Neural Point Processes for Pixel-wise Regression

Chengzhi Shi
Northeastern University

Gözde Özcan
Northeastern University

Miquel Sirera Perelló
Northeastern University

Yuanyuan Li
Northeastern University

Nina Iftikhar Shamsi
Northeastern University

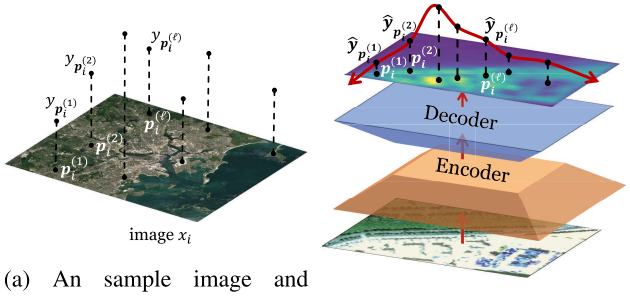
Stratis Ioannidis
Northeastern University

Abstract

We study pixel-wise regression problems with sparsely annotated images. Traditional regression methods based on mean squared error emphasize pixels with labels, leading to distorted predictions in unlabeled areas. To address this limitation, we introduce *Neural Point Processes*, a novel approach that combines 2D Gaussian Processes with neural networks to leverage spatial correlations between sparse labels on images. This approach offers two key advantages: it imposes smoothness constraints on the model output and enables conditional predictions when sparse labels are available at inference time. Empirical results on synthetic and real-world datasets demonstrate a substantial improvement in mean-squared error and R^2 scores, outperforming standard regression techniques. On the real-world dataset COWC, we achieve an R^2 of 0.769 with 81 out of $40,000$ (0.2%) points labeled, while standard regression loss (MSE) results in an R^2 of 0.060.

1 INTRODUCTION

Pixel-wise regression (Liu et al., 2021a), i.e., the inference from images at the pixel level, finds applications in several areas, including remote sensing (Yokoya et al., 2022; Wang et al., 2016; You et al., 2017), satellite (Schultz et al., 2015) and medical imaging (Yao et al., 2018), to name a few. We study pixel-wise regression in a setting where pixel labels are *sparse*: as shown in Figure 1a, only a handful of pixel labels are present per image. A canonical application is, e.g., regressing geo-located labels, such as air pollution measurements collected from sparsely deployed sensors, from aerial or satellite views, and possibly other two-dimensional covariates (e.g., traffic, temperature, and wind direction data). One possible means of addressing sparsity is by exploiting spatial correlations between pixels. Existing approaches in pixel-wise regression utilize latent feature representations to correlate pixels with similar semantics to the same pixel values, but the correlation is only implicitly enforced via the convolutional layers (Liu et al., 2021b; Baumann et al., 2023).



(a) An sample image and sparsely labeled point set

(b) Overview of NPP regression

Figure 1: (a) Our data comprises images $\mathbf{x}_i$, along with (sparsely) *labeled point sets*: points $\mathbf{p}_i^{(\ell)}$, $\ell \in \{1, \ldots, L_i\}$, represent pixel locations on image $\mathbf{x}_i$ and corresponding labels $\mathbf{y}_i^{(\ell)}$ are the associated label values: these could be, e.g., geo-located measurements of air pollution or traffic conditions, produced by sensors in the corresponding locations. (b) From a given image, the autoencoder outputs a prediction of the target values over pixels. Although giving predictions for all image points, we only use the predictions for the points that have a ground truth value to optimize the network parameters. Crucially, the labels are assumed to form a GP, parameterized by the autoencoder outputs.

Spatial information is important in remote sensing, and can naturally be imposed (e.g., across geographic areas) via Gaussian Processes (GPs) (You et al., 2017; Tanaka et al., 2019) and deep variants such as Neural Processes (NPs) (Garnelo et al., 2018a,b; Giannone and Ahmed, 2023). Gaussian processes model jointly Gaussian random variables whose mean and covariance are determined as

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

functions of their inputs, while neural processes combine deep neural networks (DNNs) with GPs; neither have been directly applied to the sparse pixel label setting we consider here. We build upon and extend this long line of work by introducing *Neural Point Processes (NPPs)*, illustrated in Figure 1b. In our setting, a DNN model regresses the labels assigned at a handful of pixels scattered across the image, which we refer to as *points*, from the image itself. Crucially, it does so by exploiting spatial correlations between points through the imposition of a GP structure on the DNN's output: this improves predictive performance and also enables us to improve predictions by leveraging partial labels available at inference time. To summarize, our contributions are as follows:

- We study pixel-wise regression via sparsely labeled points rather than the whole image and propose Neural Point Processes as a solution to point-based pixel-wise regression. NPPs naturally combine GPs with DNNs, explicitly imposing spatial correlations. In this setting, we show that maximum likelihood estimation (MLE) amounts to minimizing a Mahalanobis distance variant of the mean squared error loss, along with an appropriate kernel-dependent regularization term.
- Exploiting the ability to compute GP posteriors, we show how GPs can be used to improve predictions of NPPs if a small number of point labels are available at inference time.
- We experiment with two synthetic and two real-world datasets: PMNIST, Synthetic, SpaceNet (Rotterdam), and Cars Overhead With Context (COWC). We show that NPPs outperform competitors under different point distribution densities and patterns. On the real-world dataset COWC, we achieve an R^2 of 0.769 with $81/40000$ (0.2%) points labeled, while standard regression loss (MSE) fails with an R^2 of 0.060.

The remainder of this paper is organized as follows: In Section 2, we present an overview of the related literature. In Section 3, we give the necessary background information on Gaussian Processes. In Section 4, we formally state our objective problem and explain our main contributions in detail. We describe our experimental setup and numerical results on different datasets in Section 5. Finally, we finish the paper with concluding remarks in Section 6.

2 RELATED WORK

The integration of GPs with DNNs induces various advantages, such as capturing uncertainty and incorporating prior knowledge (Damianou and Lawrence, 2013; Bui et al., 2016; Cutajar et al., 2017; You et al., 2017). Hinton and Salakhutdinov (2007) use deep-belief networks to learn the covariance matrix to improve the performance of GPs. Damianou and Lawrence (2013) propose a deep hierarchy in which every layer constitutes a GP, with inputs being the values of the previous layer, whose parameters are learned through variational inference. Dutordoir et al. (2021) show that the forward passes of neural networks are equivalent to sparse Gaussian process models. Calandra et al. (2014) jointly learn a feature map and a GP regression over observations to improve the learnability of complex and non-differentiable functions. Wilson et al. (2016) introduce scalable deep kernels, which transform the inputs of a spectral mixture kernel with a deep architecture, benefiting in expressive power and scalability. Similarly, to scale GPs, Huang et al. (2015) pretrain a denoising autoencoder, then extract the embeddings from the last layer of a DNN and regress a Gaussian process on these embeddings. In the context of satellite images, Tanaka et al. (2019) propose spatially aggregated GPs, an appropriately defined mixture model of GPs, to regress *areal data*, i.e., values assigned to entire geographic regions (e.g., counties), rather than pixels. You et al. (2017) predict future crop yields from multispectral images; training first RNNs or DNNs as feature extractors, latent representations of images are used as inputs to GPs that regress crop yields in different areas, thereby modeling correlations across areas. All the above methods output an image-level prediction; thus, none of these approaches can be directly applied to the sparse point/pixel label setting we study here.

Neural Processes (Garnelo et al., 2018a,b) and their extensions (Louizos et al., 2019; Kim et al., 2018; Nguyen and Grover, 2022) are closer to our setting. Neural Processes comprise an encoder, that learns a latent embedding from a dataset with input and output pairs, and a decoder, that is used to predict labels over new input vectors. Similar to our approach, NPs make the output of the decoder the mean function of GPs and optimize the model by maximizing the Evidence Lower Bound (ELBO). Convolutional Neural Processes (ConvNP) (Gordon et al., 2020; Foong et al., 2020; Bruinsma et al., 2021) extend the NP family by explicitly embedding spatial structure with convolutional layers, thereby enhancing the modeling of nearby points, while "in context in context learning" (Ashman et al., 2024) allows an NP encoder to also exploit external input/output pairs from similar datasets.

Several key differences render the Neural Process family (Dubois et al., 2020), including the extentions above, unsuitable for our sparse point-wise regression setup. First, these models only take coordinates as input: they learn the mapping from locations to label values only, thereby neglecting the rich information contained in image values; incorporating image data as an additional input, as we do here, is not straightforward. Second, they output one value at a time, which limits their computational efficiency in pixel-wise regression tasks; in contrast, our approach di-

rectly predicts labels for an arbitrarily sized subset of pixels simultaneously. Finally, and most importantly, models in the neural process family only operate under the partial label revelation setting (see Sec. 4.3): they require some labeled samples to be revealed at inference time in order to predict remaining labels. This is again in contrast to NPPs, which can be used even when no such labels are available.

Diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020) reinforce correlations across nearby pixels through diffusion operations within latent layers. Baranchuk et al. (2022) propose using the diffusion model as a feature extractor for semantic segmentation when ground truth labels are scarce. The authors build a fully connected network as a pixel-wise classifier fed with pixel-level representations from a pre-trained Denoising Diffusion Probabilistic Model (DDPM). Exploiting correlations in the input is orthogonal to our approach, which attempts to exploit correlations in the output; to illustrate this, we use DDPMs (Bandara et al., 2022) as a feature extractor in our implementation of NPPs during our experimentation.

3 Background: Gaussian Processes

A vector-valued random variable $\mathbf{x} \in \mathbb{R}^n$ is said to have a *multivariate normal (or Gaussian) distribution* with mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ (i.e., $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$) if

$$p(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}}, \quad (1)$$

where $\boldsymbol{\Sigma}$ is a symmetric and positive definite and $|\boldsymbol{\Sigma}|$ is its determinant. A *Gaussian Process* (GP) is a stochastic process comprising (potentially infinitely many) random variables, such that any finite subcollection has a multivariate Gaussian distribution (Do and Lee, 2020; Rasmussen, 2003). Formally, a collection of random variables $\{g(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$ indexed by set $\mathcal{X}$ is a GP with *mean function* $m(\cdot)$ and a symmetric and positive semi-definite *covariance* or *kernel function* $k(\cdot, \cdot)$ if for any finite set of elements $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}$, the associated finite set of random variables $g(\mathbf{x}_1), \ldots, g(\mathbf{x}_m)$ are jointly distributed as:

$$\begin{bmatrix} g(\mathbf{x}_1) \\ \vdots \\ g(\mathbf{x}_m) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\mathbf{x}_1) \\ \vdots \\ m(\mathbf{x}_m) \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & \ldots & k(\mathbf{x}_1,\mathbf{x}_m) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_m,\mathbf{x}_1) & \ldots & k(\mathbf{x}_m,\mathbf{x}_m) \end{bmatrix}\right). \quad (2)$$

We denote this succinctly via: $g(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot))$. The above properties imply that, for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$,

$$m(\mathbf{x}) = \mathbb{E}[g(\mathbf{x})], \quad (3a)$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(g(\mathbf{x}) - m(\mathbf{x}))(g(\mathbf{x}') - m(\mathbf{x}'))]. \quad (3b)$$

4 LEARNING NEURAL POINT PROCESSES

We introduce the problem of sparse pixel-wise regression, defining first MSE estimation in this context. We then introduce NPPs and apply them to this setting.

4.1 Problem Formulation

For the purposes of our analysis, a *point* is an element in $\mathbb{R}^2$, and a *point set* $A \subset \mathbb{R}^2$ is a finite collection of points.[1] As illustrated in Figure 1a, we are given images and a set of labeled points (i.e., pixels) in them: our goal is to regress the label *at any given point* from the corresponding source image. Images could correspond to, e.g., a satellite image of a specific location, or any other 2d representation of a geographic area (e.g., a temperature or traffic map) and points could correspond to values of measurements collected at different coordinates (such as, e.g., air pollution, humidity, cellphone signal quality, etc.) by geographically dispersed sensors. Formally, we consider a dataset of n images $\mathbf{x}_i \in \mathbb{R}^{d_1 \times d_2}$, where $i \in [n]$.[2] Each image $\mathbf{x}_i$ is associated with a *labeled point set* $(\mathbf{P}_i, \mathbf{y}_i)$, where

$$\mathbf{P}_i = [\mathbf{p}_i^{(1)}, \ldots, \mathbf{p}_i^{(L_i)}] \text{ and } \mathbf{y}_i = [y_i^{(1)}, \ldots, y_i^{(L_i)}]. \quad (4)$$

Here, $\mathbf{p}_i^{(\ell)} \in [d_1] \times [d_2]$, $\ell \in [L_i]$, are a set of points (i.e., image pixels), and $y_i^{(\ell)} \in \mathbb{R}$, $\ell \in [L_i]$, are the corresponding labels. Note that each label is indexed by the point (i.e., pixel) at which it is measured. We denote the entire dataset by $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{P}_i, \mathbf{y}_i)\}_{i=1}^n$.

