DSA5208 Project 1: Distributed Kernel Ridge Regression with MPI

S K Ruban A0253837W
Owen Li Dong Lin A0231088H

October 2024

1 Introduction

This project focuses on predicting median house values in California using data from the 1990 Census. Our objective is to implement a kernel ridge regression model to accurately predict house values based on features such as location, demographics and economic indicators.

Given that the dataset contains over 20,000 samples, we implement the Message Passing Interface (MPI) to compute the kernel ridge regression in parallel. This approach allows us to process computations on the large dataset across multiple processes, allowing our computation operations to scale efficiently across massive dataset sizes. The effectiveness of the kernel ridge regression is evaluated using the Root Mean Square Error (RMSE) on both training and test datasets.

2 Preliminary Data Analysis

Prior to implementing the kernel ridge regression, we conducted an initial exploration of the housing dataset to understand its characteristics. The dataset, which was fortunately pre-cleaned, consists of 20,433 samples with 10 features (including one categorical variable) and no missing values. Our analysis revealed that the median house value ranges from $14,999 to $500,001, with a mean of $206,864 and a standard deviation of $115,435.

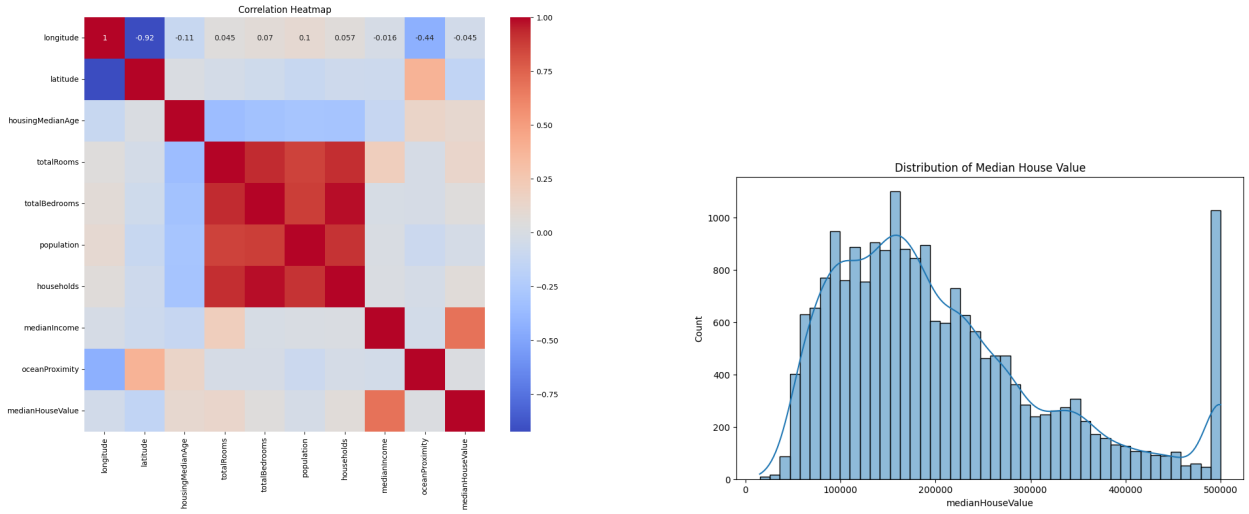


Figure 1: Correlation Heatmap and Distribution of Median House Values.

The correlation heatmap (Figure 1, Left) highlights that median income has the strongest positive correlation (0.69) with median house value. Total rooms, total bedrooms, population, and households also show

moderate positive correlations. Interestingly, longitude exhibits a slight negative correlation (-0.045) with house value, suggesting a minor tendency for prices to decrease from west to east.

The distribution of median house values (Figure 1, Right) is right-skewed with a peak around $150,000 and a notable spike at the maximum value of $500,001. This spike likely indicates data censoring, where values above $500,000 were capped. We will address this censoring issue in our modeling approach by adjusting predictions that exceed this threshold.

