consider when looking for the best split.

Gradient Boosting, XGBoost, LightGBM, and CatBoost (Table 16). See D.2 for background on tree-based baselines.

MLP (Tables 17, 18, 19). The invscaling `learning_rate` scheduler scales with $\alpha_t = \alpha_0/t^{0.5}$ where t is the step, α_0 the initial learning rate, and α_t the learning rate at step t. The adaptive `learning_rate` divides the current learning rate by 5 when two consecutive epochs fail to decrease training or validation log loss by a tolerance 1e-4. Due to compute constraints, we decreased the size of the search space as the dataset size increased by focusing on 3-layer networks, lower L2 penalties, and higher batch sizes.

k-NN (Tables 20, 21, 22). `weights` describes the weight function applied to the neighborhood, i.e., "distance" means that closer neighbors of a query point have greater influence than those further away. `algorithm` specifies the underlying k-NN algorithm, where KD Tree [7] and Ball Tree [52] are approximations of brute-force search. The "auto" setting determines an appropriate algorithm based on the input data [59]. `leaf_size` is a hyperparameter of KD Tree and Ball Tree. `p` is the power parameter for the distance metric, i.e., $\texttt{p} = 1$ yields Manhattan and $\texttt{p} = 2$ Euclidean distance. It was computationally infeasible for us to obtain reasonable results on the 11M instance Higgs Boson dataset. Even when attempting approximate 3-NN on an Azure D64 v3 instance with 256 GB RAM, we encountered an out-of-memory error.

Table 11: UCI classification dataset statistics and experimental setup details.

Dataset	Higgs Boson	Poker Hand	Forest Cover	Income	Kick	Breast Cancer
# Instances	11,000,000	1,025,010	581,012	299,285	72,983	569
# Features	28	10	54	42	32	31
# Categorical Features	0	10	44	36	18	0
# Continuous Features	28	0	10	6	14	31
# Classes	2	10	7	2	2	2
Train/Val/Test Split	0.84/0.12/0.05	0.017/0.003/0.98	0.7/0.1/0.2	0.57/0.1/0.33	0.7/0.1/0.2	0.7/0.2/0.1
Fixed Test Set	Yes	Yes	No	Yes	No	No (10-Fold CV)
Uses Minibatching	Yes	Yes	Yes	Yes	Yes	No

Table 12: UCI regression dataset statistics and experimental setup details.

Dataset	Protein	Concrete	Boston	Yacht
# Instances	45,730	1030	506	308
# Features	9	9	13	6
# Categorical Features	0	0	2	5
# Continuous Features	9	9	11	1
Train/Val/Test Split	0.7/0.1/0.2	0.7/0.2/0.1	0.7/0.2/0.1	0.7/0.2/0.1
Fixed Test Set	No	No (10-Fold CV)	No (10-Fold CV)	No (10-Fold CV)
Uses Minibatching	Yes	No	No	No

Table 13: Number of unique hyperparameter configurations swept over for each model class and dataset. Here we shorten Boston Housing to BH, Breast Cancer to BC, Poker Hand to PH, Forest Cover to FC, and Higgs Boson to HB. Datasets are ordered by increasing number of datapoints (n) from left to right.
* TabNet on Protein, Kick, and Income is tuned by sweeping over all 6 configurations listed in the original paper [2] in addition to the default configuration. Note that these configs include one tuned on Income.
† TabNet on Poker Hand, Forest Cover, and Higgs Boson use precisely the configuration specified for those datasets in the original paper [2].
‡ For some of these, we manually optimized convergence of the validation loss by adjusting non-architectural parameters such as learning rate (schedule), batch size, or number of steps in at most 3 iterations. See C.1.3.

