Samsung AFSL Home Assignment: Learning to Denoise Corrupted Binary Vectors via Local Translation-Invariant Neural Networks

Maor Kehati

July 5, 2025

1 Data Analysis

This section documents my exploration of the dataset and the corruption process that transforms binary vectors $X \in \{0,1\}^{512}$ into real-valued vectors $Y \in \mathbb{R}^{512}$. My primary goals were to validate two hypotheses: (1) that the corruption is **local**, and (2) that it is **translation invariant**.

A full record of my experimentation and visualizations is available in the accompanying Jupyter notebook: `explore.ipynb`.

1.1 Marginal Distributions and Conditional Statistics

To assess whether Y_i retains information about the original bit X_i, I plotted the empirical distributions of Y_i conditioned on $X_i = 0$ and $X_i = 1$. The distributions show clear separation, suggesting that Y_i encodes useful information about the bit it was derived from.

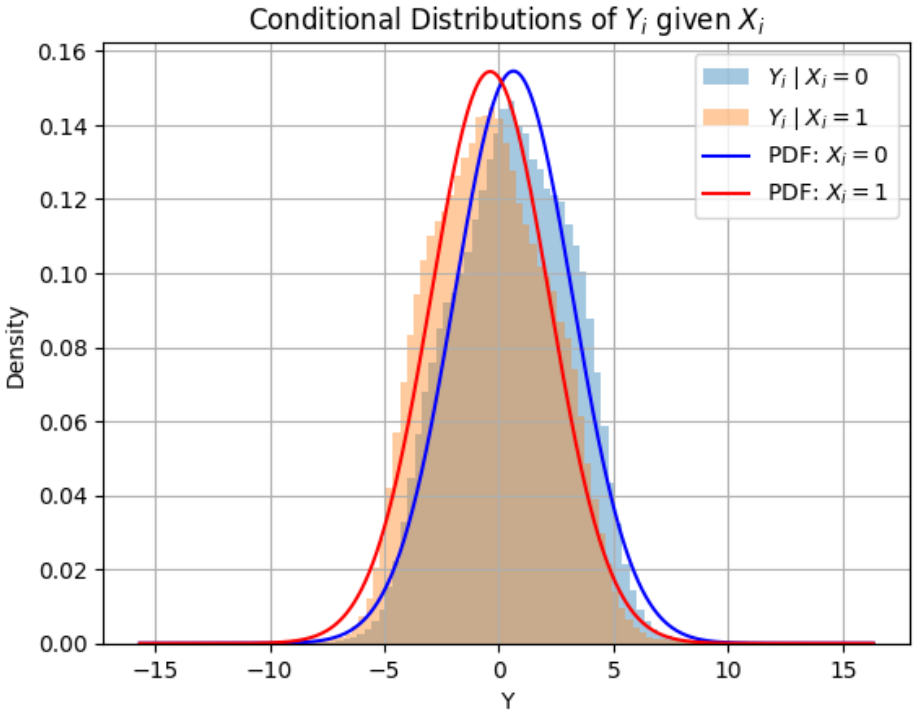

Figure 1: Empirical distributions of Y_i conditioned on $X_i = 0$ and $X_i = 1$, overlaid with Gaussian fits. The separation implies strong pointwise signal.

In addition to the pointwise relationship between X_i and Y_i, I also investigated how Y_i depends on neighboring bits. Specifically, I examined the distributions of Y_i conditioned on X_{i-1} instead of X_i. The resulting histograms were still unimodal and well-fit by Gaussian distributions—there was no evidence of multimodal structure or a need for a Gaussian mixture model in this context.

However, the correlation between X_{i-1} and Y_i was found to be even stronger than that between X_i and Y_i (as shown in Figure 4). This observation justifies the design of certain baseline models that include a small offset when predicting X from Y—for example, predicting X_{i-1} from Y_i instead of X_i.

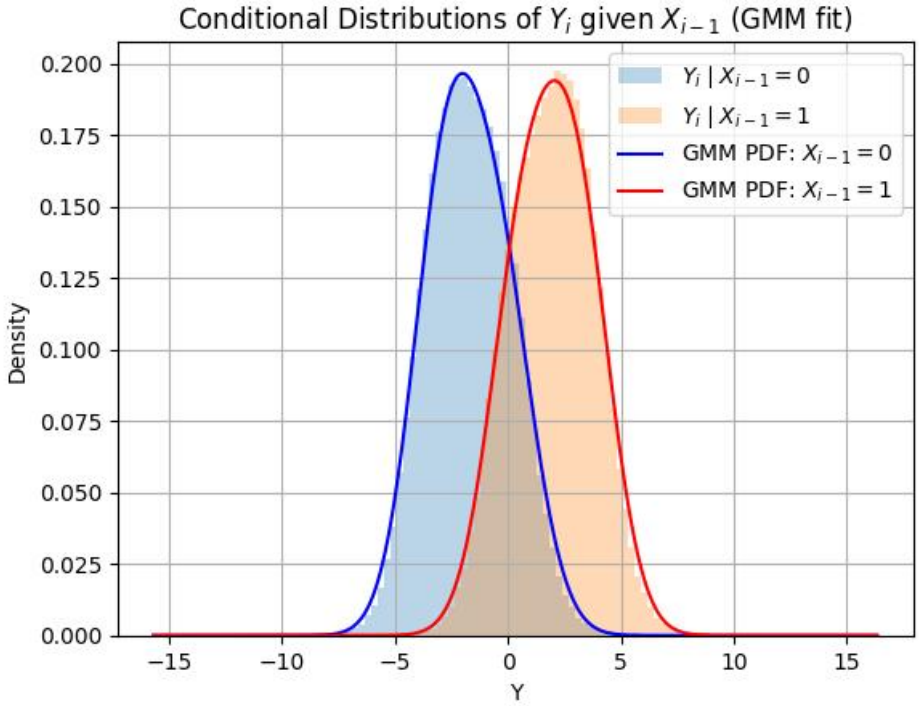

Figure 2: Distributions of Y_i conditioned on $X_{i-1} = 0$ and $X_{i-1} = 1$, along with fitted Gaussian curves. Despite the change in conditioning variable, the distributions remain unimodal.

1.2 Pointwise Correlations

To investigate the locality assumption of the corruption process, I computed the Pearson correlation between each corrupted bit Y_i and the original bits X_{i+k} for a range of offsets $k \in [-50, 50]$. The result is shown in Figure 3.

