Getting a CLUE: A Method for Explaining Uncertainty Estimates

Javier Antorán
University of Cambridge
ja666@cam.ac.uk

Umang Bhatt
University of Cambridge
usb20@cam.ac.uk

Tameem Adel
University of Cambridge
tah47@cam.ac.uk

Adrian Weller
University of Cambridge
The Alan Turing Institute
aw665@cam.ac.uk

José Miguel Hernández-Lobato
University of Cambridge
Microsoft Research
The Alan Turing Institute
jmh233@cam.ac.uk

Abstract

Both uncertainty estimation and interpretability are important factors for trustworthy machine learning systems. However, there is little work at the intersection of these two areas. We address this gap by proposing a novel method for interpreting uncertainty estimates from differentiable probabilistic models, like Bayesian Neural Networks (BNNs). Our method, Counterfactual Latent Uncertainty Explanations (CLUE), indicates how to change an input, while keeping it on the data manifold, such that a BNN becomes more confident about the input's prediction. We validate CLUE through 1) a novel framework for evaluating counterfactual explanations of uncertainty, 2) a series of ablation experiments, and 3) a user study. Our experiments show that CLUE outperforms baselines and enables practitioners to better understand which input patterns are responsible for predictive uncertainty.

1 Introduction

There is growing interest in probabilistic machine learning models, which aim to provide reliable estimates of uncertainty about their predictions [1]. These estimates are helpful in high-stakes applications such as predicting loan defaults or recidivism, or in work towards autonomous vehicles. Well-calibrated uncertainty can be as important as making accurate predictions, leading to increased robustness of automated decision-making systems and helping prevent systems from behaving erratically for out-of-distribution (OOD) test points. In practice, predictive uncertainty conveys skepticism about a model's output. However, its utility need not stop there: we posit predictive uncertainty could be rendered more useful and actionable if it were expressed in terms of model inputs, answering the question: *"Which input patterns lead my prediction to be uncertain?"*

Understanding which input features are responsible for predictive uncertainty can help practitioners learn in which regions the training data is sparse. For example, when training a loan default predictor, a data scientist (i.e., practitioner) can identify sub-groups (by age, gender, race, etc.) under-represented in the training data. Collecting more data from these groups, and thus further constraining their model's parameters, could lead to accurate predictions for a broader range of clients. In a clinical scenario, a doctor (i.e., domain expert) can use an automated decision-making system to assess whether a patient should receive a treatment. In the case of high uncertainty, the system would suggest that the doctor should not rely on its output. If uncertainty were explained in terms of which features the model finds anomalous, the doctor could appropriately direct their attention.

Preprint. Under review.

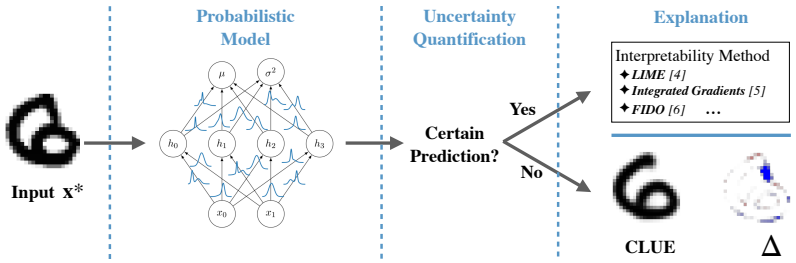


Figure 1: Workflow for automated decision making with transparency. Our probabilistic classifier produces a distribution over outputs. In cases of high uncertainty, CLUE allows us to identify features which are responsible for class ambiguity in the input (denoted by Δ and highlighted in dark blue). Otherwise, we resort to existing feature importance approaches to explain certain decisions.

While explaining predictions from deep models has become a burgeoning field [2; 3], there has been relatively little research on explaining what leads to neural networks' predictive uncertainty. In this work, we introduce Counterfactual Latent Uncertainty Explanations (CLUE), to our knowledge, the first approach to shed light on the subset of input space features that are responsible for uncertainty in probabilistic models. Specifically, we focus on explaining Bayesian Neural Networks (BNNs). We refer to the explanations given by our method as CLUEs. CLUEs answer the question: *"What is the smallest change that could be made to an input, while keeping it in distribution, so that our model becomes more certain in its decision for said input?"* CLUEs can be generated for tabular and image data on both classification and regression tasks.

To maximize transparency in the real-world deployment of a BNN, we envision CLUE complementing existing approaches to model interpretability [4; 5; 6], as shown in Figure 1. When the BNN is confident in its prediction, practitioners can generate an explanation via a feature importance technique. When the BNN has high uncertainty, CLUE can be used. We highlight the following contributions:

- We introduce CLUE, an approach that finds counterfactual explanations of uncertainty in input space, by searching in the latent space of a deep generative model (DGM). We put forth an algorithm for generating CLUEs and show how CLUEs are best displayed.
- We propose a computationally grounded approach for evaluating counterfactual explanations of uncertainty. It leverages a separate conditional DGM as a synthetic data generator, allowing us to quantify how well explanations reflect the true generative process of the data.
- We evaluate CLUE quantitatively through comparison to baseline approaches under the above framework and through ablative analysis. We also perform a user study, showing that CLUEs allow practitioners to predict on which new inputs a BNN will be uncertain.

2 Related Work

2.1 Preliminaries: Uncertainty in BNNs

Given a dataset $\mathcal{D} = \{\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\}_{n=1}^{N}$, a prior on our model's weights $p(\mathbf{w})$, and a likelihood function $p(\mathcal{D}|\mathbf{w}) = \prod_{n=1}^{N} p(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}, \mathbf{w})$, the posterior distribution over the predictor's parameters $p(\mathbf{w}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{w})p(\mathbf{w})$ encodes our uncertainty about what value $\mathbf{w}$ should take. Through marginalization, this parameter uncertainty is translated into predictive uncertainty, yielding reliable error bounds and preventing overfitting:

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w}) p(\mathbf{w}|\mathcal{D})\, d\mathbf{w}. \tag{1}$$

For BNNs, both the posterior over parameters and predictive distribution (1) are intractable. Fortunately, there is a rich literature concerning approximations to these objects [1; 7; 8]. In this work, we use scale adapted Stochastic Gradient Hamiltonian Monte Carlo (SG-HMC) [9]. For regression, we use Gaussian likelihood functions, quantifying uncertainty using their standard deviation, $\sigma(\mathbf{y}|\mathbf{x})$. For classification, we take the entropy $H(\mathbf{y}|\mathbf{x})$ of categorical distributions as uncertainty. Details are given in Appendix B. In the rest of this work, we use $\mathcal{H}$ to refer to any uncertainty metric, be it σ or H. Bayesian methods enable NNs to be used for uncertainty aware tasks, such as OOD detection [10], continual learning [11], active learning [12], and Bayesian optimization [9].

