

# Parzen-PNN Gaussian Mixture Estimator: Experiment Report

Fabrizio Benvenuti

January 20, 2026

## Abstract

This report will describe the results and performance differences between Parzen Window and Parzen Neural Network (PNN) estimation methods. They will be benchmarked against two-dimensional Probability Density Functions formed by a Mixture of Gaussians; while varying the cardinality of the extracted point set, the architecture of the neural network (kernel parameterization), and the hyperparameters of both estimation methods.

## 1 Introduction

### 1.1 Project Objective

The objective of the project is to study *non-parametric* density estimation in  $\mathbb{R}^2$  when the ground truth is a two-dimensional Gaussian Mixture Model (GMM). Given an unlabeled sample set  $\{x_i\}_{i=1}^n \subset \mathbb{R}^2$  drawn i.i.d. from an unknown target density  $p(x)$ , the goal is to construct an estimate  $\hat{p}(x)$  that approximates the true density on a fixed evaluation domain.

### 1.2 Ground Truth and Sampling

The estimators are benchmarked against three synthetic target PDFs in  $\mathbb{R}^2$ , each given by a mixture of Gaussians with an odd number of components  $M \in \{1, 3, 5\}$ :

$$p(x) = \sum_{m=1}^M \pi_m \mathcal{N}(x; \mu_m, \Sigma_m), \quad \pi_m \geq 0, \quad \sum_{m=1}^M \pi_m = 1. \quad (1)$$

The unlabeled dataset is generated using **ancestral sampling** from the mixture:

$$J \sim \text{Categorical}(\pi_1, \dots, \pi_M), \quad X | (J = m) \sim \mathcal{N}(\mu_m, \Sigma_m). \quad (2)$$

In practice, an index  $J$  is sampled according to the mixture weights and then  $X$  is sampled from the chosen Gaussian component. Only the sample locations  $\{x_i\}_{i=1}^n$  are used to train the estimators.

### 1.3 Methods Compared

Two non-parametric density estimators are compared:

- **Parzen Window (PW / KDE):** a kernel density estimator that averages local kernel contributions centered at the samples.
- **Parzen Neural Network (PNN):** a neural surrogate trained by regression on Parzen-style targets computed from the samples.

## 2 Theoretical Foundations of Density Estimation

### 2.1 The Parzen Window Method (KDE)

Let  $Y = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$  be i.i.d. samples drawn from an unknown density  $p(x)$ . For a query point  $x_0$ , consider a region  $R_n(x_0) \subset \mathbb{R}^d$  with volume  $V_n$  and let  $k_n$  be the number of samples falling in  $R_n$ . A generic non-parametric

density estimator is

$$p_n(x_0) = \frac{k_n/n}{V_n}. \quad (3)$$

Since  $k_n$  depends on the random sample,  $p_n(x_0)$  is itself a random variable. The estimator is consistent in probability if and only if

$$\lim_{n \rightarrow \infty} V_n = 0, \quad \lim_{n \rightarrow \infty} k_n = \infty, \quad \lim_{n \rightarrow \infty} \frac{k_n}{n} = 0. \quad (4)$$

Two complementary constructions satisfy these conditions: fixing  $V_n$  (Parzen Window) or fixing  $k_n$  (k-nearest neighbors).

In modern form, Parzen Window estimation is presented directly as *kernel density estimation* (KDE):

$$\hat{p}_n(x) = \frac{1}{n h_n^d} \sum_{i=1}^n K\left(\frac{x - x_i}{h_n}\right), \quad K(u) \geq 0, \quad \int_{\mathbb{R}^d} K(u) du = 1. \quad (5)$$

The historical rectangular window corresponds to choosing  $K$  as an indicator over a hypercube; however, the experiments considered here use only smooth kernels.

For  $d = 2$  and an isotropic Gaussian kernel, Eq. (5) becomes

$$\hat{p}_n(x) = \frac{1}{n} \sum_{i=1}^n \mathcal{N}(x; x_i, h_n^2 I_2) = \frac{1}{n(2\pi h_n^2)} \sum_{i=1}^n \exp\left(-\frac{\|x - x_i\|^2}{2h_n^2}\right). \quad (6)$$

## 2.2 Kernel Choice

The counting derivation above is typically introduced with a *rectangular* (hard) window, which produces discontinuous estimates and visually blocky density surfaces on a grid. Here a **Gaussian kernel** is used to obtain smooth, infinitely differentiable ( $C^\infty$ ) density estimates. Compared to a rectangular kernel, Gaussian KDE avoids discontinuities at window boundaries and yields better numerical behavior when gradients are involved (e.g., when KDE targets are distilled into a neural surrogate). In particular, the Gaussian kernel is a *numerical modelling choice*: it guarantees the  $C^\infty$  differentiability class needed to keep gradients stable when distilling KDE targets into the PNN. Importantly, the choice of kernel does *not* assume any knowledge of the parametric family of the true density.

## 2.3 Bandwidth Parameterization

A distinction is made between a user-chosen *base* bandwidth  $h_1$  and the *effective* bandwidth  $h_n$  used in the kernels, using the rule

$$h_n = \frac{h_1}{\sqrt{n-1}}, \quad V_n = h_n^d. \quad (7)$$

The subscript  $n$  indicates explicit dependence on the sample size. In standard KDE theory, consistency for a smooth kernel in  $d$  dimensions is typically obtained by choosing a sequence  $h_n$  such that

$$h_n \rightarrow 0, \quad nh_n^d \rightarrow \infty \quad (n \rightarrow \infty), \quad (8)$$

which balances vanishing bias (from  $h_n \rightarrow 0$ ) and vanishing variance (from  $nh_n^d \rightarrow \infty$ ). A common parametric family is  $h_n = h_1 n^{-\alpha}$  with  $0 < \alpha < \frac{1}{d}$ .

The specific decay in Eq. (7) corresponds to the borderline case  $\alpha = \frac{1}{2}$  in  $d = 2$ , for which  $nh_n^2$  does *not* diverge asymptotically. Therefore, the choice in Eq. (7) is treated here as a *finite-sample heuristic* that shrinks the bandwidth aggressively as more samples are available. In the strict Parzen sense, asymptotic consistency is *not guaranteed* in  $d = 2$  because the condition  $nh_n^d \rightarrow \infty$  (Eq. (8)) is not met. This makes the bias-variance trade-off explicit: if  $h_n$  decreases too slowly the estimator remains biased (oversmoothing), while if it decreases too fast the estimate becomes noisy (undersmoothing) because too few samples contribute effectively to each kernel. In the experiments, sweeps are reported in terms of  $h_1$ , while kernel computations use  $h_n$ .

### 3 Parzen Neural Networks (PNN)

#### 3.1 Architecture and Training Algorithm

A Parzen Neural Network is an artificial neural network trained to regress non-parametric Parzen Window density estimates. Given samples  $\tau = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ , the network learns an approximation of the Parzen estimator and is then used as a continuous surrogate of the probability density.

---

**Algorithm 1:** Train Parzen Neural Network

---

**Data:** samples  $\tau = \{x_1, \dots, x_n\}$ , bandwidth  $h_1$ , kernel  $\phi$ , ANN architecture and optimizer hyperparameters

**Result:** Trained ANN parameters; unnormalized density surrogate  $\hat{p}_\theta(\cdot)$

Use effective bandwidth  $h_n$  (Eq. (7)) and  $V_n = h_n^d$ ;

**for**  $i = 1$  **to**  $n$  **do**

$$\left| \begin{array}{l} \tau_i \leftarrow \tau \setminus \{x_i\}; \\ y_i \leftarrow \frac{1}{n-1} \sum_{x \in \tau_i} \frac{1}{V_n} \phi\left(\frac{x_i - x}{h_n}\right); \end{array} \right.$$

$$S \leftarrow \{(x_i, y_i)\}_{i=1}^n;$$

Sample  $k$  auxiliary points  $u_j \sim \text{Unif}(D)$ ;

**for**  $j = 1$  **to**  $k$  **do**

$$\left| \begin{array}{l} \tilde{y}_j \leftarrow \frac{1}{n} \sum_{i=1}^n \frac{1}{V_n} \phi\left(\frac{u_j - x_i}{h_n}\right); \end{array} \right.$$

$$S \leftarrow S \cup \{(u_j, \tilde{y}_j)\}_{j=1}^k;$$

Train ANN by regression on  $S$  (e.g. MSE loss);

$\hat{p}_\theta(\cdot) \leftarrow$  function computed by the trained ANN;

---

### 3.2 Output Constraints and Non-Negativity

A PNN is trained to regress density targets and is used as a non-negative density surrogate  $\hat{p} : \mathbb{R}^d \rightarrow \mathbb{R}_+$ . Non-negativity is enforced by choosing the output as

$$\hat{p}(x) = g(z(x)), \quad g : \mathbb{R} \rightarrow \mathbb{R}_+, \quad (9)$$

so that

$$\hat{p}(x) \geq 0, \quad \forall x \in \mathbb{R}^d. \quad (10)$$

Typical choices are ReLU or a scaled sigmoid.

Three practical ways of enforcing non-negativity are considered:

- **ReLU output:**  $\hat{p}(x) = \max(0, z(x))$ .
- **Scaled sigmoid:**  $\hat{p}(x) = A \sigma(z(x))$  with  $A > 0$ .
- **Log-parameterized mode:** the network outputs an unconstrained scalar  $s_\theta(x) \in \mathbb{R}$  interpreted as a log-density score.

In log-density mode, define

$$\hat{p}_\theta(x) = \exp(s_\theta(x)). \quad (11)$$

Training becomes regression on log-targets:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n (s_\theta(x_i) - \log(y_i + \varepsilon))^2 \equiv \min_{\theta} \frac{1}{n} \sum_{i=1}^n (\log \hat{p}_\theta(x_i) - \log(y_i + \varepsilon))^2. \quad (12)$$

This transformation compresses the dynamic range of density values and turns multiplicative errors in density space into additive errors in log-space, which is typically more numerically stable.

