University of Brasília
Institute of Exact Sciences
Department of Statistics

Master's Dissertation

Enhancing Accuracy through an Efficient Ensemble of Geographically Fine-Tuned Positional Encoder Graph Neural Networks

by

Vívia de Alencar Seabra

Brasília, August of 2024

Enhancing Accuracy through an Efficient Ensemble of Geographically Fine-Tuned Positional Encoder Graph Neural Networks

by

Vívia de Alencar Seabra

Dissertation submitted to Department of Statistics at the University of Brasília, as part of the requirements required to obtain the Master Degree in Statistics.

Advisor: Dr. Guilherme Souza Rodrigues

Brasília, August of 2024

Dissertation submitted to the Graduate Program in Statistics of the Department of Statistics at the University of Brasília, as part of the requirements required to obtain the Master Degree in Statistics.

Approved by:

Dr. Guilherme Souza Rodrigues
Advisor, EST/UnB

Dr. Alan Ricardo da Silva
Professor, EST/UnB

Dr. Fabrício Aguiar Silva
Professor, IEF/UFV

Science, my lad, is made up of mistakes, but they are mistakes which it is useful to make, because they lead little by little to the truth.

(Jules Verne, A Journey to the Center of the Earth)

To my Mother, the one who never doubted that I could achieve anything I set my mind to.

Acknowledgments

Thanks to the faculty of PPGEST/UnB, my supervisor Dr. Guilherme Souza Rodrigues, all professors and colleagues who contributed to this work in any way. Also thanks to Prof. Alan Ricardo da Silva and Prof. Fabrício Aguiar Silva for accepting the invitation to be part of this work's evaluation committee. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Resumo Expandido

Melhorando a Acurácia por meio de um Ensemble Eficiente de Redes Neurais em Grafos com Codificadores Posicionais Ajustados Geograficamente

A presente pesquisa investiga uma nova abordagem para melhorar a precisão preditiva e a capacidade de generalização de redes neurais em grafos (Graph Neural Networks - GNNs) aplicadas a dados espaciais. Dados espaciais oferecem *insights* valiosos sobre fenômenos geográficos e relações espaciais, apresentando desafios únicos que requerem metodologias analíticas dedicadas. Características como autocorrelação espacial, heterogeneidade espacial e não-estacionaridade espacial dificultam a aplicação de técnicas convencionais de aprendizado de máquina, que geralmente assumem a independência dos pontos de dados ou relações lineares.

Baseados no algoritmo estado da arte para regressão em dados espaciais proposto por (Klemmer, Safir, and Neill, 2023), as inovações propostas nesta pesquisa são as seguintes: primeiramente, introduzimos uma **Função de Perda Ponderada**, que implementa um mecanismo de ponderação na função de perda, permitindo que o modelo priorize o aprendizado de pontos de dados com base em sua proximidade espacial. Em segundo lugar, é proposto a **Escolha Eficiente de Modelos Localizados**, onde, baseados em mecanismos de clusterização, modelos locais são criados reduzindo o número de modelos necessários e aumentando a eficiência computacional. Em terceiro, é utilizado um **Ensemble de Modelos Locais**, combinando previsões

dos múltiplos modelos localizados para suavizar erros e reduzir o impacto de imprecisões de qualquer modelo individual. Quarto, a **Utilização de Pesos Pré-Treinados** inicializa modelos locais com pesos de um modelo global previamente treinado, melhorando a eficiência do treinamento e fornecendo uma base robusta. Por fim, introduzimos uma **Matriz de Distância com Redução Dimensional**, que usa distâncias entre clusters em vez de entre todos os pontos, simplificando a carga computacional.

Para avaliar a eficácia da abordagem proposta, utilizamos dois conjuntos de dados reais que contêm informações geográficas. O primeiro é o California Housing, que inclui preços de mais de 20.000 casas na Califórnia, coletados a partir do censo dos EUA de 1990 e com objetivo de previsão dos preços das casas com base em características como idade da casa e número de quartos, além de suas localizações geográficas. O segundo conjunto de dados é o Air Temperature, que contém coordenadas de 3.000 estações meteorológicas ao redor do mundo. Neste caso, a tarefa de regressão é prever as temperaturas médias a partir da precipitação média e da localização das estações.

A função de perda ponderada, que prioriza o aprendizado a partir de pontos de dados com base em sua proximidade espacial, mostrou-se eficaz em melhorar a sensibilidade às variações locais, aumentando significativamente o desempenho do modelo para o conjunto de dados Air temperature. Além disso, a análise de sensibilidade revelou que aumentar o número de clusters ou a largura de banda geralmente melhora a precisão do modelo, mas até certo ponto, após o qual as melhorias se estabilizam ou diminuem. Esses achados indicam que a simples elevação desses parâmetros sem considerar suas interações pode resultar em resultados subótimos, destacando a necessidade de métodos mais sofisticados para a seleção desses valores.

Palavras-chave: Dados geográficos, Redes Neurais em Grafos, Função de Perda Ponderada

Abstract

Spatial data analysis presents unique challenges due to the inherent properties of spatial autocorrelation, heterogeneity, and non-stationarity. Traditional approaches often struggle with these complexities, leading to models that either underfit or misinterpret spatial dynamics. This work introduces an innovative approach to enhance the predictive power and generalizability of spatial data models by integrating localized modeling techniques with the advanced capabilities of Graph Neural Networks (GNNs).

Our method incorporates the clustering of geographical coordinates to train localized models effectively. This approach leverages the strength of GNNs to capture and utilize complex spatial relationships. By segmenting the data into clusters, we create localized models that learn from specific spatial contexts, aiming to improve model accuracy and performance.

We introduce a novel weighted loss function that prioritizes geographical proximity between clusters. Additionally, we employ pre-trained weights from a global model to initialize these localized models, which speeds up the training process and gives the models a comprehensive understanding of spatial relationships before adjusting them to fit specific local data.

This work contributes to the field of spatial data analysis by providing a scalable, efficient, and effective framework for modeling complex spatial relationships.

Contents

List of Tables

List of Figures

Abbreviations and Acronyms

GNN	Graph Neural Networks
GWR	Geographically Weighted Regression
PE-GNN	Positional Encoder Graph Neural Networks
LEPE-GNN	Localized Ensemble Positional Encoder Graph Neural Networks

Chapter 1

Introduction

1.0.1 Contextualization

Spatial data offers invaluable insights into geographic phenomena and spatial relationships. Unlike traditional datasets, spatial data contains information about where things are and what shape they have. It is a way of describing things in the real world in terms of their position, like their latitude and longitude, and their form, which could be points (specific locations), lines (like roads or rivers), or areas (like the boundary of a city or a lake). It is critical in many fields, from urban planning and environmental science to public health and logistics. However, the inherently distinct nature of spatial data introduces specific complexities that require dedicated analytical methodologies.

When working with spatial data, we must deal with a number of unique characteristics that distinguish it from other types of data. One of the most significant of these is spatial autocorrelation. This principle, also known as Tobler's First Law of Geography (Tobler, 1970), asserts that points closer in space are more likely to exhibit similar values than points that are further apart. This can create complications for conventional statistical learning techniques and machine learning algorithms which typically operate under the assumption that data points are independent of one another. In cases where spatial autocorrelation is present, this assumption

is invalidated, which can lead to biased estimates.