Dataset	Yacht	BH	BC	Concrete	Protein	Kick	Income	PH	FC	HB
Random Forest	24	24	24	24	24	24	24	24	24	24
Gradient Boosting	48	48	48	48	48	48	48	48	48	48
XGBoost	48	48	48	48	48	48	48	48	48	48
CatBoost	48	48	48	48	48	48	48	48	48	48
LightGBM	48	48	48	48	48	48	48	48	48	48
MLP	11,340	11,340	11,340	11,340	270	270	270	270	270	6
k-NN	480	480	480	480	40	40	40	40	40	-
TabNet	48	48	48	48	7*	7*	7*	$1^{\dagger}$	$1^{\dagger}$	$1^{\dagger}$
NPT	8	8	8	8	$1^{\ddagger}$	$1^{\ddagger}$	$1^{\ddagger}$	$1^{\ddagger}$	$1^{\ddagger}$	$1^{\ddagger}$

Table 14: Random Forest classification hyperparameters.

Hyperparameter	`criterion`	`n_estimators`	`max_features`
Setting	gini, entropy	[50, 1000]	auto, sqrt, log2

Table 15: Random Forest regression hyperparameters.

Hyperparameter	`criterion`	`n_estimators`	`max_features`
Setting	mae, mse	[50, 1000]	auto, sqrt, log2

Table 16: Gradient Boosting, XGBoost, LightGBM, and CatBoost hyperparameters (for both regression and classification).

Hyperparameter	learning_rate	n_estimators	max_depth
Setting	[1e-3, 0.3]	[3, 10]	[50, 1000]

Table 17: MLP hyperparameters for small datasets (Boston Housing, Breast Cancer, Concrete, and Yacht).

Hyperparameter	hidden_layer_sizes	l2_penalty
Setting	[(25)-(500), (25,25)-(500,500), (25,25,25)-(500,500,500)]	[0, 1]

Hyperparameter	batch_size	learning_rate	learning_rate_init
Setting	[32, 256]	constant, invscaling, adaptive	[1e-5, 1e-1]

Table 18: MLP hyperparameters for medium and large datasets other than Higgs Boson (Protein, Kick, Income, Poker Hand, Forest Cover).

Hyperparameter	hidden_layer_sizes	l2_penalty
Setting	[(25,25,25)-(500,500,500)]	[0, 1e-2]

Hyperparameter	batch_size	learning_rate	learning_rate_init
Setting	[128, 256]	constant, invscaling, adaptive	[1e-5, 1e-1]

Table 19: MLP hyperparameters for the Higgs Boson dataset.

Hyperparameter	hidden_layer_sizes	l2_penalty
Setting	(500,500,500)	0

Hyperparameter	batch_size	learning_rate	learning_rate_init
Setting	[128, 256]	constant, invscaling, adaptive	[1e-5, 1e-1]

Table 20: k-NN hyperparameters for small datasets (Boston Housing, Breast Cancer, Concrete, and Yacht).

Hyperparameter	n_neighbors	weights	algorithm	leaf_size	p
Setting	[2, 100]	uniform, distance	ball_tree, kd_tree, brute	[10, 100]	1, 2

Table 21: k-NN hyperparameters for medium-large datasets (Protein, Kick, Income, Poker Hand).

Hyperparameter	`n_neighbors`	`weights`	`algorithm`	`leaf_size`	`p`
Setting	[2, 1000]	distance	auto	[10, 100]	2

Table 22: k-NN hyperparameters for Forest Cover.

Hyperparameter	`n_neighbors`	`weights`	`algorithm`	`leaf_size`	`p`
Setting	[2, 25]	distance	auto	[10, 100]	2

F Societal Impacts of NPT

We have introduced Non-Parametric Transformers, a novel deep learning architecture that predicts by including learned interactions between points of the dataset. In this work, we take first steps towards exploring NPTs and their properties. We do not recommend that NPTs are carelessly applied in production settings, because we do not yet know enough about them. We now list common concerns in applying machine learning models, discuss how they may apply to NPTs, and how to potentially mitigate them.

Many countries of the world, such as the US, UK, and the countries of the EU, are implementing "Right to Explanation"-schemes that grant those affected by autonomous decisionmaking the right to an explanation of why and how decisions were made. In general, Transformer-based architectures such as NPT have been shown to be amenable to explanations, see e.g., [76]. One could argue that our experiments in §4.4 move in an explanatory direction. However, we have not sufficiently investigated the explanations of individual NPT decisions, and believe this to be exciting future work.