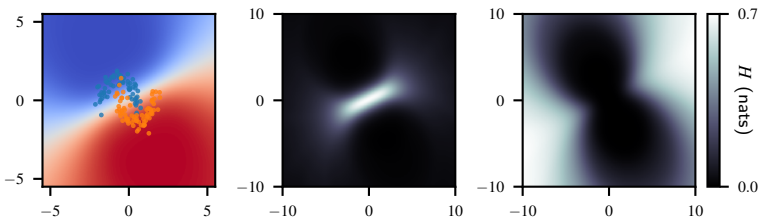


Figure 2: Left: Training points and predictive distribution for variational Bayesian Logistic Regression on the Moons dataset. Center: Aleatoric entropy H_a matches regions of class non-separability. Right: Epistemic entropy H_e grows away from the data. Both uncertainties are detailed in Appendix B.2.

Predictive uncertainty can be separated into two components, as shown in Figure 2. Each conveys different information to practitioners [13]. Irreducible or *aleatoric uncertainty* is caused by inherent noise in the generative process of the data, usually manifesting as class overlap. Model or *epistemic uncertainty* represents our lack of knowledge about $\mathbf{w}$. Stemming from a model being under-specified by the data, epistemic uncertainty arises when we query points off the training manifold or points in sparse regions. Capturing model uncertainty is the main advantage of BNNs over regular NNs.

2.2 Uncertainty Sensitivity Analysis

To the best of our knowledge, the only existing method for interpreting uncertainty estimates is Uncertainty Sensitivity Analysis [14]. This method quantifies the global importance of an input dimension to a chosen metric of uncertainty $\mathcal{H}$ using a sum of linear approximations centered at each test point:

$$I_i = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{n=1}^{|\mathcal{D}_{\text{test}}|} \left| \frac{\partial \mathcal{H}(\mathbf{y}_n | \mathbf{x}_n)}{\partial x_{n,i}} \right| . \tag{2}$$

As discussed by Rudin [15], linear explanations of non-linear models, such as BNNs, can be misleading. Even generalized linear models, which are often considered to be "inherently interpretable," like logistic regression, produce non-linear uncertainty estimates in input space. This can be seen in Figure 2. Furthermore, high-dimensional input spaces limit the actionability of these explanations, as $\nabla_{\mathbf{x}} \mathcal{H}$ will likely not point in the direction of the data manifold. In Figure 3 and Appendix D, we show how this can result in sensitivity analysis generating meaningless explanations.

Our method, CLUE, leverages the latent space of a DGM to avoid working with high-dimensional input spaces and to ensure explanations are in-distribution. CLUE does not rely on crude linear approximations. The counterfactual nature of CLUE guarantees explanations have tangible meaning.

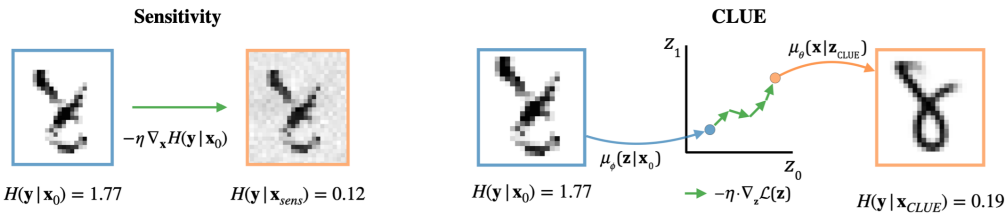


Figure 3: Left: Taking a step in the direction of maximum sensitivity leads to a seemingly noisy input configuration for which H is small. Right: Minimizing CLUE's uncertainty-based objective in terms of a DGM's latent variable $\mathbf{z}$ produces a plausible digit with a corrected lower portion.

2.3 Counterfactual Explanations

The term "counterfactual" captures notions of what would have happened if something had been different. Two meanings have been used by ML subcommunities. (1) Those in causal inference make causal assumptions about interdependencies among variables and use these assumptions to incorporate appropriate consequential adjustments when particular variables are set to new values [16; 17]. (2) In contrast, the interpretability community recently used "counterfactual explanations" to explore how input variables must be modified to change a model's output *without* making explicit

causal assumptions [18]. In this work, we use "counterfactual" in a sense similar to (2): we seek small changes to an input in order to reduce its uncertainty without explicit causal assumptions.

Multiple counterfactual explanations can exist for any given input, as the functions we are interested in explaining are often non-injective [19]. Generally, we are concerned with counterfactual input configurations that are close to the original input $\mathbf{x}_0$ according to some pairwise distance metric $d(\cdot, \cdot)$. Given a desired outcome c different from the original one $\mathbf{y}_0$ produced by predictor p_I, counterfactual explanations $\mathbf{x}_c$ are usually generated by solving an optimization problem that resembles:

$$\mathbf{x}_c = \arg\max_{\mathbf{x}} \left(p_I(\mathbf{y}{=}c|\mathbf{x}) - d(\mathbf{x}, \mathbf{x}_0)\right) \quad \text{s.t.} \quad \mathbf{y}_0{\neq}c. \tag{3}$$

Naively optimizing (3) in high-dimensional input spaces may result in the creation of adversarial examples which are not actionable [20]. Telling a person that they would have been approved for a loan had their age been -10 is of very little use. To right this, recent work introduces DGMs to ensure explanations are in-distribution [6; 21; 22; 23]. We dub these *auxiliary DGMs*. Others define linear constraints on explanations [24; 19]. CLUE avoids the above issues by searching for counterfactuals in the lower-dimensional latent space of an auxiliary DGM. This choice is well suited for uncertainty, as the DGM effectively constrains CLUE's search space to the data manifold. When faced with an OOD input, CLUE returns the nearest in-distribution analog, as shown in Figure 3.

3 Proposed Method

Without loss of generality, we use $\mathcal{H}$ to refer to any differentiable estimate of uncertainty (σ or H). We introduce an auxiliary latent variable DGM: $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\, d\mathbf{z}$. In the rest of this paper, we will use the decoder from a variational autoencoder (VAE). Its encoder is denoted as $q_\phi(\mathbf{z}|\mathbf{x})$. We write these models' predictive means as $\mathbb{E}_{p_\theta(\mathbf{x}|\mathbf{z})}[\mathbf{x}]{=}\mu_\theta(\mathbf{x}|\mathbf{z})$ and $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\mathbf{z}]{=}\mu_\phi(\mathbf{z}|\mathbf{x})$ respectively.