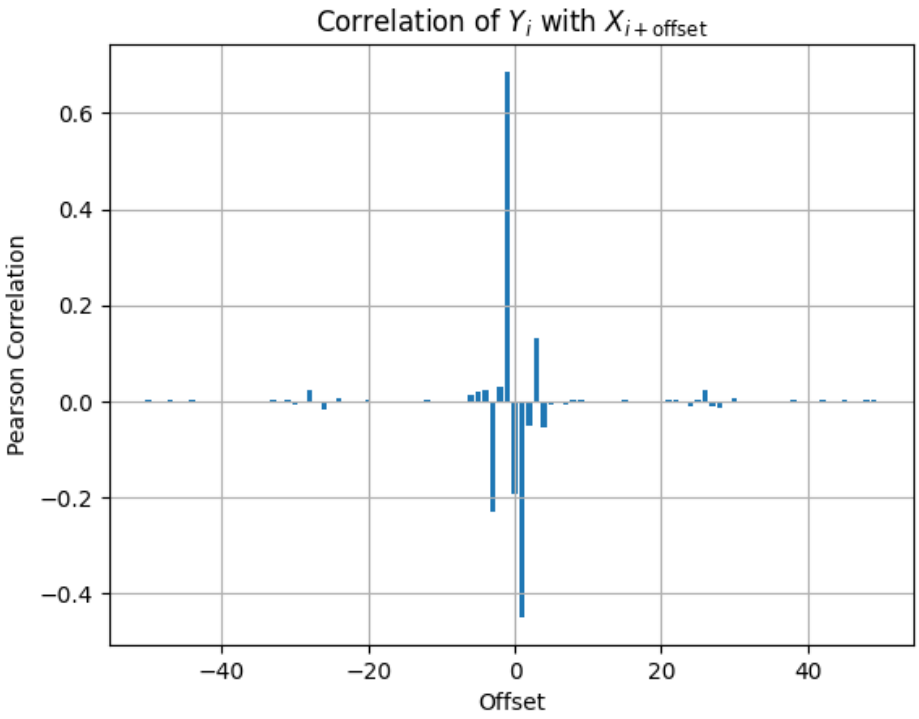

Figure 3: Correlation between Y_i and X_{i+k} for offsets $k \in [-50, 50]$.

While there is some weak structure visible around offsets $k \approx \pm 30$, the magnitude of those correlations is minimal and can be considered negligible. This supports the hypothesis that the corruption process is *local*, with the significant signal confined to a narrow band of neighboring bits.

To examine the local structure more closely, I zoomed in on the range $k \in [-5, 5]$ in Figure 4.

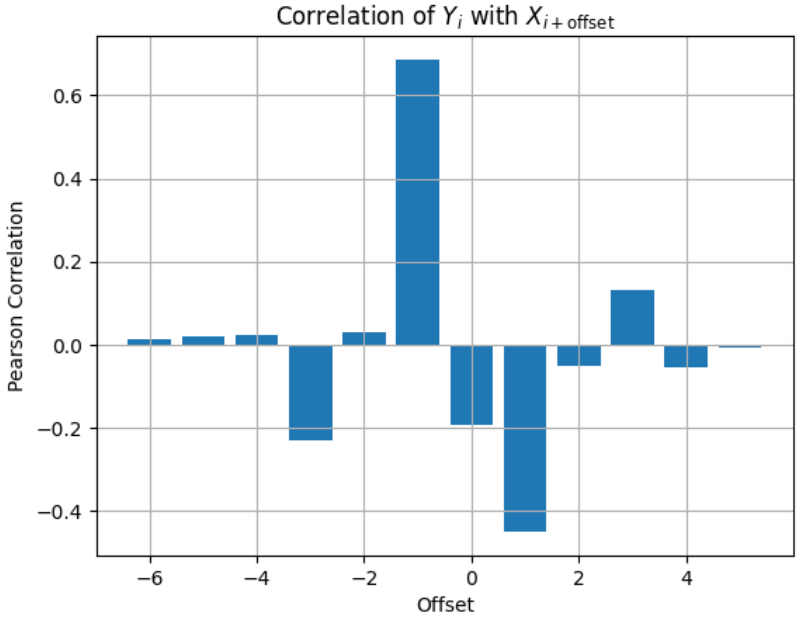

Figure 4: Zoomed-in correlation between Y_i and X_{i+k} for offsets $k \in [-5, 5]$.

The zoomed view clearly shows that Y_i exhibits strong correlation with X_i and meaningful but weaker correlations with nearby bits within a window of radius 2–3. This reinforced my design choice of using local receptive fields in the proposed neural model.

1.3 Higher-Order Dependency Analysis

While the Pearson correlation analysis in the previous section confirms a strong linear (first-order) relationship between Y_i and local bits X_{i+k}, I also investigated whether *nonlinear* or higher-order relationships exist beyond what is captured by linear correlation.

To isolate higher-order effects, I first performed a linear regression of Y_i on each shifted version of X (i.e., X_{i+k} for $k \in [-6, 5]$). I then computed the residuals—i.e., the portion of Y_i not explained by the linear component—and measured the mutual information between these residuals and X_{i+k}. This mutual information serves as a proxy for nonlinear dependencies: if Y_i contains nonlinear signal from X_{i+k}, it will manifest as mutual information in the residuals.

Table 1: Estimated mutual information between X_{i+k} and the residuals of Y_i after linear regression, for small offsets k.

Offset k	Nonlinear Mutual Information
-6	0.0018
-5	0.0020
-4	0.0013
-3	0.0031
-2	0.0044
-1	0.0034
0	0.0046
+1	0.0042
+2	0.0031
+3	0.0019
+4	0.0029
+5	0.0019

The results, shown in Table 1, reveal that the residual mutual information values are consistently very low (all below 0.005). This implies that once the linear contribution is accounted for, there is little to no remaining dependency between Y_i and X_{i+k}. Therefore, the transformation from X to Y is effectively linear (or at least well-approximated as such) within the examined local window.

Conclusion: The corruption process appears to be almost entirely first-order and local. No substantial higher-order (nonlinear) relationships were detected, which simplifies the modeling assumptions and justifies the use of linear baselines and locally constrained models. Consequently, models designed to capture complex higher-order interactions were deemed unnecessary for this task.