Spatial heterogeneity and spatial nonstationarity (Bivand et al., 2008) are other important concepts in geographical data analysis, though they address different aspects of spatial variation. Spatial heterogeneity refers to the variability in relationships between variables across different geographical locations. For instance, the impact of population density on air pollution may vary between urban and rural areas. This variability challenges traditional statistical models which often assume that relationships observed in one geographical area apply universally. On the other hand, spatial nonstationarity deals with changes in the statistical properties of a process, such as mean and variance, over space. This concept is crucial when statistical characteristics are expected to remain constant across different locations, which is often not the case. For example, assuming a constant average temperature across a country could lead to inaccuracies in models that do not account for regional differences. Meaning, nonstationarity requires models that can adapt their parameters to accommodate specific geographical variations. This complexity necessitates advanced modeling techniques that can adapt to such diversity in data characteristics.

Spatial data can be efficiently viewed, stored, and analyzed as a graph (Nikparvar and Thill, 2021). Their intrinsic structure, composed of nodes and edges, allows for a direct and intuitive representation of real-world phenomena (Veličković, 2023). Nodes in a graph can depict various spatial elements: cities, landmarks, individual buildings, etc. While edges can express the relationships or connections between these entities, whether they are physical pathways like roads or railways, or abstract connections like trade routes or social links.

As we see, the graph structure provides a natural way to capture spatial autocorrelation, as the proximity of the nodes in the graph can represent their geographical closeness. This makes them particularly suitable for spatial data analysis, where neighboring data points often exhibit similar properties due to spatial auto-correlation.

In recent years, the field of machine learning has witnessed a significant rise in interest in the application of Graph Neural Networks (GNNs) to solve complex problems in various domains,

including spatial data-related tasks (Mai et al., 2020; Yin et al., 2019).

GNNs leverage the relationships between spatial entities, represented as graph structures, to model spatial dependencies and uncover hidden patterns. They excel at handling the challenges posed by spatial autocorrelation and heterogeneity, modeling complex, variable relationships across different geographical locations. This characteristic of GNNs to incorporate neighborhood information and capture local spatial dependencies makes them particularly valuable in addressing spatial non-stationarity. They deliver locally adaptive models that can account for the changing relationships between variables across geographic space.

A novel advancement called PE-GNN (Klemmer, Safir, and Neill, 2023) has emerged, offering an innovative way to overcome some inherent challenges in traditional GNNs, unlike conventional GNNs, which often rely on Euclidean distances to construct the input graphs. PE-GNN introduces a new framework that incorporates spatial context and correlation into the modeling process in two big tasks: (1) learns a context-aware vector encoding of the geographic coordinates, and (2) predicts spatial autocorrelation in the data while also handling the main task. This method improves our understanding of space by identifying and responding to the complex spatial structures found in real-world situations, providing a more flexible and detailed way to represent spatial data.

1.0.2 Objectives

In this dissertation, a new method, Localized Ensemble Positional Encoder Graph Neural Network (LEPE-GNN), aims to build a more flexible model using geographically localized models based on clustering. By dividing the spatial data into clusters, each cluster can be treated as a separate entity with its own model that captures local spatial relationships and variations. This approach is based on PE-GNN and incorporates several innovations into its original structure with the goal of performance improvement:

- **Weighted Loss Function**: A weighting mechanism within the loss function allows the

model to prioritize learning from data points based on their spatial proximity, enhancing the model's sensitivity to local spatial correlations.

- **Efficient Choice of Localized Model Locations**: Instead of fitting a new model for each point (resulting in N models), we can select a set of K locations based on prior clustering. This method reduces the number of models we need to manage and enhances computational efficiency.

- **Ensemble of Local Models**: Combining predictions from multiple localized models trained on different clusters helps to smooth out errors and reduce the impact of inaccuracies from any single model, leveraging diversity for improved prediction stability.

- **Utilization of Pre-Trained Weights**: Initializing local models with weights pre-trained from a global model enhances training efficiency and provides a robust starting point that encapsulates previously learned spatial relationships.

- **Dimensionally Reduced Distance Matrix**: Instead of considering the distance between every pair of points (resulting in an $n \times n$ matrix), we use distances between clusters. This reduces the distance matrix to a size of $k \times k$, where k is much smaller than n, simplifying the computational load.

1.0.3 Text Organization

The remainder of this dissertation is organized as follows: Chapter 2 reviews existing approaches to spatial data analysis and highlights the limitations of traditional GNNs. It also discusses the innovations introduced by PE-GNNs and their potential pitfalls. Chapter 3 describes the LEPE-GNN method. Chapter 4 details the experimental setup, outlining the clustering methods, weighting mechanisms, and other parameters used in the experiments, the results and discussion, analyzing the outcomes of the experiments, and discussing their implications

for spatial data analysis. Chapter 5 concludes with a summary of the key findings and suggestions for future research, providing a broader perspective on how localized models can be further developed and applied.

Chapter 2

Background

2.1 Graph Theory

Graphs are a fundamental structure in mathematics and computer science, represented using two key components: nodes (or vertices) and edges. Mathematically, a graph G can be defined as an ordered pair $G = (V, E)$ where V is a set of vertices $V = \{v_1, v_2, ..., v_n\}$ and E is a set of edges $E = \{e_1, e_2, ...e_m\}$, with each edge e_k connecting a pair of vertices. The edges may be directed, indicating a one-way relationship with an ordered pair of vertices (v_i, v_j), or undirected, indicating a bidirectional relationship represented by an unordered pair $\{v_i, v_j\}$. Additionally, graphs can be classified as weighted if each edge e_k is assigned a weight w_k, reflecting the strength or cost of the connection. This mathematical framework allows for the representation of a wide array of systems, from social networks and communication infrastructures to biological networks and transportation systems, making graphs a versatile tool for modeling and analyzing complex relationships and structures (Barabási, 2016).

2.1.1 The Adjacency Matrix

In a graph, two vertices are said to be adjacent if there's an edge directly connecting them. Degree refers to the number of edges incident to a node.

An adjacency matrix is a square matrix used to represent a graph. The matrix has dimensions $n \times n$, where n is the number of vertices in the graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

We define the adjacency matrix for a graph G with n vertices is denoted as $A = [a_{ij}]$, where $1 \leq i, j \leq n$. The matrix element a_{ij} is defined as follows:

$$a_{ij} = \begin{cases} 1, & \text{if there is an edge from vertex } i \text{ to vertex } j \\ 0, & \text{otherwise} \end{cases}. \tag{2.1}$$

For an undirected graph, the adjacency matrix is symmetric, meaning $a_{ij} = a_{ji}$ for all i, j because if node i is adjacent to node j, then node j is also adjacent to node i. This symmetry reflects the bidirectional nature of the connections in undirected graphs.

In the case of directed graphs, the adjacency matrix does not necessarily have to be symmetric, as the presence of an edge from i to j does not imply the presence of an edge from j to i.

For a weighted graph, instead of using 1 to represent an edge, aij represents the weight of the edge between vertices i and j. If this edge does not exists the value typically set to 0 or, in some contexts, to a special value or infinity to denote the absence of an edge.

2.2 Graph Neural Networks

Traditional graph methods focus on extracting information directly from the graph's structure, leveraging mathematical and algorithmic approaches to identify key characteristics and relationships within the data. Some of the core methods used in traditional graph analysis are:

- Shortest Path Calculation: This involves finding the shortest paths between nodes in a graph, which is fundamental in applications like routing and navigation.

- Centrality Measures: These metrics identify the most important vertices within a graph.

- Clustering Coefficient: This measures the likelihood that two adjacent vertices of a vertex are connected. It provides a sense of the clustering in the whole network and indicates the degree to which nodes tend to cluster together.

However, these approaches have their limits due to the need for carefully hand-engineered statistics (Cai, Zheng, and Chang, 2018).