CLUE aims to find points in latent space which generate inputs similar to an original observation $\mathbf{x}_0$ but are assigned low uncertainty. This is achieved by minimizing (4). CLUEs are then decoded as (5).

$$\mathcal{L}(\mathbf{z}) = \mathcal{H}(\mathbf{y}|\mu_\theta(\mathbf{x}|\mathbf{z})) + d(\mu_\theta(\mathbf{x}|\mathbf{z}), \mathbf{x}_0), \tag{4}$$

$$\mathbf{x}_{\text{CLUE}} = \mu_\theta(\mathbf{x}|\mathbf{z}_{\text{CLUE}}) \;\text{ where }\; \mathbf{z}_{\text{CLUE}} = \arg\min_{\mathbf{z}} \mathcal{L}(\mathbf{z}). \tag{5}$$

The pairwise distance metric takes the form $d(\mathbf{x}, \mathbf{x}_0) = \lambda_x d_x(\mathbf{x}, \mathbf{x}_0) + \lambda_y d_y(f(\mathbf{x}), f(\mathbf{x}_0))$ such that we can enforce similarity between uncertain points and CLUEs in both input and prediction space. The hyperparameters (λ_x, λ_y) control the trade-off between producing low uncertainty CLUEs and CLUEs which are close to the original inputs. In this work, we take $d_x(\mathbf{x}, \mathbf{x}_0) = \|\mathbf{x} - \mathbf{x}_0\|_1$ to encourage sparse explanations. For regression, $d_y(f(\mathbf{x}), f(\mathbf{x}_0))$ is mean squared error. For classification, we use cross-entropy. Note that the best choice for $d(\cdot, \cdot)$ will be task-specific.

The CLUE algorithm and a diagram of our procedure are provided in Algorithm 1 and Figure 4 respectively. CLUE can be applied to batches of inputs simultaneously, allowing us to leverage GPU-accelerated matrix computation. The hyperparameter λ_x is selected for each dataset and type of uncertainty by cross validation. We set λ_y to 0 for our main experiments but explore different values

Algorithm 1: CLUE

Inputs: original datapoint $\mathbf{x}_0$, distance function $d(\cdot, \cdot)$, BNN uncertainty estimator $\mathcal{H}$, DGM decoder $\mu_\theta(\cdot)$, DGM encoder $\mu_\phi(\cdot)$

Set initial value of $\mathbf{z} = \mu_\phi(\mathbf{z}|\mathbf{x}_0)$;
while *loss* $\mathcal{L}$ *is not converged* **do**
  Decode: $\mathbf{x} = \mu_\theta(\mathbf{x}|\mathbf{z})$;
  Use BNN to obtain $\mathcal{H}(\mathbf{y}|\mathbf{x})$;
  $\mathcal{L} = \mathcal{H}(\mathbf{y}|\mathbf{x}) + d(\mathbf{x}, \mathbf{x}_0)$;
  Update $\mathbf{z}$ with $\nabla_{\mathbf{z}}\, \mathcal{L}$;
end
Decode explanation: $\mathbf{x}_{\text{CLUE}} = \mu_\theta(\mathbf{x}|\mathbf{z})$;

Output: Counterfactual example $\mathbf{x}_{\text{CLUE}}$

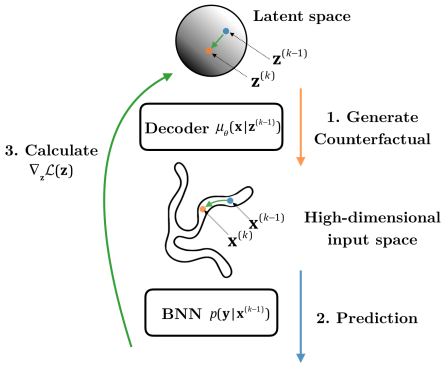
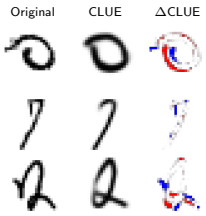


Figure 4: Latent codes are decoded into inputs for which a BNN generates uncertainty estimates; its gradients are backpropagated to latent space.

in Appendix H.1. We minimize (4) with Adam. To facilitate optimization, the initial value of $\mathbf{z}$ is chosen to be $\mathbf{z}_0 = \mu_\phi(\mathbf{z}|\mathbf{x}_0)$. Optimization runs for a minimum of three iterations and a maximum of 35 iterations, with a learning rate of 0.1. If the decrease in $\mathcal{L}(\mathbf{z})$ is smaller than $\mathcal{L}(\mathbf{z}_0)/100$ for three consecutive iterations, we apply early stopping. Our implementation is detailed in full in Appendix B.

As noted by Wachter et al. [18], individual counterfactuals may not shed light on all important features. Fortunately, we can exploit the non-convexity of CLUE's objective to address this. We initialize CLUE with $\mathbf{z}_0 = \mu_\phi(\mathbf{z}|\mathbf{x}_0) + \epsilon$, where $\epsilon = \mathcal{N}(\mathbf{z}; \mathbf{0}, \sigma_0 \mathbf{I})$, and perform Algorithm 1 multiple times to obtain different CLUEs. We find $\sigma_0 = 0.15$ to give a good trade-off between optimization speed and CLUE diversity. Appendix C shows examples of different CLUEs obtained for the same inputs.

We want to ensure noise from auxiliary DGM reconstruction does not affect CLUE visualization. For tabular data, we use the change in percentile of each input feature with respect to the training distribution as a measure of importance. We only highlight continuous variables for which CLUEs are separated by 15 percentile points or more from their original inputs. All changes to discrete variables are highlighted. For images, we report changes in pixel values by applying a sign-preserving quadratic function to the difference between CLUEs and original samples: $\Delta\text{CLUE} = |\Delta\mathbf{x}| \cdot \Delta\mathbf{x}$ with $\Delta\mathbf{x} = \mathbf{x}_{\text{CLUE}} - \mathbf{x}_0$. This is showcased in Figure 5 and in Appendix G.



(a) MNIST

	Original
LSAT	41.0
UGPA	3.7
Race	Asian
Sex	Female

	CLUE
LSAT	41.0
UGPA	3.0
Race	White
Sex	Female

(b) LSAT

Figure 5: Example image and tabular CLUEs.

4 A Framework for Evaluating Counterfactual Explanations of Uncertainty

Evaluating explanations quantitatively (without resorting to expensive user studies) is a hard task [25; 26]. We put forth a computational framework to evaluate counterfactual explanations of uncertainty. In the spirit of [27], we desire counterfactuals that are *1) informative*: they should highlight features which affect our BNN's uncertainty, and *2) relevant*: they should lie close to the original inputs. Counterfactuals must also represent plausible parameter settings, lying close to the data manifold. Recall, from Figure 3, that low uncertainty inputs can be constructed by applying adversarial perturbations to high uncertainty ones. We make our evaluation robust to this by introducing an additional DGM to act as a "ground truth" data generating process (g.t. DGM). Specifically, we use a variational autoencoder with arbitrary conditioning [28] (g.t. VAEAC). It jointly models inputs and targets $p_{\text{gt}}(\mathbf{x}, \mathbf{y})$, as well as the conditional distribution over targets given inputs, $p_{\text{gt}}(\mathbf{y}|\mathbf{x})$. This allows us to evaluate if counterfactuals address the true sources of uncertainty in the data, as opposed to exploiting adversarial vulnerabilities. The evaluation procedure, shown in Figure 6, is as follows:

1. Train a g.t. VAEAC on a real dataset to obtain $p_{\text{gt}}(\mathbf{x}, \mathbf{y})$ as well as conditionals $p_{\text{gt}}(\mathbf{y}|\mathbf{x})$.
2. Sample artificial data $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \sim p_{\text{gt}}(\mathbf{x}, \mathbf{y})$. Use them to train a BNN and an auxiliary DGM.
3. Sample more artificial data. Generate counterfactual explanations $\bar{\mathbf{x}}_c$ for uncertain samples.
4. Use the g.t. VAEAC to obtain the conditional distribution over targets given counterfactual inputs $p_{\text{gt}}(\mathbf{y}|\bar{\mathbf{x}}_c)$ and evaluate if counterfactuals are on-manifold through $\log p_{\text{gt}}(\bar{\mathbf{x}}_c)$.