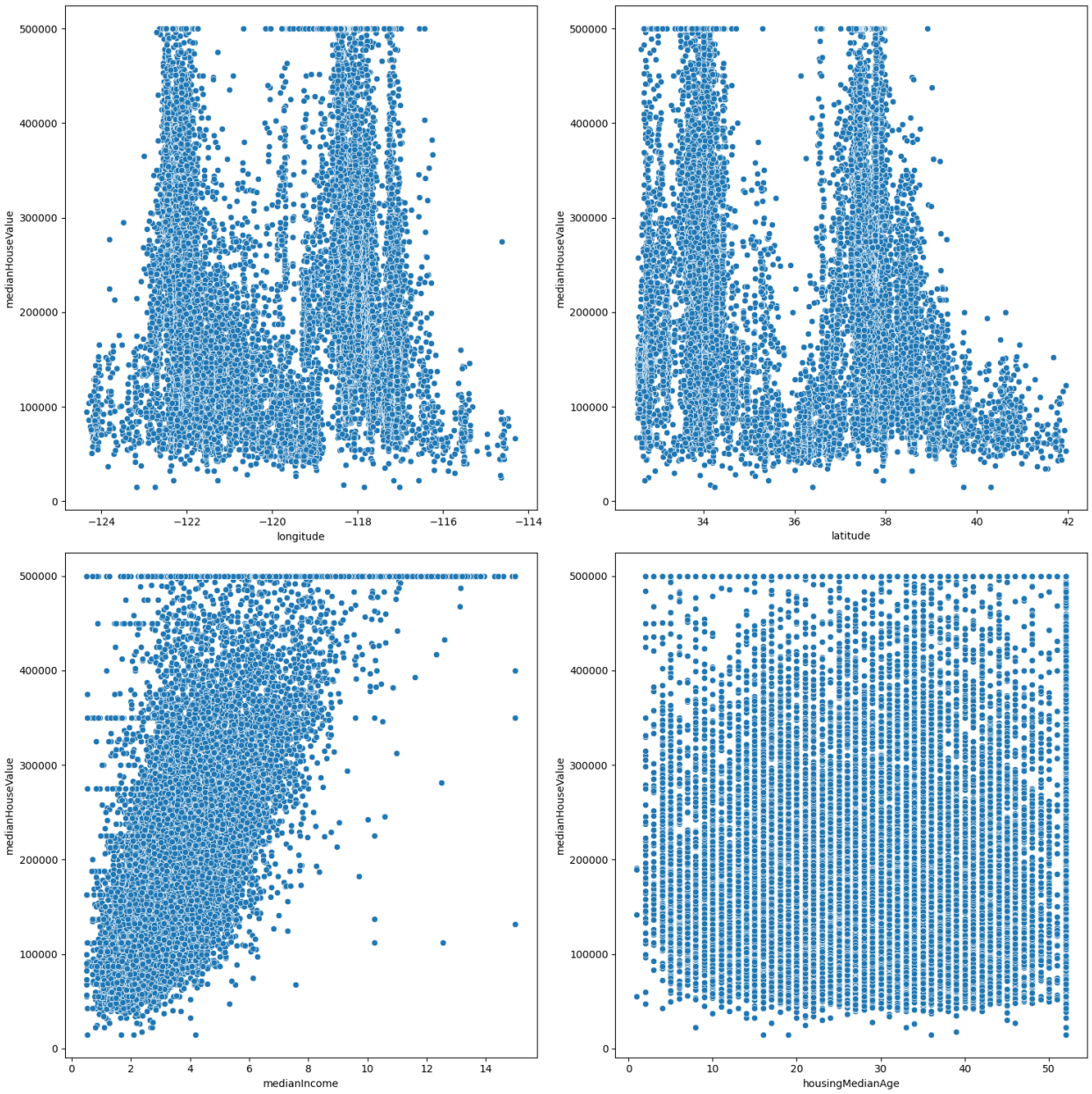
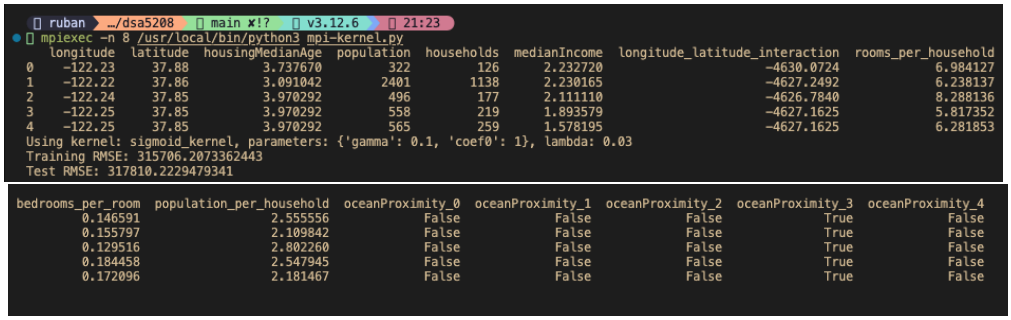