Graph Neural Networks (GNNs) offer a more dynamic and scalable solution to representation learning in graphs. The core principle behind GNNs is to learn node representations through iterative updating, where the representation of a node at each layer is refined by aggregating and combining the features of its neighboring nodes along with its own current features (Scarselli et al., 2008). This process allows GNNs to propagate and integrate information across the network, capturing both local and global structural information. Unlike traditional methods that operate with static features, GNNs adaptively learn the most relevant features for the task, making them highly effective for a wide range of applications, from social network analysis to bioinformatics and beyond (Hamilton, Ying, and Leskovec, 2017).

The functionality of GNNs can be explained through two main operations described by Wu et al. (2022):

- **Aggregate**: This operation refers to the process of collecting information from the neighbors of each node in the graph. This is crucial because the properties of a node in a graph are often highly influenced by its neighbors. The aggregate function collects the neighborhood information, creating a single output that encapsulates the overall context of each node's surroundings. It can involve simple operations like averaging or summing the features of neighboring nodes, or more complex functions such as using convolution operations or applying attention mechanisms.

- **Combine**: After the aggregate operation, the Combine operation is performed to update the representation of each node based on the aggregated information. It integrates the node's previous representation and the aggregated neighborhood information to create a

new, updated representation. The goal of the combine operation is to generate a comprehensive feature that contains information about the node itself as well as its relevant context within the graph. This operation can be as simple as concatenation followed by a linear transformation, or more complex mechanisms like gating or the application of non-linear functions can be employed.

Formally, the general framework of graph neural networks can be defined:

$$a_v^k = AGGREAGTE^k\{h_u^{k-1} : u \in N(v)\} \tag{2.2}$$

$$h_v^k = COMBINE^k\{h_v^{k-1}, a_v^k\} \tag{2.3}$$

where $h_v^{(k)}$ is the feature vector of node v at the k-th layer. And $h_v^{(0)} = \mathbf{X}$, and $N(v)$ is the set of neighbors for the v-th node.

After k iterations of aggregation, a node's representation captures the structural information within its k-hop network neighborhood (Xu et al., 2019). The node representation h^k in the last layer can be treated as the final node representation.

The choice of $AGGREGATE^{(k)}(.)$ and $COMBINE^{(k)}(.)$ determine different types of GNN arctectures.

Graph Convolutional Networks (GCN) (Berg, Kipf, and Welling, 2017) generalize the concept of convolution from grid data, such as images, to graph-structured data. The core idea behind GCNs is to apply a convolutional operation on the graph, enabling each node to aggregate features from its local neighborhood in a way that respects the structure of the graph. A typical layer of a GCN can be represented as:

$$\mathbf{H}^{(k+1)} = \sigma(\bar{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\bar{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(k)}\mathbf{W}^{(k)}) \tag{2.4}$$

Here, $\mathbf{H}^k$ is the matrix of node features at layer l, $\bar{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix of the graph $\mathbf{A}$ with added self-connections $\mathbf{I}$, $\bar{\mathbf{D}}$ is the degree matrix of $\bar{\mathbf{A}}$, σ is a non-linear activation

such as the ReLu, and $\mathbf{W}^{(k)}$ are potentially learnable parameters. The GCN moel model can be interpreted in terms of the Equations (2.2) and (2.3) as:

$$h_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \cdot \frac{\sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}}{|\mathcal{N}(v)|} + \mathbf{B}^{(k)} \cdot h_v^{(k-1)}\right), \tag{2.5}$$

where we have the mean aggregation of v's neighbour's embeddings at step $k-1$ linearly concatenate with node v's embedding at step $k-1$ and $\mathbf{W}^{(k)}$ and $\mathbf{B}^{(k)}$ are potentially learnable parameters.

Graph Attention Networks (GATs) (Veličković et al., 2017) are a type of Graph Neural Network that leverage attention mechanisms to specify different weights to different nodes in a neighborhood, allowing for more nuanced feature aggregation. The fundamental concept of GAT is to assign varying levels of importance to the nodes in the local neighborhood of a central node when aggregating their features, rather than treating all neighbors as equally important as in others GNN models. Graph Attention Network (GAT), which uses attention weights to define a weighted sum of the neighbors:

$$h_v^{(k)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu} \mathbf{W} h_u\right) \tag{2.6}$$

where α_{vu} denotes the attention on neighbor $u \in \mathcal{N}(v)$ when we are aggregating information at node v. In the original GAT paper, the attention weights are defined as:

$$\alpha_{vu} = \frac{\exp\left(\mathbf{a}^T[\mathbf{W}h_v \oplus \mathbf{W}h_u]\right)}{\sum_{u' \in \mathcal{N}(v)} \exp\left(\mathbf{a}^T[\mathbf{W}h_v \oplus \mathbf{W}h_{u'}]\right)}, \tag{2.7}$$

where $\mathbf{a}$ is a trainable attention vector, $\mathbf{W}$ is a trainable matrix, and $\oplus$ denotes the concatenation operation.

The attention coefficients indicate the relevance of the neighboring nodes' features to the central node, and these coefficients are used to weight the features accordingly during aggregation.

It is also possible to rewrite the GAT model in terms of Equations (2.2) and (2.3):

$$h_v^{(k)} = \sigma^{(k)} \left(\mathbf{W}^{(k)} \cdot \left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(k-1)} h_u^{(k-1)} + \alpha_{vv}^{(k-1)} h_v^{(k-1)} \right] \right) \tag{2.8}$$

In this equation $h_v^{(k)}$ is the feature vector of node v at iteration k, σ is a non-linear function, such as an activation function, α_{vu} is the attention coefficient between node v and node u. The brackets are used to denote concatenation of the aggregated neighbors features (h_u) and the node's own features (h_v).

2.3 Positional Encoder

The Transformer architecture, first introduced by Vaswani et al. (2017), represents a significant shift in the design of natural language processing (NLP) models. This groundbreaking approach moves away from traditional sequence-to-sequence models, focusing instead on self-attention mechanisms to capture relationships within sequential data. By using multi-layered stacks of attention modules, the Transformer architecture revolutionizes the way dependencies are managed, especially over long sequences. The "attention" concept allows the model to assign varying levels of importance to different elements in a sequence, providing a robust solution to the long-range dependency issues commonly faced by recurrent models. Additionally, the inclusion of positional encodings ensures the model can effectively process sequential order, establishing the Transformer as a cornerstone in modern NLP and achieving state-of-the-art results in various benchmarks.