A straightforward way to regress labeled point values from an image is by fitting a neural network via, e.g., a mean squared error (MSE) loss on the labeled points. Consider a neural network $f : \mathbb{R}^{d_1 \times d_2} \times \mathbb{R}^{m_\theta} \to \mathbb{R}^{d_1 \times d_2}$, parameterized by $\theta \in \mathbb{R}^{m_\theta}$, that receives as input an image $\mathbf{x}$ and produces the values where $f(\mathbf{x}; \theta) \in \mathbb{R}^{d_1 \times d_2}$ *at all possible points* $\mathbf{p} \in [d_1] \times [d_2]$. Parameters θ can thus be learned by minimizing:

$$\begin{aligned} \mathcal{L}_{\texttt{MSE}}(\theta, \mathcal{D}) &= \textstyle\sum_{i=1}^n \sum_{\ell=1}^{L_i} \left(f_{p_i^{(\ell)}}(\mathbf{x}_i; \theta) - y_i^{(\ell)}\right)^2 \\ &= \textstyle\sum_{i=1}^n \|f_{\mathbf{P}_i}(\mathbf{x}_i; \theta) - \mathbf{y}_i\|_2^2, \end{aligned} \quad (5)$$

where by $f_{\mathbf{P}_i}(\cdot; \cdot)$ we denote the restriction/projection of the output of the neural network to the coordinates $\mathbf{P}_i$. However, Eq. (5) *does not consider potential spatial correlations between nearby points*. Such correlations may indeed manifest if, e.g., measurements occur at nearby locations. As a result, the neural network trained thusly might have a poor bias/variance trade-off, failing to exploit correlations due to proximity. An additional drawback is that,

[1] This can be thought of an instantiation of a *point process* (Daley et al., 2003), which is a random map from every subset $S \subset \mathbb{R}^d$ to $\mathbb{N}$ (counting the number of points in this set).

[2] We use the notation $[k] \equiv \{1, \ldots, k\}$ for $k \in \mathbb{N}$.

Table 1: Common parametric kernels. Inputs $\mathbf{x}$, $\mathbf{y}$ are assumed to be in $\mathbb{R}^d$. Note that the RBF/Gaussian kernels are a special case of the specral mixture kernel.

Name	**Kernel Function**	**Parameters** ζ	**Dimension** m_ζ
Linear (Seeger, 2004)	$k(\mathbf{x},\mathbf{y}) = \mathbf{x}^\top AA^\top \mathbf{y}$	embedding matrix $A \in \mathbb{R}^{d\times d'}$	$d \times d'$
Polynomial (**?**)	$k(\mathbf{x},\mathbf{y}) = (\mathbf{x}^\top AA^\top \mathbf{y} + c)^d$	embedding matrix $A \in \mathbb{R}^{d\times d'}$, coefficient $c \in \mathbb{R}$, degree $d \in \mathbb{N}$	$d \times d' + 2$
RBF (Seeger, 2004)	$k(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\ell^2}\right)$	length scale ℓ	1
Gaussian (**?**)	$k(\mathbf{x},\mathbf{y}) = \frac{\lvert\Sigma\rvert^{\frac{1}{2}}}{(2\pi)^{d/2}} \exp\left(-\frac{(\mathbf{x}-\mathbf{y})^\top\Sigma(\mathbf{x}-\mathbf{y})}{2}\right)$	bandwidth matrix $\Sigma \in \mathbb{S}^{d\times d}$	$\frac{d\times(d+1)}{2}$
Rational Quadratic (Seeger, 2004)	$k(\mathbf{x},\mathbf{y}) = \left(1 + \frac{\|A\mathbf{x}-A\mathbf{y}\|^2}{2\alpha\ell^2}\right)^{-\alpha}$	shape $\alpha > 0$, length scale $\ell > 0$, embedding matrix $A \in \mathbb{R}^{d\times d'}$	$d \times d' + 2$
Spectral mixture (Wilson et al., 2016)	$\sum_{q=1}^{Q} a_q \frac{\lvert\Sigma_q\rvert^{\frac{1}{2}}}{(2\pi)^{d/2}} e^{-\frac{1}{2}\|\Sigma_q^{1/2}(\mathbf{x}-\mathbf{y})\|^2} \cos\langle \mathbf{x}-\mathbf{y}, 2\pi\mu_q\rangle$	mixture weights $a_q \in [0,1]$, diagonal covariance matrices $\Sigma_q \in \mathbb{S}^{d\times d}$, frequency vectors $\mu_q \in \mathbb{R}^d$	$Q \times (2d+1)$

at test time, the neural network can only make use of the input $\mathbf{x}$. If, however, we are given both $\mathbf{x}$, and a *few labeled points*, we would like to exploit that information during predictions; the neural network trained via Eq. (5) ignores information provided by such labeled points.

4.2 Neural Point Processes

We address these two issues with our proposed *neural point processes*, as illustrated in Figure 1b. Intuitively, we treat the labels at every pixel as a Gaussian Process, whose means are regressed by the neural network, and whose covariance is determined by a (known) kernel. An immediate consequence is that point labels are now correlated; this not only leads to a different estimation process than Eq. (5) but also allows incorporating any available point labels at inference time in a principled fashion. Consider again a sample image $\mathbf{x}_i$ with a corresponding labeled point set $(\mathbf{P}_i, \mathbf{y}_i)$ given by Eq. (4). We assume that, for point $\mathbf{p}_i \in [d_1]\times[d_2]$, the corresponding label $y_i \in \mathbb{R}$ is given by:

$$y_i = g_i(\mathbf{p}_i) + \varepsilon, \tag{6}$$

where $\varepsilon \sim N(0, \sigma_0^2)$ is i.i.d. noise, and $g_i(\cdot)$ is a GP:

$$g_i(\cdot) \sim \mathcal{GP}(m_i(\cdot), k_i(\cdot,\cdot)), \tag{7}$$

whose mean function $m_i : \mathbb{R}^2 \to \mathbb{R}$ and kernel function $k_i(\cdot,\cdot)$ are given by

$$m_i(\mathbf{p}) = f_{\mathbf{p}}(\mathbf{x}_i;\theta) \in \mathbb{R}, \quad k_i(\mathbf{p},\mathbf{p}') = k(\mathbf{p},\mathbf{p}';\zeta_i), \quad \text{for } \mathbf{p},\mathbf{p}' \in [d_1]\times[d_2]. \tag{8}$$

Thus, means are again the projections of the output of a neural network $f : \mathbb{R}^{d_1\times d_2} \times \mathbb{R}^{m_\theta} \to \mathbb{R}^{d_1\times d_2}$ parameterized by θ, and $k(\cdot,\cdot)$ is a parametric positive semidefinite kernel parameterized by $\zeta_i \in \mathbb{R}^{m_\zeta}$ such as, e.g., the RBF kernel. Moreover, the kernel parameter may or may not depend on the image $\mathbf{x}_i$ as we have three regimes to compute ζ_i. Additional examples of parametric kernel candidates and their parameters are provided in Table 1. Under this assumption, we can estimate model parameters from a dataset $\mathcal{D}$ through maximum likelihood estimation (MLE):

Theorem 1. *If (a) labels are generated according to Eq. (6), with i.i.d. noise σ_0, and (b) GPs g_i given by Eq. (7) are also independent across images, then the maximum likelihood estimate of θ can be obtained by minimizing*

$$\mathcal{L}_{\texttt{NPP}}(\theta,\zeta;\mathcal{D}) = \frac{1}{2}\sum_{i=1}^{n}\Big[\log|k(\mathbf{P}_i,\mathbf{P}_i;\zeta) + 2\sigma_0^2\mathbf{I}| + (\mathbf{y}_i - f_{\mathbf{P}_i}(\mathbf{x}_i;\theta))^\top (k(\mathbf{P}_i,\mathbf{P}_i;\zeta)+\sigma_0^2\mathbf{I})^{-1}(\mathbf{y}_i - f_{\mathbf{P}_i}(\mathbf{x}_i;\theta))\Big]. \tag{9}$$

where $k(\mathbf{P}_i,\mathbf{P}_i;\zeta) \in \mathbb{R}^{L_i\times L_i}$ is the PSD covariance matrix of GP g_i over points $\mathbf{P}_i$.

The proof can be found in Appendix A in the supplement. We refer to Eq. (9) as the *NPP loss*. We observe that, compared to standard MSE (Eq. (5)), which treats all residual error terms equally in the Euclidean space, the error terms are measured in the *squared Mahalanobis distance*, as imposed by the kernel/covariance matrix $k(\mathbf{P}_i,\mathbf{P}_i;\zeta)$. Intuitively, this enforces correlations between the values of points that are proximal. Parameter σ_0^2 that corresponds to label noise can either be set to a small value, to ensure invertibility or can be treated as a hyperparameter that can be determined by measuring performance on a validation set.

We note there are several approaches to model the kernel parameters ζ: (a) **Static Kernel**: the parameters be treated as hyperparameters, tuned on a validation set along with σ_0. (b) **Learnable Kernel**: alternatively, as the NPP loss is differentiable w.r.t. ζ, these can be estimated via gradient descent (GD) along with θ. Note that the second term in Eq. (9) acts as an anisotropic regularization term in this case. (c) **Context-aware Kernel**: finally, the kernel parameters can be regressed from the input as well. This is particularly useful in scenarios where each input image may have distinct underlying spatial structures or correlations, such as satellite images of different geographic regions.

The inclusion of image-dependent kernel prediction thus equips the model with the flexibility to adapt its learned representations to diverse data distributions, making it robust in scenarios with heterogeneous input data. In this case, each sample is endowed with a different kernel k_i parameterized by ζ_i, where $\zeta_i = f'(\mathbf{x}_i; \theta')$ for some neural network $f : \mathbb{R}^{d_1 \times d_2} \times \mathbb{R}^{m_{\theta'}} \to \mathbb{R}^{m_\zeta}$ parameterized by $\theta' \in \mathbb{R}^{m_{\theta'}}$. Regression would then amount to minimizing Eq. (9) w.r.t. both θ and θ', again via GD.

We emphasize that our approach scales gracefully to high-dimensional inputs (e.g., high-resolution images) because the GP computation depends primarily on the number of labeled points L, rather than the total number of pixels; this is true also about the partial label revelation setting discussed below. Moreover, the sparse-label setting we study ($L \ll d_1 \times d_2$) is relevant to many distributed sensing applications. We discuss computational complexity issues in detail in Appendix B.