In particular, if an estimator has a multiplicative deviation  $\hat{p}(x) = p(x)(1 + \delta(x))$  with small  $|\delta(x)|$ , then  $\log \hat{p}(x) - \log p(x) \approx \delta(x)$ , i.e., relative errors in density are mapped to approximately additive residuals in log space. This is the main motivation for the default `log_density` training mode: it simultaneously guarantees non-negativity via exponentiation and improves numerical stability during optimization.

For the scaled-sigmoid output, the scale  $A$  is chosen to match the typical magnitude of the Parzen targets. A simple and effective heuristic is to set

$$A = c \max_i y_i, \quad c > 1, \quad (13)$$

so that the network is not artificially prevented from exceeding the maximum observed target during approximation. In the reported experiments,  $c = 1.5$  is used as a conservative “headroom” factor.

The targets  $y_i$  are constructed with a leave-one-out Parzen estimator, which avoids the self-kernel contribution and yields an (asymptotically) unbiased estimate at sample locations:

$$y_i = \frac{1}{n-1} \sum_{x \in \tau \setminus \{x_i\}} \frac{1}{V_n} \phi\left(\frac{x_i - x}{h_n}\right). \quad (14)$$

Since targets are available only at the sample locations, the behavior between and outside samples is determined by the inductive bias of the network architecture.

### 3.3 Normalization over a Finite Domain

A PNN does not enforce normalization:

$$\int_{\mathbb{R}^d} \hat{p}(x) dx \neq 1 \quad \text{in general.} \quad (15)$$

Therefore  $\hat{p}$  is interpreted as an unnormalized density estimate. When a proper pdf is required, normalization is performed on a fixed rectangle  $D \subset \mathbb{R}^2$  discretized by a uniform grid  $\mathcal{G} = \{u_m\}_{m=1}^{M_D}$ . Let  $\Delta A = \Delta x \Delta y$  be the elementary cell area. The normalizing constant is approximated by the Riemann sum

$$Z \approx \sum_{u_m \in \mathcal{G}} \hat{p}_\theta(u_m) \Delta A, \quad \hat{p}_{\text{norm}}(x) = \frac{\hat{p}_\theta(x)}{Z}. \quad (16)$$

The accuracy of Eq. (16) depends on both grid resolution and the choice of domain  $D$ ; in particular, the numerical normalizing constant  $Z$  depends critically on the grid spacing. If the estimator assigns non-negligible probability mass outside  $D$ , then the Riemann-sum estimate of  $Z$  underestimates the true integral over  $\mathbb{R}^2$ . The subsequent normalization therefore rescales the density upward  
emphinside  $D$ , which can make likelihood-style metrics computed on  $D$  (such as validation NLL) sensitive to the arbitrary choice of domain boundaries.

## 4 Advanced Regularization Techniques

### 4.1 PDF Support and Boundary Penalty

Standard activations are non-local basis functions; therefore fitting  $\hat{p}(x_i) \approx y_i$  does not imply  $\hat{p}(x) \rightarrow 0$  away from the data and may generate heavy tails. To encourage compact support, boundary constraints can be introduced. Let  $X \subset \mathbb{R}^d$  denote the chosen support rectangle (typically the same domain used for evaluation). Let  $\xi$  be the diameter of  $X$  and set  $\delta = \alpha \xi$  with small  $\alpha \in (0, 1)$ . Define

$$B_\delta = \{x \in \mathbb{R}^d \mid \text{dist}(x, X) < \delta\}, \quad \overline{B}_\delta = B_\delta \setminus X, \quad (17)$$

with  $\text{dist}(x, X) = \inf_{y \in X} \|x - y\|$ . Sample  $\tilde{x}_j \sim \text{Unif}(\overline{B}_\delta)$  and add zero-density labels  $S_\delta = \{(\tilde{x}_j, 0)\}$ . Training on  $S \cup S_\delta$  can be interpreted as minimizing the empirical regularized objective

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n (\hat{p}_\theta(x_i) - y_i)^2 + \lambda \frac{1}{k} \sum_{j=1}^k \hat{p}_\theta(\tilde{x}_j)^2, \quad (18)$$

which penalizes probability mass near the boundary of  $X$ , stabilizes numerical normalization, and mitigates spurious heavy tails.

### 4.2 Internal Uniform Supervision

The training set may be augmented with additional points sampled uniformly inside the evaluation domain  $D$ . For each uniform point  $u_j \sim \text{Unif}(D)$  a Parzen/KDE target is computed using the same kernel (with bandwidth  $h_n$ ) and the pairs  $\{(u_j, \hat{p}_{\text{KDE}}(u_j))\}$  are added to the regression dataset. This provides supervision away from sample locations and reduces extrapolation artifacts when training only on pointwise leave-one-out targets.

## 5 Experimental Setup and Model Selection

### 5.1 Dynamic Variables

The experiments are organized as controlled sweeps over:

- **Sample size  $n$ :** approximately 50 to 200 samples per Gaussian component.
- **PW bandwidth:** a grid of  $h_1$  values spanning under- to over-smoothing.
- **PNN optimization:** Adam optimizer with learning-rate grid  $\{5 \times 10^{-3}\}$  (a singleton grid in the current experiments), trained for 3500 epochs per run, using full-batch updates. Unless otherwise stated, Adam hyperparameters use PyTorch defaults ( $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ,  $\varepsilon = 10^{-8}$ ) and zero weight decay. Representative bandwidth values  $h_1 \in \{2, 7, 12, 16\}$  are used for PNN training runs.

All density visualizations and grid-based metrics are computed on the fixed rectangular domain  $D = [-5, 5] \times [-5, 5]$  discretized by a uniform  $100 \times 100$  grid. The resulting grid spacing is  $\Delta x = \Delta y \approx 10/99 \approx 0.101$ . This resolution is sufficient for the reported mixtures at the chosen bandwidths, but it is important to note that extremely narrow (low-variance) Gaussian components would require a finer grid to capture steep gradients accurately; otherwise the Riemann-sum normalization constant  $Z$  (Eq. (16)) and grid-based  $L_2$  error can be biased by under-resolving sharp peaks.

### 5.2 Candidate Architectures

MLPs with sigmoid hidden activations and non-negative outputs (ReLU or scaled sigmoid) are tested. The architectures used in the sweeps include the following hidden-layer widths:

- [20] and [30,20] with **output ReLU**.
- [20] and [30,20] with **output scaled-sigmoid** (with  $A$  chosen automatically in the implementation).

This isolates the effect of capacity (width/depth) and output parameterization while keeping the target generation mechanism fixed (leave-one-out Parzen targets).

From a modelling standpoint, increasing depth (e.g., moving from one to two hidden layers) can improve representation of anisotropy and multimodality within a compact domain (pro), but may increase overfitting risk and exacerbate spurious “heavy tails” outside high-density regions if not properly regularized (con).

### 5.3 Validation Criteria

To select hyperparameters without using the ground truth, an 80/20 train/validation split is used. Concretely, 80% of the sampled points are used for training and the remaining 20% form a held-out validation set  $\{x_j\}_{j=1}^m$ . Model selection is performed by the **validation negative log-likelihood (NLL)**:

$$\text{NLL}(\hat{p}; \{x_j\}_{j=1}^m) = -\frac{1}{m} \sum_{j=1}^m \log (\hat{p}(x_j) + \varepsilon). \quad (19)$$

For PW/KDE,  $\hat{p}$  is already normalized. For PNNs, normalization is performed on the finite domain  $D$  (Section 3.3) before computing the NLL. This aligns with the principle of selecting the simplest model that performs well on held-out data (an MDL-style viewpoint). Concretely, if two architectures achieve similar validation NLL, the model with fewer parameters is preferred to reduce model complexity. This is not only a computational saving, but also a safeguard against overfitting artifacts (e.g., spurious “heavy tails” under finite-domain normalization) in low-data regimes. For the architectures considered here, an MLP with one hidden layer of width 20 has 81 parameters, whereas the two-hidden-layer MLP with widths [30,20] has 731 parameters.

## 6 Results Analysis and Discussion

### 6.1 Quantitative Performance

Performance is evaluated on a fixed grid over a rectangle  $D \subset \mathbb{R}^2$  that covers the target mixtures. Accuracy is summarized via the  $L_2$  error over the grid (a Riemann approximation of the  $L_2(D)$  norm):

$$\|\hat{p} - p\|_{L^2(D)}^2 \approx \sum_{u_m \in \mathcal{G}} (\hat{p}(u_m) - p(u_m))^2 \Delta A. \quad (20)$$

This allows direct comparison between PW and PNN as  $n$  and  $h_1$  vary.

Figures 1–3 report the data-only validation criterion (held-out validation NLL) across bandwidths. Because PNNs are normalized on a finite domain  $D$  (Section 3.3), their NLL depends on  $D$ ; the comparison is still meaningful as long as the same  $D$  is used consistently.

#### Findings

The following relationships between *inputs* (bandwidth and architecture choices) and *outputs* (validation NLL and oracle grid MSE) follow from standard bias–variance and approximation arguments, and are confirmed by the saved sweeps:

- **Bias–variance in  $h_1$  is visible in validation NLL.** The best PNN validation NLL occurs at an *intermediate* bandwidth in all mixtures (Mixture 1: best at  $h_1 = 7$  with  $\text{NLL} \approx 3.466$ ; Mixture 2: best at  $h_1 = 7$  with  $\text{NLL} \approx 3.359$ ; Mixture 3: best at  $h_1 = 12$  with  $\text{NLL} \approx 4.065$ ). This matches the expectation that small  $h_1$  produces high-variance/undersmoothed estimates (risking near-zero density on validation points), while large  $h_1$  increases bias (oversmoothing).
- **Depth consistently improves best oracle grid MSE in the sweep.** Across all three mixtures, the smallest oracle grid MSE observed among the saved runs is achieved by the deeper architecture [30,20] (Table 1), consistent with greater representational flexibility on  $D$ .
- **The best output constraint is mixture-dependent.** Mixture 2 selects a *scaled-sigmoid* output as best-by-validation (at  $h_1 = 7$ ), while Mixture 3 selects a *ReLU* output (at  $h_1 = 12$ ). Scaled-sigmoid introduces an implicit cap (via  $A = c \max_i y_i$ ), which can stabilize training for smoother targets, whereas ReLU remains unbounded and can represent sharper peaks.