Inspired by the Transformer architecture's success, particularly in its application to geographical data (Mai et al., 2020), PE-GNN (Klemmer, Safir, and Neill, 2023) incorporates a unique positional encoding (PE) comprising two key components: a sinusoidal transformation and a fully-connected neural network (NN). The sinusoidal transformation is a deterministic process that concatenates a series of sinusoidal functions, varying in frequency and scale, to

create a comprehensive spatial representation. Given a matrix $C_B = [c_1, ..., c_{n_B}]^T$, containing the spatial coordinates of a batch of data points, typically of dimension $n_B \times 2$, where each coordinate c_i represents a latitude-longitude pair, the sinusoidal transformation is defined as shown in Equation (2.9). It uses parameters σ_{min} and σ_{max}, which define the minimum and maximum grid scales, and S , which indicates the number of grid scales considered.

$$ST(C_B, \sigma_{\min}, \sigma_{\max}) = \begin{bmatrix} ST_0(C_B, \sigma_{\min}, \sigma_{\max}) \\ \dots \\ ST_{S-1}(C_B, \sigma_{\min}, \sigma_{\max}) \end{bmatrix}, \tag{2.9}$$

where

$$ST_s(C_B, \sigma_{\min}, \sigma_{\max}) = \begin{bmatrix} ST_{s,1}(C_B, \sigma_{\min}, \sigma_{\max}) \\ ST_{s,2}(C_B, \sigma_{\min}, \sigma_{\max}) \end{bmatrix}, \tag{2.10}$$

and

$$ST_{s,v}(C_B, \sigma_{\min}, \sigma_{\max}) = \begin{bmatrix} \cos\left(\frac{C_B^{[v]}}{\sigma_{\min} \cdot g^{s/(S-1)}}\right) \\ \sin\left(\frac{C_B^{[v]}}{\sigma_{\min} \cdot g^{s/(S-1)}}\right) \end{bmatrix}, \tag{2.11}$$

for all

$$s \in \{0, \dots, S-1\}, \quad v \in \{1, 2\}. \tag{2.12}$$

The first component of this transformation separates spatial dimensions (latitude and longitude) and handles them individually, while the second component, the fully connected NN, processes the output of the sinusoidal transformation to produce a vector-space representa-

tion. This transformation culminates in the coordinate embedding matrix, denoted as $C_{emb} = PE(C_B, \sigma_{min}, \sigma_{max}, \Theta_{PE})$, where Θ_{PE} epresents the parameters of the fully-connected NN. This embedding provides a critical spatial context for the PE-GNN model, allowing it to accurately understand and predict based on geographical relationships.

2.4 Positional Encoder Graph Neural Network

The Positional Encoder Graph Neural Network (PE-GNN) model (Klemmer, Safir, and Neill, 2023) incorporates a positional encoder (PE) (Vaswani et al., 2017; Mai et al., 2020), designed to learn a contextual embedding for point coordinates throughout the training process. This approach is flexible and modular, designed for predictive modeling with geographical data. It supports any GNN backbone and incorporates multiple innovations that enhance traditional GNN methods when dealing with spatial data.

In traditional GNN approaches with geographical data, coordinates are mainly used to compute distances for constructing a graph with a set number of nearest neighbors. Once this graph is built, the traditional approach no longer utilizes spatial location. PE-GNN, on the other hand, goes a step further. In addition to using coordinates to construct the graph, it learns a spatial embedding through a PE. The PE processes Each pair of coordinates, producing a vector representing the spatial context. This vector is then concatenated with node features before applying the GNN operator. For a given batch B of randomly sampled data points, the input to the first GNN layer is:

$$\mathbf{H}^{(0)} = concat(\mathbf{X}, \mathbf{C}_{emb}), \tag{2.13}$$

where $\mathbf{X}$ represents the node features and $\mathbf{C}_{emb}$ is the spatial embedding obtained from the PE.

The method also added an auxiliary task alongside the main task during training. While traditional GNN approaches output the predicted target variable for each node, PE-GNN introduces a parallel prediction task for Moran's I, a measure of spatial autocorrelation (Klemmer

and Neill, 2021). The Local Moran's I is computed for the target variable y_i, incorporating the spatial context. This additional task provides further information to the model, improving its learning process. The Local Moran's I is given by

$$I_i = (n-1) \cdot \frac{(y_i - \bar{y})}{\sum_{j=1}^{n}(y_j - \bar{y})^2} \cdot \sum_{j=1, j\neq i}^{n} a_{i,j} \cdot (y_j - \bar{y}), \tag{2.14}$$

where $\bar{y}$ is the sample mean of y and $a_{i,j} \in A$ denotes adjacency of observations i and j.

The third innovation in PE-GNN lies in the batch-based training procedure. At each training step, a random batch B of nodes is sampled, and the entire process of constructing the training graph, generating spatial embeddings, concatenating with node features, and applying the GNN operator is carried out using only this batch. This method helps minimize issues like Moran's I scale sensitivity by varying the neighborhood of each data point at every training step. Additionally, this strategy allows PE-GNN to learn a more generalizable spatial embedding because a data point can have a different neighborhood in each training step. This approach contrasts with the unsupervised method proposed by (Mai et al., 2020), with the PE being jointly learned with the other parameters of PE-GNN.

The loss function used in PE-GNN is based on Mean Squared Error (MSE) and incorporates the auxiliary task with a weight parameter λ. The loss is calculated using:

$$L_B = \text{MSE}(\hat{y}_B, y_B) + \lambda \cdot \text{MSE}(I(\hat{y}_B), I(y_B)), \tag{2.15}$$

where $MSE(\hat{y}_B, y_B)$ is the mean squared error between the predicted and actual target values, and $MSE(I(\hat{y}_B), I(y_B))$ accounts for the auxiliary task, with λ as its weight.

The positional encoding in PE-GNN involves creating spatial embeddings based on global spatial relationships. This could introduce scale sensitivity, where the model's performance might vary depending on the spatial distribution of data or geographical scope.

Chapter 3

Localized Ensemble Positional Encoder Graph Neural Network

In this chapter, is described the Localized Ensemble Positional Encoder Graph Neural Network (LEPE-GNN) method. All innovations are listed below and Algorithm 1 presents the step-by-step procedure to train an ensemble of geographically localized models.

Considering a set of data points belonging to a training dataset S_{train}: $p_i = \{y_i, \mathbf{x}_i, \mathbf{c}_i\}$ for $i = 1, ..., n$ where y_i is the target value, $\mathbf{x}_i$ are the features, $\mathbf{c}_i$ are the geographical coordinates associated with observation i.

3.1 Efficient Choice of Localized Model Locations

The first innovation is to cluster the geographical coordinates $\mathbf{C}$ using a Gaussian Mixture Model (GMM) into a predefined number K of exhaustive and mutually exclusive zones. This probabilistic model treats each cluster as a Gaussian distribution, and points are probabilistically assigned to a cluster based on their geographical proximity. Then each point is associated with a cluster label $k_i = 1, ..., K$.

3.2 Dimensionally Reduced Distance Matrix

The next step is to calculate the distances between the centroids of the clusters k_i rather than between every individual point. This approach significantly reduces the computational load by constructing a manageable $K \times K$ distance matrix $\mathbf{D}$ where $d_{j,m}$ represents the great-circle distance between centroid j and centroid m.

The great-circle distance is calculated as follows:

$$d_{j,m} = r \times \theta, \tag{3.1}$$

where r is the radius of the Earth and θ is the angular distance in radians, computed as:

$$\theta = 2 \times \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1) \times \cos(\phi_2) \times \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right), \tag{3.2}$$

here, ϕ_1 and ϕ_2 are the latitudes of centroids j and m in radians, and λ_1 and λ_2 are the longitudes of centroids j and m in radians. This method accurately accounts for the curvature of the Earth and provides a precise measurement of the distance between points based on their geographical coordinates.

Those distances are then used during the training stage in a custom loss function, our third innovation.

3.3 Weighted Loss Function

Unlike traditional loss functions that consider all points equally, this function uses a weighting mechanism to prioritize data points that are geographically closer to the cluster being trained. This process involves the following:

Each point in a training dataset has an associated distance from its cluster centroid to the centroid of the cluster being trained. These distances are normalized using a bandwidth parame-

ter, which controls the influence range. The Epanechnikov kernel, a function that assigns higher weights to points with smaller normalized distances, is then used to generate the weights. The weight w for an observation i when training the model j is calculated as:

$$w_{i,j} = \begin{cases} \frac{3}{4} \times (1 - u_{i,j}^2), & \text{if } |u_{i,j}| \leq 1 \\ 0, & \text{otherwise} \end{cases}, \tag{3.3}$$

where $u_{i,j} = \frac{d_{m_k,j}}{h}$ with $d_{m_k,j}$ being the distance between the data point's centroid and the centroid of the cluster being trained, and h representing the bandwidth parameter. The bandwidth affects how weights are distributed across the data points. A smaller bandwidth leads to weights that decrease rapidly with distance, emphasizing closer points more strongly. A larger bandwidth gives more uniform weights, reducing the impact of distance.