1.4 Translation Invariance

To test whether the corruption process is translation invariant—that is, whether the relationship between X and Y is consistent across positions—I computed the Pearson correlation between Y_i and X_{i+k} for several fixed offsets $k \in [-6, 5]$, separately at each position i in the vector. I then aggregated the correlations by computing their mean and standard deviation across positions.

Table 2: Mean and standard deviation of the Pearson correlation between Y_i and X_{i+k} across positions.

Offset k	Mean Correlation	Std Dev
-6	-0.0003	0.0293
-5	-0.0065	0.0329
-4	-0.0551	0.0301
-3	0.1372	0.0344
-2	-0.0522	0.0321
-1	-0.4641	0.0497
0	-0.1975	0.0338
+1	0.7069	0.0610
+2	0.0297	0.0301
+3	-0.2380	0.0392
+4	0.0241	0.0318
+5	0.0192	0.0308

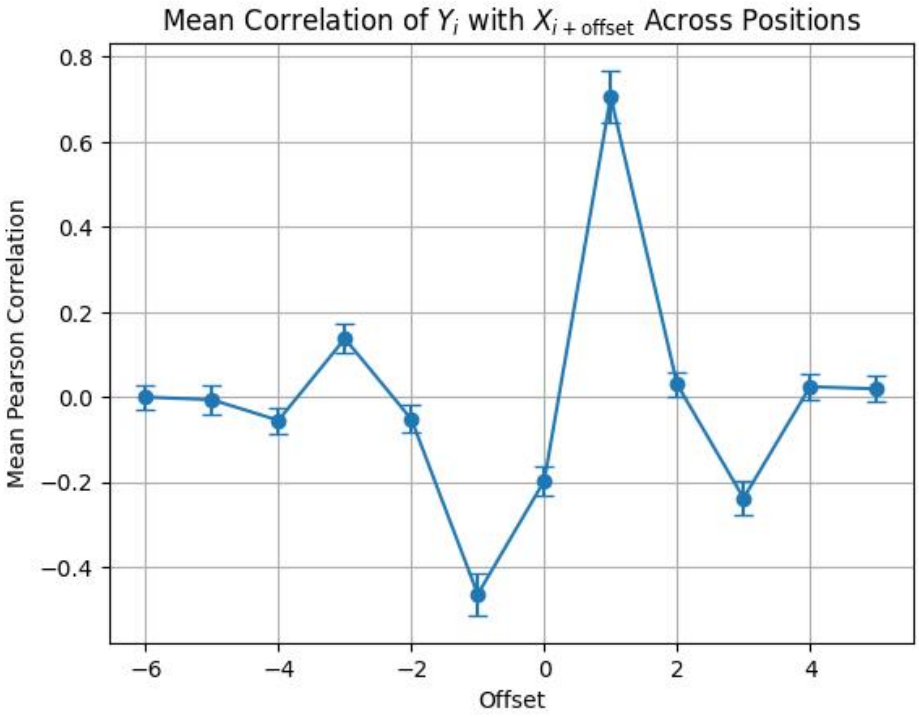

Figure 5: Mean Pearson correlation between Y_i and X_{i+k} across positions, for offsets $k \in [-6, 5]$. Error bars indicate standard deviation across positions.

The results, shown in Table 2 and Figure 5, reveal that the correlation structure is highly consistent across positions. The standard deviations are small (typically around 0.03–0.06), indicating that the relationship between Y_i and nearby bits X_{i+k} is approximately position-independent. This supports the assumption that the corruption process is translation invariant.

Conclusion: The observed consistency in local correlations across positions justifies the use of translation-invariant architectures, such as convolutional or locally shared models, in the denoising task.

2 Baseline Models

As a point of comparison, I evaluated several baseline models on the denoising task of predicting X from Y. These models serve to quantify how much information is recoverable using low-capacity architectures.

2.1 MAP Estimator $\hat{X}_i \mid Y_i$ with Gaussian Corruption and Offset

This baseline assumes a simple generative model in which the observed value Y_i is drawn from a Gaussian distribution whose parameters depend not on X_i, but on a shifted bit $X_{i+\delta}$. That is, the corruption process is modeled as:

$$Y_i \mid X_{i+\delta} = b \sim \mathcal{N}(\mu_b, \sigma_b^2)$$

The MAP classifier then predicts $X_{i+\delta}$ from Y_i, exploiting the localized structure revealed in Section 1.2. By varying the offset δ, the model effectively searches for the alignment between bits in X and the observed signal in Y that provides the strongest predictive power.

Classification rule:

$$\hat{X}_{i+\delta} = \arg\max_{b \in \{0,1\}} \left[-\frac{(Y_i - \mu_b)^2}{2\sigma_b^2} + \log P(X_{i+\delta} = b) \right]$$

This formulation differs from a standard denoising approach (which would model $Y_i \mid X_i$) by explicitly acknowledging that the most predictive bit for Y_i may be located at a small offset from i. Empirical correlation plots show that Y_i is often more correlated with $X_{i+\delta}$ for small $\delta \neq 0$, justifying this offset-based modeling.

In practice, the offset δ is treated as a hyperparameter and selected via cross-validation over a small range (e.g., $\delta \in [-5, 5]$).

Assuming equal priors and approximately equal variances ($\sigma_0 \approx \sigma_1$), the MAP rule reduces to a threshold test on Y_i:

$$\hat{X}_{i+\delta} = \begin{cases} 0 & \text{if } Y_i \geq \tau \\ 1 & \text{if } Y_i < \tau \end{cases} \quad \text{where} \quad \tau = \frac{\mu_0 + \mu_1}{2}$$

This approach provides a surprisingly strong baseline by leveraging local, shifted statistics between Y and X, while remaining fully interpretable.

2.2 MAP Estimator $\hat{X}_i | Y_{i-L:i+L}$ with Multivariate Gaussian Corruption

This model generalizes the previous MAP approach by modeling the distribution of a local window of corrupted observations around each index i, rather than just the scalar Y_i. Specifically, I assume that each corrupted scalar Y_i arises from a linear combination of nearby binary inputs $X_{i-L}, \ldots, X_{i+L}$, plus a class-conditioned bias and Gaussian noise:

$$Y_i = \sum_{j=-L}^{L} \alpha_j \cdot X_{i+j} + \mu_b + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma_b^2)$$

where:

- α_j are shared local weights,
- $\mu_b \in \mathbb{R}$ is a bias term that depends on the value of $X_i = b \in \{0, 1\}$,
- ε_i is zero-mean Gaussian noise, with class-conditioned variance σ_b^2.