4.3 Using Labels Revealed at Test/Inference Time

Using a probabilistic GP as a model allows us to exploit the presence of labeled points that may be partially revealed at test/inference time. Consider the following *partial label revelation* setting, in which a few labels are revealed at certain points in a test image. We denote by $\mathbf{P}^\dagger$ and $\mathbf{P}^*$ be the points with revealed and missing labels in the test image, respectively. The joint distribution of corresponding labels $\mathbf{Y}^\dagger$ and $\mathbf{Y}^*$ is:

$$\begin{bmatrix}\mathbf{Y}^\dagger \\ \mathbf{Y}^*\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}m(\mathbf{P}^\dagger) \\ m(\mathbf{P}^*)\end{bmatrix}, \begin{bmatrix}k(\mathbf{P}^\dagger, \mathbf{P}^\dagger) + \sigma_0^2\mathbf{I} & k(\mathbf{P}^\dagger, \mathbf{P}^*) \\ k(\mathbf{P}^*, \mathbf{P}^\dagger) & k(\mathbf{P}^*, \mathbf{P}^*)\end{bmatrix}\right).$$

Conditioned on observations $\mathbf{Y}^\dagger$, the posterior distribution of $\mathbf{Y}^*$ is Gaussian with the following mean and covariance:

$$\mathbb{E}[\mathbf{Y}^*|\mathbf{Y}^\dagger] = m(\mathbf{P}^*) + k(\mathbf{P}^*, \mathbf{P}^\dagger)\mathbf{K}_{\dagger,\dagger}^{-1}(\mathbf{y}^\dagger - m(\mathbf{P}^\dagger)), \quad (10a)$$

$$\mathrm{cov}(\mathbf{Y}^*) = k(\mathbf{P}^*, \mathbf{P}^*) - k(\mathbf{P}^*, \mathbf{P}^\dagger)\mathbf{K}_{\dagger,\dagger}^{-1}k(\mathbf{P}^\dagger, \mathbf{P}^*), \quad (10b)$$

where $\mathbf{K}_{\dagger,\dagger} \equiv k(\mathbf{P}^\dagger, \mathbf{P}^\dagger) + \sigma_0^2\mathbf{I}$. Hence, in the presence of a labeled point set $(\mathbf{P}^\dagger, \mathbf{Y}^\dagger)$, we can combine the estimate produced by the neural network, producing the means via Eq. (8). Thus, along with the observed values, by exploiting the available labels, Gaussian Processes can produce not only a MAP estimate of $\mathbf{Y}^*$ via the expectation in Eq. (10), but also the full Bayesian posterior, taking the covariance into account as well.

5 EXPERIMENTS

We compare our proposed NPPs against competitors over four different datasets (two synthetic and two real-world datasets). We make our code publicly available. [3]

[3] https://github.com/neu-spiral/Neural-Point-Processes

Table 2: Dataset configurations. The column n represents the number of images, d_1 and d_2 are the height and width of an image. L_{sparse} and L_{dense} are the number of labeled pins on an image in the sparse and dense setting. We used five to ten times more labeled points for COWC compared to Rotterdam due to the larger image sizes.

Dataset	Point Distr.	n	$d_1 \times d_2$	L_{sparse}	L_{dense}
Synthetic Heatmaps	grid	1000	28×28	9	100
	random	1000	28×28	10	100
PMNIST	grid	1000	28×28	9	100
	random	1000	28×28	10	100
Rotterdam	grid	1000	100×100	16	121
	random	1000	100×100	10	100
COWC	grid	1000	200×200	81	529
	random	1000	200×200	100	500

We summarize our experimental setup here, and provide additional details in Appendix C. Specifically, we describe our dataset generation procedures, detailed network architectures, hyperparameter tuning strategies, hardware and runtime configurations, and the formulas for all metrics used. We also provide additional experiments analyzing the sensitivity to kernel parameters and label sparsity.

5.1 Datasets

Our four datasets are summarized in Table 2. *Synthetic Heatmaps* contains synthetic "elevation/heat maps" as inputs, and labels on points are determined by a GP parameterized by these inputs. *Point MNIST (PMNIST)* is MNIST (LeCun et al., 1998) augmented with points dispersed over the image, with labels measuring the pixel density in a region around these points. *Rotterdam* contains satellite images from the city of Rotterdam; point labels indicate the density of buildings in the area surrounding each point. *COWC* contains satellite images of cars with labels measuring the number of cars within the region of points. Each dataset is studied under two point distributions, *grid* and *random*, illustrated in Figure 4 in Appendix C. Both are studied in a sparse (L_{sparse}) and dense (L_{dense}) setting, corresponding to different numbers of labels L, as described in Table 2. All datasets are described in detail in Appendix C.

5.2 Algorithms

We implement five different algorithms for regressing point labels: *Plain*, i.e., training through the MSE loss in Eq. (5), *Neural Process (NP)*, the neural processes of Garnelo et al. (2018b), *specifically* for the partial label scenario, *Convolutional Neural Processes (ConvNP)*, introduced by Gordon et al. (2020), which explicitly embed spatial structure, and compare them to two variants of our approach, *NPP* and *NPP-GP*, corresponding to Eqs. (9) and (10), respectively. In short, both use the same trained model, but in NPP-GP, we allow the use of partial labels revealed at test time (see Sections 4.3 and 5.4.3). Note that our competitor NP can

Table 3: Experiment results for methods that use labels (ConvNP, NP, NPP-GP) and those that do not (Plain, NPP) on four datasets. The best metrics are marked as **Bold**. The table compares MSE and R^2 metrics in the regression setting with no label given and the label revelation setting where we incorporate labels in inference time. We observe that across all datasets and settings, NPP and NPP-GP remain the best in their settings (no label or partial label revelation) respectively for most cases. For cases where the best performing methods are approximately tied (equal up to the 3-rd digit), we repeat experiments with three runs from different seeds and report standard deviations in Appendix D.

	Datasets		Rotterdam				COWC				Partial label revelation
	Point pattern		Grid		Random		Grid		Random		
	Metric		MSE ↓	R^2 ↑	MSE ↓	R^2 ↑	MSE ↓	R^2 ↑	MSE ↓	R^2 ↑	
Real-world	Sparse	Plain	1.79	−0.058	1.14	0.297	17.99	0.060	17.7	0.078	✗
		NPP (ours)	**1.76**	**−0.046**	**0.834**	**0.437**	**4.89**	**0.767**	**7.77**	**0.594**	
		NPP-GP (ours)	1.76	−0.046	**0.833**	**0.437**	**4.86**	**0.769**	7.65	0.600	✓
		NP	**1.44**	**0.047**	1.64	−0.027	14.0	0.263	15.9	0.171	
		ConvNP	0.725	−4.44	1.12	−8.10	16.6	−0.529	**5.68**	**0.768**	
	Dense	Plain	1.55	0.100	0.486	0.678	10.9	0.432	14.51	0.208	✗
		NPP (ours)	**1.25**	**0.286**	**0.453**	**0.700**	**5.67**	**0.704**	**5.31**	**0.710**	
		NPP-GP (ours)	1.25	**0.287**	**0.443**	**0.707**	**5.60**	**0.706**	5.02	0.726	✓
		NP	**1.20**	0.196	1.199	0.197	17.9	0.066	13.2	0.272	
		ConvNP	0.581	−7.27	0.344	0.379	19.7	−3.12	**3.47**	**0.922**	
	Datasets		PMNIST				Synthetic Heatmaps				Partial label revelation
	Point pattern		Grid		Random		Grid		Random		
	Metric		MSE ↓	R^2 ↑	MSE ↓	R^2 ↑	MSE ↓	R^2 ↑	MSE ↓	R^2 ↑	
Synthetic	Sparse	Plain	**67.5**	**0.087**	0.456	0.992	1298	−15.0	14192	−173	✗
		NPP (ours)	72.0	0.026	**0.451**	**0.993**	**94.6**	**−0.164**	**27.1**	**0.666**	
		NPP-GP (ours)	**56.3**	**0.239**	**0.451**	**0.993**	**75.2**	**0.075**	27.5	0.661	✓
		NP	78.2	−0.072	44.8	0.404	111	−0.381	104	−0.587	
		ConvNP	63.2	−19.4	29.7	0.293	79.8	−5.28	**12.1**	**0.761**	
	Dense	Plain	0.467	0.994	0.181	**0.998**	104	−0.28	27.1	0.666	✗
		NPP (ours)	**0.307**	**0.996**	**0.128**	**0.998**	**94.6**	**−0.164**	**26.9**	**0.669**	
		NPP-GP (ours)	**0.300**	**0.996**	**0.120**	**0.999**	74.71	0.081	26.9	0.669	✓
		NP	64.6	0.121	27.2	0.627	59.2	0.269	43.2	0.467	
		ConvNP	12.7	0.765	10.0	0.861	**24.9**	**0.246**	**19.1**	**0.727**	

only predict labels in this partial label revelation setting.

A summary of optimal hyperparameters/design choices is given in Table 8 in Appendix C of the supplement. Every hyperparameter selection is based on the MSE loss over the validation set (see Section 5.3). We explore the following common hyperparameters for all four methods. We set the batch size to 32 and explore for the optimal initial learning rate within $\{0.0001, 0.001, 0.01\}$. We use the Adam (Kingma and Ba, 2015) optimizer and decrease the learning rate by $0.1\times$ until 0.0001 if the validation loss does not decrease for 5 epochs.

We also explore architectural choices for the first three methods. We use two different architectures as neural networks f: a simple autoencoder (AE) Kramer (1991) and an autoencoder combined with a DDPM feature extractor Bonito et al. (2023). We use 3 variants of each approach (in terms of layers) as candidate networks, and treat the optimal architecture (out of 6 in total) as a hyperparameter to be tuned over the validation set. Additional details of these architectures are provided in Appendix C. For NP, we use the latent embedding architecture originally proposed by Garnelo et al. (2018a). Finally, for ConvNP we follow the corresponding implementation from the public Neural Processes Family package (NPF) (Dubois et al., 2020).

W.r.t. NPP (and NPP-GP), we explore two kernels: an RBF Kernel and an SM kernel (see Table 1). As mentioned in Section 4.2, we have three approaches to model the kernel parameters: (a) a static kernel, whose parameters ζ are learned via validation (b) a learnable kernel, whose parameters ζ are treated as parameters learned via loss minimization, and (c) a context-aware Kernel, whose parameters ζ are regressed from features $\mathbf{x}_i$. For approach (a) we explore the length-scale parameter $\ell \in \{0.01, 0.1, 0.2, 0.5, 1, 2, 5\}$ for RBF; the optimal value via validation is used as a starting point for approach (b). For SM, which contains a large number of parameters, we only explore the number of mixtures $Q \in \{1, \ldots, 5\}$ and learn remain parameters via (b); in particular, we initialize the kernel with weights $\mu_q = \frac{1}{Q}$ and an identity matrix $\Sigma_q = \mathbf{I}$. We assume the noise is of a magnitude of $\sigma_0 = 10^{-5}$. Finally, to implement the last last approach (c), we add a fully connected layer to the the encoder part of the AE to regress the parameters for the kernel. We treat these approaches as hyperparameter choices, with the optimal kernel choice (static, learned, or context-aware) determined by MSE on the validation set. Note that the same optimal values are also used in NPP-GP.

5.3 Experiment Setup

The datasets are split into training (70%), validation (10%), and test (20%) sets. Labels for each dataset are generated in the training and validation set in two manners: in a regular grid (*grid*), and in a random fashion (*random*), at points selected uniformly at random (u.a.r). To ensure a fair comparison between different training methods (grid vs. random), labels in the test set are always of the *random* type, i.e., at u.a.r. selected pixels. We select the best-performing model hyperparameters based on the MSE on validation set out of five distinct runs with different seeds (selected from $0, 1, 2, 3, 4$). In the test set, we hide 50% of the available point labels per image: to ensure a consistent comparison across methods, this 50% of hidden labels is always used as the target/ground truth prediction when we report test metrics. Methods *Plain* and *NPP* do not make any use of the remaining 50% of labels at test time. In contrast *NPP-GP* and *NP* observe these labels (i.e., operate in the partial label revelation setting). For the latter two methods, we also considered a *gradual revelation* scenario: In particular, we reveal 0%, 25%, 50%, 75%, and 100% of the non-hidden labeled (i.e., 50%) points.

We evaluate performance on the (hidden) test set labels w.r.t. two metrics: (a) *MSE*, i.e., the mean-square-error (MSE) of predictions on hidden labels on the image, averaged across images on the test set, (b) R^2, i.e., the R^2 value of predictions against the average label value in the test set. Formulas for these metrics are provided in Appendix C.

5.4 Results

For both synthetic and real-world datasets, we first compare the proposed method against the MSE without labels at test time to verify if its smoothness prior is useful in learning the regression model. Next, we evaluate the performance of NPP against its Gaussian Process-refined version (NPP-GP) and NPF, including the NP and ConvNP, to examine their ability to leverage available labels during testing. Additionally, we explore the impact of different label densities and distribution patterns (e.g., grid and random). The optimal hyperparameters for all experiments are detailed in Appendix C.