The loss function used to calculate the weighted mean squared error (WMSE) for a model j is then defined as:

$$L_j = \frac{1}{N} \sum_{i=1}^{N} w_{i,j} \times (y_i - \hat{y}_{i,j})^2, \tag{3.4}$$

where $w_{i,j}$ is the weight derived from the Epanechnikov kernel based on the normalized distance, y_i is the actual target value, and $\hat{y}_{i,j}$ is the predicted value from the model j.

3.4 Utilization of Pre-Trained Weights

We can start the training with random weights or introduce a pre-trained global PE-GNN that has already learned comprehensive spatial relationships across the entire dataset, out third innovation. This pre-trained model serves as a foundational framework, providing a robust starting point that encapsulates a broad understanding of spatial dynamics. By leveraging these pre-trained weights, each localized model is equipped with an advanced baseline from which further, more focused learning can occur. This approach accelerates the initial training phase, as

Algorithm 1 Ensemble Model Training and Prediction

Require:

- Dataset $\mathbf{S}$ with target ($\mathbf{y}_{(n\times 1)}$), features ($\mathbf{X}_{(n\times p)}$) and coordinates ($\mathbf{C}_{(n\times 2)}$) matrices
- Number of clusters K
- Bandwidth parameter h
- Number of training epochs (n_E), the batch size (n_B), and learning rate (α)

Split dataset $\mathbf{S}$ into training $\mathbf{S}_{train}$ and validation $\mathbf{S}_{validation}$ sets.
Perform clustering on the training geographical coordinates using matrix $\mathbf{C}_{train}$ to create K clusters.
Assign each data point of $\mathbf{S}_{train}$ and $\mathbf{S}_{validation}$ to a cluster.
Compute the great-circle distances between all cluster centroids to create a $K \times K$ matrix $\mathbf{D}_{centroids}$.
Initialize a set of models M_K individual PE-GNN models.
if A global model is provided **then**
    Load the pre-trained weights for each model M_K.
end if
for epoch from 1 to n_E **do**
    Sample minibatch B of n_B datapoints: $\mathbf{X}_{B(n_B\times p)}$, $\mathbf{C}_{B(n_B\times p)}$, $\mathbf{y}_{B(n_B\times 1)}$.
    for each model **do**
        Compute weights $w_{i,k}$ for each data point in n_B based on distances to the cluster centroid of the model M_i and bandwidth parameter h.
        Calculate the loss for the model using weighted mean squared error (MSE): $L_k = \frac{1}{N}\sum_{i=1}^{N} w_{i,k} \times (y_i - \hat{y}_{i,k})^2$.
        Perform gradient descent to update learner model parameters with learning rate α.
    end for
    Calculate ensemble predictions for the validation set $\mathbf{S}_{validation}$: $\hat{y}_{i,k} = \sum_{k=1}^{M} w_{i,k}\hat{y}_{i,k} / \sum_{k=1}^{M} w_{i,k}$.
    Compute $\mathbf{S}_{validation}$ mean squared error.
    if early stopping condition is met **then**
        Break the training loop to avoid overfitting.
    end if
end for
Output: The ensemble model parameters $\mathbf{\Theta} = \theta_1, \ldots, \theta_K$,$\mathbf{D}_{\text{centroids}}$

the models begin with an informed perspective, reducing the time and computational resources required to reach convergence.

3.5 Ensemble of Local Models

After each training round, the models are combined using an ensemble method (E) that generates predictions from each model and then calculates a weighted average to create a single prediction for the ensemble. This weighted averaging process helps to smooth out errors and reduce the impact of any single model's inaccuracies by assigning more weight to closer models in terms of their distance in the cluster.

Mathematically, if $\hat{y}_{i,j}$ is the prediction from the jth model for the ith data point and $w_{i,j}$ is the weight assigned to the jth model's prediction, then the ensemble's final prediction $\hat{y}_{i,j}$ is given by:

$$\hat{y}_{i,j} = \frac{\sum_{j=1}^{M} w_{i,j}\hat{y}_{i,j}}{\sum_{j=1}^{M} w_{i,j}}. \tag{3.5}$$

The Mean Squared Error (MSE) metric is used to evaluate the performance of the ensemble.

Finally, a custom early-stopping strategy ensures that the models are trained just enough to achieve optimal performance on the validation dataset without overfitting. This strategy involves monitoring the Mean Squared Error (MSE) of the ensemble E on the validation dataset after each training epoch. It records the best MSE observed and employs a patience parameter, which determines the number of consecutive epochs without significant MSE improvement (beyond a minimal δ threshold) before halting the training.

The method presented in this chapter combines clustering with advanced graph neural network techniques to boost spatial data analysis. This approach leverages the similarities in geographical and feature characteristics shared between neighboring clusters, allowing models to learn more robust and broadly applicable patterns.

Chapter 4

Experimental Setup and Results

4.1 Experimental Setup

In this section, the experimental setup is designed to evaluate the performance of the proposed method configurations. This evaluation targets the predictive power of the LEPE-GNN method compared to the standard PE-GNN method (Klemmer, Safir, and Neill, 2023), and an XGB-Model (Chen et al., 2015). The spatial correlation of the residuals for each method is also calculated.

To allow for a fair comparison between the different approaches, the two methods based on neural networks (PE-GNN, LEPE-GNN), consist of two GCN layers with ReLU activation and dropout, followed by linear layer regression heads. We test both approaches with two auxiliary task weights $\lambda = 0, 0.5$, where $\lambda = 0$ implies no auxiliary task as implemented by Klemmer, Safir, and Neill (2023). Training for the GNN models is conducted using PyTorch library in Python software (Paszke et al., 2019) and PyTorch Geometric (Fey and Lenssen, 2019). The Adam algorithm (Kingma and Ba, 2014) is used to optimize the models and the mean squared error (MSE) loss. All training is conducted on a single CPU.

The XGBoost model was trained with and without the geographical information (latitude/-longitude) and optimized using the Optuna package (Akiba et al., 2019) for each case.

To assess the degree of spatial correlation of the models the Global Moran's I (Moran, 1950) index was calculated for the datasets and the residuals of the models.

A sensitivity analysis is also performed to analyze how different parameter settings affect the performance of the LEPE-GNN method. We systematically vary key parameters such as the number of clusters, the bandwidth values, and the use of pre-trained weights, as discussed below.

4.1.1 Using Pre-Trained Weights of a global PE-GNN

To improve training efficiency and model stability, we use pre-trained weights from a previously PE-GNN model trained with 30 epochs without clustering. These weights serve as a starting point for the training process, reducing the overall training time and providing a more stable foundation. This approach allows us to evaluate whether reusing pre-trained weights improves the model's convergence and generalization.

4.1.2 Different Number of Clusters

To assess the influence of the number of clusters on the model's performance, we conduct experiments with different cluster counts (10, 30, 50, and 100 clusters). This variation helps determine the optimal level of spatial granularity required for the best results.