Although the generative assumption defines how each individual value Y_i is produced from a local window of binary values $X_{i-L:i+L}$, the classification model operates in the reverse direction: it infers X_i from the observed neighborhood $\mathbf{y}_i = [Y_{i-L}, \ldots, Y_{i+L}]^\top$. This is consistent with the assumption, since local correlations in Y arise directly from overlapping windows of X, and modeling $P(\mathbf{y}_i \mid X_i = b)$ as a multivariate

Gaussian captures this structure. Thus, the estimator leverages the full local observation window to infer the central bit.

For each class $b \in \{0, 1\}$, I model:

$$\mathbf{y}_i \mid X_i = b \sim \mathcal{N}(\mu_b, \Sigma_b)$$

where $\mu_b \in \mathbb{R}^{2L+1}$ is the class-conditional mean vector and $\Sigma_b \in \mathbb{R}^{(2L+1)\times(2L+1)}$ is the full covariance matrix.

MAP Decision Rule:

$$\hat{X}_i = \arg\max_{b\in\{0,1\}} \left[-\frac{1}{2}(\mathbf{y}_i - \mu_b)^\top \Sigma_b^{-1}(\mathbf{y}_i - \mu_b) - \frac{1}{2}\log|\Sigma_b| + \log P(X_i = b) \right]$$

This approach generalizes the pointwise MAP estimator from Section 2.1, which is encapsulated as the special case $L = 0$. Hence, I expect this multivariate model to yield superior performance by leveraging contextual information from neighboring observations.

2.3 Linear Regression with Local Shared Weights

This baseline model assumes a linear relationship between a corrupted observation window $\mathbf{y}_i = [Y_{i-L}, \ldots, Y_i, \ldots, Y_{i+L}]^\top$ and the original binary bit X_i. I define the model as:

$$\hat{x}_i = \sigma\left(\mathbf{w}^\top \mathbf{y}_i + b\right)$$

where $\mathbf{w} \in \mathbb{R}^{2L+1}$ is a learned weight vector, b is a learned bias, and $\sigma(\cdot)$ is the step function used to threshold the output into binary predictions.

Shared Weights and Windowing. I use the same weight vector $\mathbf{w}$ and bias b across all positions i, making the model translation invariant. At each position, the input is a local window of size $2L+1$ centered at Y_i. At sequence boundaries where the window would extend beyond available indices, I apply masking so that only valid entries contribute to the prediction.

Training via Least Squares. I obtain the weights $\mathbf{w}$ and bias b via closed-form least squares regression. Specifically, I solve the optimization problem:

$$\min_{\mathbf{w},b} \sum_{i=1}^{N\cdot S} \left(X_i - \mathbf{w}^\top \mathbf{y}_i - b\right)^2$$

where the total number of training samples is $N \cdot S$, and each sample corresponds to a bit X_i and its surrounding corrupted observations.

Interpretation. This model captures linear dependencies between the corrupted signal and the clean bit labels, while assuming that the corruption process remains translation invariant across the sequence. It serves as a simple yet strong baseline when the corruption noise is close to Gaussian and the mapping from Y to X is primarily linear.

This model can be seen as a simplified special case of the multivariate Gaussian MAP estimator from Section 2.2: it assumes a shared linear mapping from local $\mathbf{y}_i$ windows to x_i, without modeling class-specific statistics or noise. Since it discards both class-conditioned means and covariances, I expect it to underperform the full MAP estimator that models these explicitly.

3 Neural Net Solution

Motivated by the data analysis and corruption assumptions, I designed a neural architecture with the following goals:

- Capture **local dependencies** across nearby bits
- Exploit **translation invariance** by weight sharing
- Allow for **nonlinear interactions** between bits

My solution consists of a configurable 1D convolutional neural network (CNN) that maps the corrupted vector $Y \in \mathbb{R}^{512}$ to a predicted output vector $\hat{X} \in \mathbb{R}^{512}$. The network applies a sequence of convolutional layers with symmetric padding and shared weights, enabling local, translation-invariant modeling across the entire sequence.

The architecture has the following configurable parameters:

- **Depth:** number of convolutional layers
- **Width:** number of channels in each hidden layer
- **Kernel size:** controls the local receptive field (typically odd)
- **Activation:** nonlinear function between layers (e.g., ReLU, GELU, Tanh)

All layers are 1D convolutions with padding chosen to preserve input length. The final layer outputs a single channel with no activation, allowing the model to produce either logits (if trained with BCE) or continuous denoised signals.

Formally, the model approximates:

$$\hat{X}_i = f(Y_{i-L}, \ldots, Y_i, \ldots, Y_{i+L})$$

where $f(\cdot)$ is a learned nonlinear function parameterized by the CNN.

I train the model using binary cross-entropy (BCE) loss when treating this as a probabilistic binary reconstruction task.

3.1 Hyperparameter Search and Results

To identify the most effective neural architecture, I conducted a systematic greedy hyperparameter search across several architectural and training parameters. The search was performed in stages, tuning one hyperparameter at a time while holding the others fixed to the best-known values. Each configuration was trained for 50 epochs using 5-fold cross-validation, and evaluated using a suite of metrics including accuracy, F1 score, ROC-AUC, binary cross-entropy, and expected calibration error.

The hyperparameters and values explored were:

- **Learning rate:** 1e-5 to 3e-1 (log scale)
- **Batch size:** 8, 16, 32, 64, 128
- **Activation function:** ReLU, GELU, Tanh, Sigmoid, Leaky ReLU
- **Depth:** 1, 2, 3, 4, 6, 8, 16, 32
- **Width:** 8, 16, 32, 64, 128, 256
- **Kernel size:** 1, 3, 5, 7, 9, 11, 15

Each trial involved full training and evaluation, with results logged and saved for comparison. The full sweep results are provided in Appendix B.

Conclusion: The optimal configuration chosen for final evaluation was:

- Learning rate: `3e-2`
- Batch size: `128`
- Activation: `gelu`

- Depth: `8`
- Width: `32`
- Kernel size: `3`

This setup achieved the best trade-off between performance and model complexity, consistently outperforming other configurations across all evaluation metrics.