5.4.1 Real-world Datasets

We show the results of real-world datasets including Rotterdam and COWC in the top half of the Table 3. When only images are available (no partial label revelation at inference time), NPP consistently outperforms the Plain MSE across all eight settings, demonstrating the effectiveness of incorporating smoothness priors from the GP component.

On the other hand, in the partial label revelation setting, NPP-GP ranks as the top method in five out of eight scenarios. One exception is the sparse *grid* setting on Rotterdam, where only 16 labeled points are provided on 100×100 images; notably, all methods perform very poorly in this setting, including the best-performing method (NP, with a low R^2 value of 0.047). Interestingly, both NPP and NPP-GP significantly outperform NP ($R^2 = 0.437$ vs. -0.027) in the sparse *random* setting of Rotterdam, even with fewer labels than the grid setting (10 points vs. 16). Meanwhile, NPP-GP also achieves the best performance in the sparse grid setting for COWC, where 81 labeled points are provided per 200×200 image. This demonstrates that NPP-GP (and NPP) can still learn effectively from sparse grid distributions as long as the number of points exceeds a certain threshold. Specifically, the models can learn from as few as 0.2% points labeled (81 points compared to 40,000 per image), achieving strong performance ($R^2 = 0.769$). However, using the standard MSE loss in the plain setting results in distorted prediction, with an R^2 of just 0.06. Additional visual results are presented in Section 5.4.4 and Appendix D.

The other two exceptions are the *random* settings of COWC, where we observe that ConvNP outperforms the other methods by a noticeable margin. As ConvNP does not perform as well in Rotterdam, we conclude that ConvNP requires more labels with a dense distribution of labeled objects (i.e., the car distribution in COWC is much more dense than the building distribution in Rotterdam). Although ConvNP shows superior performance in Table 3, it is worth noticing that NPP outperforms trained ConvNP when 125 labels are revealed (see Appendix D).

Lastly, we observe that for all cases, the NPP-GP improves the performance over NPP, using the kernel identified via validation MSE. Thus, we conclude that the GP correction technique is effective, especially when the best-performing method learns or predicts the appropriate parameters for the kernel during training. We include the hyperparameter table in Appendix C, and it shows that multiple best-performing models learn or predict the kernel parameters.

5.4.2 Synthetic Datasets

The bottom of Table 3 provides a performance comparison w.r.t. both MSE and R^2 over synthetic datasets (Synthetic Heatmaps and PMNIST). Similar to real-world datasets, NPP again outperforms the Plain MSE baseline in seven out of eight cases, highlighting the value of regularization during learning from sparse labels.

In the partial label revelation setting, NPP-GP demonstrates superior performance over the other methods in all four cases on PMNIST. Nevertheless, NPP-GP underperforms ConvNP on the Synthetic Heatmap for the random point distribution setting, as well as for the dense grid point distribution. This is expected as the Synthetic Heatmap is generated by a GP (see Appendix C; it comprises GP-

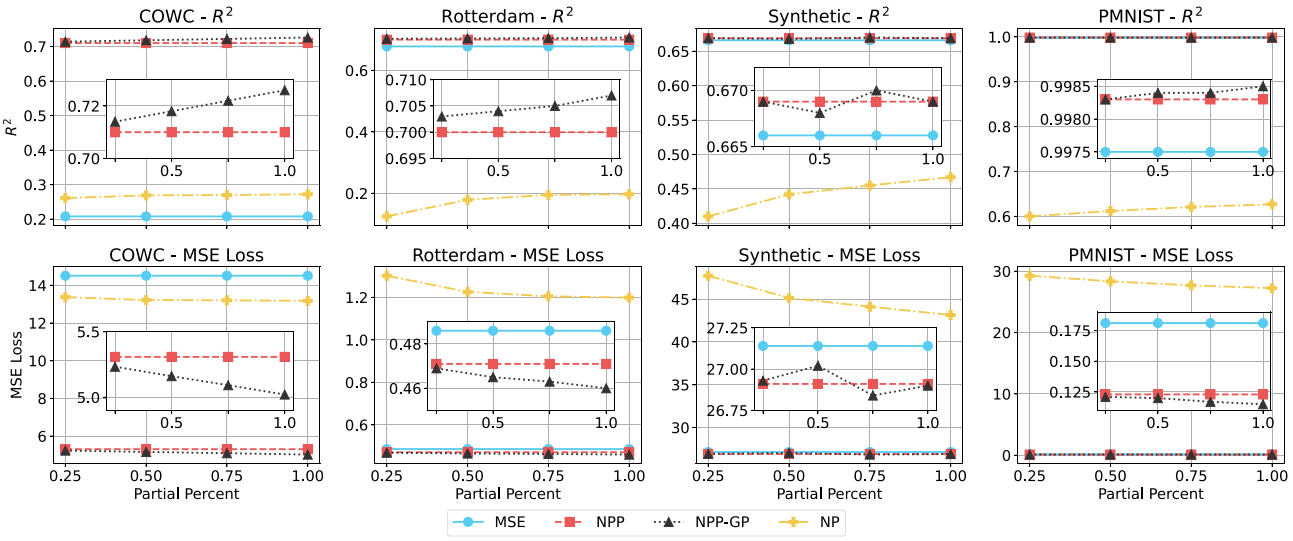


Figure 2: Performance with gradual label revelation on all datasets in the dense random regime. The x-axis and y-axis denote the percent of labels revealed and performance metrics MSE (↓) and R^2 (↑). At the inset, we zoom into the gap between NPP-GP and NPP. We observe that both NPP-GP and NP constantly improve their performance when more labels are revealed, but NPP-GP significantly outperforms NP.

generated smooth waves), which perfectly fits the assumption of NP and ConvNP. In contrast, ConvNP struggles on PMNIST. This is due to sharp edges present in digits, as both ConvNP and NP cannot access the original images. While ConvNP outperforms other methods, NPP achieves comparable RMSE and R^2 even without access to partial labels, by directly utilizing the raw image input.

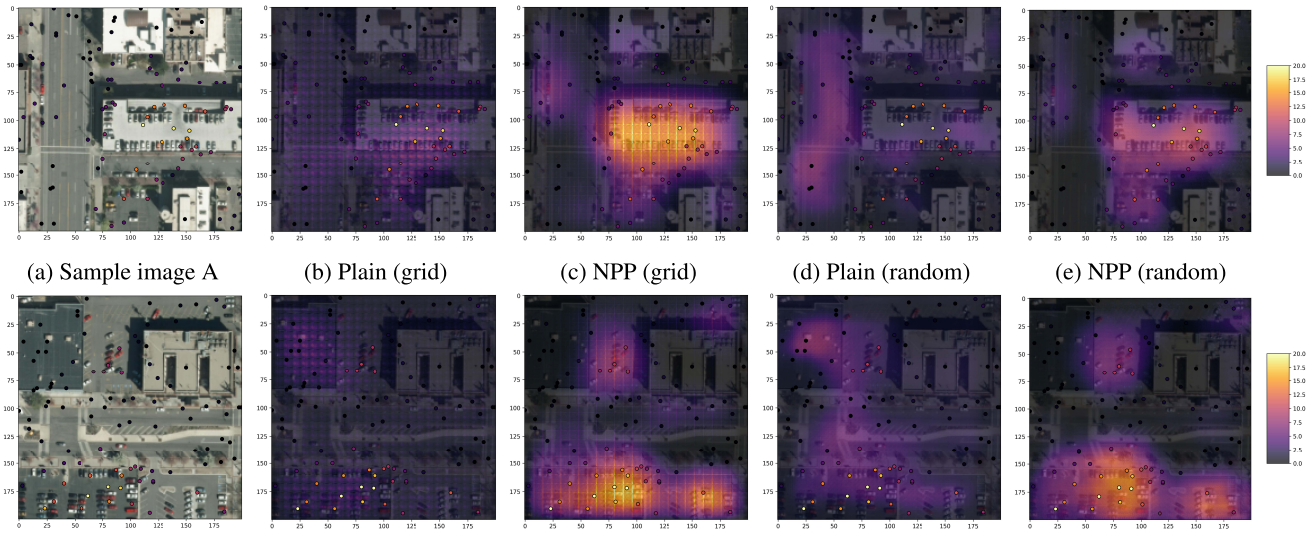
Lastly, we observe all methods perform poorly in the *grid* setting of Synthetic Heatmaps. This can be attributed to the periodic structure of the Synthetic Heatmaps dataset (defined via sin/cos functions, as discussed in Appendix C), which makes grid-distributed point learning more difficult. This issue appears to be both dataset and distribution-specific, as both NPP and NPP-GP perform well across different point distributions, including sparse grid, in other datasets. Overall, NPP and NPP-GP exhibit stronger performance across the majority of settings, making them suitable choices for most scenarios.

5.4.3 Exploiting Revealed Labels

To highlight the effectiveness of the GP component based on the kernel from NPP, we compare the performance of NPP-GP against other methods under a gradual label revelation setting. As shown in Figure 2, we observe that for both metrics, the performance of NPP-GP generally improves with more labels revealed. This trend holds across different scenarios, including the dense random, dense grid, and sparse grid settings. However, in the sparse random case, the results are more mixed, with some experiments showing a decline in performance. This may be because randomly distant points are less correlated, limiting the ability to enforce smoothness constraints effectively, for certain images. In conclusion, NPP-GP successfully leverages the kernel inferred from the NPP optimization–whether through validation set tuning for the Static kernel, gradient-based optimization for the Learnable kernel, or a separately trained layer to regress the Context-aware kernel–allowing the GP component to refine the predictions made by the neural network. This synergy between NPP and GP enables NPP-GP to adapt to various label distributions, particularly when label density is sufficient to capture meaningful spatial relationships. This is also observed in other setups, as shown in Appendix D.

5.4.4 Visualizations

In Figure 3, we illustrate inference via Plain and NPP on two sample two sample images from the test set of the COWC dataset, where labels indicate car densities. Colored points represent the 100 labels, whose values are encoded via their color. Using the same color scheme, we superimpose the prediction of NPP and Plain (trained in the sparse setting) over the corresponding input images. We observe that, despite using the same network structure as NPP, the plain methods, trained under both sparse grid and sparse random settings, fail to detect the car-dense areas: they falsely relate roads to cars, and generally ignore cars in parking lots. Moreover, predictions are not as smooth as under NPP, due to the lack of smoothness constraints. In contrast, NPP detects a high car-density in the parking lot, by the side of the roads, or next to a building, and is also correct quantitatively: the colors (values) of the prediction match those of the test set labels.



(a) Sample image A (b) Plain (grid) (c) NPP (grid) (d) Plain (random) (e) NPP (random)

(f) Sample image B (g) Plain (grid) (h) NPP (grid) (i) Plain (random) (j) NPP (random)

Figure 3: Visual comparison of Plain and NPP methods trained under sparse grid/random setup on two test images of the COWC dataset, where labels indicate car densities. The scattered colored points indicate the 100 labeled pixels with corresponding ground truth values. We predict every pixel label with Plain or NPP in the sparse setup, using either grid or random pixels during training. Then, we superimpose the semi-transparent prediction over each test image. Predictions and labels share the same color map. Thus, color proximity indicates better predictions. We observe that, in contrast to Plain, NPP correctly identifies regions of high car density.

6 CONCLUSION

We propose a novel method for sparse pixel-wise image regression tasks called *Neural Point Processes*. We leverage spatial correlations on 2D images by combining Gaussian processes with deep neural networks. This enhances the correlation between point labels and allows incorporating labels made available at inference time. We show that maximum likelihood estimation corresponds to an MSE-like loss expressed w.r.t. the Mahalanobis distance, and regularized through the log-determinant of the GP kernel. We show experimentally that using the NPP loss results in improved MSE and R^2 scores over the standard MSE loss in both real and synthetic datasets.

For future directions, we propose applying NPPs to broader domains such as air pollution, traffic flow prediction, and other settings that involve geo-located sensor measurements. Additionally, extending NPPs to 2D/3D segmentation or other tasks with sparse labeling (e.g., medical imaging) is an exciting prospect. While applying our approach to 3D regression is straightforward, full segmentation introduces extra challenges (pixel-wise classification). We leave this for future work.