4.1.3 Bandwidth

The bandwidth parameter plays a crucial role in the weighting mechanism of the loss function. It determines the range of influence that each cluster has on the model's training. In this setup, different bandwidth values are tested to explore their impact on the model's ability to learn from spatial relationships. A smaller bandwidth emphasizes closer points, while a larger bandwidth encompasses a broader context.

4.1.4 Data

The method is going to be evaluated in two real data datasets that contain geographical data:

- California Housing: This dataset contains the prices of over 20,000 California houses from the 1990 U.S. census (Pace and Barry, 2003). The regression task at hand is to predict house prices using the following features:
  - MedInc: average household income in the census block.
  - HouseAge: average age of the houses in the block.
  - AveRooms: average number of rooms per household.
  - AveBedrms: average number of bedrooms per household.
  - Population: total number of people residing in the block.
  - AveOccup: average number of occupants per household.
  - Latitude: latitude of the block.
  - Longitude: longitude of the block.
- Air temperature: The air temperature dataset (Hooker, Duveiller, and Cescatti, 2018) contains the coordinates of 3,000 weather stations around the globe. This regression task is to predict mean temperatures from a single feature, mean precipitation, and location (latitude/longitude).

Both datasets are divided into 60% train, 20% validation, and 20% test and all experiments were conducted on a single CPU (Mac M2 with 8GB of RAM).

4.2 Results

4.2.1 Predictive Performance

This section presents the findings from the experimental evaluation of localized models (LEPE-GNN) versus non-localized models (XGBoost and PE-GNN) and the influence of incorporating

an auxiliary task through varying values of the lambda λ parameter using the California Housing Dataset and the Air Temperature Dataset. The primary metrics for evaluation were Mean Squared Error (MSE) and Mean Absolute Error (MAE) which provide insights into the models' accuracy and predictive performance.

The results indicate that localizing the model training to specific geographic clusters does not significantly enhance the model's performance in lowering the MSE or MAE for the California Housing dataset (Table 4.1). However, the LEPE-GCN models demonstrate a notable performance improvement for the Air Temperature dataset. The localized models, especially with $\lambda = 0.5$ achieve the lowest MSE (0.0033) and MAE (0.0394) among the configurations tested. This indicates that localizing the training process to geographic clusters for datasets with pronounced spatial patterns can significantly improve model accuracy.

Table 4.2 shows the parameter used by the LEPE-GNN in each dataset. Both of them were trained per 500 epochs and bandwidth of $\frac{\text{max}}{2}$, where max is the maximum distance between clusters.

Table 4.3 shows the values of Global Moran's I for each dataset, both of them have a large spatial correlation on the response variable. Table 4.4 shows the values of the Moran's I for the residuals of the three best models models with the PE-GNN model with the lower values in the California Housing Dataset and the LEPE-GNN model showing the lower values for the Air temperature dataset.

Figure 4.1 shows the performance of three models: PE-GNN, LEPE-GNN, and LEPE-GNN with pre-trained weights, over 250 epochs, measured by the root mean squared error (RMSE). We see that the LEPE-GNN models stabilize faster compared to the standard PE-GNN approach. The flatter trajectory of the RMSE curves for the LEPE-GNN models beyond the initial epochs evidences this. These models also exhibit less variation in error as training progresses, indicating a more consistent and reliable performance.

When analyzing the residuals of the two GNN models in Figure 4.2 the models perform similarly for the California House Dataset. However, we see smaller residuals for the LEPE-

Table 4.1: Comparison of Model Evaluation Metrics for the California Housing Dataset and Air Temperature Dataset

Model	**Cali. Housing**		**Air Temp**	
	MSE	**MAE**	**MSE**	**MAE**
XGBoost without lat/log	0.0158	0.0920	0.0256	0.1248
XGBoost with lat/log	**0.013**	**0.0759**	0.0059	0.0530
PE-GCN $\lambda = 0$	0.0158	0.0892	0.0049	0.0542
PE-GCN $\lambda = 0.5$	0.0160	0.0887	0.0045	0.0492
LEPE-GCN $\lambda = 0$	0.0161	0.0894	0.0034	0.0399
LEPE-GCN $\lambda = 0.5$	0.0163	0.0894	**0.0033**	**0.0394**

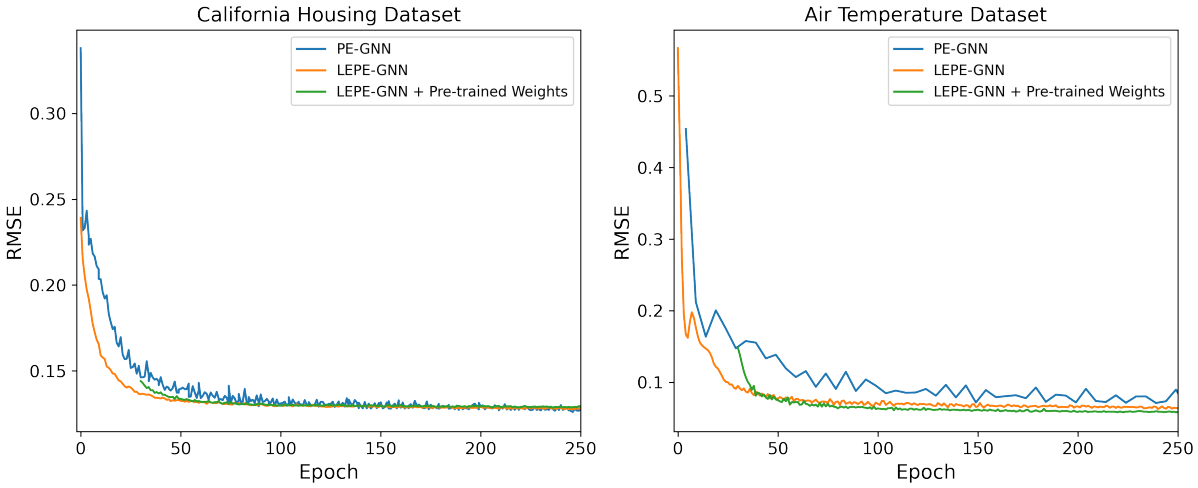

Figure 4.1: Validation error curves of PE-GCN, LEPE-GCN, and LEPE-GCN with pre-trained weights, measured by the RMSE metric for the California Housing Dataset (Left) and Air Temperature Dataset (Right).

Table 4.2: LEPE-GNN Parameters

Dataset	Number of Clusters	Batch Size
California Housing	30	512
Air Temperature	20	256

Table 4.3: Comparison of Global Moran's I for the Datasets

Dataset	Global Moran's I
Cali. Housing	0.7096
Air Temp	0.8339

Table 4.4: Comparison of Global Moran's I for the Residuals of Different Models

	Cali. Housing	Air Temp
XGBoost	0.09353	0.51649
PE-GNN	**0.06501**	0.39198
LEPE-GNN	0.11792	**0.17278**

GNN in the Air Temperature Dataset.

The Kruskal-Wallis test (Hollander, Wolfe, and Chicken, 2013) was performed for both datasets and for the Air Temperature Dataset the distribution between the two models is statistically different at 5% significance level.

In Figure 4.3 we compare two model predictions against real data values and against each other, highlighting geographic areas where model performances diverge significantly.