4 Results

All models were evaluated using 5-fold cross-validation over the training data, which provides a more rigorous and thorough assessment of generalization performance. This procedure is especially suitable here given the relatively small size of the training set, allowing full use of the available data without requiring a separate validation split.

4.1 Evaluation Metrics

To comprehensively assess model performance, I report a mix of classification and calibration metrics. These are computed per-bit across all samples, and then averaged. Each metric provides a distinct view into different aspects of model behavior when denoising a corrupted binary vector.

- **Accuracy:** The proportion of correctly predicted bits across the entire vector. In this denoising task, where the goal is to reconstruct a clean binary vector $\hat{X} \in \{0,1\}^n$ from a corrupted observation Y, accuracy reflects the overall bit-level agreement.
- **Precision:** The proportion of predicted 1s that are actually correct:

  $$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

  where TP and FP are true and false positives respectively. In imbalanced datasets, precision can be misleading, but here, the data is balanced with $\mathbb{P}(x_i = 1) = \mathbb{P}(x_i = 0) = 0.5$, making precision directly interpretable.
- **Recall:** The proportion of actual 1s that are correctly predicted:

  $$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

  Since the dataset is balanced, recall and precision are symmetric in importance. High recall ensures that true bits are recovered reliably.
- **F1 Score:** The harmonic mean of precision and recall. It balances the trade-off between them and is particularly meaningful when both false positives and false negatives are costly, as is the case here where each bit prediction matters:

  $$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

  Again, due to the balanced data distribution, the F1 score offers a fair summary of model effectiveness.
- **ROC-AUC:** The area under the Receiver Operating Characteristic curve. This metric measures the model's ability to rank positive samples above negatives across all thresholds. In the binary denoising task, where logits or probabilities are produced before thresholding, ROC-AUC evaluates discriminative power. With balanced classes, ROC-AUC is especially informative and not biased by class prevalence.

- **Hamming Distance:** The average number of differing bits between the predicted vector and the true vector, normalized over length:

$$\text{Hamming}(X, \hat{X}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[x_i \neq \hat{x}_i]$$

  This directly captures how many bits the model gets wrong and is particularly well-suited for evaluating reconstruction tasks like binary denoising.

- **Binary Cross-Entropy (BCE):** A standard loss function for binary classification, BCE quantifies the average negative log-likelihood of the predicted probabilities against the ground truth labels. In this setting, BCE measures how well the model's predicted probabilities align with the true bit values.

- **Brier Score:** The mean squared error between predicted probabilities and true binary outcomes. Unlike BCE, Brier is a proper scoring rule that is more sensitive to overconfident incorrect predictions and useful for evaluating probabilistic calibration.

- **Expected Calibration Error (ECE):** Measures the difference between predicted confidence and actual accuracy. In a denoising context, a model with low ECE provides more trustworthy probability estimates, which is crucial when downstream decisions depend on confidence levels.

Together, these metrics provide a well-rounded evaluation: classification accuracy, ranking ability, probabilistic quality, and calibration.

4.2 Models Results

Each model's best configuration (as determined via the hyperparameter search described in Appendix A and Appendix B) was selected and evaluated. The table below reports the mean performance across all folds using a variety of metrics: accuracy, F1 score, precision, recall, ROC-AUC. For the NN solution, I also report the Hamming distance, binary cross-entropy (BCE), Brier score, and Expected Calibration Error (ECE).

Model	Accuracy	F1	Precision	Recall	ROC-AUC	HamDist	BCE	Brier	ECE
MAP (Offset)	0.8207	0.8207	0.8215	0.8189	0.8207	–	–	–	–
Multivariate Gaussian	0.9065	0.9065	0.9066	0.9061	0.9065	–	–	–	–
Linear Regression	0.6029	0.6029	0.5572	0.9981	0.6033	–	–	–	–
Neural Network (CNN)	**0.9385**	**0.9385**	**0.9364**	**0.9408**	**0.9384**	**0.0615**	**0.5345**	**0.0615**	**0.0320**

Table 3: Final performance metrics for each model. The CNN outperforms other approaches across all available metrics. Missing values (–) indicate metrics not computed for that model.

4.3 Final Test Set Prediction on Full Training Set

To produce the final predictions on the test set, the chosen model (NN) was retrained on the full training set using the best hyperparameters obtained from cross-validation. This ensures that the model benefits from all available training data and avoids artificial data wastage.

5 Discussion

In this report, I addressed the problem of reconstructing a clean binary vector $X \in \{0, 1\}^{512}$ from a corrupted real-valued vector $Y \in \mathbb{R}^{512}$. Through empirical analysis, I showed that the corruption process is local, translation invariant, and approximately linear with additive Gaussian noise. These properties guided the design of both classical baseline models and a convolutional neural network (CNN) solution.

5.1 Multivariate MAP Estimator: A Strong Classical Baseline

Among the baseline methods, the multivariate MAP estimator achieved a high accuracy of 90.6% by modeling the distribution of local corrupted windows as class-conditional multivariate Gaussians. This strong result suggests that the underlying data distribution aligns well with the model's assumptions. In particular, the local Gaussian structure appears to capture most of the signal needed for denoising.

The fact that such a relatively simple, interpretable model performs so well reinforces the conclusion that the corruption process is mostly linear and locally stationary. However, the estimator still falls short of the CNN in final accuracy, indicating room for improvement. Possible directions include refining the model's handling of class-conditioned noise or introducing mild nonlinearity in the generative structure.

5.2 Neural Network: Top Accuracy but Calibration Limitations

The 1D CNN outperformed all other models, achieving 93.85% accuracy. This improvement over the multivariate MAP baseline likely stems from the network's capacity to capture subtle nonlinear dependencies or to better approximate the interaction between local patterns and noise.

Despite its strong classification performance, the CNN's calibration metrics tell a more nuanced story. The binary cross-entropy loss (0.5345), Brier score (0.0615), and expected calibration error (ECE) of 0.0320 are notably less impressive than its accuracy would suggest. This indicates that while the model is often correct, it may be overconfident or poorly aligned in its probability estimates.

Such calibration gaps are not uncommon in deep learning models and can be problematic in settings that require reliable confidence estimates—e.g., downstream decision-making or uncertainty-aware inference. Further calibration techniques such as temperature scaling or isotonic regression could be applied post hoc to improve the trustworthiness of the model's probabilistic outputs.