Figure 2: Comparison of House Value with other features.

Scatter plots (Figure 2) further illustrate these relationships, with median income demonstrating the clearest positive trend with house value. Geographic variables (longitude and latitude) display clustered patterns, reflecting the concentration of high-value properties in specific regions of California. The housing median age shows a weak relationship with house value, suggesting that other factors may be more influential.

3 Methodology

We begin with data preparation and feature engineering. The project uses a cropped version of the housing dataset with 20,000 samples so that it can be distributed equally among an even number of processors even after splitting the data. The dataset was then preprocessed by engineering new features such as longitude-latitude interaction, rooms per household, bedrooms per room, and population per household. To address potential non-linear relationships, we apply log transformations to median income and housing median age. To mitigate multicollinearity, we remove the total rooms and total bedrooms features. The categorical variable 'ocean proximity' is converted to one-hot encoding to allow its use in the model.



```
ruban …/dsa5208 main ✘!? v3.12.6 21:23
mpiexec -n 8 /usr/local/bin/python3 mpi-kernel.py
   longitude  latitude  housingMedianAge  population  households  medianIncome  longitude_latitude_interaction  rooms_per_household
0    -122.23     37.88          3.737670         322         126      2.232720                        -4630.0724             6.984127
1    -122.22     37.86          3.091042        2401        1138      2.230165                        -4627.2492             6.238137
2    -122.24     37.85          3.970292         496         177      2.111110                        -4626.7840             8.288136
3    -122.25     37.85          3.970292         558         219      1.893579                        -4627.1625             5.817352
4    -122.25     37.85          3.970292         565         259      1.578195                        -4627.1625             6.281853
Using kernel: sigmoid_kernel, parameters: {'gamma': 0.1, 'coef0': 1}, lambda: 0.03
Training RMSE: 315706.2073362443
Test RMSE: 317810.2229479341
```

bedrooms_per_room	population_per_household	oceanProximity_0	oceanProximity_1	oceanProximity_2	oceanProximity_3	oceanProximity_4
0.146591	2.555556	False	False	False	True	False
0.155797	2.109842	False	False	False	True	False
0.129516	2.802260	False	False	False	True	False
0.184458	2.547945	False	False	False	True	False
0.172096	2.181467	False	False	False	True	False

Figure 3: data.head() of features after processing.

The prepared dataset is then split into training (70%) and test (30%) sets. To ensure optimal performance of our KRR model, we standardize both the features and the target variable using StandardScaler. This scaling step is crucial as it brings all features to a comparable scale and often leads to faster convergence in gradient-based optimization methods. We also tried other scalars like the MinMaxScalar and RobustScalar, but the StandardScaler performed best.

The core of our methodology is the implementation of Kernel Ridge Regression (KRR). KRR combines Ridge Regression with the kernel trick, allowing us to capture non-linear relationships in the data. The model can be formulated as:

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i)$$

where $K(x, x_i)$ is the kernel function, x_i are the training samples, and α_i are the model parameters to be learned. We employ a Gaussian kernel, defined as:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

where σ is a hyperparameter controlling the kernel width.
The optimization problem for KRR can be expressed as:

$$\min_{\alpha} \|y - K\alpha\|^2 + \lambda \alpha^T K \alpha$$

where K is the kernel matrix, y are the target values, and λ is the regularization parameter.

To solve this optimization problem efficiently, we implement the conjugate gradient method. This iterative approach is particularly well-suited for large datasets as it avoids the need for explicit matrix inversion.