In the bottom right panel, we see the difference between the residues of the two models calculated as:

$$\Delta_r = |residuals_{\text{PE-GNN}}| - |residuals_{\text{LEPE-GNN}}|\,, \tag{4.1}$$

here red areas denote regions where the LEPE-GNN model exhibits smaller residuals than the PE-GNN, indicating superior predictive accuracy by the localized model in these areas. Conversely, blue areas highlight regions where the PE-GNN model outperforms the Localized PE-GNN, with smaller residuals and higher accuracy. The presence of grey or neutral areas

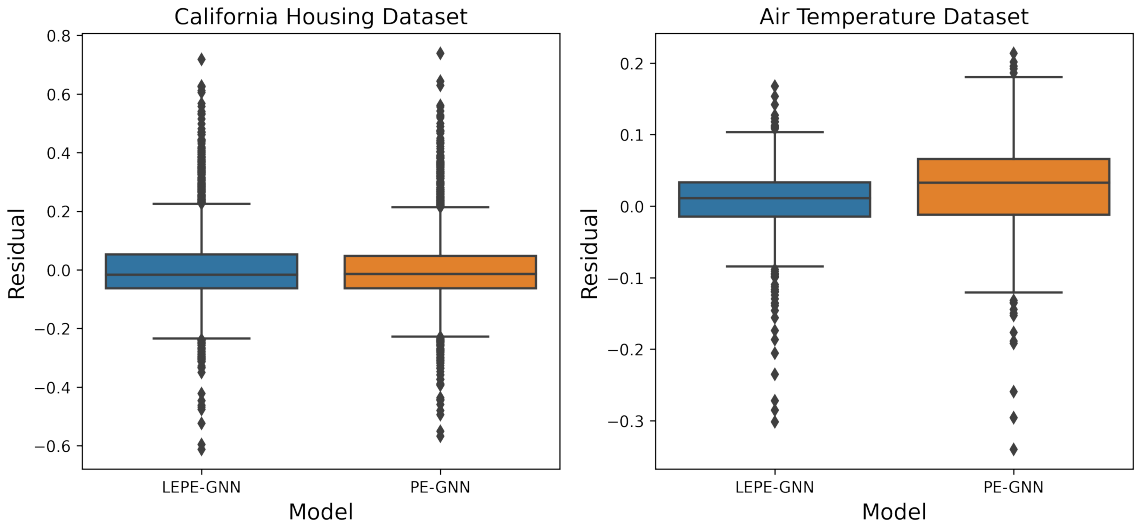

Figure 4.2: Comparison of residuals for the two GNN methods for the California Housing Dataset (Left) and Air Temperature Dataset (Right).

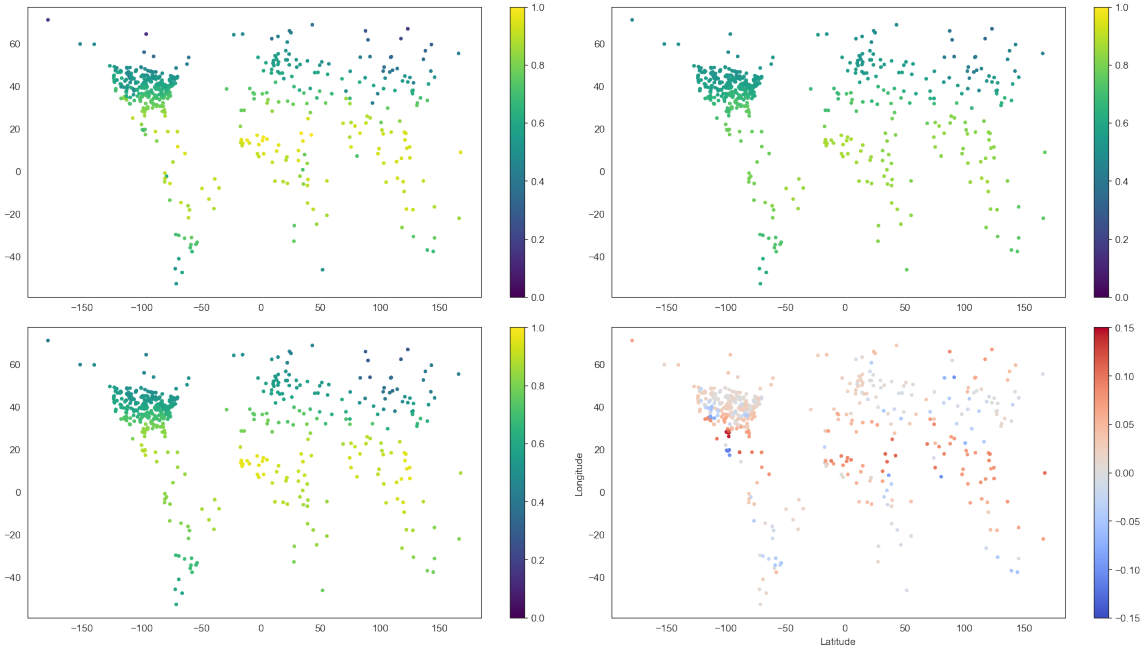

Figure 4.3: Real Values (Top Left), Predictions from PE-GNN (Top Right), Predictions from LEPE-GNN (Bottom Left), Difference in Residual Magnitudes (PE-GNN - LEPE-GNN) (Bottom Right) for the Air Temperature Dataset.

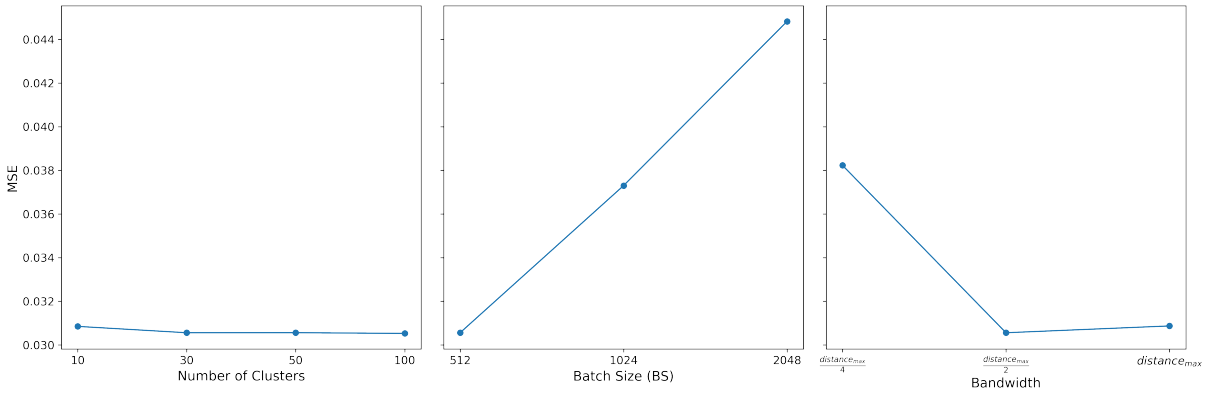

Figure 4.4: Sensitivity analysis: Number of clusters (left), Batch Size (Middle), Bandwidth (Right) for the California Housing Dataset

suggests comparable performance between the two models in those locations. This differential performance underscores the potential advantage of localized modeling in capturing complex, localized patterns within the data.

4.2.2 Sensitivity Analysis

This subsection focuses on understanding how different parameter settings influence our model's performance, here we analyze the effects of changing the bandwidth parameter of the loss function, the number of clusters, and the batch size of the localized model. This study was conducted with the California Housing Dataset, all models were trained using 100 epochs.

The left panel of Figure 4.4 shows the MSE as a function of the number of clusters used in the model. All models were trained with a batch size of 512 and bandwidth of $\frac{distance_{max}}{2}$. We see that the MSE decreases sharply with an increase in the number of clusters from 10 to 30, stabilizes between 30 and 50, and reaches its lowest at 100 clusters, indicating that more clusters can improve model performance but the gains can be limited when used the same configuration of bandwidth for different number of clusters.