5.3 Implications and Future Directions

The close performance of the CNN and the multivariate Gaussian baseline implies that the latter already captures much of the relevant structure in the data. The CNN's marginal improvement suggests that the true generative process is nearly—but not exactly—Gaussian. Introducing slight modifications to the generative model, such as nonlinear mappings or context-aware noise modeling, could close this performance gap without requiring deep architectures.

Additionally, enhancing the calibration of neural models will be essential if they are to be deployed in confidence-sensitive applications. This includes post-training calibration or explicitly regularizing for uncertainty during training.

Conclusion: This task of denoising corrupted binary vectors with the given specific distribution appears largely governed by local, Gaussian-like structure. Classical models are highly competitive under these conditions. However, neural networks offer a modest edge by exploiting remaining nonlinearities—at the cost of poorer calibration. Further progress may come from blending the interpretability of generative models with the flexibility of learned representations.

A Baseline Model Hyperparameter Search

This section documents the results of hyperparameter sweeps for the baseline models presented in the main text. Each model was evaluated using accuracy, precision, recall, F1 score, and ROC-AUC. For the MAP estimator, a sweep over the alignment offset (shift) parameter was performed.

A.1 MAP Estimator: Offset Parameter Sweep

The MAP estimator predicts the bit $X_{i+\delta}$ from observation Y_i, where δ is a tunable offset capturing the dependency structure observed in the data. A sweep over shift values from -5 to $+5$ was conducted to determine the optimal alignment. The results are shown below:

Shift	Accuracy	F1	Precision	Recall	ROC-AUC
-5	0.5068	0.5068	0.5065	0.4840	0.5068
-4	0.5110	0.5110	0.5109	0.4956	0.5111
-3	0.5954	0.5954	0.5940	0.5995	0.5954
-2	0.5262	0.5262	0.5284	0.4779	0.5262
-1	**0.8207**	**0.8207**	**0.8215**	**0.8189**	**0.8207**
0	0.5795	0.5795	0.5792	0.5783	0.5795
1	0.6596	0.6596	0.6587	0.6609	0.6596
2	0.5163	0.5163	0.5167	0.4891	0.5163
3	0.5494	0.5494	0.5493	0.5444	0.5494
4	0.5190	0.5190	0.5191	0.5007	0.5190
5	0.5010	0.5010	0.5005	0.4798	0.5012

Table 4: MAP estimator performance across offset values δ. The best shift is $\delta = -1$, indicating strongest alignment between Y_i and X_{i-1}.

These results support the earlier observation that the corrupted value Y_i often aligns more closely with X_{i-1} than with X_i, motivating the use of a shifted MAP rule for improved accuracy.

A.2 Multivariate Gaussian Model: Context Window Sweep

The Multivariate Gaussian baseline models the joint distribution of local input-output patterns using a multivariate Gaussian over a symmetric context window of size $2L + 1$, where L is a tunable parameter. This allows the model to incorporate both past and future context surrounding each bit when estimating the most likely value.

To assess the effect of window size, I swept over values of L from 1 to 200. The accuracy results are presented in Table **??**.

L	Accuracy	F1	Precision	Recall	ROC-AUC
1	0.8401	0.8401	0.8401	0.8396	0.8401
2	0.8680	0.8680	0.8680	0.8677	0.8680
3	0.8896	0.8896	0.8897	0.8891	0.8896
4	0.8997	0.8997	0.8993	0.8999	0.8997
5	0.9011	0.9011	0.9007	0.9013	0.9011
6	0.9042	0.9042	0.9037	0.9046	0.9042
7	0.9048	0.9048	0.9042	0.9053	0.9048
8	0.9051	0.9051	0.9045	0.9055	0.9051
9	0.9052	0.9052	0.9047	0.9056	0.9052
10	0.9052	0.9052	0.9047	0.9056	0.9052
30	0.9061	0.9061	0.9058	**0.9061**	0.9061
50	**0.9065**	**0.9065**	**0.9066**	0.9061	**0.9065**
100	0.9059	0.9059	0.9065	0.9050	0.9059
200	0.8960	0.8960	0.8969	0.8946	0.8960

Table 5: Multivariate Gaussian model performance across context window sizes L. Best values per column are highlighted in bold.

Performance improves steadily with increasing context, peaking at $L = 50$, which corresponds to a full window of 101 bits. Larger values begin to degrade accuracy, likely due to the increased dimensionality of the covariance matrix and limited training data. For final evaluation, $L = 50$ was selected as the optimal context size.

A.3 Linear Regression: Context Window Sweep

The linear regression baseline uses a symmetric context window of size $2L + 1$ centered around the bit to be predicted. A separate linear model is trained to map the corrupted input window to the target bit, minimizing squared error. This simple model serves as a baseline for understanding how much predictive power can be extracted using only local linear structure in the corrupted data.

I performed a sweep over values of L from 1 to 200, with accuracy results summarized in Table **??**.

L	Accuracy	F1	Precision	Recall	ROC-AUC
1	0.5810	0.5810	0.5438	**0.9983**	0.5814
2	0.5876	0.5876	0.5478	0.9981	0.5881
3	0.5926	0.5926	0.5509	0.9978	0.5931
4	0.5957	0.5957	0.5528	0.9978	0.5962
5	0.5965	0.5965	0.5533	0.9979	0.5970
6	0.5975	0.5975	0.5539	0.9979	0.5980
7	0.5978	0.5978	0.5541	0.9978	0.5983
8	0.5981	0.5981	0.5542	0.9978	0.5985
9	0.5981	0.5981	0.5543	0.9979	0.5986
10	0.5982	0.5982	0.5543	0.9979	0.5986
30	0.5993	0.5993	0.5550	0.9979	0.5997
50	0.5998	0.5998	0.5553	0.9979	0.6003
100	0.6021	0.6021	0.5567	0.9981	0.6026
200	**0.6029**	**0.6029**	**0.5572**	0.9981	**0.6033**

Table 6: Linear regression model performance across context window sizes L. Best values per column are highlighted in bold.

As expected, the linear model benefits from larger context windows, albeit with diminishing returns. Accuracy improves slowly with increasing L, eventually plateauing near 60.3%. The relatively low performance compared to the Multivariate Gaussian model highlights the limitations of a purely linear approach, which does not account for bit correlations or distributional uncertainty.