A key aspect of our methodology is the use of parallel processing via the Message Passing Interface (MPI). We implement a 'circular kernel computation method' to enable efficient distributed computation of our that distributes the computation of the kernel matrix across multiple processes. This approach significantly reduces the computational time for large datasets.

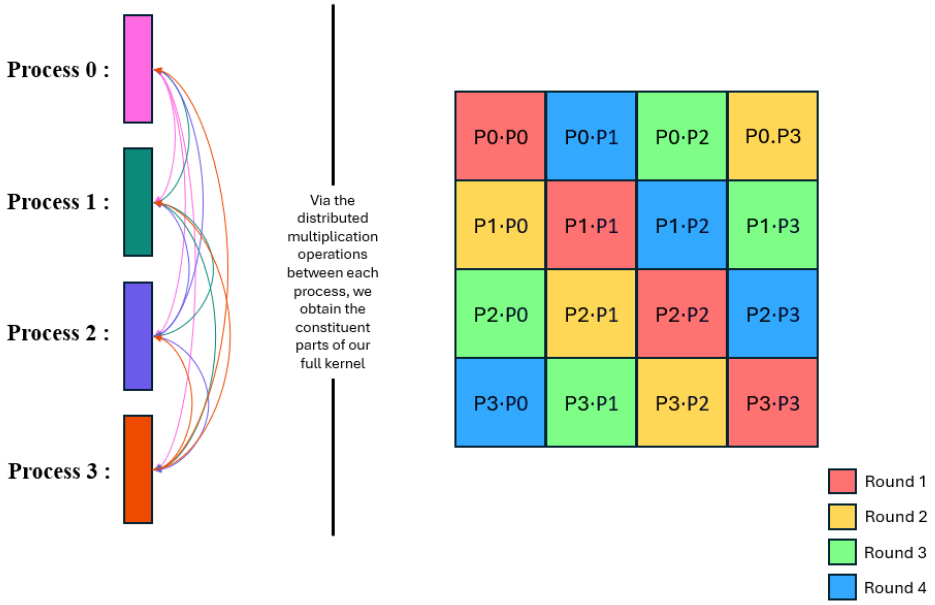


Figure 4: Overview of our distributed kernel computation

Our kernel computation computes the multiplication operations between processes in a pairwise manner. The multiplication operations happen in rounds, and the products are added to the each process' subset of the full kernel. For example, in Figure 4, we can see that Process 0 finishes the circular kernel computation with the top-most row of the full kernel, with each computation product (eg. P0.P0) being filled in by round order.