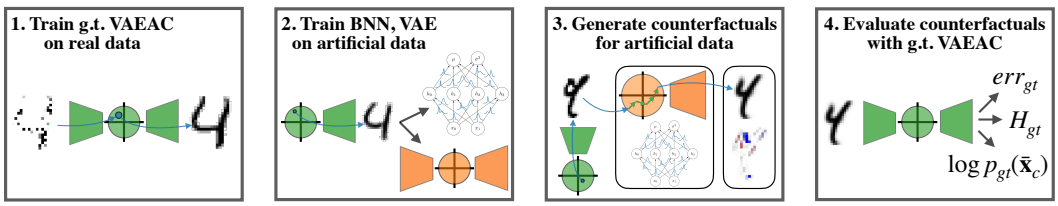


Figure 6: Pipeline for computational evaluation of counterfactual explanations of uncertainty. The VAEAC which we treat as a data generating process is colored in green. Colored in orange is the auxiliary DGM used by the approach being evaluated. For approaches that do not use an auxiliary DGM, like Uncertainty Sensitivity Analysis, the orange element will not be present.

Given an uncertain artificially generated test point $\bar{\mathbf{x}}_0 \sim p_{\text{gt}}$ and its corresponding counterfactual explanation $\bar{\mathbf{x}}_c$, we quantify *informativeness* as the amount of uncertainty that has been explained away. The variance (or entropy) of $p_{\text{gt}}(\mathbf{y}|\mathbf{x})$ reflects the ground truth aleatoric uncertainty associated with $\mathbf{x}$. Hence, for aleatoric uncertainty, we quantify *informativeness* as $\Delta\mathcal{H}_{\text{gt}} = \mathbb{E}_{p_{\text{gt}}}[\mathcal{H}_{\text{gt}}(\mathbf{y}|\bar{\mathbf{x}}_0) - \mathcal{H}_{\text{gt}}(\mathbf{y}|\bar{\mathbf{x}}_c)]$. Epistemic uncertainty only depends on our BNN. It cannot be directly computed from $p_{\text{gt}}(\mathbf{y}|\mathbf{x})$. However, its reduction can be measured implicitly through the reduction in the BNN's prediction error with respect to the labels outputted by the g.t. VAEAC: $\Delta err_{\text{gt}} = \mathbb{E}_{p_{gt}}[err_{\text{gt}}(\bar{\mathbf{x}}_0) - err_{\text{gt}}(\bar{\mathbf{x}}_c)]$. Here $err_{\text{gt}}(\mathbf{x}) = d_y(p(y|\mathbf{x}), \arg\max_y p_{\text{gt}}(y|\mathbf{x}))$. Approaches that exploit adversarial weaknesses in the BNN will not transfer to the g.t. VAEAC, failing to reduce uncertainty or error. We assess the *relevance* of counterfactuals through their likelihood under the g.t. VAEAC $\log p_{gt}(\bar{\mathbf{x}}_c)$ and through their ℓ_1 distance to the original inputs $\|\Delta\bar{\mathbf{x}}\|_1 = \|\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_c\|_1$. Further discussion is included in Appendix I.

5 Experiments

We validate CLUE on 5 datasets: LSAT academic performance regression [29], UCI Wine quality regression, UCI Credit classification [30], a 7 feature variant of COMPAS recidivism classification [31], and MNIST image classification [32]. For each, we select roughly the 20% most uncertain test points as those for which we reject our BNNs' decisions. We only generate CLUEs for "rejected" points. Rejection thresholds, model architectures, and hyperparameter settings are in Appendix B.

As a baseline, we introduce a localized version of Uncertainty Sensitivity Analysis. It produces counterfactuals by taking a single step in the direction of the gradient of an input's uncertainty estimates $\mathbf{x}_c = \mathbf{x}_0 - \eta\nabla_{\mathbf{x}}\mathcal{H}(\mathbf{y}|\mathbf{x}_0)$. Averaging $|\mathbf{x}_0 - \mathbf{x}_c|$ across a test set, we recover (2). As a second baseline, we adapt FIDO [6], a counterfactual feature importance method, to explain uncertainty. We dub it U-FIDO. This method places a binary mask $\mathbf{b}$ over the set of input variables $\mathbf{x}_U$. The mask is modeled by a product of Bernoulli random variables: $p_{\boldsymbol{\rho}}(\mathbf{b}) = \prod_{u\in U} \text{Bern}(b_u; \rho_u)$. The set of masked inputs $\mathbf{x}_B$ is substituted by its expectation under an auxiliary conditional generative model $p(\mathbf{x}_B|\mathbf{x}_{U\setminus B})$, fixed to be a VAEAC. U-FIDO finds the masking parameters $\boldsymbol{\rho}$ which minimize (6):

$$\mathcal{L}(\boldsymbol{\rho}) = \mathbb{E}_{p_{\boldsymbol{\rho}}(\mathbf{b})}[\mathcal{H}(\mathbf{y}|\mathbf{x}_c(\mathbf{b})) + \lambda_b\|\mathbf{b}\|_1], \tag{6}$$

$$\mathbf{x}_c(\mathbf{b}) = \mathbf{b}\odot\mathbf{x}_0 + (1-\mathbf{b})\odot\mathbb{E}_{p(\mathbf{x}_B|\mathbf{x}_{U\setminus B})}[\mathbf{x}_B]. \tag{7}$$

Counterfactuals are generated by (7), where $\odot$ is the Hadamard product. We also compare and contrast CLUE with existing non-counterfactual feature importance methods [4; 33] in Appendix F.

5.1 Computational Evaluation

We compare CLUE, Localized Sensitivity, and U-FIDO using the evaluation framework put forth in Section 4. We would like counterfactuals to explain away as much uncertainty as possible while staying as close the to original inputs as possible. We manage this *informativeness* (large $\Delta\mathcal{H}_{\text{gt}}$) to *relevance* (small $\|\Delta\bar{\mathbf{x}}\|_1$) trade-off with the hyperparameters η, λ_x, and λ_b for Local Sensitivity, CLUE, and U-FIDO respectively. We perform a logarithmic grid search over hyperparameters and plot Pareto-like curves. Our two metrics of interest take minimum values of 0 but their maximum is dataset and method dependent. For Sensitivity, $\|\Delta\bar{\mathbf{x}}\|_1$ grows linearly with η. For CLUE and U-FIDO, these metrics saturate for large and small values of λ_x (or λ_b). As a result, the values obtained by these methods do not overlap. As shown in Figure 7, CLUE is able to explain away more uncertainty ($\Delta\mathcal{H}_{\text{gt}}$) than U-FIDO, and U-FIDO always obtains smaller values of $\|\Delta\bar{\mathbf{x}}\|_1$ than CLUE.