Acknowledgments

Research was sponsored by the United States Army Core of Engineers (USACE) Engineer Research and Development Center (ERDC) Geospatial Research Laboratory (GRL) and was accomplished under Cooperative Agreement Federal Award Identification Number (FAIN) W9132V-22-2-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of USACE EDRC GRL or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

Ashman, M., Diaconu, C., Weller, A., and Turner, R. E. (2024). In-context in-context learning with transformer neural processes.

Bandara, W. G. C., Nair, N. G., and Patel, V. M. (2022). DDPM-CD: Denoising Diffusion Probabilistic Models as Feature Extractors for Change Detection. *arXiv preprint arXiv:2206.11892*.

Baranchuk, D., Voynov, A., Rubachev, I., Khrulkov, V., and Babenko, A. (2022). Label-Efficient Semantic Segmentation with Diffusion Models. In *ICLR*.

Baumann, A., Roßberg, T., and Schmitt, M. (2023). Probabilistic MIMO U-Net: Efficient and Accurate Uncertainty Estimation for Pixel-wise Regression. In *ICCVW*.

Bonito, L., Requeima, J., Shysheya, A., and Turner, R. E. (2023). Diffusion-Augmented Neural Processes. *NeurIPS Workshop on Diffusion Models*.

Bruinsma, W. P., Requeima, J., Foong, A. Y. K., Gordon, J., and Turner, R. E. (2021). The gaussian neural process.

Bui, T., Hernández-Lobato, D., Hernandez-Lobato, J., Li, Y., and Turner, R. (2016). Deep Gaussian Processes for Regression using Approximate Expectation Propagation. In *ICML*.

Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. (2014). Manifold gaussian processes for regression. *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3338–3345.

Chen, H. and Shi, Z. (2020). A Spatial-Temporal Attention-Based Method and a New Dataset for Remote Sensing Image Change Detection. *Remote Sens.*

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807.

Cutajar, K., Bonilla, E. V., Michiardi, P., and Filippone, M. (2017). Random Feature Expansions for Deep Gaussian Processes. In *ICML*.

Daley, D. J., Vere-Jones, D., et al. (2003). *An Introduction to the Theory of Point Processes: Volume I: Elementary Theory and Methods*. Springer.

Damianou, A. and Lawrence, N. D. (2013). Deep Gaussian Processes. In *AISTATS*.

Do, C. B. and Lee, H. (2020). Gaussian Processes. Lecture Notes for CS229.

Dubois, Y., Gordon, J., and Foong, A. Y. (2020). Neural process family. `http://yanndubs.github.io/Neural-Process-Family/`.

Dutordoir, V., Hensman, J., van der Wilk, M., Ek, C. H., Ghahramani, Z., and Durrande, N. (2021). Deep Neural Networks as Point Estimates for Deep Gaussian Processes. *NeurIPS*.

Elfwing, S., Uchibe, E., and Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*.

Foong, A. Y. K., Bruinsma, W. P., Gordon, J., Dubois, Y., Requeima, J., and Turner, R. E. (2020). Meta-learning stationary stochastic process prediction with convolutional neural processes.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. (2018a). Conditional Neural Processes. In *ICML*.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018b). Neural Processes. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*.

Giannone, G. and Ahmed, F. (2023). Diffusing the Optimal Topology: A Generative Optimization Approach. In *IDETC-CIE*.

Gordon, J., Bruinsma, W. P., Foong, A. Y. K., Requeima, J., Dubois, Y., and Turner, R. E. (2020). Convolutional conditional neural processes.

Hinton, G. E. and Salakhutdinov, R. R. (2007). Using Deep Belief Nets to Learn Covariance Kernels for Gaussian Processes. *NeurIPS*.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising Diffusion Probabilistic Models. *NeurIPS*.

Huang, W., Zhao, D., Sun, F., Liu, H., and Chang, E. (2015). Scalable Gaussian Process Regression Using Deep Neural Networks. In *IJCAI*.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. (2018). Attentive Neural Processes. In *ICLR*.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *ICLR*.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*.

LeCun, Y., Cortes, C., and Burges, C. J. (1998). MNIST handwritten digit database.

Liu, H., Liu, F., Fan, X., and Huang, D. (2021a). Polarized self-attention: Towards high-quality pixel-wise regression. *arXiv preprint arXiv:2107.00782*.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021b). Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows. In *ICCV*.

Louizos, C., Shi, X., Schutte, K., and Welling, M. (2019). The Functional Neural Process. *NeurIPS*.

Mundhenk, T. N., Konjevod, G., Sakla, W. A., and Boakye, K. (2016). A large contextual dataset for classification, detection and counting of cars with deep learning. *CoRR*, abs/1609.04453.

Nguyen, T. and Grover, A. (2022). Transformer Neural Processes: Uncertainty-Aware Meta Learning Via Sequence Modeling. In *ICML*.

Rasmussen, C. E. (2003). Gaussian Processes in Machine Learning. In *Summer School on Machine Learning*. Springer.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*.

Schultz, B., Immitzer, M., Roberto Formaggio, A., Del'Arco Sanches, I., José Barreto Luiz, A., and Atzberger, C. (2015). Self-guided segmentation and classification of multi-temporal landsat 8 images for crop type mapping in southeastern brazil. *Remote Sensing*, 7(11):14482–14508.

Seeger, M. (2004). Gaussian processes for machine learning. *International journal of neural systems*, 14(02):69–106.

Shermeyer, J., Hogan, D., Brown, J., Van Etten, A., Weir, N., Pacifici, F., Hansch, R., Bastidas, A., Soenen, S., Bacastow, T., et al. (2020). SpaceNet 6: Multi-Sensor All Weather Mapping Dataset. In *CVPRW*.

Shewchuk, J. (1994). *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Carnegie-Mellon University. Department of Computer Science.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *ICML*.

Tanaka, Y., Tanaka, T., Iwata, T., Kurashima, T., Okawa, M., Akagi, Y., and Toda, H. (2019). Spatially Aggregated Gaussian Processes with Multivariate Areal Outputs. *NeurIPS*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention Is All You Need. *NeurIPS*.

Wang, L., Zhou, X., Zhu, X., Dong, Z., and Guo, W. (2016). Estimation of biomass in wheat using random forest regression algorithm and remote sensing data. *The Crop Journal*.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016). Deep kernel learning. In Gretton, A. and Robert, C. C., editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 370–378, Cadiz, Spain. PMLR.

Yao, W., Zeng, Z., Lian, C., and Tang, H. (2018). Pixel-wise regression using U-Net and its application on pansharpening. *Neurocomputing*.

Yokoya, N., Yamanoi, K., He, W., Baier, G., Adriano, B., Miura, H., and Oishi, S. (2022). Breaking Limits of Remote Sensing by Deep Learning From Simulated Data for Flood and Debris-Flow Mapping. *IEEE Trans. Geosci. Remote Sens.*

You, J., Li, X., Low, M., Lobell, D., and Ermon, S. (2017). Deep Gaussian Process for Crop Yield Prediction Based on Remote Sensing Data. In *AAAI*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Not Applicable]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Supplementary Material for Paper: Neural Point Processes for Pixel-wise Regression

A Proof of Theorem 1

In this section, we present the detailed proof of Theorem 1.

Proof. Given point positions $\mathbf{P}_i = \left[\mathbf{p}_i^{(1)}, \ldots, \mathbf{p}_i^{(L_i)}\right]^\top$ and corresponding labels $\mathbf{y}_i = \left[\mathbf{y}_i^{(1)}, \ldots, \mathbf{y}_i^{(L_i)}\right]^\top$, consider the Gaussian process in Eq. (7) with means obtained by the neural network $f(\mathbf{x}_i;\theta) \in \mathbb{R}^{d_1 \times d_2}$, and added i.i.d. Gaussian noise as in Eq. (6). Then, the labels of points on image i are distributed according to:

$$\mathbf{Y}_i = g_i(\mathbf{P}_i) \sim \mathcal{GP}(f_{\mathbf{P}_i^{(\ell)}}(\mathbf{x}_i;\theta),\ k(\mathbf{P}_i,\mathbf{P}_i;\zeta) + \sigma_0^2\mathbf{I}) \in \mathbb{R}^{L_i}, \tag{11}$$

where $k(\mathbf{P}_i,\mathbf{P}_i) \in \mathbb{R}^{L_i \times L_i}$ is the PSD covariance matrix of GP g_i over points $\mathbf{P}_i$: in other words, its coordinates are $k(\mathbf{p}_i^{(\ell)}, \mathbf{p}_i^{(\ell')})$ over all pairs $\ell, \ell' \in [L_i]$. Note that as $k(\mathbf{P}_i,\mathbf{P}_i;\zeta)$ is symmetric and positive semi-definite, $k(\mathbf{P}_i,\mathbf{P}_i;\zeta) + \sigma_0^2\mathbf{I}$ is positive definite and is therefore invertible. The joint probability density of $\mathbf{Y}_i$ evaluated at observed labels is:

$$f_{\mathbf{Y}_i}(\mathbf{y}_i) = \frac{\exp\left\{-\frac{1}{2}\left(\mathbf{y}_i - f_{\mathbf{P}_i^{(\ell)}}(\mathbf{x}_i;\theta)\right)^\top \left(k(\mathbf{P}_i,\mathbf{P}_i;\zeta) + \sigma_0^2\mathbf{I}\right)^{-1}\left(\mathbf{y}_i - f_{\mathbf{P}_i^{(\ell)}}(\mathbf{x}_i;\theta)\right)\right\}}{2\pi^{L_i/2}|k(\mathbf{P}_i,\mathbf{P}_i;\zeta) + \sigma_0^2\mathbf{I}|^{1/2}}.$$

Consequently, the maximum likelihood loss over i.i.d. images is:

$$\mathcal{L}_{\texttt{MLE}}(\mathcal{D};\theta,\zeta) = \prod_{i=1}^{n} f_{\mathbf{Y}_i}(\mathbf{y}_i), \tag{12}$$

and the corresponding negative log-likelihood is:

$$\begin{aligned}\mathcal{L}_{\texttt{NLL}}(\mathcal{D};\theta,\zeta) &= -\sum_{i=1}^{n} \log f_{\mathbf{Y}_i}(\mathbf{y}_i) \\ &= \sum_{i=1}^{n}\Bigg[\frac{1}{2}(\mathbf{y}_i - f_{\mathbf{P}_i}(\mathbf{x}_i;\theta))^\top (k(\mathbf{P}_i,\mathbf{P}_i;\zeta) + \sigma_0^2\mathbf{I})^{-1}(\mathbf{y}_i - f_{\mathbf{P}_i}(\mathbf{x}_i;\theta)) \\ &\quad + \frac{1}{2}\log|k(\mathbf{P}_i,\mathbf{P}_i;\zeta) + \sigma_0^2\mathbf{I}| + \frac{L_i}{2}\log(2\pi)\Bigg].\end{aligned}$$

Removing constant terms, we obtain the Neural Point Processes (NPP) loss:

$$\begin{aligned}\mathcal{L}_{\texttt{NPP}}(\mathcal{D};\theta,\zeta) &= \sum_{i=1}^{n}\Bigg[\frac{1}{2}(\mathbf{y}_i - f_{\mathbf{P}_i}(\mathbf{x}_i;\theta))^\top (k(\mathbf{P}_i,\mathbf{P}_i;\zeta) + \sigma_0^2\mathbf{I})^{-1}(\mathbf{y}_i - f_{\mathbf{P}_i}(\mathbf{x}_i;\theta)) \\ &\quad + \frac{1}{2}\log|k(\mathbf{P}_i,\mathbf{P}_i;\zeta) + \sigma_0^2\mathbf{I}|\Bigg].\end{aligned} \tag{13}$$

Therefore, minimizing this w.r.t θ is equivalent to maximizing the likelihood. □

B Computational Complexity

We derive the following computational complexity analysis for the baseline MSE and the proposed NPP. Note that in our setting, we have sparse labels and here we assume WLOG all images have the same amount of labeled points, i.e., $L \ll d_1 \times d_2$. Besides, we ignore the calculation regarding the kernel parameters as its number is negligible w.r.t the neural network parameters, i.e., $m_\zeta \ll m_\theta$. We also want to stress that the sparsity of the labels (small value of L) is a major motivating aspect of our work: if labels are dense, indeed the standard MSE approach would suffice.