B Neural Net Full Hyperparameter Search Results

This appendix presents the full results of the greedy hyperparameter search performed to optimize the convolutional neural network architecture. In each stage, I swept over values for a single hyperparameter while holding all others fixed at their current best setting. The goal was to incrementally discover a performant configuration with minimal training overhead.

Unless otherwise noted, all experiments used:

- Epochs: 10
- Batch size: 32
- Depth: 3
- Width: 32
- Kernel size: 3
- Activation: ReLU
- Learning rate: 1e-3

B.1 Learning Rate Sweep

I tested the learning rates detailed in table 7 below. As a result, the learning rate value 3×10^{-2} was chosen. The model was trained with each learning rate for 10 epochs using a fixed architecture.

LR	Acc	F1	Prec	Rec	ROC-AUC	Ham	BCE	Brier	ECE
1×10^{-5}	0.4839	0.4839	0.4792	0.3514	0.4836	0.5161	0.7457	0.5161	0.1923
3×10^{-5}	0.5870	0.5870	0.6697	0.4961	0.5867	0.4130	0.6991	0.4130	0.1616
1×10^{-4}	0.8008	0.8008	0.8275	0.7694	0.8007	0.1992	0.5993	0.1992	0.0841
3×10^{-4}	0.8419	0.8419	0.8431	0.8398	0.8419	0.1581	0.5822	0.1581	0.0781
1×10^{-3}	0.8877	0.8877	0.8863	0.8892	0.8877	0.1123	0.5598	0.1123	0.0570
3×10^{-3}	0.8964	0.8964	0.8932	0.9002	0.8964	0.1036	0.5557	0.1036	0.0538
1×10^{-2}	0.9028	0.9028	**0.9040**	0.9012	0.9028	0.0972	0.5518	0.0972	**0.0478**
3×10^{-2}	**0.9056**	**0.9056**	0.9034	**0.9081**	**0.9056**	**0.0944**	**0.5509**	**0.0944**	0.0485
1×10^{-1}	0.9009	0.9009	0.8982	0.9042	0.9009	0.0991	0.5534	0.0991	0.0513
3×10^{-1}	0.7368	0.7368	0.7743	0.8035	0.7368	0.2632	0.6430	0.2632	0.1649

Table 7: Grid search results for various learning rates. Metrics include accuracy (Acc), F1 score (F1), precision (Prec), recall (Rec), ROC-AUC, Hamming distance (Ham), binary cross-entropy (BCE), Brier score, and expected calibration error (ECE). Best results highlighted in bold.

B.2 Epochs Sweep

Epochs	Accuracy	F1	Precision	Recall	ROC-AUC	Hamming	BCE	Brier	ECE
10	0.9056	0.9056	0.9034	0.9081	0.9056	0.0944	0.5509	0.0944	0.0485
50	0.9132	0.9132	0.9110	0.9157	0.9132	0.0868	0.5471	0.0868	0.0447
100	0.9145	0.9145	0.9097	**0.9203**	0.9145	0.0855	0.5469	0.0855	0.0457
200	0.9151	0.9151	**0.9150**	0.9151	0.9151	0.0849	**0.5459**	0.0849	**0.0425**
500	**0.9155**	**0.9155**	0.9132	0.9182	**0.9155**	**0.0845**	0.5460	**0.0845**	0.0436

Table 8: Validation performance across different epoch settings. All experiments used a learning rate of 3×10^{-2} and batch size 32.

While training for 500 epochs achieved the highest performance across most metrics, the gain compared to 50 epochs is minimal. I therefore chose 50 epochs as the default, as it offers a favorable tradeoff between convergence time and performance.

B.3 Batch Size Sweep

BS	LR	Accuracy	F1	Precision	Recall	ROC AUC	Hamming	BCE	Brier
8	0.913283	0.913283	0.911070	0.915751	0.913267	0.086717	0.547092	0.086717	0.044654
16	0.913277	0.913277	**0.911257**	0.915503	0.913263	0.086723	**0.547065**	0.086723	**0.044535**
32	0.913285	0.913285	0.911090	0.915719	0.913273	0.086715	0.547086	0.086715	0.044635
64	0.913211	0.913211	0.911222	0.915397	0.913197	0.086789	0.547094	0.086789	0.044549
128	**0.913387**	**0.913387**	0.910500	**0.916667**	**0.913376**	**0.086613**	0.547137	**0.086613**	0.045006

Table 9: Batch size sweep results (fixed learning rate = 3e-2). Best metric values across rows are wrapped in asterisks. The performance differences are minimal across batch sizes, but a batch size of 128 was selected for its consistent performance and potential computational benefits due to improved parallelism.

B.4 Activation Function Sweep

Activation	Epochs	Batch	LR	Accuracy	F1	Precision	Recall	ROC AUC	Hamming
ReLU	50	128	3e-2	0.913295	0.913295	0.911917	0.914739	0.913279	0.086705
Tanh	50	128	3e-2	0.913203	0.913203	0.910721	**0.916068**	0.913214	0.086797
GELU	50	128	3e-2	**0.914131**	**0.914131**	**0.912516**	0.915861	**0.914129**	**0.085869**
Sigmoid	50	128	3e-2	0.909055	0.909055	0.908773	0.909197	0.909065	0.090945
Leaky ReLU	50	128	3e-2	0.911572	0.911572	0.911617	0.911326	0.911572	0.088428

Table 10: Activation function sweep (fixed epochs = 50, batch size = 128, learning rate = 3e-2). GELU activation achieves the best overall results across nearly all metrics, including accuracy, F1, and BCE. Therefore, GELU was selected as the default activation for subsequent experiments.