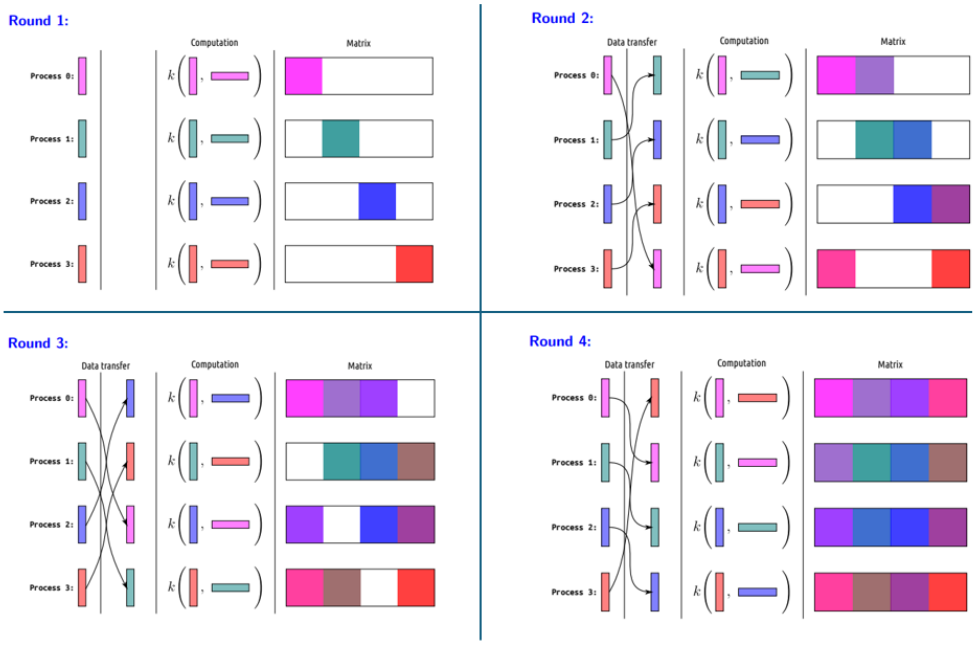


Figure 5: Circular kernel computation

In Figure 5, we can see the round-wise distributed kernel computation in action. The distributed kernel computation is considered Circular, because of the characteristic way our multiplication candidates "wrap around" the respective process locations, in order to maintain the integrity of the round-wise kernel computation and location placement.

To achieve this in our program, we utilized the unique variable 'rank', as well as 'round' and 'size'. For instance, the sequence in which the process-wise multiplication candidates are allocated to each other is determined by the modulus of (rank-1) against size, and (rank+1) against size, Additionally, the kernel row-wise placement of each distributed multiplication product is determined by the modulus of (rank + round) against size. The terminal log of our circular kernel computation can be seen in Figure 6, as additional validation of the sequential kernel computation.

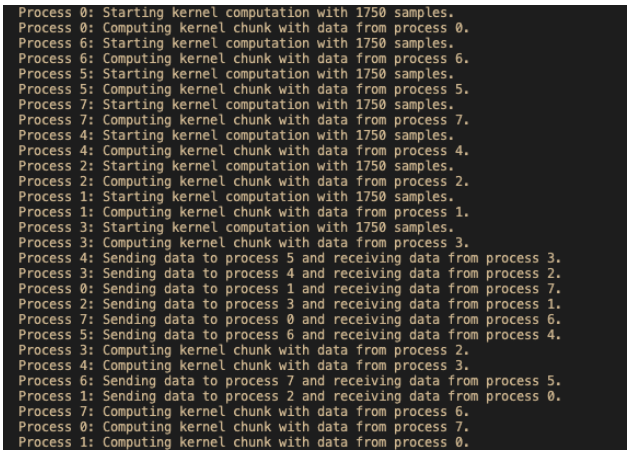


Figure 6: Terminal running log for kernel computation

In the training phase, we computed the kernel matrix in parallel using our circular kernel computation method. Then, we applied our regularization term to the kernel result. Afterwards, we solved for the optimal α values using the Conjugate Gradient Method.

For prediction, we compute the kernel between the test points and training points, then apply the learned α values.

Post-processing involves inverse-transforming the predictions back to the original scale. To address the data censoring issue identified in our exploratory data analysis, we apply a clipping function that limits predictions to the range [0, 500001].

Model evaluation is performed using the Root Mean Square Error (RMSE) on both the training and test sets. The RMSE is calculated as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i are the true values and $\hat{y}_i$ are the predicted values.

4 Results

4.1 Data Scaling Method Comparison

We first compared different scaling methods to determine the most effective approach for the data standardisation. The results are summarized in Table 1.

Table 1: Comparison of Scaling Methods

Scaling Method	Training RMSE	Test RMSE
Standard Scaler	**45,312.82**	**52,789.82**
MinMax Scaler	62,616.35	61,986.53
Robust Scaler	46,340.57	53,784.44

StandardScaler showed better performance in both training and test RMSE, and was thus selected.

4.2 Kernel Selection

We then tested four different kernels: sigmoid, linear, Gaussian (RBF), and polynomial. For each kernel, we explored a range of lambda values and kernel-specific parameters to thoroughly assess their performance. The effectiveness of each configuration was measured using Root Mean Square Error (RMSE) on both training and test sets. After testing, we summarized some of the most representative results for each kernel type. These results, using a lambda value of 0.03 (chosen later by hyperparameter tuning) for consistency, are as follows:

Table 2: Comparison of Kernel Functions and Their RMSE Values

Kernel Type	Parameters	Training RMSE	Test RMSE
Sigmoid	$\gamma = 0.1$, $r = 1$	315,706.21	317,810.22
Linear	—	188,653.55	186,624.46
Gaussian	$\sigma = 2.3$	**45,312.82**	**52,789.82**
Polynomial	$d = 3$, $r = 1$	285,417.88	287,054.80

The Gaussian kernel significantly outperformed the others, having a better performance on both training and test sets. Given its substantially lower RMSE values, we selected the Gaussian kernel for our final model.