To construct a single performance metric, we scale all measurements by the maximum values obtained between U-FIDO or CLUE, e.g. $(\sqrt{2}\cdot\max(\Delta\mathcal{H}_{\text{gt U-FIDO}}, \Delta\mathcal{H}_{\text{gt CLUE}}))^{-1}$, linearly mapping them to $[0, 1/\sqrt{2}]$. We then negate $\Delta\mathcal{H}_{\text{gt}}$, making its optimum value 0. We consider each method's best performing hyperparameter configuration, as determined by its curve's point nearest the origin, or *knee-point*. The euclidean distance from each method's knee-point to the origin acts as a metric of relative performance. The best value is 0 and the worst is 1. Knee-point distances, computed across three runs, are shown for both uncertainty types in Table 1.

Local Sensitivity performs poorly on all datasets except COMPAS. We attribute this to only two features being necessary to predict targets [34]. U-FIDO's input space masking mechanism allows for counterfactuals that leave features unchanged. It performs well in low dimensional problems but leads to high variance as dimensionality grows. CLUE performs best on higher dimensional datasets.

Latent space optimization makes CLUE more robust to input space complexity. In Appendix H.2, we replace input space similarity $\|\Delta\bar{\mathbf{x}}\|_1$ with proximity to the data manifold $\log p_{gt}(\bar{\mathbf{x}}_c)$. Therein, CLUE produces the most in-distribution counterfactuals, performing best in 8 out of 10 experiments.

Table 1: Relative performance measure obtained by all methods on all datasets under consideration. Lower is better. The dimensionality of each dataset is listed next to their names. *e* and *a* indicate results for epistemic (Δerr_{gt}) and aleatoric ($\Delta\mathcal{H}_{\text{gt}}$) uncertainty respectively.

Method	LSAT (4)		COMPAS (7)		Wine (11)		Credit (23)		MNIST (784)	
	e	a	e	a	e	a	e	a	e	a
Sensitivity	0.70	0.67	**0.71**	**0.13**	0.69	0.03	0.63	0.50	0.66	0.68
CLUE	0.52	0.64	**0.71**	0.18	**0.01**	0.14	0.52	**0.29**	**0.26**	**0.27**
U-FIDO	**0.36**	**0.51**	**0.71**	0.31	0.22	**0.02**	**0.45**	0.63	0.38	0.50

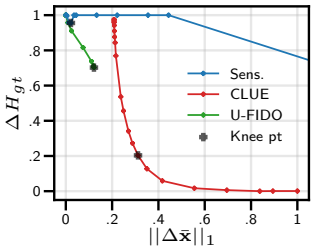


Figure 7: MNIST knee-points.

5.2 User Study

CLUE's promising results in computational evaluation do not substitute for human-based evaluation [25]. We want to assess if CLUEs help machine learning practitioners identify sources of uncertainty more than simple linear approximations (Local Sensitivity) or human intuition. To do this, we propose a forward-simulation task. We show practitioners one datapoint below our "rejection" threshold and one datapoint above. The former is labeled as certain and the latter as uncertain; we refer to these as *context points*. The certain *context point* serves as a counterfactual explanation of the uncertain *context point's* uncertainty. Using *context points* for reference, practitioners are asked to predict whether a new test point will be above or below our threshold (i.e., will our BNN's uncertainty be high or low for the new point). Our survey compares the utility of certain *context points* generated by CLUE relative to those generated by baseline approaches. An example question is shown in Figure 9.

In our survey, we compare four different methods, varying how we select certain *context points*. We either 1) select a certain point at random from the test set as a control, generate a counterfactual certain point with 2) Local Sensitivity or with 3) CLUE, or 4) display a human selected certain point (*Human CLUE*). To generate a *Human CLUE*, we ask participants (who will not take the main survey) to pair uncertain *context points* with similar certain points. We select the points used in our main survey with a pilot procedure similar to Grgic-Hlaca et al. [35]. This procedure, shown in Figure 8, prevents us from injecting biases into point selection and ensures *context points* are relevant to test points. In our procedure, a participant is shown a pool of randomly selected certain and uncertain points. We ask this participant to select points from this pool: these will be test points. We then ask the participant to map each selected test point to a similar uncertain point without replacement. In this way, we obtain uncertain *context points* that are relevant to test points.

We use the LSAT and COMPAS datasets in our user study. We have ten different participants take each variant of the main survey: our participants are students who have taken at least one machine learning course. The main survey consists of 18 questions, 9 from each dataset. The average accuracy of a participant by variant is: CLUE (82.22%), *Human CLUE* (62.22%), Random (61.67%), and Local Sensitivity (52.78%). To measure statistical significance, we treat each participant-question pair as an observation; we have 180 observations per variant. Using the Nemenyi test [36], we determine that the differences in average rank between CLUE and other methods are significant. The critical distance (CD) for a statistically significant difference between average ranks at α=0.05 is 0.35. The average rank of CLUE is 2.15, while *Human CLUE*, Random, and Local Sensitivity obtain 2.55, 2.56, and 2.73 respectively: our CD is 0.4. Additional analysis is included in Appendix H.3.

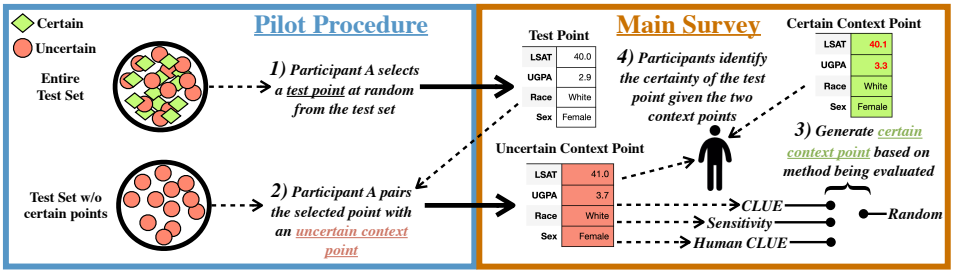


Figure 8: Experimental workflow for our tabular data user study.

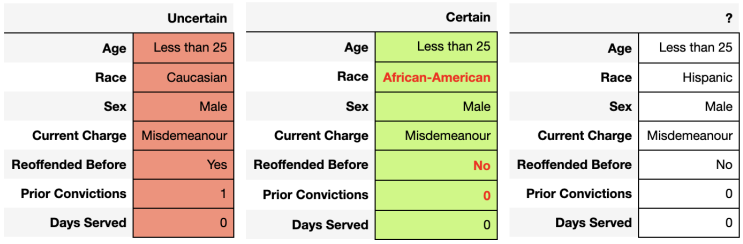


Figure 9: Example question shown to main survey participants for the COMPAS dataset: *Given the uncertain example on the left and the certain example in the middle, will the model be certain on the test example on the right?* The red text highlights the features that differ between *context points*.