Regarding the choice of the batch size the Figure 4.4 - middle panel, illustrates the impact of increasing batch size on MSE, calculated using a fixed number of 30 clusters and bandwidth of $\frac{distance_{max}}{2}$. As batch size increases from 512 to 2048, the MSE steadily rises, suggesting that

larger batch sizes may degrade model performance. Klemmer, Safir, and Neill (2023) uses 2048 for the California Housing dataset, but we see in the figure that the localized approach benefits from a smaller size.

The right panel displays the MSE as a function of bandwidth, defined as fractions of the maximum distance, meaning that values in the right have more influence on more distant points, with the analysis conducted using 30 clusters and a batch size of 512. A significant decrease in MSE is observed as the bandwidth increases from a quarter of the maximum distance to half, followed by a plateau when using the maximum distance, indicating that larger bandwidths yield better performance up to a certain threshold when keeping the number of clusters and batch size constant.

4.2.3 Computational Time

This subsection presents how the computational time of the LEPE-GNN is affected by the different number of clusters and compares it with the PE-GNN model (1 cluster).

Tables 4.5 and 4.6 show that as we increase the number of clusters (localized models) we increase the computational time necessary to train the model.

Table 4.5: Computational Time for the California Housing Dataset

Number of Clusters	Computational Time
1	2min 30s
10	22min 5s
30	1h 6min 45s
50	1h 50min 26s
100	3h 40min 45s

Table 4.6: Computational Time for the Air Temperature Dataset

Number of Clusters	**Computational Time**
1	9.81 s
10	2min 49s
30	8min 14s
50	13min 40s
100	28min 22s

Chapter 5

Conclusion

In this dissertation, a novel approach was developed, that utilizes localized models to improve the predictive accuracy and generalization of graph neural networks applied to spatial data. By segmenting spatial data into clusters and training dedicated models for each segment, the limitations commonly faced by global models due to the complex nature of spatial autocorrelation and heterogeneity are addressed.

Both localized (LEPE-GNN) and non-localized configurations of the PE-GCN model exhibit similar error metrics for the California Housing dataset, which suggests that the benefits of localized training might be constrained by other factors such as the intrinsic characteristics of the dataset, the clustering methodology, or possibly the scale of spatial variability present within the data.

In terms of MSE values, the XGBoost model achieved the lower value for the California Dataset but showed the highest Moran's I value for the residuals, indicating that while non-geographical models can perform well they can't capture and deal with the spatial autocorrelation like the geographical models can.

For the Air Temperature Dataset we obtained a significant increase in performance, the LEPE-GNN model showed a bigger representation capacity and the smallest Moran's I value.

The sensitivity analysis shows that increasing the number of clusters or the bandwidth gen-

erally enhances model accuracy, but only up to a certain point. Beyond specific thresholds, these improvements plateau or even decrease. This pattern implies that simply escalating these parameters without considering their interaction can result in suboptimal results. Hence, a more sophisticated method for selecting these values is necessary. Techniques like multi-parameter optimization algorithms, grid search, or even more advanced methods such as Bayesian optimization could be useful. These strategies enable simultaneous adjustments of multiple parameters and assess their effect on model performance.

The computational time needed to train the LEPE-GNN increases as more local models are added, and it also depends on the size of the dataset. For both datasets, the number of local models was selected that gave the smallest error metrics and in a feasible time.

The method's main limitation is the number of hyperparameters that the user should choose, such as the number of clusters and the bandwidth beyond the other parameters from the PE-GNN: batch size for example.

For future works, the suggestions are:

- Exploring alternative clustering techniques and shifting from clustering geographical coordinates to using the spatial embeddings generated by PE-GNNs. This change aims to leverage the spatial information encoded in embeddings, potentially uncovering deeper insights and relationships within the data.

- Using hyperparameters tunning techniques like Bayesian optimization to choose the best setting for a given dataset.

- The LEPE-GNN results were generated with all innovations, an ablation study should be conducted to measure the contribution of each individual innovation.

References

Akiba, Takuya et al. (2019). "Optuna: A next-generation hyperparameter optimization framework". *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631.

Barabási, Albert-László (2016). *Network Science*. Cambridge University Press.

Berg, Rianne van den, Kipf, Thomas N, and Welling, Max (2017). "Graph convolutional matrix completion". *arXiv preprint arXiv:1706.02263*.

Bivand, Roger S et al. (2008). *Applied spatial data analysis with R*. Vol. 747248717. Springer.

Cai, Hongyun, Zheng, Vincent W, and Chang, Kevin Chen-Chuan (2018). "A comprehensive survey of graph embedding: Problems, techniques, and applications". *IEEE transactions on knowledge and data engineering* 30.9, pp. 1616–1637.

Chen, Tianqi et al. (2015). "Xgboost: extreme gradient boosting". *R package version 0.4-2* 1.4, pp. 1–4.

Fey, Matthias and Lenssen, Jan Eric (2019). "Fast graph representation learning with PyTorch Geometric". *arXiv preprint arXiv:1903.02428*.

Hamilton, William L, Ying, Rex, and Leskovec, Jure (2017). "Representation learning on graphs: Methods and applications". *arXiv preprint arXiv:1709.05584*.

Hollander, Myles, Wolfe, Douglas A, and Chicken, Eric (2013). *Nonparametric statistical methods*. John Wiley & Sons.

Hooker, Josh, Duveiller, Gregory, and Cescatti, Alessandro (2018). "A global dataset of air temperature derived from satellite remote sensing and weather stations". *Scientific data* 5.1, pp. 1–11.

Kingma, Diederik P and Ba, Jimmy (2014). "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980*.

Klemmer, Konstantin and Neill, Daniel B (2021). "Auxiliary-task learning for geographic data with autoregressive embeddings". *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, pp. 141–144.

Klemmer, Konstantin, Safir, Nathan S, and Neill, Daniel B (2023). "Positional encoder graph neural networks for geographic data". *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 1379–1389.

Mai, Gengchen et al. (2020). "Multi-scale representation learning for spatial feature distributions using grid cells". *arXiv preprint arXiv:2003.00824*.

Moran, Patrick AP (1950). "Notes on continuous stochastic phenomena". *Biometrika* 37.1/2, pp. 17–23.

Nikparvar, Behnam and Thill, Jean-Claude (2021). "Machine learning of spatial data". *ISPRS International Journal of Geo-Information* 10.9, p. 600.

Pace, R Kelley and Barry, R (2003). "Spatial statistics toolbox 2.0". *FDELW2. m. February* 15.

Paszke, Adam et al. (2019). "Pytorch: An imperative style, high-performance deep learning library". *Advances in neural information processing systems* 32.

Scarselli, Franco et al. (2008). "The graph neural network model". *IEEE transactions on neural networks* 20.1, pp. 61–80.

Tobler, Waldo R (1970). "A computer movie simulating urban growth in the Detroit region". *Economic geography* 46.sup1, pp. 234–240.

Vaswani, Ashish et al. (2017). "Attention is all you need". *Advances in neural information processing systems* 30.

Veličković, Petar (2023). "Everything is connected: Graph neural networks". *Current Opinion in Structural Biology* 79, p. 102538.

Veličković, Petar et al. (2017). "Graph attention networks". *arXiv preprint arXiv:1710.10903*.

Xu, Keyulu et al. (2019). "How Powerful are Graph Neural Networks?" *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=ryGs6iA5Km.

Yin, Yifang et al. (2019). "Gps2vec: Towards generating worldwide gps embeddings". *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 416–419.