4.3 Hyperparameter Tuning

We then performed a hyperparameter tuning process using grid search cross-validation. This process was conducted over three rounds, progressively refining our search for the optimal combination of λ (regularization parameter) and σ (kernel width parameter).

Through this iterative process, we progressively refined our search, starting with a broad range of values and narrowing down to more precise combinations. This approach led to a significant improvement in our model's performance with $\lambda = 0.03$ and $\sigma = 2.3$.

Interestingly, when we applied these optimal parameters ($\lambda = 0.03$ and $\sigma = 2.3$) to our Gaussian kernel model without using grid search cross-validation, we achieved an even lower test RMSE of **52,789.82**. This slight improvement suggests that our model generalizes well to unseen data and that our hyperparameter tuning process was effective in identifying near-optimal parameters.

Round 1 RMSE Values

λ / σ	0.01	0.1	1.0
0.1	230336.76	230777.38	233305.50
1.0	64112.31	60524.00	64978.86
5.0	55554.02	57598.22	60790.42

Round 2 RMSE Values

λ / σ	0.005	0.01	0.05
2.0	55706.91	54680.76	54054.38
3.0	54733.71	54285.85	54529.76
4.0	54818.67	54787.18	55688.82

Round 3 RMSE Values

λ / σ	0.01	0.03	0.05
2.0	54680.76	54053.72	54054.38
2.3	54450.16	**53979.50**	54027.27
2.5	54365.44	54004.25	54103.27

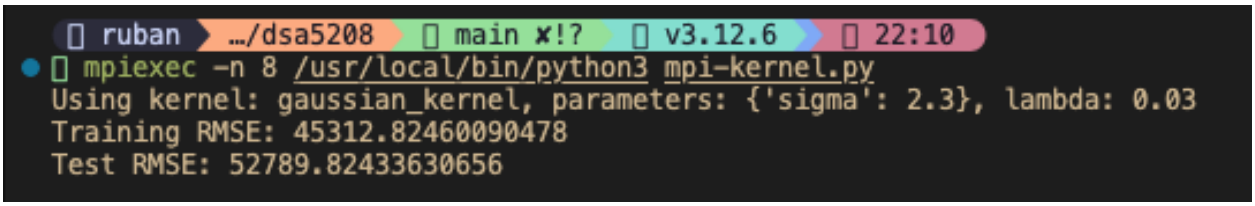


Figure 7: Final RMSE Results

5 Conclusion

In this project, we successfully implemented a Kernel Ridge Regression (KRR) model to predict median house values in California, using data from the 1990 Census.

By leveraging parallel computation through the Message Passing Interface (MPI), we efficiently processed large datasets and reduced computation time. The integration of feature engineering, regularization, and hyperparameter tuning led to a model that achieved a final RMSE of 53,979.50.

Our model captured important non-linear relationships in the data, with median income proving to be the strongest predictor of house values. The use of a circular kernel computation method was crucial in distributing kernel calculations, allowing us to handle large-scale data effectively.

Moving forward, another promising direction could be implementing more sophisticated machine learning models such as Gaussian Processes or deep learning methods (e.g., neural networks), which may capture even more complex patterns in the data. Furthermore, addressing the issue of data censoring more rigorously, perhaps with quantile regression or custom loss functions, could help improve predictions at the extreme ends of the housing price distribution.

Finally, scalability improvements could involve implementing more distributed computing frameworks, like Apache Spark, to further speed up computations for even larger datasets or exploring GPU acceleration for more efficient parallel processing.

6 References

1. DSA5208, Lecture 4 Slides
2. ChatGPT, https://chatgpt.com/share/67013846-9b68-8000-af1b-466c9bd7a914
3. ChatGPT, https://chatgpt.com/share/67013831-7b64-8011-9de3-7e10b4d24917
4. ChatGPT, https://chatgpt.com/share/67015b68-32f8-8011-937f-ea8ca54cdb09