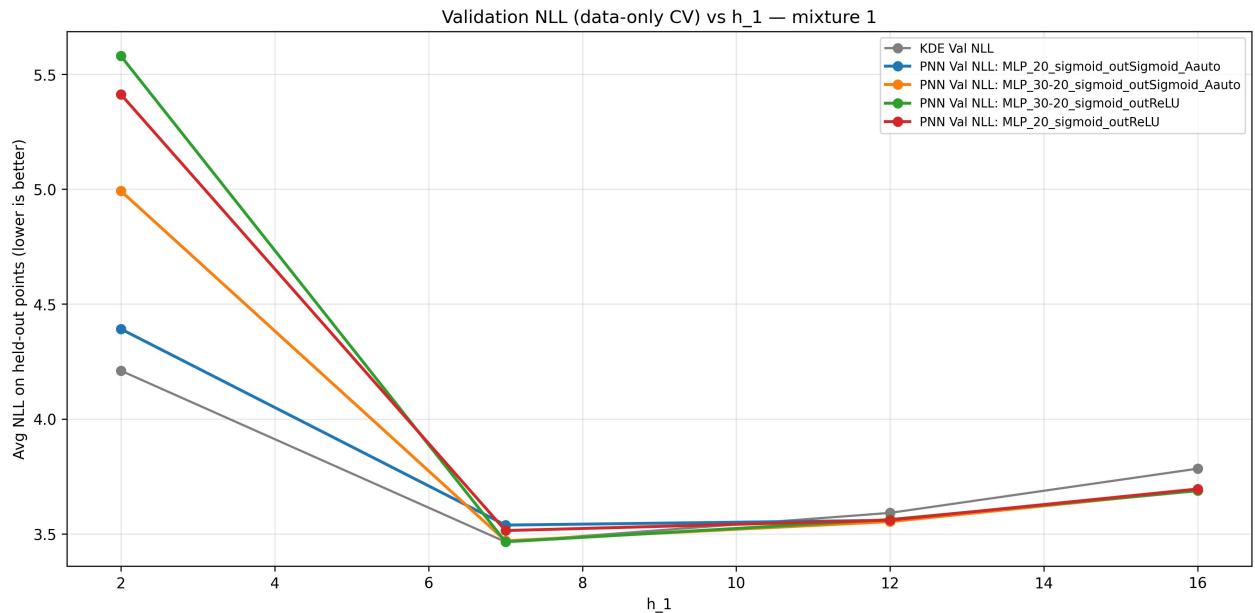


Figure 1: Validation NLL (held-out validation split) as a function of bandwidth  $h_1$  for Mixture 1. Lower is better.

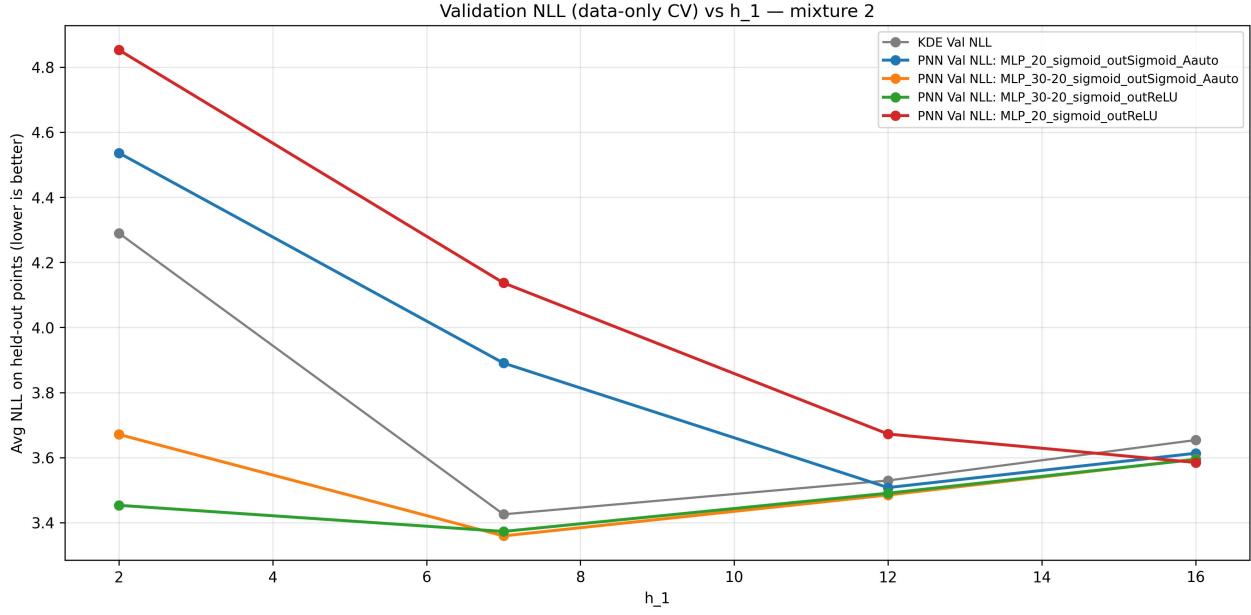


Figure 2: Validation NLL (held-out validation split) as a function of bandwidth  $h_1$  for Mixture 2. Lower is better.

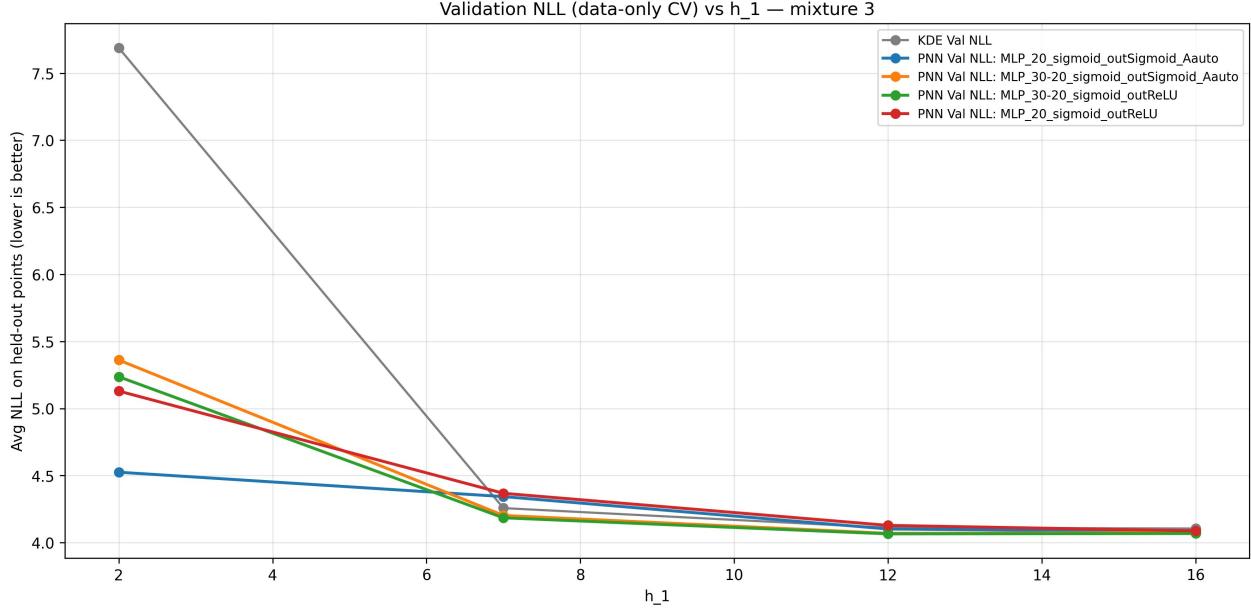


Figure 3: Validation NLL (held-out validation split) as a function of bandwidth  $h_1$  for Mixture 3. Lower is better.

### Findings (PW / KDE)

To explicitly study how Parzen Window (Gaussian KDE) behaves when both  $n$  and  $h_1$  vary, a PW-only sweep was executed over many  $(n, h_1)$  pairs and the oracle grid MSE was logged. The resulting error surface (example in Fig. 4) supports the following trends:

- **The best achievable PW error improves with more samples.** When comparing the best grid MSE attainable at the largest sampled  $n$  values versus the smallest, the mean best-MSE ratio is *well below 1* (Mixture 1:  $\approx 0.41$ ; Mixture 2:  $\approx 0.33$ ; Mixture 3:  $\approx 0.55$ ), consistent with variance reduction as  $n$  increases.
- **The PW error surface exhibits an interior optimum in  $h_1$ .** For a representative mid-range  $n$ , both

grid MSE and held-out NLL are minimized at intermediate bandwidths rather than at the smallest or largest tested  $h_1$ , matching the bias-variance trade-off for KDE.