We find that linear explanations (Local Sensitivity) of a non-linear function (BNN) mislead practitioners and perform worse than random. While *Human CLUE* uses observed points as explanations, CLUE generates explanations from a VAE. We conjecture that this results in CLUEs representing more typical feature configurations. This might make CLUEs relevant to a wider range of test points. In our tabular data user study, we only show one pair of *context points* per test point. We find that otherwise the survey is difficult for practitioners to follow, due to non-expertise in college admissions or criminal justice. Using MNIST, we run a smaller scale study, wherein we show larger sets of *context points* to practitioners. Results are in Appendix J.2: again, CLUE outperforms baselines.

5.3 Analysis of CLUE's Auxiliary Deep Generative Model

We study CLUE's reliance on its auxiliary DGM. Further ablative analysis is found in Appendix H.

Initialization Strategy: We compare Algorithm 1's encoder-based initialization $\mathbf{z}_0 = \mu_\phi(\mathbf{z}|\mathbf{x}_0)$ with $\mathbf{z}_0 = \mathbf{0}$. As shown in Figure 10, for high dimensional datasets, like MNIST, initializing $\mathbf{z}$ with the encoder's mean leads to CLUEs that require smaller changes in input space to explain away similar amounts of uncertainty (i.e., more *relevant*). In Appendix H.1, similar behavior is observed for Credit, our second highest dimensional dataset. On other datasets, both approaches yield indistinguishable results. This shows CLUEs can be generated with differentiable DGMs that lack an encoding mechanism, such as GANs. These could prove useful when dealing with more complex data.

Capacity of CLUE's DGM: Figure 10 shows how auto-encoding uncertain MNIST samples with low-capacity VAEs significantly reduces these points' predictive entropy. CLUEs generated with these VAEs highlight features that the VAEs are unable to reproduce but are not reflective of our BNN's uncertainty. This results in large values of $\|\Delta\mathbf{x}\|_1$; although counterfactual examples are indeed more certain than the original samples, they contain unnecessary changes. As our auxiliary DGMs' capacity increases, the amount of uncertainty preserved when auto-encoding inputs increases as well. $\|\Delta\mathbf{x}\|_1$ decreases while the predictive entropy of our CLUEs stays the same. More expressive DGMs allow for generating sparser CLUEs. Fortunately, even in scenarios where our predictor's training dataset is limited, we can train powerful DGMs by leveraging unlabeled data.

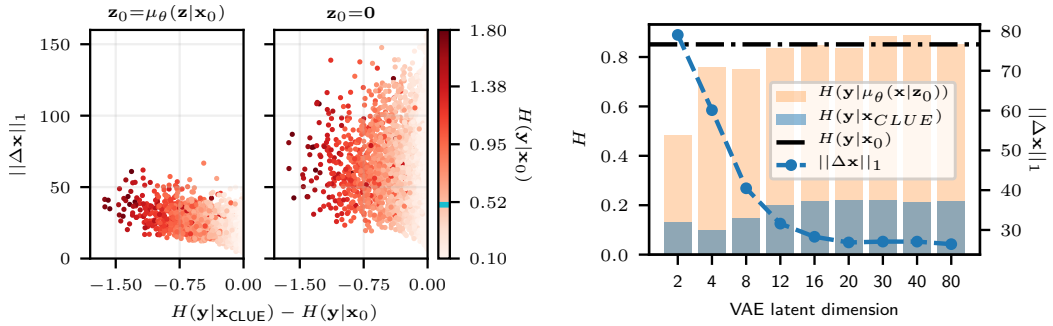


Figure 10: Left: CLUEs are similarly *informative* under encoder-based and encoder-free initializations. The colorbar indicates the original samples' uncertainty. Its horizontal blue line denotes our rejection threshold. Right: Auxiliary DGMs with more capacity result in more *relevant* CLUEs.

6 Conclusion

We propose Counterfactual Latent Uncertainty Explanations (CLUE), a method that reveals which input features can be changed to reduce the uncertainty of a probabilistic model. CLUE leverages an auxiliary DGM to ensure counterfactuals lie on the data manifold. For certain inputs, existing feature importance methods may suffice; however, for uncertain inputs, CLUE excels at providing useful explanations of which features are responsible for predictive uncertainty. We propose a computational evaluation framework for counterfactual explanations of uncertainty, and find that CLUE outperforms baselines quantitatively. We also qualitatively verify CLUE's utility to practitioners through a user study. CLUE's latent space optimization mechanism allows it to cope well with high dimensional data. Future work can leverage recent advances in scalable BNNs [37] and generative modeling [38] to explore using CLUE for more complex data, such as natural images and natural language.

Broader Impact

As machine learning models are deployed in high-stakes scenarios, there has been a call for algorithmic transparency into models' behavior. While existing transparency techniques have focused on which features are important to prediction, our approach tackles scenarios where a reliable prediction can not be made. This is of special interest to those working with probabilistic models that can capture both noise and model uncertainty. This paper specifically discusses how to put a model's predictive uncertainty in terms of input features.

We view predictive uncertainty as a crucial form of transparency into model behavior. Our technique has implications on machine learning practitioners who can understand where in input space their model's failures lie: which features or feature interactions do we need to solicit to make our system more robust or in an active learning regime? We foresee practitioners using CLUE not only for model debugging but also for discovering global uncertainty trends, and better understanding their datasets.

Acknowledgments and Disclosure of Funding

JA acknowledges support from Microsoft Research through its PhD Scholarship Program. UB acknowledges support from DeepMind and the Leverhulme Trust via the Leverhulme Centre for the Future of Intelligence (CFI), and from the Partnership on AI. AW acknowledges support from the David MacKay Newton research fellowship at Darwin College, The Alan Turing Institute under EPSRC grant EP/N510129/1 & TU/B/000074, and the Leverhulme Trust via CFI.

References

[1] David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Comput.*, 4(3):448–472, May 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.448. URL http://dx.doi.org/10.1162/neco.1992.4.3.448.

[2] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1 – 15, 2018. ISSN 1051-2004. doi: https://doi.org/10.1016/j.dsp.2017.10.011. URL http://www.sciencedirect.com/science/article/pii/S1051200417302385.

[3] Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José M. F. Moura, and Peter Eckersley. Explainable machine learning in deployment. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, FAT* '20, page 648–657, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450369367. doi: 10.1145/3351095.3375624. URL https://doi.org/10.1145/3351095.3375624.

[4] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1135–1144, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778. URL http://doi.acm.org/10.1145/2939672.2939778.

[5] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 3319–3328. JMLR.org, 2017. URL http://dl.acm.org/citation.cfm?id=3305890.3306024.

[6] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1MXz20cYQ.

[7] José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1861–1869. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045316.