Forward pass: Since plain MSE and NPP losses are associated with the same model, the inference step should have similar complexities. We denote the inference complexity as M_{d_1,d_2,m_θ}, where the network takes $d_1 \times d_2$ inputs given m_θ parameters.

For the loss computation, the MSE loss calculated in a batch has a complexity of $\mathcal{O}(BL)$. On the other hand, the NPP loss calculates the inverse of the kernel and the products, i.e., $\mathcal{O}(BL^3)$. Here, we assume that inversion is $O(L^3)$, though this can be reduced to $O(L^{2.371552})$ through, e.g., variants of Strassen's algorithm. However, if the kernel is static, the kernel only needs to be computed once at the beginning; thus, the complexity becomes $\mathcal{O}(BL^2)$.

Notably, at inference time, if no partial labels are revealed, we simply take the neural network's output; in that scenario, inference runs as fast as the baseline MSE estimate. Only when partial labels are present at inference time do we pay an $\mathcal{O}(L^3)$ overhead (with L being the number of revealed labels).

Backward propagation: With the scalar loss computed in the forward pass, the backward propagation refers to taking the derivative w.r.t. the neural network parameters (and kernel parameters if the kernel is not static).

For MSE loss, the derivative computation is direct and the complexity is $\mathcal{O}(BLm_\theta)$. For NPP, the computation contains taking the derivative of the network output via the inverted kernel, i.e., $\mathcal{O}(BL^2 + BLm_\theta)$ (and also the derivative for the log determinant of the kernel in the learnable and context-aware kernel case $\mathcal{O}(BL^3)$) in Equation (9).

Comparison and analysis: Summarizing the computational complexity, for the forward pass and backward propagation of MSE, the complexity is $\mathcal{O}(BL + BM_{d_1,d_2,m_\theta})$ for the forward pass and $\mathcal{O}(BLm_\theta)$ for the backward pass. In the case of NPP loss with a static kernel, the forward pass complexity is $\mathcal{O}(BL^2 + BM_{d_1,d_2,m_\theta})$ and the backward propagation complexity is $\mathcal{O}(B(L^2 + Lm_\theta))$. When using learnable or context-aware kernels, the forward complexity becomes $\mathcal{O}(BL^3 + BM_{d_1,d_2,m_\theta})$, and the backward complexity increases to $\mathcal{O}(B(L^3 + L^2 + Lm_\theta))$.

The MSE and NPP (whether using static, learnable, or context-aware kernels) share the same upper bound in both forward and backward passes under certain conditions. In the case of static kernels, if $L \ll \min(M^{1/2}_{d_1,d_2,m_\theta}, m_\theta)$, the forward pass and backward propagation complexities for both MSE and NPP reduce to $\mathcal{O}(BM_{d_1,d_2,m_\theta})$ and $\mathcal{O}(BLm_\theta)$, respectively.

For learnable or context-aware kernels, this condition becomes $L \ll \min(M^{1/3}_{d_1,d_2,m_\theta}, m^{1/2}_\theta)$.

Conclusion: In sparsely labeled setups where conditions are satisfied, the complexity of MSE and NPP are bounded similarly for both forward and backward passes. However, as L grows closer to or exceeds m_θ, the NPP loss becomes more computationally demanding due to the quadratic and cubic terms in L, particularly for non-static kernels.

On the cubic dependence on L: Although the complexity of the inversion of the kernel is cubic in L, where L is the number of (sparse) labels partially revealed at inference time, we do not explicitly need to invert; operations reduce to solving a linear system (e.g., $r^T K^{-1} r$ can be computed as $r^T v$ where $Kv = r$. As we add $\sigma_0^2 I$ to the kernel matrix K, it is always positive definite; we can thus control its condition number through σ_0. We also empirically observe that the optimal kernel bandwidths for static, learnable, and context-aware settings are all small, and, hence, K is also sparse. Thus, we can use a conjugate gradient method to solve the corresponding linear systems in $O(\sqrt{\kappa}Q)$ time, where κ and Q are the condition number and the cardinality of the support of the sparse matrix (Shewchuk, 1994). Thus, using the conjugate gradient method, the actual complexity of NPP can be greatly reduced when K is sparse.

C Additional Experimental Details

C.1 Datasets

We describe all four datasets in detail in this section. We provide exemplars from each dataset in Figure 4.

Synthetic Heatmaps: We generate representations of random elevations as heat maps. We generate a dataset $\mathcal{D}_{\text{SH}} =$

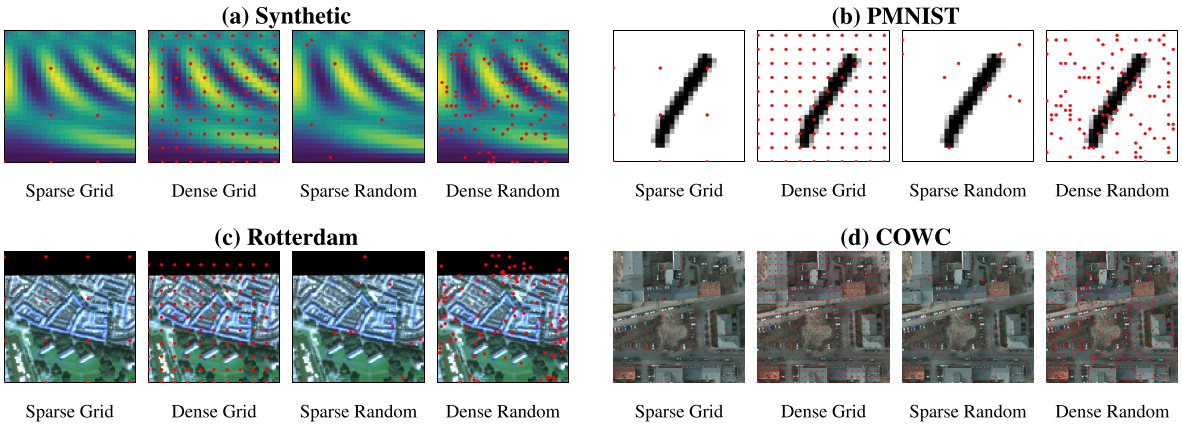


Figure 4: Comparative examples from the Synthetic (a), PMNIST (b), Rotterdam (c), and COWC (d) datasets illustrating different point layouts (grid and random) and sparsities (sparse and dense).

$\{(\mathbf{x}_i, \mathbf{P}_i, \mathbf{y}_i)\}_{i=1}^n$ of size $n = 1000$ images. Each image $\mathbf{x}_i \in \mathbb{R}^{d_1 \times d_2}$ is a $d_1 \times d_2$ matrix where $d_1 = d_2 = 28$. For each image i, its constituent pixels $\mathbf{p} = (p_1, p_2)$ are associated with an "elevation" function $h(\mathbf{p}) = c_1 h_1^{c_2}(p_1) + c_3 h_2(c_4 p_1 p_2) h_3(p_1)$ where the scalars c_1, c_2, c_3, c_4 are integers picked uniformly at random from a finite range, while functions h_1, h_2, h_3 are randomly picked to be either $\sin(\cdot)$ or $\cos(\cdot)$, across images. This elevation variable is used to produce "heatmap" images $\mathbf{x}_i$ using `PyPlot`'s default colormap: this results in colored "peaks" and "valleys", as shown in Fig. 4. We select the points $\mathbf{P}_i$ to be labeled in one of these two ways, either u.a.r. or to be placed on a mesh grid, with different sparsity levels, as illustrated in Table 2. Labels $\mathbf{y}_i$ for points $\mathbf{P}_i$ are sampled from a GP with mean $m(\mathbf{p}) = ah(\mathbf{p}) + b$, where $a = 3$, $b = 0.5$, and RBF kernel $k(\mathbf{p}, \mathbf{p}') = \exp\left(-\|\mathbf{p}' - \mathbf{p}'\|^2/\sigma^2\right)$ with $\sigma^2 = 100$.

PMNIST: In PMNIST, we use 1000 images from MNIST (LeCun et al., 1998) selected uniformly at random (u.a.r.)., as inputs $\mathbf{x}_i$, with pixel values normalized between $[-1, 1]$ to ensure numerical stability. We again generate points $\mathbf{P}_i$ either u.a.r. or in a grid, at differing densities as shown in Table 2. To produce labels $\mathbf{y}_i$, we sum the values of pixels in a disk of radius equal to 3 pixel surrounding the given point.

Rotterdam: We select 1000 images u.a.r. as inputs $\mathbf{x}_i$ from the SpaceNet6 dataset (Shermeyer et al., 2020), a collection of satellite imagery over Rotterdam, Netherlands. We resize images to 100×100 pixels. We generate points $\mathbf{P}_i$ either u.a.r. or in a grid, again at differing densities as shown in Table 2. To produce labels $\mathbf{y}_i$, we count the number of unique buildings in a disk of radius equal to 30 surrounding the given point.

COWC : We selected 1000 images u.a.r. from the Cars Overhead With Context (COWC) dataset (Mundhenk et al., 2016), which provides high-resolution (2048×2048) satellite imagery focused on vehicle detection across diverse geographic regions. For our experiments, we first crop these images to a 800×800, and then resize them to 200×200 pixels. In line with the dataset's original methodology, training and validation images are taken from a disjoint set from the test images. Specifically, we sampled 700 images for testing and 100 for validation, and an additional 200 test images u.a.r. Labels are generated either u.a.r. or on a grid, where each label represents the number of cars within a 100-pixel radius around the point.

C.2 Model Architectures

We implement three networks to predict the point values: an autoencoder (Kramer, 1991), implemented as a base model, a DDPM (Ho et al., 2020), and the three-stage architecture used by Neural Processes. The plain, NPP, and NPP-GP algorithms are tested with the same autoencoder architectures while exploring different depths. We use the NP architecture from the public PyTorch Neural Processes repository[4].

Autoencoder (Kramer, 1991): We implement a standard autoencoder, whose encoder comprises two convolutional (Conv) layers with 64 and 32 output channels with kernels of size 3 and padding to keep the shape unchanged. After each Conv layer, we apply Rectified Linear Unit (ReLU) activation functions and max-pooling layers with a kernel size of 2 and a stride of 2. The decoder comprises two transposed Conv layers with 64 and 1 output channels. For Rotterdam,

[4]https://github.com/EmilienDupont/neural-processes/tree/master

Table 4: Autoencoder architectures for different datasets. The input size denotes the channel number and shape of the inputs. In the Encoder, Conv2d(x, 3, 1) denotes a 2D convolutional layer with 'x' output channels, a kernel size of 3, and padding of 1. MaxPool2d(2, 2) indicates a pooling layer with a kernel size and stride of 2. In the Decoder, ConvT2d(x, 2, 2) represents a transposed convolution with 'x' output channels, a kernel size of 2, and a stride of 2, with an optional output padding input in the end with a default of 0.

Dataset	PMNIST (Heatmaps)	Rotterdam	COWC
Input Size	$1(3) \times 28 \times 28$	$4 \times 100 \times 100$	$3 \times 200 \times 200$
Encoder	3 × (Conv2d(64, 3, 1) + ReLU) MaxPool2d(2, 2) 3 × (Conv2d(32, 3, 1) + ReLU) MaxPool2d(2, 2)	3 × (Conv2d(64, 3, 1) + ReLU) MaxPool2d(2, 2) 3 × (Conv2d(32, 3, 1) + ReLU) MaxPool2d(2, 2) 3 × (Conv2d(16, 3, 1) + ReLU) MaxPool2d(2, 2)	3 × (Conv2d(64, 3, 1) + ReLU) MaxPool2d(2, 2) 3 × (Conv2d(32, 3, 1) + ReLU) MaxPool2d(2, 2) 3 × (Conv2d(16, 3, 1) + ReLU) MaxPool2d(2, 2) 3 × (Conv2d(8, 3, 1) + ReLU) MaxPool2d(2, 2)
Decoder	ConvT2d(32, 2, 2, 1) + ReLU 2 × (Conv2d(64, 3, 1) + ReLU) ConvT2d(64, 2, 2) + ReLU	ConvT2d(16, 2, 2, 1) + ReLU 2 × (Conv2d(32, 3, 1) + ReLU) ConvT2d(32, 2, 2) + ReLU 2 × (Conv2d(64, 3, 1) + ReLU) ConvT2d(64, 2, 2) + ReLU	ConvT2d(8, 2, 2, 1) + ReLU 2 × (Conv2d(16, 3, 1) + ReLU) ConvT2d(16, 2, 2) + ReLU 2 × (Conv2d(32, 3, 1) + ReLU) ConvT2d(32, 2, 2) + ReLU 2 × (Conv2d(64, 3, 1) + ReLU) ConvT2d(64, 2, 2) + ReLU

which is more complex and needs a deeper architecture, we treat the depth of both the encoder and decoder as a hyper-parameter, adding additional convolutional layers: we experimented with 3 different architectures in total, illustrated in Appendix C.2 of the supplement.