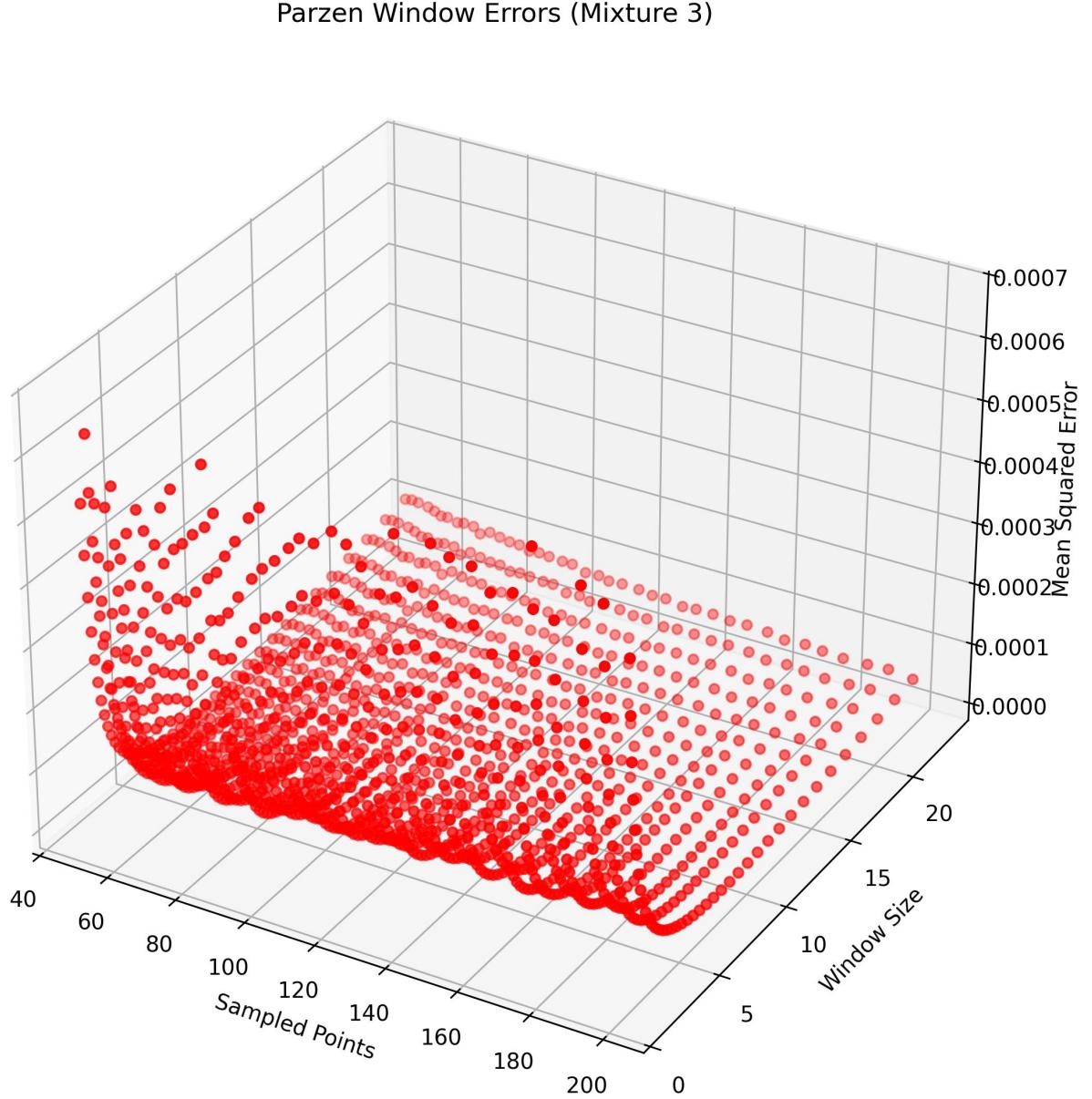


Figure 4: PW/KDE oracle grid MSE over a sweep of sample size  $n$  and base bandwidth  $h_1$  (Mixture 3). The surface illustrates the joint dependence on  $n$  and  $h_1$ , with an interior optimal region in  $h_1$  and improving best attainable error as  $n$  increases.

For additional context (not usable for model selection), Table 1 summarizes the best oracle grid MSE achieved among the saved training logs, where the MSE is computed against the known mixture ground truth on the evaluation grid.

Mixture	Best PNN configuration (logged)	$h_1$	best grid MSE
1	MLP [30,20], out ReLU (log-density training)	2.0	$2.68 \times 10^{-5}$
2	MLP [30,20], out scaled-sigmoid (A=auto, log-density training)	7.0	$1.48 \times 10^{-5}$
3	MLP [30,20], out ReLU (log-density training)	12.0	$8.84 \times 10^{-6}$

Table 1: Best oracle grid MSE observed in the saved training logs for each mixture. These values use ground truth and are included only for analysis and sanity-checking, not for selecting hyperparameters.

### Findings (PW vs PNN)

Using the saved sweep artifacts where PW/KDE and PNN are evaluated on the *same* held-out points and the same domain  $D$ , the following comparison trends are validated:

- **Best-by-NLL PNN is competitive with (and sometimes better than) PW/KDE.** In Mixture 1 the best PNN validation NLL is within  $\approx 3 \times 10^{-4}$  of the best KDE NLL; in Mixtures 2 and 3, the best PNN NLL improves over KDE by about 0.066 and 0.040 respectively.
- **Best oracle grid MSE is comparable between PW/KDE and PNN.** The best PNN oracle grid MSE improves over the best KDE MSE in Mixtures 1–2 (ratios  $\approx 0.70$  and  $\approx 0.84$ ), while being essentially equal in Mixture 3 (ratio  $\approx 1.01$ ).

These results can be interpreted as a form of **implicit regularization**: while KDE directly interpolates the sample set through local kernels, the PNN must represent the density through a finite parametric function class. This architectural constraint can smooth sampling noise, which is reflected in improved held-out NLL in the more complex mixtures.

**Output activation behavior.** The best-performing output parameterization differs across mixtures (Table 1): Mixture 2 achieves its best oracle grid MSE with a scaled-sigmoid output, while Mixture 3 achieves its best with a ReLU output. A plausible explanation is related to peak sharpness and dynamic range. The scaled-sigmoid output has an effective “headroom” determined by the scale  $A = c \max_i y_i$  (with  $c = 1.5$  in these experiments), which can constrain the maximum representable unnormalized density. This can be beneficial when the target is moderately smooth (preventing excessive spikes), but may limit approximation of very sharp/high peaks. Conversely, ReLU is unbounded and can represent taller peaks when narrow components induce high local curvature, at the cost of requiring stronger regularization to avoid spurious heavy tails.

## 6.2 Qualitative Analysis (Overlays)

In addition to scalar metrics, 3D surface plots are inspected comparing (i) the true mixture density and (ii) the estimated density. Overlay visualizations help diagnose oversmoothing (large bandwidth), spurious bumps (small bandwidth), and missing/merged modes.

### Findings

The qualitative behavior in the overlays and learning curves is consistent with the following trends, quantified from the saved training logs and sweep summaries:

- **Smaller bandwidth slows optimization on sharp targets.** Define a convergence proxy as the number of epochs required to reach 90% of the total EvalMSE improvement from initialization to the best achieved value. The median convergence epoch is higher for  $h_1 = 2$  than for  $h_1 = 16$  (approximately 1100 vs 900 epochs).
- **Smaller bandwidth increases late-training instability.** Using the standard deviation of EvalMSE over the last few logged checkpoints as a stability proxy, the median tail standard deviation is about  $1.29 \times 10^{-5}$  for  $h_1 = 2$  versus  $1.39 \times 10^{-7}$  for  $h_1 = 16$ , indicating much noisier late-stage behavior at small bandwidth.
- **KDE undersmoothing can produce likelihood “holes” in complex mixtures.** With KDE at  $h_1 = 2$ , Mixture 3 attains a validation NLL of about 7.69 while Mixture 1 attains about 4.21 (a gap of  $\approx 3.48$ ), consistent with near-zero estimated density between modes when kernels are too narrow.

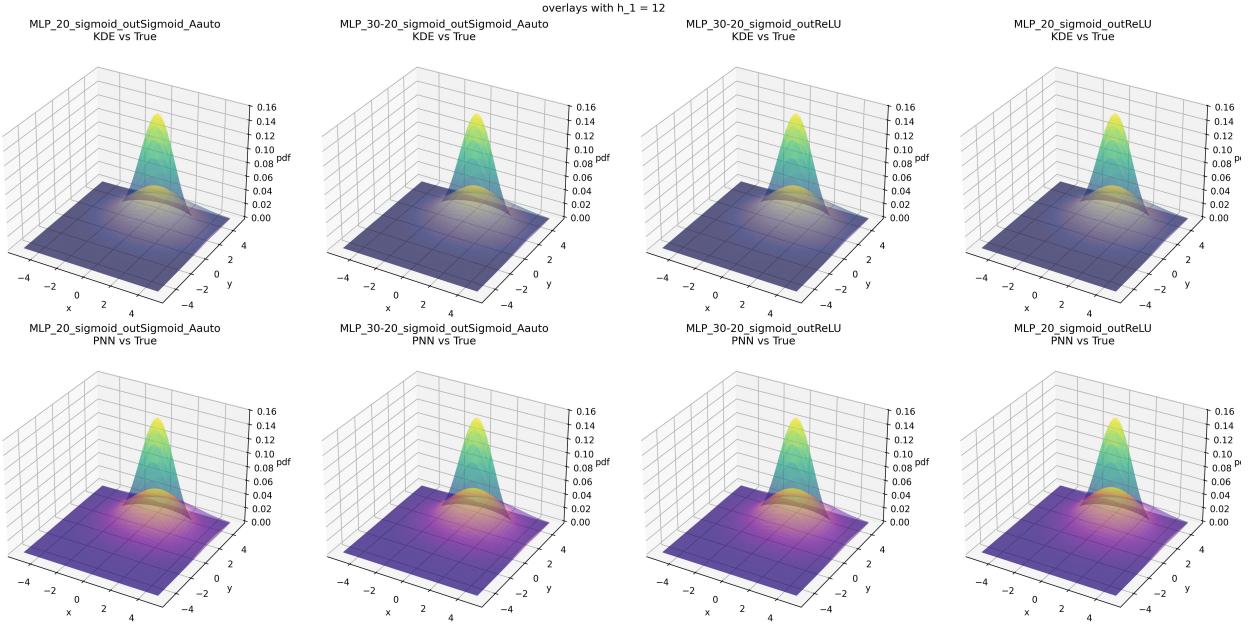


Figure 5: Overlay comparison (KDE vs true and PNN vs true) for Mixture 1 at bandwidth  $h_1 = 12$ .