Machine learning models are increasingly used in autonomous decision making that affects human beings in some capacity, e.g., clinical diagnosis, autonomous driving, and detection of toxic comments online.[14] It is of great importance that those decisions are *fair*, i.e., that they do not discriminate against underrepresented groups in some manner. We have not yet investigated how NPTs respond to common techniques of calibrating machine learning models to fulfil some definition of fairness. We believe that their special predictive behavior from similar datapoints likely poses both challenges and opportunities in this domain. For example, instead of needing to retrain the model to elicit changes in prediction – which could be infeasible in a real-world deployment – NPT could be "prompted" with a different set of context datapoints to modify its predictive behavior towards a more socially desirable response.

In large architectures based on Transformers, the memorization of training data is a common concern. If the model memorizes training data, adversarial attacks can be used to extract training data from the model weights, see e.g., [14]. This can lead to violations of privacy if, for example, a publicly available model was trained on data that must remain private. This can also cause more subtle problems; for example, if training data "lives on" in the model but must be deleted at some point in time to comply with privacy regulations. As NPT directly relies on training data as input for prediction, NPT is not a "private" model per definition. However, we can imagine future work tackling this question; for example, by learning to predict from a set of anonymous representative points instead of the training data directly.

At the model sizes presented in the paper, the environmental impact of training and using NPT is relatively small compared to some of the large architectures currently in fashion, see e.g., [12]. However, NPT could be scaled up to larger sizes at which point the energy used for training and

[14] For example, see:
Filos, Angelos, et al. "A systematic comparison of bayesian deep learning robustness in diabetic retinopathy tasks." NeurIPS Workshop on Bayesian Deep Learning (2019).
Michelmore, Rhiannon, Marta Kwiatkowska, and Yarin Gal. "Evaluating uncertainty quantification in end-to-end autonomous driving control." arXiv:1811.06817 (2018).
van Aken, Betty, et al. "Challenges for toxic comment classification: An in-depth error analysis." arXiv:1809.07572 (2018).

prediction would become a serious concern. When considering tabular data, training a *single* NPT model is expensive compared to training a *single* one of our tree-based baselines such as XGBoost. However, we find that such baselines are often more sensitive to correctly tuned hyperparameters than NPT, such that the total compute including hyperparameter tuning of NPT and the baselines is actually often similar, particularly on larger datasets. Sparse approximations as referenced in Section 5 may further reduce the computational impact of NPT.

NPT is a new – and exciting – architecture. Therefore, in applications where explanations, fairness, or privacy are desired or legally required, we do not recommend that NPT be used at this stage.

G Code, Computational Resources, and License

Code. We release code for NPTs at github.com/OATML/Non-Parametric-Transformers. The code relies on PyTorch [58] and NumPy [34], and we use Scikit-Learn [59] for many of the baseline experiments.

Computational Resources. For the experiments we mainly rely on a shared internal cluster that has both NVIDIA Titan RTX GPUs (24 GB memory) as well as NVIDIA GeForce RTX 2080 Tis (12 GB memory). For tuning baselines, which are often compute-heavy workloads, we use Azure D-series compute-optimized VMs. For small datasets (< 1000 datapoints) such as Breast Cancer, training and evaluation of a single NPT model takes about 10 minutes. For larger datasets such as Protein ($< 100\,000$ datapoints), training and evaluation of NPT takes about 10 hours. For the largest datasets, e.g., Higgs Boson with 11 million datapoints, training and evaluation of NPT takes about 5 days. We did not optimize NPT for efficiency or training speed in this paper and suspect that convergence could be drastically improved with relatively little effort. The total amount of compute used for this paper is given by all NPT and baseline runs with repetitions for cross-validation, which amounts to more than 30 GPU days.

License Agreements.
License agreement of the CIFAR-10 dataset: CIFAR-10 is published under MIT license.

License agreement of the MNIST dataset: License: Yann LeCun and Corinna Cortes hold the copyright of MNIST dataset, which is a derivative work from original NIST datasets. MNIST dataset is made available under the terms of the Creative Commons Attribution-Share Alike 3.0 license.

UCI Machine Learning Repository: Licenses for all datasets can be found at archive.ics.uci.edu/ml/.