[8] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[9] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4134–4142. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6117-bayesian-optimization-with-robust-bayesian-neural-networks.pdf.

[10] Erik Daxberger and José Miguel Hernández-Lobato. Bayesian variational autoencoders for unsupervised out-of-distribution detection, 2019.

[11] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL https://openreview.net/forum?id=BkQqq0gRb.

[12] Stefan Depeweg, Jose-Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1184–1193, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/depeweg18a.html.

[13] Stefan Depeweg. *Modeling Epistemic and Aleatoric Uncertainty with Bayesian Neural Networks and Latent Variables*. PhD thesis, Technical University of Munich, 2019.

[14] Stefan Depeweg, José Miguel Hernández-Lobato, Steffen Udluft, and Thomas A. Runkler. Sensitivity analysis for predictive uncertainty. In *ESANN*, 2017.

[15] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1:206–215, May 2019.

[16] Judea Pearl. The seven tools of causal inference, with reflections on machine learning. 62(3), 2019. ISSN 0001-0782. doi: 10.1145/3241036. URL https://doi.org/10.1145/3241036.

[17] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4066–4076. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6995-counterfactual-fairness.pdf.

[18] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard Journal of Law & Technology*, 31(2), 2018.

[19] Chris Russell. Efficient search for diverse coherent explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, page 20–28, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361255. doi: 10.1145/3287560.3287569. URL https://doi.org/10.1145/3287560.3287569.

[20] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572.

[21] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 592–603. Curran Associates, Inc., 2018.

[22] Shalmali Joshi, Oluwasanmi Koyejo, Been Kim, and Joydeep Ghosh. xGEMs: Generating examplars to explain black-box models. *arXiv preprint arXiv:1806.08867*, 2018.

[23] Rafael Poyiadzi, Kacper Sokol, Raul Santos-Rodriguez, Tijl De Bie, and Peter Flach. FACE: Feasible and actionable counterfactual explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 344–350, 2020.

[24] Shubham Sharma, Jette Henderson, and Joydeep Ghosh. Certifai: A common framework to provide explanations and analyse the fairness and robustness of black-box models. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, AIES '20, page 166–172, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371100. doi: 10.1145/3375627.3375812. URL https://doi.org/10.1145/3375627.3375812.

[25] Finale Doshi-Velez and Been Kim. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv e-prints*, art. arXiv:1702.08608, Feb 2017.

[26] Adrian Weller. Transparency: Motivations and challenges. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 23–40. Springer, 2019.

[27] Umang Bhatt, Adrian Weller, and José MF Moura. Evaluating and aggregating feature-based model explanations. *arXiv preprint arXiv:2005.00631*, 2020.

[28] Oleg Ivanov, Michael Figurnov, and Dmitry Vetrov. Variational autoencoder with arbitrary conditioning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SyxtJh0qYm.

[29] L.F. Wightman, H. Ramsey, and Law School Admission Council. *LSAC national longitudinal bar passage study*. LSAC research report series. Law School Admission Council, 1998. URL https://books.google.co.uk/books?id=WdA7AQAAIAAJ.

[30] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[31] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. there's software used across the country to predict future criminals. and it's biased against blacks. URL https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

[32] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[33] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[34] Julia Dressel and Hany Farid. The accuracy, fairness, and limits of predicting recidivism. *Science advances*, 4(1):eaao5580–eaao5580, Jan 2018. ISSN 2375-2548. doi: 10.1126/sciadv.aao5580. URL https://www.ncbi.nlm.nih.gov/pubmed/29376122.

[35] Nina Grgic-Hlaca, Elissa M Redmiles, Krishna P Gummadi, and Adrian Weller. Human perceptions of fairness in algorithmic decision making: A case study of criminal risk prediction. In *Proceedings of the 2018 World Wide Web Conference*, pages 903–912, 2018.

[36] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[37] Kazuki Osawa, Siddharth Swaroop, Mohammad Emtiyaz E Khan, Anirudh Jain, Runa Eschenhagen, Richard E Turner, and Rio Yokota. Practical deep learning with Bayesian principles. In *Advances in Neural Information Processing Systems*, pages 4289–4301, 2019.

[38] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

[39] Guy W. Cole and Sinead A. Williamson. Avoiding resentment via monotonic fairness. *ArXiv*, abs/1909.01251, 2019.

[40] Muhammad Bilal Zafar, Isabel Valera, Manuel Rodriguez, Krishna Gummadi, and Adrian Weller. From parity to preference-based notions of fairness in classification. In *Advances in Neural Information Processing Systems*, pages 229–239, 2017.

[41] Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization, 2020.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[43] Bin Dai and David Wipf. Diagnosing and enhancing VAE models. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=B1e0X3C9tQ`.

[44] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*, April 2020.

[45] Yoshua Bengio. Estimating or propagating gradients through stochastic neurons. *CoRR*, abs/1305.2982, 2013. URL `http://arxiv.org/abs/1305.2982`.

[46] J. Antoran and A. Miguel. Disentangling and learning robust representations with natural clustering. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 694–699, 2019.

[47] Mihaela Rosca, Balaji Lakshminarayanan, and Shakir Mohamed. Distribution Matching in Variational Inference. *arXiv e-prints*, art. arXiv:1802.06847, Feb 2018.

Appendix

This appendix is formatted as follows.

1. We discuss the **datasets** used in Appendix A.
2. **Implementation details** for our experiments are provided in Appendix B.
3. We provide examples of the **multiplicity of CLUEs** in Appendix C.
4. We discuss the application of **uncertainty sensitivity analysis** in high dimensional spaces in Appendix D.
5. We visualize CLUE's **optimization in the latent space** in Appendix E.
6. We compare CLUE to existing **feature importance** techniques in Appendix F.
7. We provide additional **examples** of CLUEs and U-FIDO counterfactuals in Appendix G.
8. We provide additional **experimental results** in Appendix H.
9. We note additional details of our **computational evaluation** framework for counterfactual explanations of uncertainty in Appendix I.
10. We include more details on the **setup** of our user studies in Appendix J.

A Datasets

We employ 5 datasets in our experiments, 4 tabular and one composed of images. All of them are publicly available. Their details are given in Table 2.

Table 2: Summary of datasets used in our experiments. (*) We use a 7 feature version of COMPAS, however, other versions exist.

Name	Targets	Input Type	N. Inputs	N. Train	N. Test
LSAT	Continuous	Continuous & Categorical	4	17432	4358
COMPAS	Binary	Continuous & Categorical	7*	5554	618
Wine (red)	Continuous	Continuous	11	1438	160
Credit	Binary	Continuous & Categorical	24	27000	3000
MNIST	Categorical	Image (greyscale)	28×28	60000	10000

We use the LSAT loading script from Cole and Williamson [39]'s github page. The raw data can be downloaded from (`https://raw.githubusercontent.com/throwaway20190523/MonotonicFairness/master/data/law_school_cf_test.csv`) and (`https://raw.githubusercontent.com/throwaway20190523/MonotonicFairness/master/data/law_school_cf_train.csv`).