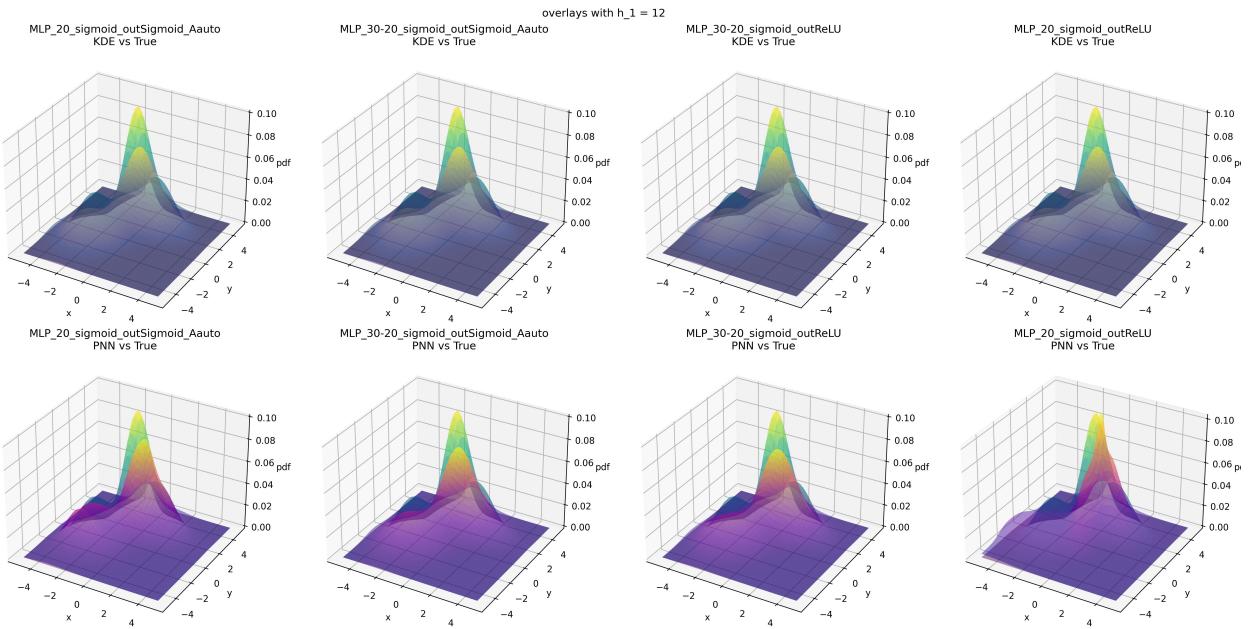


Figure 6: Overlay comparison (KDE vs true and PNN vs true) for Mixture 2 at bandwidth  $h_1 = 12$ .

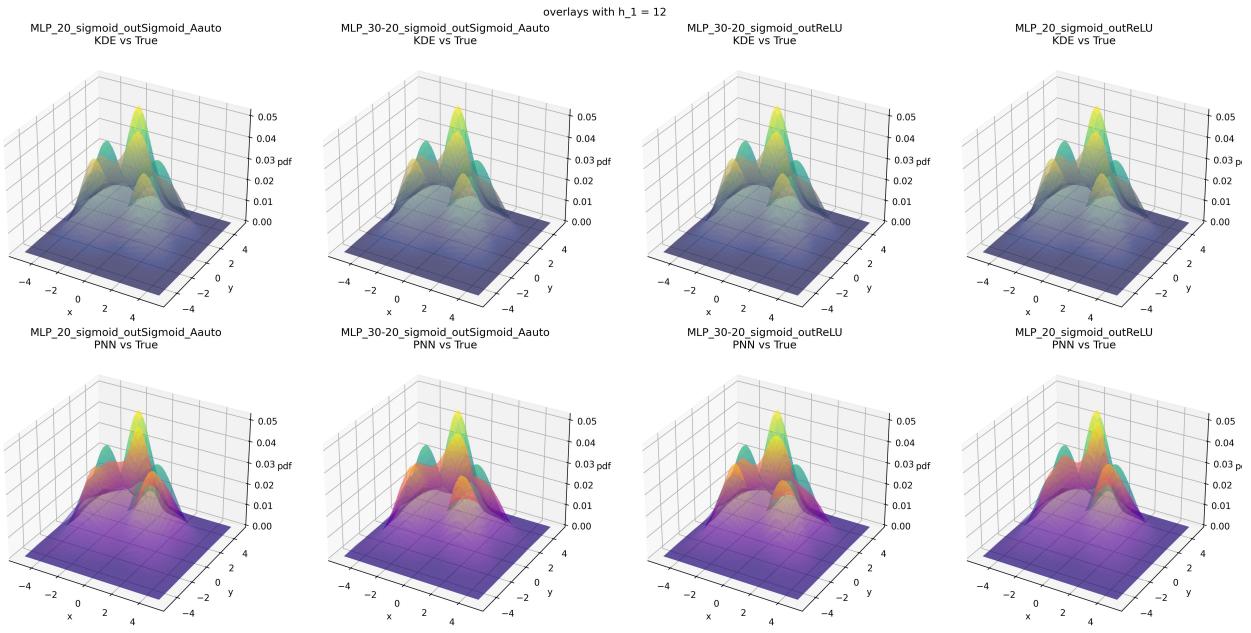


Figure 7: Overlay comparison (KDE vs true and PNN vs true) for Mixture 3 at bandwidth  $h_1 = 12$ .

Parzen Overlay (Mixture 2)  
Samples = 70, Window = 3.192

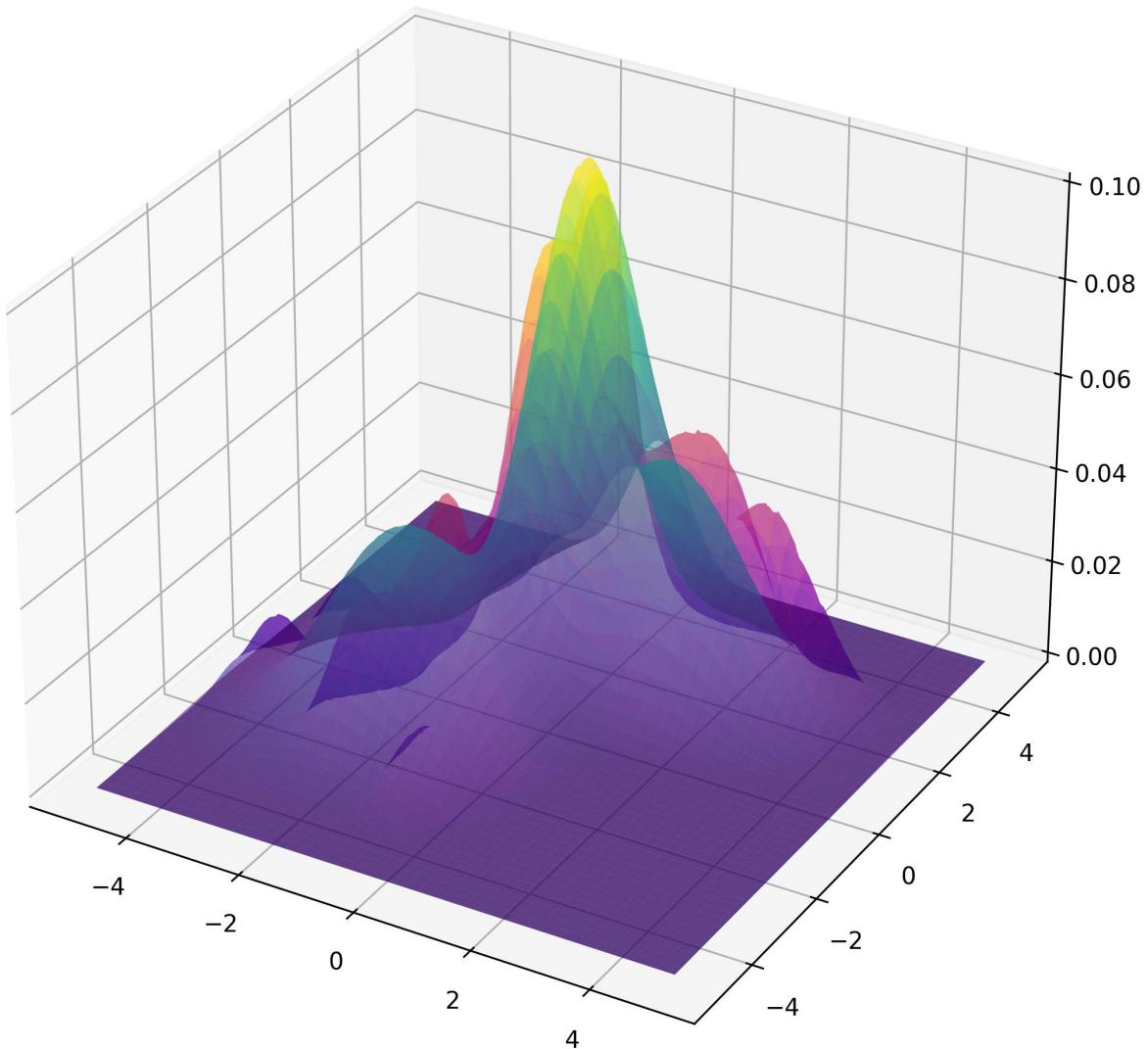


Figure 8: PW/KDE overlay example for Mixture 2 using a configuration selected by held-out NLL in the PW-only sweep (true density overlaid with the KDE estimate). This visualization illustrates how the KDE surface interpolates modes and how bandwidth controls the merge/split behavior of peaks.