DDPM (Ho et al., 2020): Inspired by Baranchuk et al. (2022), we use DDPM as a feature extractor: after we train it we use its feature maps as an alternative form of input to the autoencoder. We then train an AE on these maps as inputs, in the same fashion as described above. We obtained the DDPM architecture from a public GitHub repository.[5] The DDPM is trained in a self-supervised fashion, to reconstruct the original input. The training comprises a forward and a backward process. The forward process keeps adding noise to the original image input in a sequential way from step 0 to step T, while the backward process aims at predicting the noise from the final noisy image at step T backward to step 0 using a UNet (Ronneberger et al., 2015) architecture, thus removing the noise.

After training, we evaluate the DDPM model by visualizing the generated samples. For PMNIST and Synthetic Heatmaps datasets, we set the steps to be 1000, and the $\beta_{\min}$ and $\beta_{\max}$ for noise to be 10^{-4} and 0.02 respectively. Here, we apply Sinusoidal embeddings (Vaswani et al., 2017) to encode temporal information. In the UNet, we have the encode and decode paths. The encoder path has 4 blocks (with LayerNorm and 6 Conv layers followed by the SiLU activation (Elfwing et al., 2018)) connected by a Conv layer of stride 2 to downsample the size and the resulting channels are $[10, 20, 40, 80]$. The decoder has 3 blocks connected with a transposed Conv layer for upsampling and the results channels are $[40, 20, 1]$. Notably, we concatenate the time embeddings into the input of each block.

However, since Rotterdam is a much more complicated dataset than the other two Synthetic Heatmaps datasets, we failed to achieve a good DDPM model. Thus, we load the pre-trained model from DDPM-CD (Bandara et al., 2022), which is trained on various building-related datasets LEVIR-CD (Chen and Shi, 2020) and the generated images are verified by visualization. The code and model are publicly available.[6]

In the experiments, we use shallow AE for PMNIST and Synthetic Heatmaps datasets, and we use all three AE with the Rotterdam dataset. When we feed feature maps extracted from the DDPM, although the input channels are increased, the intermediate channels remain the same. The detailed structures are shown in Tables 5–6, and discussed next.

Detailed Autoencoder Structures: For each dataset, we attempt to find the minimal structure that enables the baseline method *Plain* to work successfully (achieving positive R^2) in the random dense setting of each dataset. To explore the minimum structure, we gradually add one block (consisting of three convolutional layers followed by ReLU with one max-pooling layer). We list the optimal structure found for each dataset based on the validation set in Table 4. For PMNIST and Synthetic Heatmaps, the shallow AE is used. Furthermore, we add one (two) extra block(s) for the encoder and decoder for Rotterdam (COWC).

Note that for the *Context-aware kernel*, we predict the model by connecting an extra FC layer to the encoder. The output dimension depends on the kernel choices.

[5] https://github.com/BrianPulfer/PapersReimplementations

[6] https://github.com/wgcban/ddpm-cd

Table 5: DDPM architectures with UNet as inputs across different datasets for reproducibility.

Dataset	Input Shape	DDPM Initialization with UNet
Synthetic Heatmaps	$3 \times 28 \times 28$	DDPM(UNet(3, shape=(28,28)), n_steps=1000, min_beta=10^{-4}, max_beta=0.02, device=device)
PMNIST	$1 \times 28 \times 28$	DDPM(UNet(1, shape=(28,28)), n_steps=1000, min_beta=10^{-4}, max_beta=0.02, device=device)
Building	$4 \times 100 \times 100$	DDPM(UNet(4, shape=(100,100)), n_steps=1000, min_beta=10^{-4}, max_beta=0.02, device=device)
Cars	$3 \times 200 \times 200$	DDPM(UNet(3, shape=(200,200)), n_steps=1000, min_beta=10^{-4}, max_beta=0.02, device=device)

Table 6: UNet Architecture Used Within DDPM Across Different Datasets. Each Conv block in the upsampling path takes both the previous upsampled feature maps and the results from the corresponding downsampling layers as input (concatenated).

Component	Architecture Details
Time Embedding	nn.Embedding(n_steps=1000, time_emb_dim=100) Sinusoidal Embedding: $t \to \mathbb{R}^{100}$
Conv1	Conv2d(input channels, 10, 3) + SiLU 5 × (Conv2d(10, 10, 3) + SiLU) Downsample: Conv2d(10, 10, 4, stride=2)
Conv2	Conv2d(10, 20, 3) + SiLU 5 × (Conv2d(20, 20, 3) + SiLU) Downsample: Conv2d(20, 20, 4, stride=2)
Conv3	Conv2d(20, 40, 3) + SiLU 5 × (Conv2d(40, 40, 3) + SiLU) Downsample: Conv2d(40, 40, 4, stride=2)
Bottleneck	Conv2d(40, 20, 3) + SiLU 7 × (Conv2d(20, 20, 3) + SiLU) Conv2d(20, 40, 3) + SiLU
Upsampling1	ConvTranspose2d(40, 40, 4, stride=2) + SiLU ConvTranspose2d(40, 40, 2)
Conv4	Takes inputs from Upsampling1 and Conv3 (concatenated) Conv2d(80, 40, 3) + SiLU Conv2d(40, 40, 3) + SiLU Conv2d(40, 20, 3) + SiLU 3 × (Conv2d(20, 20, 3) + SiLU)
Upsampling2	ConvTranspose2d(20, 20, 4, stride=2) + SiLU
Conv5	Takes inputs from Upsampling2 and Conv2 (concatenated) Conv2d(40, 20, 3) + SiLU Conv2d(20, 20, 3) + SiLU Conv2d(20, 10, 3) + SiLU 3 × (Conv2d(10, 10, 3) + SiLU)
Upsampling3	ConvTranspose2d(10, 10, 4, stride=2) + SiLU
Conv6	Takes inputs from Upsampling3 and Conv1 (concatenated) Conv2d(20, 10, 3) + SiLU 5 × (Conv2d(10, 10, 3) + SiLU) Conv2d(10, input channel, 3) + SiLU

Detailed DDPM Structure: For DDPM, we use a common structure for all experiments and further use a pre-trained DDPM specifically for Rotterdam (Bandara et al., 2022). To be specific, we conducted experiments with the model structure summarized as in Table 5 with the default UNet and DDPM as in Table 6 and Table 7. Moreover, for Rotterdam, due to the low performance, we also tested both MSE and NPP with a pre-trained DDPM that is publicly available as mentioned on similar satellite datasets, such as s LEVIR-CD.

The UNet architecture employs skip connections to combine feature maps from different levels. During the upsampling path, the upsampled feature maps are concatenated with the corresponding feature maps from the downsampling path to enhance the representation. Specifically, **Conv4** takes inputs from both **Upsampling1** and **Conv3**, **Conv5** integrates features from **Upsampling2** and **Conv2**, and **Conv6** merges the upsampled features from **Upsampling3** with those from **Conv1**.

As Neural Processes-based methods cannot process images directly (they operate in the label space by learning the mapping from pixel coordinate to labels), we follow the structures in their work. The structures are shown below.

Neural Processes (NP) (Garnelo et al., 2018b): Recall that the NP model consists of three parts: the encoder, aggregator, and decoder. The encoder is composed of three fully connected (FC) layers with two ReLU activation functions in between, the aggregator is composed of 3 FC layers followed by one Sigmoid function, and the decoder is made of 3 FC layers with each followed by a ReLU activation.

Convolutional Neural Processes (ConvNP) (Gordon et al., 2020): The ConvNP model extends the neural process framework to handle spatial data by leveraging convolutional architectures for efficient and translation-equivariant processing. Its encoder begins with two parallel convolutional layers—each with a kernel size of 11—where one computes a so-called signal representation and the other estimates density by convolving the masks. These outputs are normalized and concatenated before being passed through a single linear layer that projects the 2-channel input into a 128-dimensional latent space. Next, a convolutional neural network comprising 5 convolutional blocks (each using a kernel size of 9) refines this latent grid representation by aggregating local spatial information (Chollet, 2017). Finally, the decoder, typically implemented as a 2-layer multilayer perceptron (one hidden layer with ReLU activation followed by an output layer), maps the refined features to output predictions, ensuring that the model can seamlessly handle varying context sizes while preserving spatial consistency and translation equivariance.

C.3 Metrics

We evaluate our models with pixel-wise MSE loss and both global and local R^2. Given a label $\mathbf{y}_i = [y_i^{(1)}, \ldots, y_i^{(L_i)}] \in \mathbb{R}^{L_i}$ and the corresponding prediction $\hat{\mathbf{y}}_i = f_{\mathbf{P}_i}(\mathbf{x}_i; \theta)$, the average pixel-wise MSE over all n images can be expressed as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{L_i}\sum_{j=1}^{L_i}\left(y_i^{(j)} - \hat{y}_i^{(j)}\right)^2,$$

where $\hat{y}_i^{(j)}$ denotes the prediction of image i on the j-th point $\mathbf{p}_i^{(j)}$. We also compute R^2, the coefficient of determination, defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\sum_{j=1}^{L_i}\left(y_i^{(j)} - \hat{y}_i^{(j)}\right)^2}{\sum_{i=1}^{n}\sum_{j=1}^{L_i}\left(y_i^{(j)} - \bar{y}_{\text{global}}\right)^2},$$

where

$$\bar{y}_{\text{global}} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{L_i} y_i^{(j)}}{\sum_{i=1}^{n} L_i}$$

is the global average of the labels. The numerator represents the residual sum of squares (RSS), and the denominator represents the total sum of squares (TSS), reflecting the variance in the true values.

C.4 Code and Hardware Implementation Details

All cited repositories used in this work are released under the MIT license. In terms of the hardware configurations, all experiments were conducted on a Tesla V100-SXM2-32GB GPU with CUDA Version 12.3. The system was equipped with an Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz.

C.5 Optimal Hyperparameters

As mentioned, we treat learning rates, features (original images or DDPM extracted features), kernels, and kernel parameters as hyperparameters. Then, we explore the best combination based on the validation set. Here, we report the optimal hyperparameters in Table 8 corresponding to the results shown in Table 3 for reproducing purposes.

D Additional Experimental Results

D.1 Performance Trend in the Partial Label Setting

In Section 5, we show the relation between performance w.r.t. different percentages of revealed labels in the dense random setting. Here, we present the relation in the other three settings: sparse random, dense grid, and sparse grid in Figure 5.

Table 7: DDPM Diffusion Process and Architecture

Component	Details
n_steps	1000 (Number of diffusion steps)
β_t	Linearly spaced between min_beta=10^{-4} and max_beta=0.02
α_t	$\alpha_t = 1 - \beta_t$ (Scaling factor for retaining the original image data at time step t)
$\bar{\alpha}_t$	$\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ (Represents the cumulative product of α_s
Forward Process	At time step t, the noisy image x_t is computed as: $x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \eta$ where x_0 is the original image and η is Gaussian noise. Noise is added incrementally over n_steps to reach x_T, a fully noisy image.
Reverse Process	The model learns to predict the noise η_t added at each time step t. At each reverse step, the UNet estimates the noise η_t given x_t and t. The denoised image at step $t-1$ is computed as: $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot \eta_t \right)$ This progressively removes noise from x_t to recover x_0.

Table 8: The optimal hyperparameters corresponding to the results shown in Table 3. * indicates that the model takes DDPM extracted features as inputs. - indicates not applicable (the method does not use a kernel).