B.5 Depth Sweep

Depth	Accuracy	F1	Precision	Recall	ROC-AUC	Hamming	BCE	Brier	ECE
1	0.8834	0.8834	0.8830	0.8837	0.8834	0.1166	0.5618	0.1166	0.0585
2	0.8834	0.8834	0.8830	0.8837	0.8834	0.1166	0.5618	0.1166	0.0585
3	0.9141	0.9141	0.9125	0.9159	0.9141	0.0859	0.5466	0.0859	0.0439
4	0.9261	0.9261	0.9274	0.9247	0.9261	0.0739	0.5402	0.0739	0.0363
6	0.9367	0.9367	0.9320	**0.9420**	0.9367	0.0633	0.5357	0.0633	0.0343
8	**0.9385**	**0.9385**	**0.9357**	0.9418	**0.9385**	**0.0615**	**0.5346**	**0.0615**	**0.0324**
16	0.5832	0.5832	0.4852	0.3849	0.5836	0.4168	0.6881	0.4168	0.1094
32	0.4995	0.4995	0.5152	0.2060	0.5002	0.5005	0.7184	0.5005	0.1034

Table 11: Effect of CNN depth on denoising performance (activation: GELU, width: 32). Deeper models improve accuracy up to depth 8, after which performance collapses.

B.6 Width Sweep

To evaluate the impact of channel width on performance, I ran a sweep over several values for the convolutional layer width while keeping all other hyperparameters fixed. The results are shown in Table 12.

Width	Accuracy	F1	Precision	Recall	ROC AUC	Hamming	BCE	Brier	ECE
8	0.9311	0.9311	0.9272	0.9358	0.9311	0.0689	0.5384	0.0689	0.0368
16	0.9373	0.9373	0.9334	0.9417	0.9373	0.0627	0.5353	0.0627	0.0336
32	**0.9387**	**0.9387**	**0.9347**	**0.9432**	**0.9387**	**0.0613**	**0.5346**	**0.0613**	**0.0330**
64	0.9355	0.9355	0.9334	0.9377	0.9355	0.0645	0.5360	0.0645	0.0334
128	0.9197	0.9197	0.9083	0.9381	0.9197	0.0803	0.5458	0.0803	0.0494
256	0.9239	0.9239	0.9157	0.9342	0.9240	0.0761	0.5427	0.0761	0.0432

Table 12: Performance metrics across different convolutional layer widths. Best values per column are marked with asterisks.

Although several widths performed well, the model with width 32 achieved the best overall performance across nearly all metrics, including accuracy, F1 score, ROC AUC, and calibration-based scores like Brier and ECE. Therefore, I selected a width of 32 for the final model configuration.

B.7 Kernel Size Sweep

The kernel size controls the width of the local window each convolutional filter processes, directly affecting the receptive field and the type of local patterns the network can capture. In the context of denoising binary vectors, a moderate kernel size allows the model to exploit local dependencies while avoiding over-smoothing or overfitting due to excessive contextual aggregation.

I experimented with several odd-valued kernel sizes ranging from 1 to 15. The results, shown below, indicate that a kernel size of 3 yields the best overall performance across all key metrics. Both smaller (e.g., 1) and larger (e.g., 9 or higher) kernel sizes degrade performance significantly. Very large kernels likely introduce unnecessary complexity and dilute the locality prior observed in the data.

Kernel	Accuracy	F1	Precision	Recall	ROC-AUC	HamDist	BCE	Brier	ECE
1	0.5793	0.5793	0.5818	0.5614	0.5793	0.4207	0.7117	0.4207	0.2017
3	**0.9385**	**0.9385**	**0.9364**	**0.9408**	**0.9384**	**0.0615**	**0.5345**	**0.0615**	**0.0320**
5	0.9350	0.9350	0.9352	0.9346	0.9350	0.0650	0.5359	0.0650	0.0324
7	0.7588	0.7588	0.7555	0.7383	0.7583	0.2412	0.6217	0.2412	0.1109
9	0.5833	0.5833	0.4884	0.5870	0.5837	0.4167	0.7122	0.4167	0.2100
11	0.4985	0.4985	0.3977	0.5983	0.4999	0.5015	0.7661	0.5015	0.3002
15	0.4996	0.4996	0.4998	0.3865	0.5008	0.5004	0.7400	0.5004	0.1933

Table 13: Effect of kernel size on model performance (depth=8, width=32, activation=GELU). Best values per column are marked with asterisks.

I therefore selected a kernel size of 3 for the final configuration, as it offers the best balance between locality and model expressiveness, in line with the properties observed in the data.

B.8 Final Configuration

After completing the full greedy search, the best configuration was selected as follows:

- Learning rate: `3e-2`
- Batch size: `128`
- Activation: `gelu`
- Depth: `8`
- Width: `32`
- Kernel size: `3`

This configuration was used for final training and evaluation of the model.

C MAP Estimator Derivation

I begin by modeling the generative process that corrupts the binary vector $X \in \{0,1\}^{512}$ into a real-valued vector $Y \in \mathbb{R}^{512}$. For each index $i \in \{1, \ldots, 512\}$, I assume:

- The corruption is **local and independent across indices**, so I can treat each Y_i independently given X_i.
- The distribution of Y_i depends only on X_i, and follows a Gaussian:

$$Y_i \mid X_i = b \sim \mathcal{N}(\mu_b, \sigma_b^2), \quad b \in \{0,1\}$$

- The prior distribution of X_i is:

$$P(X_i = 1) = p, \quad P(X_i = 0) = 1 - p$$

C.1 MAP Objective

I want:

$$\hat{X}_i = \arg\max_{b \in \{0,1\}} P(X_i = b \mid Y_i)$$

By Bayes' Rule:

$$P(X_i = b \mid Y_i) \propto P(Y_i \mid X_i = b) \cdot P(X_i = b)$$

Using the Gaussian likelihood:

$$\log P(Y_i \mid X_i = b) = -\frac{(Y_i - \mu_b)^2}{2\sigma_b^2} - \frac{1}{2}\log(2\pi\sigma_b^2)$$

So:

$$\log P(X_i = b \mid Y_i) \propto -\frac{(Y_i - \mu_b)^2}{2\sigma_b^2} + \log P(X_i = b)$$

Which gives the final rule:

$$\hat{X}_i = \arg\max_{b \in \{0,1\}} \left[-\frac{(Y_i - \mu_b)^2}{2\sigma_b^2} + \log P(X_i = b) \right]$$

C.2 Special Case: Equal Variance

If $\sigma_0 = \sigma_1 = \sigma$, I obtain a threshold rule:

$$\hat{X}_i = \begin{cases} 1 & \text{if } Y_i > \tau \\ 0 & \text{otherwise} \end{cases}$$

with

$$\tau = \frac{\mu_0 + \mu_1}{2} + \frac{\sigma^2}{\mu_1 - \mu_0} \log\left(\frac{1-p}{p}\right)$$