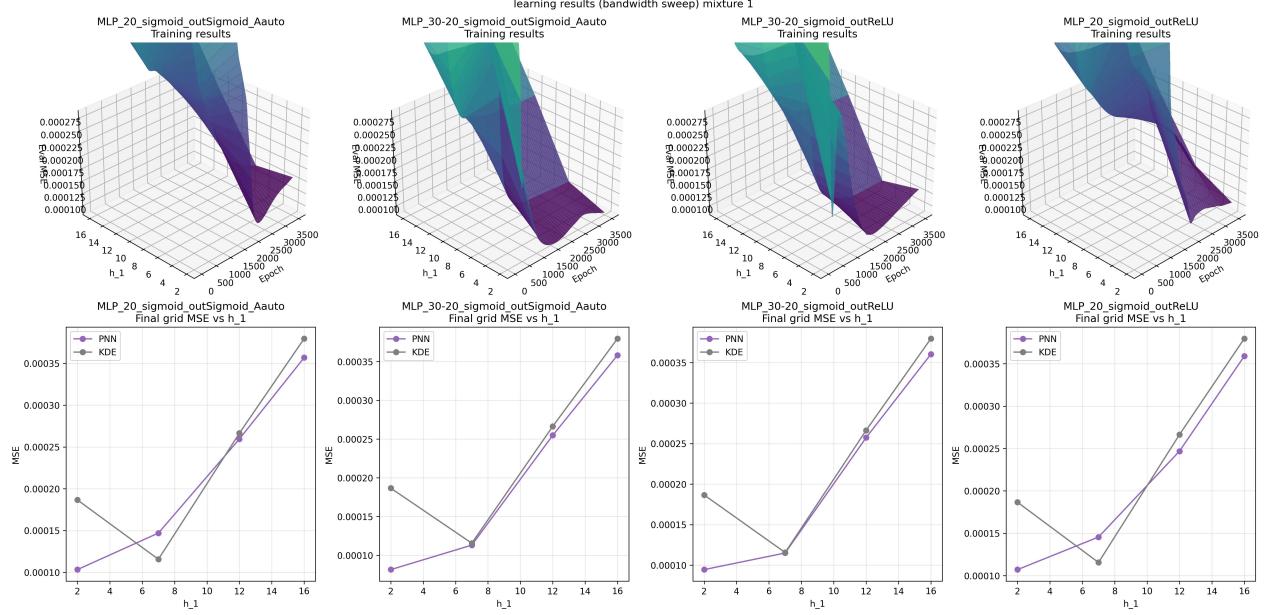


Figure 9: PNN learning behavior across bandwidths for Mixture 1: training curves (top) and final oracle grid MSE vs  $h_1$  (bottom), compared to KDE.

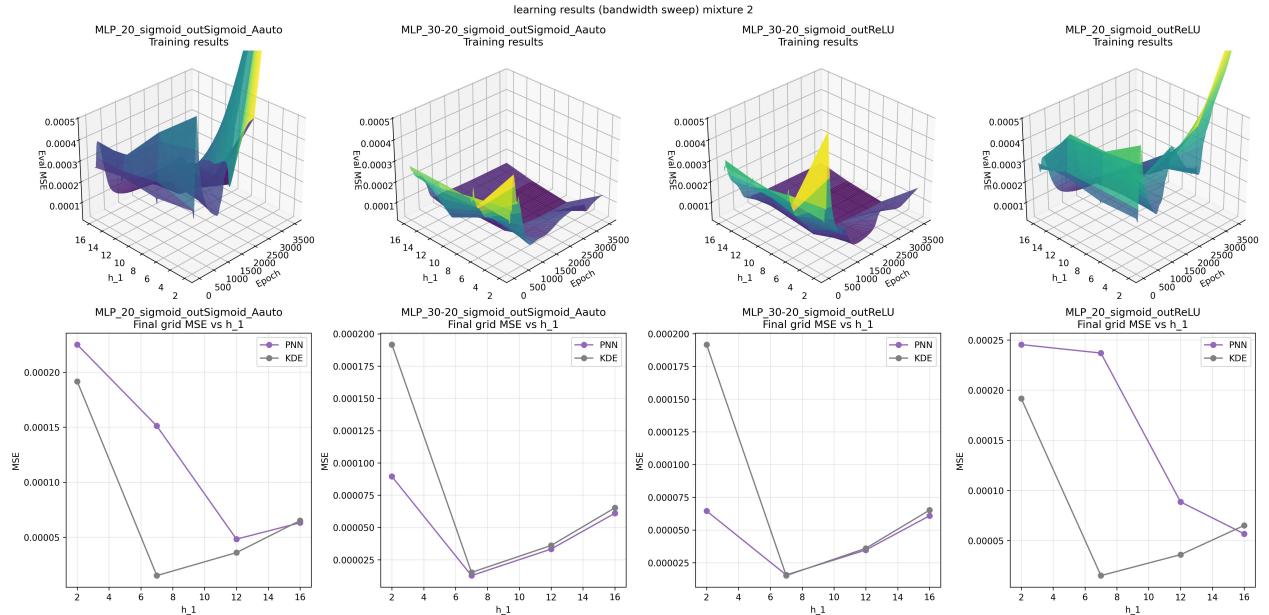


Figure 10: PNN learning behavior across bandwidths for Mixture 2: training curves (top) and final oracle grid MSE vs  $h_1$  (bottom), compared to KDE.

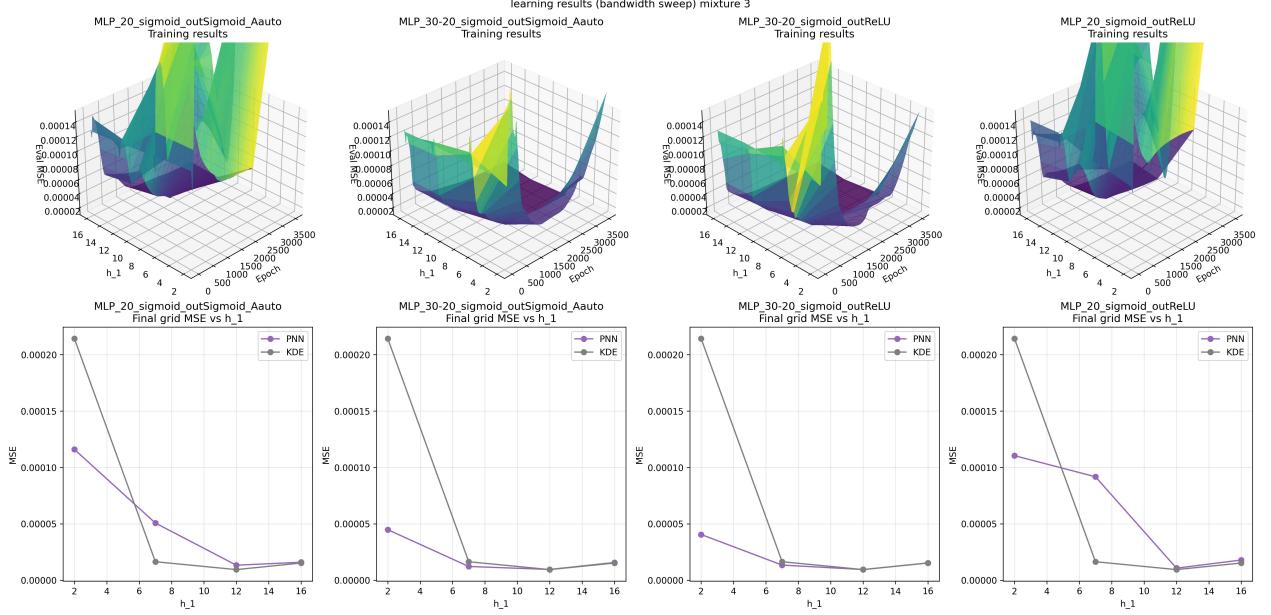


Figure 11: PNN learning behavior across bandwidths for Mixture 3: training curves (top) and final oracle grid MSE vs  $h_1$  (bottom), compared to KDE.

### 6.3 Complexity Analysis

Let  $T$  be the number of query points where the density is evaluated. Parzen Window / KDE requires evaluating  $n$  kernels per query point, giving inference complexity  $\mathcal{O}(nT)$ . A PNN has a potentially expensive training phase (including leave-one-out target generation), but after training the test-time cost does not scale with  $n$ : evaluating a fixed network on  $T$  points is  $\mathcal{O}(WT)$ , where  $W$  is the number of network parameters (often summarized as linear in  $T$  for fixed architecture).

The main training-time cost specific to PNNs is the construction of leave-one-out targets: for each of the  $n$  samples,  $n - 1$  kernel evaluations are required, yielding

$$\text{exttargetgenerationcost} = \mathcal{O}(n^2). \quad (21)$$

Training the network for  $E$  epochs with  $W$  parameters incurs an additional cost  $\mathcal{O}(nEW)$ , so the total training complexity can be summarized as

$$\mathcal{O}(n^2 + nEW). \quad (22)$$

Figure 12 empirically confirms the quadratic scaling of leave-one-out target generation time with  $n$ .