	Datasets		Rotterdam				COWC			
	Point pattern		Grid		Random		Grid		Random	
	Metric		Kernel (ζ_0)	LR	Kernel (ζ_0)	LR	Kernel (ζ_0)	LR	Kernel (ζ_0)	LR
Real-world	Sparse	Plain	-	0.001	-	0.001	-	0.001	-	0.001
		NPP/NPP-GP	$\text{SM}_{\text{context}}$ (1.0)	0.001	$\text{SM}_{\text{learnable}}$ (2.0)	0.001	$\text{RBF}_{\text{static}}$ (0.01)	0.001	$\text{RBF}_{\text{static}}$ (0.1)	0.001
		NP	-	0.001	-	0.001	-	0.0001	-	0.0001
	Dense	Plain	-	0.001	-	0.001	-	0.001	-	0.001
		NPP/NPP-GP	$\text{RBF}_{\text{static}}$ (0.01)	0.001	$\text{RBF}_{\text{learnable}}$ (0.2)	0.001	$\text{RBF}_{\text{static}}$ (0.01)	0.001	$\text{SM}_{\text{static}}$ (2.0)	0.001
		NP	-	0.0001	-	0.0001	-	0.0001	-	0.001
	Datasets		PMNIST				Synthetic Heatmaps			
	Point pattern		Grid		Random		Grid		Random	
	Metric		Kernel (ζ_0)	LR	Kernel (ζ_0)	LR	Kernel (ζ_0)	LR	Kernel (ζ_0)	LR
Synthetic	Sparse	Plain	-	0.001	-	0.001	- *	0.01	-*	0.001
		NPP/NPP-GP	$\text{RBF}_{\text{static}}$ (0.2)	0.001	$\text{SM}_{\text{learnable}}$ (2.0)	0.001	$\text{RBF}_{\text{learnable}}$ (2.0) *	0.01	$\text{RBF}_{\text{static}}$ (0.01)	0.001
		NP	-	0.0001	-	0.0001	-	0.001	-	0.001
	Dense	Plain	-	0.001	-	0.001	-	0.01	-	0.001
		NPP/NPP-GP	$\text{SM}_{\text{learnable}}$ (2.0)	0.001	$\text{RBF}_{\text{static}}$ (0.2)	0.001	$\text{RBF}_{\text{context}}$ (0.1)	0.01	$\text{RBF}_{\text{static}}$ (0.01)	0.01
		NP	-	0.0001	-	0.001	-	0.001	-	0.001

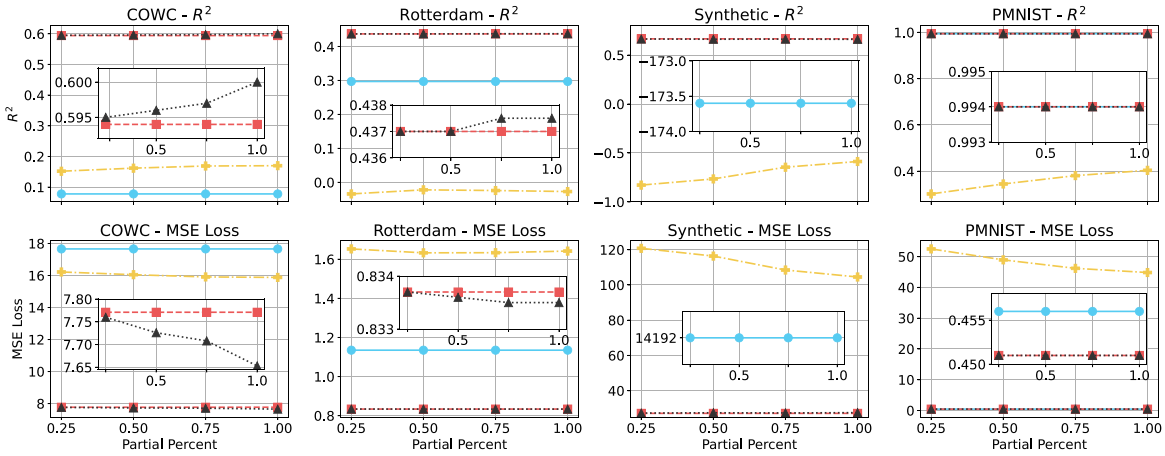
The behavior w.r.t. both R^2 and MSE loss is consistent with the one reported in Section 5. Except for three settings where the performance is saturated (NPP itself achieves 0.995 R^2 in both dense grid and sparse random of PMNIST) or all the methods fail (all methods achieve around 0 R^2 in sparse grid of Rotterdam), we observe that with the increasing amount of revealed labels, the performance all increases for both NPP-GP and NP. Meanwhile, NPP-GP outperforms the rest in a majority of cases. Thus, we conclude that the NPP-GP method based on the NPP kernel is robust and effective across different settings and datasets.

D.2 Visualization on COWC

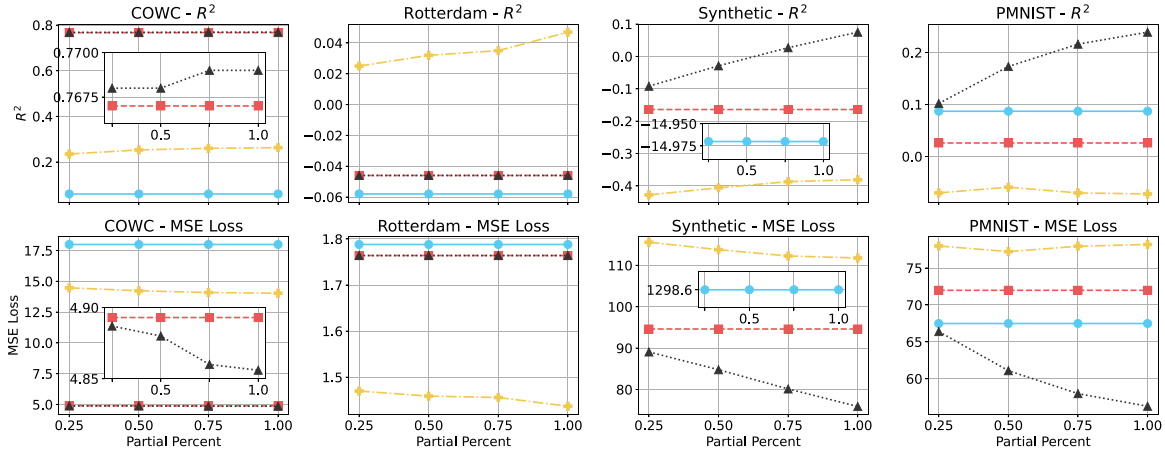
We also provide a qualitative figure in Figure 6 to visually compare the performance of Plain and NPP methods under a dense random setup. The figure presents two test images from the Rotterdam dataset, alongside their respective predictions using both methods. The scattered colored points indicate the 100 labeled pixels with corresponding ground truth values. By superimposing the semi-transparent predictions over the test images, we can assess how well each method captures car density regions. As observed, in contrast to Plain, NPP does not misidentify regions of high car density, demonstrating its superior performance in preserving spatial structures.

D.3 Detailed Results of the Tied Methods on PMNIST

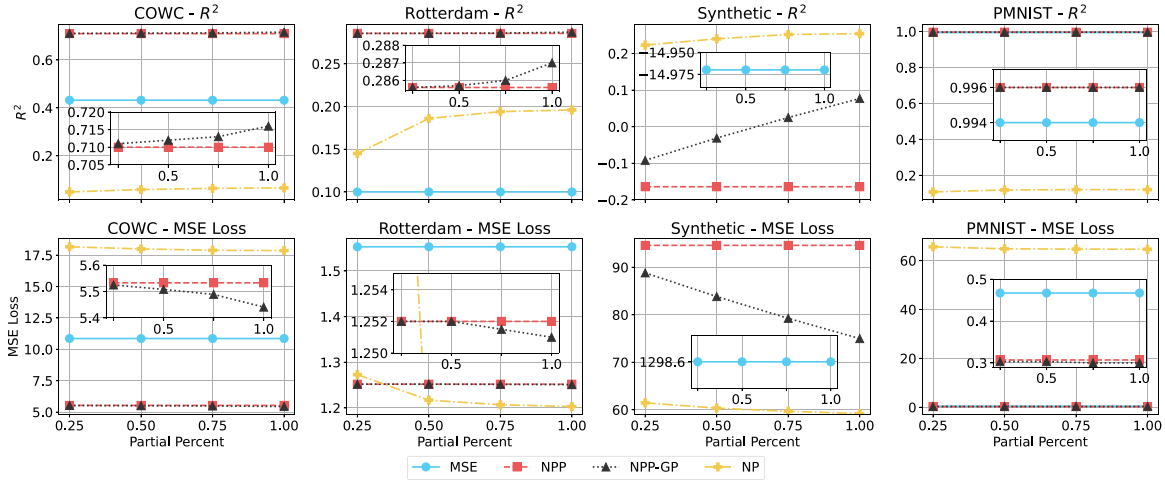
To distinguish the tied results in Table 3, we provide the standard deviation by running the experiments on 3 different seeds in Table 9.



(a) Sparse random setting

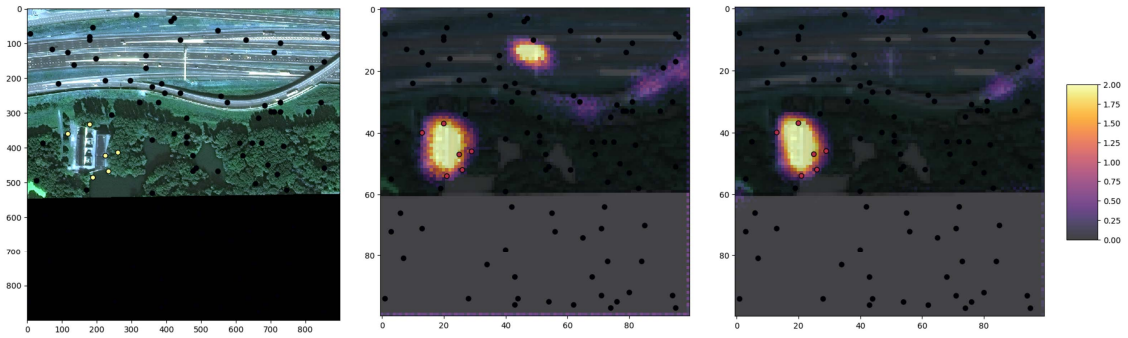


(b) Sparse grid setting



(c) Dense grid setting

Figure 5: Performance across all datasets with gradual label revelation in different settings: (a) sparse random, (b) sparse grid, and (c) dense grid. The x-axis represents the percentage of labels revealed, and the y-axis shows the performance metrics MSE (↓) and R^2 (↑). In most scenarios, NPP-GP consistently outperforms NP and Plain, with the most noticeable improvements occurring in denser settings. The legend, shared across all subfigures, is displayed in (c).



(a) Sample image (b) Plain (random) (c) NPP (random)

Figure 6: Visual comparison of Plain and NPP methods trained under a sparse random setup on two test images of the COWC dataset, where labels indicate car densities. The scattered colored points indicate the labeled pixels with corresponding ground truth values. We predict every pixel label with Plain or NPP using random pixels during training. Then, we superimpose the semi-transparent prediction over each test image. Predictions and labels share the same color map. Thus, color proximity indicates better predictions. We observe that, in contrast to Plain, NPP does not misidentify regions of high car density.

Table 9: Performance comparison in R^2 on PMNIST

	Plain (MSE)	NPP	NPP-GP
Random Dense	0.9976 ± 0.0002	0.9982 ± 0.00005	0.9985 ± 0.00004
Random Sparse	0.9923 ± 0.0015	0.9926 ± 0.0016	0.9926 ± 0.0017

Table 10: Comparison of NPP, NPP-GP, and ConvNP in the COWC dense random setting.

Method	Labels Used	$R^2 \uparrow$
NPP (w/o label)	0	0.71
NPP-GP	500	0.73
ConvCNP	125	0.56
	250	0.89
	375	0.92
	500	0.92

D.4 Detailed Results in COWC Dense Random Setting

We provide a detailed comparison of NPP, NPP-GP, and ConvNP with different amounts of labels revealed in Table 10. We notice that NPP with raw image alone outperforms ConvNP with 125 labeled context points, indicating the effectiveness of utilizing the raw image.