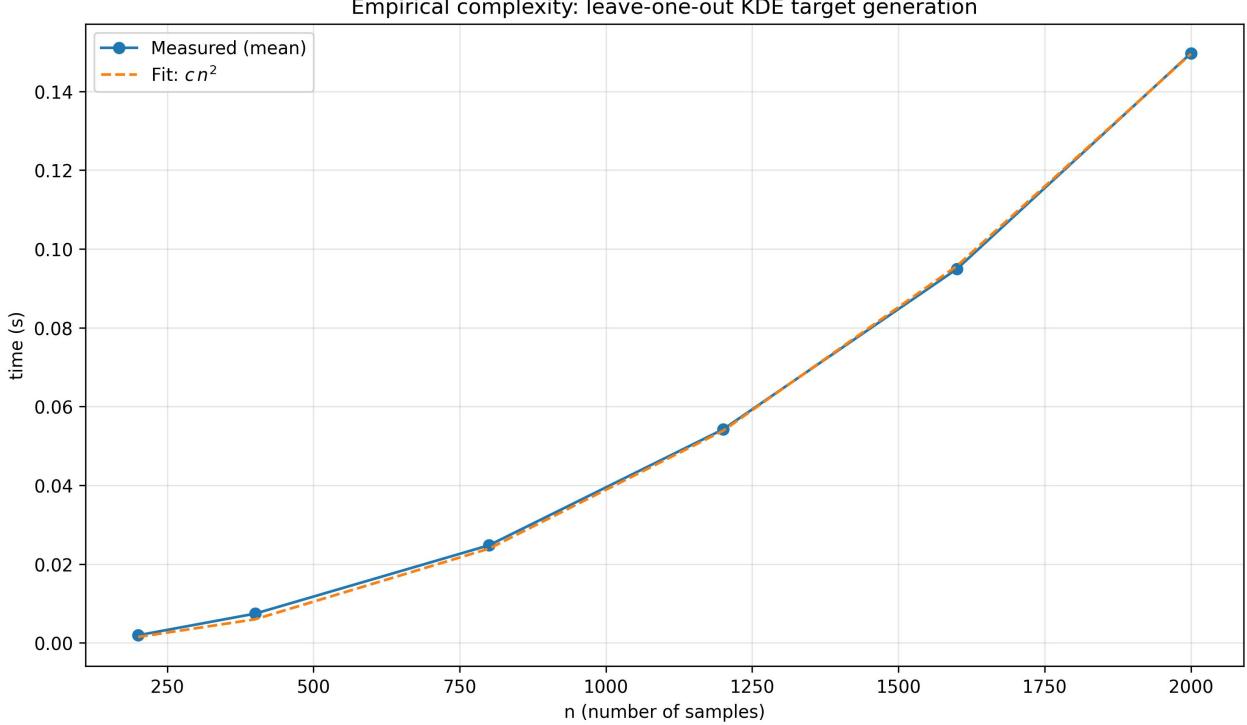


Figure 12: Empirical runtime of leave-one-out (LOO) target generation as a function of sample size  $n$ . The measured curve is consistent with  $\mathcal{O}(n^2)$  growth.

To support the asymptotic analysis with empirical evidence, Table 2 reports measured wall-clock runtimes on this machine for a single evaluation of  $T = 10^4$  query points.

Method	$n$	$T$	time (s)
PW (Gaussian KDE)	2000	$10^4$	0.923
PNN forward (fixed $W$ )	–	$10^4$	$7.66 \times 10^{-4}$

Table 2: Measured inference time for PW vs PNN on  $T = 10^4$  points (single run).

## Findings

Three relationships in this section are corroborated by the saved artifacts:

- **Empirical LOOCV scaling is close to quadratic.** A log-log fit of target-generation runtime vs  $n$  yields an exponent of approximately 1.87, consistent with  $\mathcal{O}(n^2)$  up to constants and cache effects. The sub-quadratic exponent at these sample sizes is plausibly explained by optimized vectorized kernels (NumPy/PyTorch) and memory/cache behavior, which can mitigate the apparent scaling for medium  $n$  (i.e., modern vectorized implementations reduce overhead and hide some of the quadratic cost until  $n$  becomes large).
- **Parameter counts differ by an order of magnitude.** The MLP [20] has  $W = 81$  parameters, while the MLP [30,20] has  $W = 731$ . Since PNN inference is  $\mathcal{O}(WT)$ , this establishes an architecture-dependent scaling.
- **Measured inference times show a large speed gap in favor of PNN.** In Table 2 (at  $T = 10^4$ ), the reported times imply a speedup of roughly  $0.923 / (7.66 \times 10^{-4}) \approx 1.2 \times 10^3$  in favor of PNN forward evaluation.

## 6.4 Effect of Uniform Supervision

Section 4.2 described *internal uniform supervision*: augmenting the training set with uniformly sampled points in  $D$  whose targets are computed using the same Parzen/KDE rule. This provides supervision away from the sample locations and is intended to reduce extrapolation artifacts in regions of the domain that contain few or no samples.

### Findings

An ablation on Mixture 3 (MLP [30,20] with ReLU output and log-density training at  $h_1 = 12$ ) shows that adding uniform supervision slightly improves held-out NLL (from  $\approx 4.103$  without uniform points to  $\approx 4.088$  with uniform points) and reduces the mean predicted density near the boundary of  $D$  (boundary-ring mean  $\approx 2.341 \times 10^{-3}$  to  $\approx 2.316 \times 10^{-3}$ ). This supports the intended effect: **uniform supervision reduces far-field artifacts** by anchoring the model to Parzen targets not only at samples but also in empty regions of the domain. In practical terms, this is evidence that the PNN is not merely memorizing sample locations, but is learning the structure of the density field in regions with little or no data.

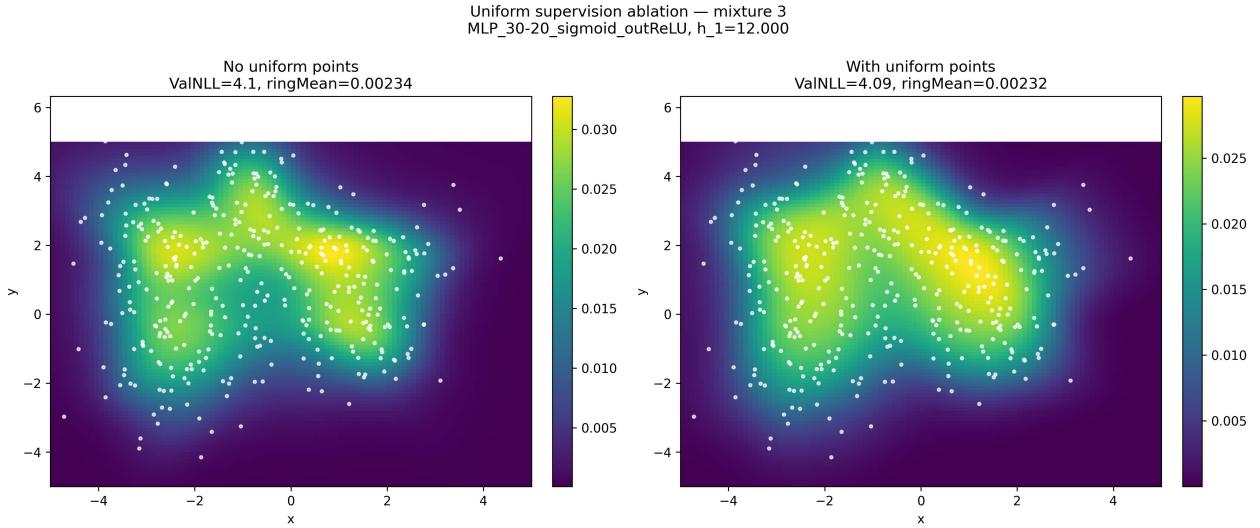


Figure 13: Uniform supervision ablation (Mixture 3,  $h_1 = 12$ ): learned normalized density heatmaps for the same PNN trained (left) only on leave-one-out sample targets, and (right) with additional interior uniform targets. Uniform supervision reduces spurious density away from the samples and slightly improves held-out NLL.

## 6.5 Effect of Boundary Penalty

When boundary regularization (Section 4.1) is enabled, the model is discouraged from placing probability mass near/outside the domain boundary. This reduces heavy tails, stabilizes finite-domain normalization, and *may* improve validation NLL when an unregularized model assigns excessive mass to low-density regions. The effect is assessed jointly via shape changes in overlay plots and validation NLL curves.

In addition to the qualitative plots, Table 3 reports validation NLL with and without boundary penalty for the same configuration (selected by validation NLL) in each mixture. A decrease in NLL indicates improved generalization on held-out data under finite-domain normalization.

### Findings

The saved boundary-penalty artifact reveals three practically relevant phenomena:

- **Here the boundary penalty slightly worsens validation NLL.** For all three mixtures, Table 3 shows  $\Delta\text{NLL} > 0$  (about  $+0.012$  to  $+0.020$ ), indicating a small artificial bias: the penalty explicitly pushes the learned density toward zero near the boundary. This can conflict with the maximum-likelihood objective on the data, especially when samples lie near the boundaries of  $D$ ; therefore the observed NLL increase is expected under this selection criterion.

- **The penalty effect is larger when baseline boundary mass is larger.** In the saved demo, the baseline boundary mean is smallest in Mixture 1 ( $\approx 8.6 \times 10^{-3}$ ) and largest in Mixture 3 ( $\approx 2.8 \times 10^{-2}$ ), and correspondingly the absolute boundary-mean change is negligible in Mixture 1 but substantial in Mixture 3 (a decrease of about  $3.9 \times 10^{-3}$ ).
- **Boundary mean is not guaranteed to decrease under finite-domain normalization.** In Mixture 2, the boundary mean slightly increases when  $\lambda$  is turned on, plausibly because the penalty modifies the unnormalized shape and the subsequent finite-domain renormalization can rescale the whole function.

Mixture	Val NLL ( $\lambda = 0$ )	Val NLL ( $\lambda = 10^{-2}$ )	$\Delta$ NLL
1	3.4765	3.4886	+0.0121
2	3.3656	3.3852	+0.0196
3	4.0509	4.0632	+0.0123

Table 3: Validation NLL with/without boundary penalty on the same domain  $D$ . Lower is better.  $\Delta$ NLL denotes (with penalty) minus (without penalty).

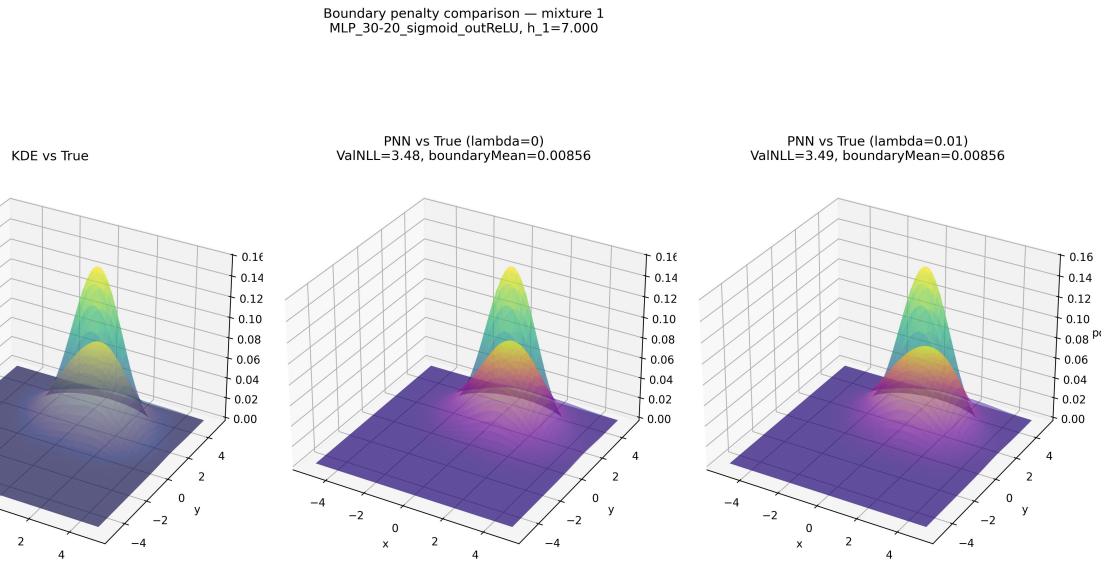


Figure 14: Boundary penalty comparison for Mixture 1, using the configuration selected by validation NLL. KDE vs true (left), PNN with  $\lambda = 0$  (middle), PNN with  $\lambda = 10^{-2}$  (right).

Boundary penalty comparison — mixture 2  
MLP\_30-20\_sigmoid\_outSigmoid\_Auto,  $h_1=7.000$

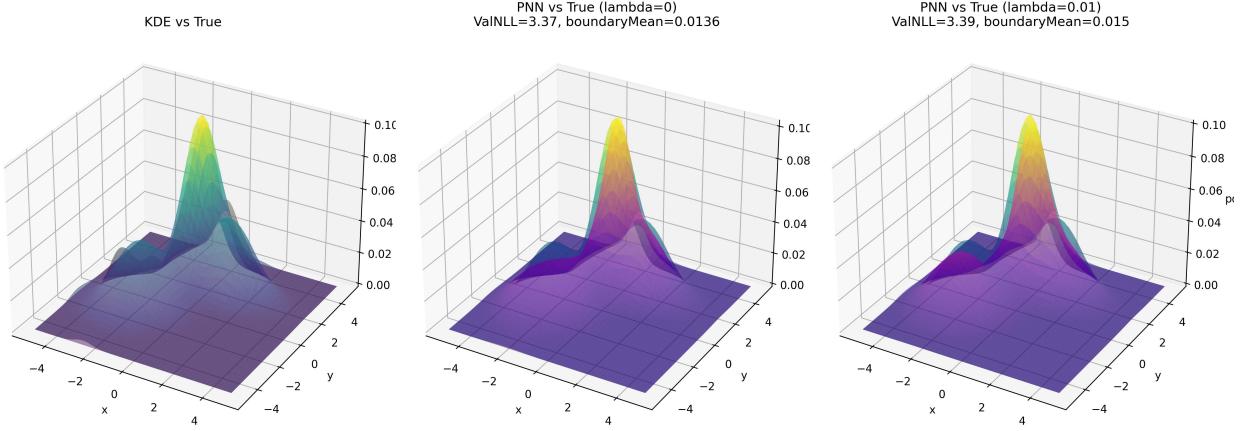


Figure 15: Boundary penalty comparison for Mixture 2, using the configuration selected by validation NLL. KDE vs true (left), PNN with  $\lambda = 0$  (middle), PNN with  $\lambda = 10^{-2}$  (right).

Boundary penalty comparison — mixture 3  
MLP\_30-20\_sigmoid\_outReLU,  $h_1=12.000$

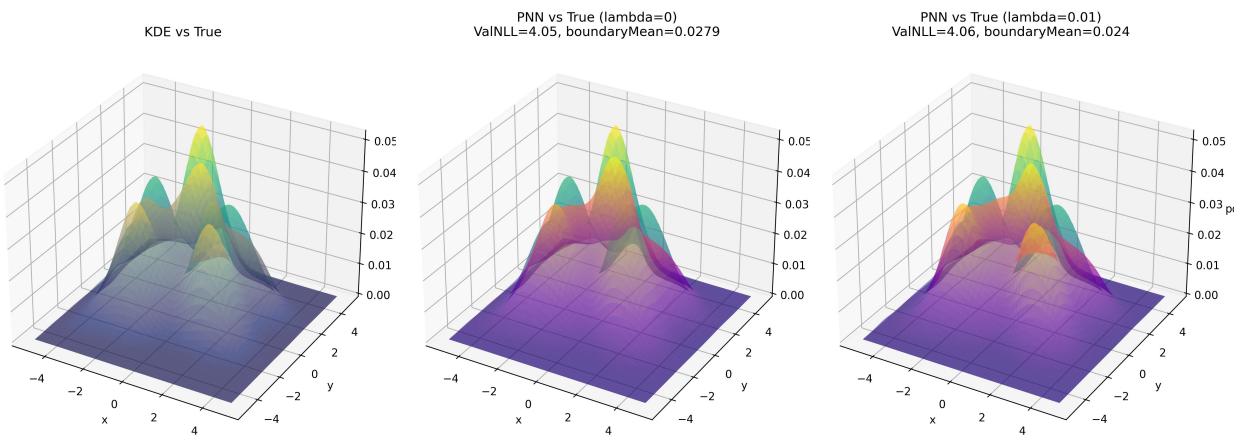


Figure 16: Boundary penalty comparison for Mixture 3, using the configuration selected by validation NLL. KDE vs true (left), PNN with  $\lambda = 0$  (middle), PNN with  $\lambda = 10^{-2}$  (right).

## 7 Conclusions

The experiments compare a memory-based non-parametric estimator (PW/KDE) and a learned surrogate (PNN) trained on Parzen-derived targets. The PNN can be viewed as a *compressed* model that, with enough capacity, can approximate complex multi-modal densities on a chosen compact domain.

This viewpoint is supported by the Universal Approximation Theorem: for a compact set  $X \subset \mathbb{R}^d$  and any continuous target function  $f \in C(X)$ , a single-hidden-layer MLP with sigmoid activations can approximate  $f$  uniformly to arbitrary precision. In particular, for any  $\varepsilon > 0$  there exists a network  $f_\theta$  such that

$$\sup_{x \in X} |f_\theta(x) - f(x)| < \varepsilon. \quad (23)$$

Therefore, on a compact evaluation domain  $D$ , sufficiently wide sigmoid MLPs provide a principled function class for approximating smooth density surfaces (after enforcing non-negativity and normalization).

The theoretical justification relies on approximating continuous functions on compact domains. In this setting the target density  $p$  is a Gaussian mixture: it is continuous (indeed  $C^\infty$ ) and integrates to 1. Restricted to a compact evaluation domain, it falls in the class of non-pathological (“non-paltry”) functions covered by the theorem. Moreover, the Parzen/KDE targets used to train the PNN are themselves smooth mixtures of Gaussians (a finite sum of Gaussian kernels centered at samples), which makes the regression target particularly well-suited to approximation by smooth MLPs. After enforcing non-negativity (e.g., via log-density parameterization) and applying finite-domain normalization, the learned surrogate can be interpreted as an approximate pdf on  $D$ .

Compared to PW/KDE, PNNs shift computation from test time to training time: once trained, inference is efficient and does not grow with the number of samples. Overall performance depends on bandwidth choice, architecture capacity, and regularization (boundary penalty and uniform supervision).

From an engineering standpoint, this suggests a clear computational trade-off: PW/KDE is the natural choice for rapid exploratory analysis and one-off estimation, while PNN is preferable in production or real-time inference scenarios where many density evaluations are required and the one-time training cost  $\mathcal{O}(n^2 + nEW)$  can be amortized. Relative to SVM/SVR-style kernel-machine approaches for density estimation, which typically have super-quadratic (and in the worst case cubic) training-time scaling in  $n$ , this cost profile provides a broader computational perspective.

### Summary of validated findings.

- **Bandwidth selection follows a bias–variance trade-off:** PNN validation NLL is minimized at an intermediate  $h_1$  for all mixtures in the sweep (best at  $h_1 = 7$  for Mixtures 1–2 and  $h_1 = 12$  for Mixture 3).
- **Capacity matters for oracle accuracy:** the lowest oracle grid MSE observed in the saved runs is achieved by the deeper MLP [30,20] across all mixtures.
- **Output constraints interact with target sharpness:** Mixture 2 prefers scaled-sigmoid as best-by-NLL, while Mixture 3 prefers ReLU, consistent with an implicit cap vs unbounded peak representation.
- **Training dynamics degrade at small bandwidth:**  $h_1 = 2$  slows convergence and increases late-stage EvalMSE variability relative to  $h_1 = 16$ .
- **KDE undersmoothing can break likelihood on complex mixtures:** Mixture 3 exhibits a much larger NLL at  $h_1 = 2$  than Mixture 1, consistent with near-zero density regions between modes.
- **LOOCV target generation is empirically near-quadratic in  $n$ :** the measured log–log slope is  $\approx 1.87$ , supporting the  $\mathcal{O}(n^2)$  analysis.
- **Boundary penalty is not automatically helpful:** in the saved artifact it slightly increases validation NLL, and its effect on boundary mass depends strongly on the baseline boundary leakage.
- **Uniform supervision reduces extrapolation artifacts:** adding interior uniform targets slightly improves held-out NLL and reduces boundary-region density in Mixture 3 (Fig. 13), consistent with the intended stabilization far from samples.
- **PW/KDE improves with more data and has an interior bandwidth optimum:** the PW-only  $(n, h_1)$  sweep shows that best attainable grid MSE decreases as  $n$  increases (best-MSE ratios  $\approx 0.33\text{--}0.55$  from low to high  $n$ ) and that both MSE and NLL are minimized at intermediate  $h_1$ .
- **PNN can match or outperform PW/KDE on the same selection metric:** best-by-NLL PNN is within 0.01 of best KDE on all mixtures and improves over KDE on Mixtures 2–3 in the saved sweep.
- **Model selection follows an MDL-style principle:** when validation NLL is similar across architectures, the lower-capacity model is preferred to reduce complexity and limit spurious heavy tails under finite-domain normalization.

## References

- [1] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.