For the COMPAS criminal recidivism prediction dataset we use a modified version of Zafar et al. [40]'s loading and pre-processing script. It can be found at (`https://github.com/mbilalzafar/fair-classification/blob/master/disparate_mistreatment/propublica_compas_data_demo/load_compas_data.py`). We add an additional feature: "days served" which we compute as the difference, measured in days, between the "c_jail_in" and "c_jail_out" variables. The raw data is found at (`https://github.com/propublica/compas-analysis/blob/master/compas-scores-two-years.csv`).

The red wine quality prediction dataset can be obtained from and is described in detail at (`https://archive.ics.uci.edu/ml/datasets/wine+quality`).

The default of credit card clients dataset, which we refer to as "Credit" in this work, can be obtained from and is described in detail at (`https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`). Note that this dataset is different from the also commonly used German credit dataset.

The MNIST handwritten digit image dataset can be obtained from (`http://yann.lecun.com/exdb/mnist/`).

B Implementation Details

B.1 Inference in BNNs

We choose a Monte Carlo (MC) based inference approach for our BNNs due to these not being limited to localized approximations of the posterior. Specifically, we make use of scale adapted SG-HMC [9], an approach to stochastic gradient Hamiltonian Monte Carlo with automatic hyperparameter discovery. This technique estimates the mass matrix and the noise introduced by stochasticity in the gradients using exponentially decaying moving average filters during the chain's burn-in phase. We use a fixed step size of $\epsilon = 0.01$ and batch sizes of 512. We set a diagonal 0 mean Gaussian prior $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mathbf{0}, \sigma_w^2 \cdot I)$ over each layer of weights. We place a per-layer conjugate Gamma hyperprior over σ_w^2 with parameters $\alpha = \beta = 10$. We periodically update σ_w^2 for each layer using Gibbs sampling.

On MNIST, we burn in our chain for 25 epochs, using the first 15 to estimate SG-HMC parameters. We re-sample momentum parameters every 10 steps and perform a Gibbs sweep over the prior variances every 45 steps. We save parameter settings every 2 epochs until a total of 300 sets of weights are stored. This makes for a total of 625 epochs.

For tabular datasets, we perform a burn-in of 400 epochs, using the first 120 to estimate SG-HMC parameters. We save weight configurations every 20 epochs until a total of 100 sets if weights are saved. This makes for a total of 2500 epochs. Momentum is re-sampled every 10 epochs and the prior over weights is re-sampled every 50 epochs. We use a batch size of 512 for all datasets.

B.2 Computing Uncertainty Estimates

In this work, we consider NNs which parametrize two types of distributions over target variables: the categorical for classification problems and the Gaussian for regression. For classification, our networks output a probability vector with elements $f_k(\mathbf{x}, \mathbf{w})$, corresponding to classes $\{c_k\}_{k=1}^K$. The likelihood function is $p(y|\mathbf{x}, \mathbf{w}) = Cat(y; f(\mathbf{x}, \mathbf{w}))$. Given a posterior distribution over weights $p(\mathbf{w}|\mathcal{D})$, we use marginalization (1) to translate uncertainty in $\mathbf{w}$ into uncertainty in predictions. Unfortunately, this operation is intractable for BNNs. We resort to approximating the predictive posterior with M MC samples:

$$\begin{aligned} p(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) &= \mathbb{E}_{p(\mathbf{w}|\mathcal{D})}[p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w})] \\ &\approx \frac{1}{M} \sum_{m=0}^{M} f(\mathbf{x}^*, \mathbf{w}); \quad \mathbf{w} \sim p(\mathbf{w}|\mathcal{D}). \end{aligned}$$

The resulting predictive distribution is categorical. We quantify its uncertainty using entropy:

$$H(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) = \sum_{k=1}^{K} p(y^*{=}c_k|\mathbf{x}^*, \mathcal{D}) \log p(y^*{=}c_k|\mathbf{x}^*, \mathcal{D}).$$

This quantity contains aleatoric and epistemic components (H_a, H_e). The former is estimated as:

$$H_a = \mathbb{E}_{p(\mathbf{w}|\mathcal{D})}[H(y^*|\mathbf{x}^*, \mathbf{w})] \approx \frac{1}{M} \sum_{m}^{M} H(y^*|\mathbf{x}^*, \mathbf{w}); \quad \mathbf{w} \sim p(\mathbf{w}|\mathcal{D}).$$

The epistemic component can be obtained as the difference between the total and aleatoric entropies. This quantity is also known as the mutual information between $\mathbf{y}^*$ and $\mathbf{w}$:

$$H_e = I(\mathbf{y}^*, \mathbf{w}|\mathbf{x}^*, \mathcal{D}) = H(y^*|\mathbf{x}^*, \mathcal{D}) - \mathbb{E}_{p(\mathbf{w}|\mathcal{D})}[H(y^*|\mathbf{x}^*, \mathbf{w})].$$

For regression, we employ heteroscedastic likelihood functions. Their mean and variance are parametrized by our NN: $p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w}) = \mathcal{N}(\mathbf{y}; f_\mu(\mathbf{x}^*, \mathbf{w}), f_{\sigma^2}(\mathbf{x}^*, \mathbf{w}))$. Marginalizing over $\mathbf{w}$ with MC induces a Gaussian mixture distribution over outputs. Its mean is obtained as:

$$\boldsymbol{\mu}_a \approx \frac{1}{M} \sum_{m=0}^{M} f_\mu(\mathbf{x}^*, \mathbf{w}); \quad \mathbf{w} \sim p(\mathbf{w}|\mathcal{D}).$$

There is no closed-form expression for the entropy of this distribution. Instead, we use the variance of the GMM as an uncertainty metric. It also decomposes into aleatoric and epistemic components (σ_a^2, σ_e^2):

$$\sigma^2(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) = \underbrace{\mathbb{E}_{p(\mathbf{w}|\mathcal{D})}[\sigma^2(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w})]}_{\sigma_a^2} + \underbrace{\sigma^2_{p(\mathbf{w}|\mathcal{D})}[\mu(\mathbf{y}^*|\mathbf{x}, \mathbf{w})]}_{\sigma_e^2}.$$

These are also estimated with MC:

$$\sigma^2(\mathbf{y}^*|\mathbf{x}^*, \mathcal{D}) \approx \underbrace{\frac{1}{M}\sum_m^M \mu(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w})^2 - (\frac{1}{M}\sum_m^M \mu(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w}))^2}_{\sigma_e^2} + \underbrace{\frac{1}{M}\sum_m^M \sigma^2(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w})}_{\sigma_a^2};\ \mathbf{w} \sim p(\mathbf{w}|\mathcal{D}).$$

Here, σ_e^2 reflects model uncertainty - our lack of knowledge about $\mathbf{w}$ - while σ_a^2 tells us about the irreducible uncertainty or noise in our training data.

In Figure 11, we show the fit obtained with a BNN with scale adapted SG-HMC on the toy moons dataset. We would like to highlight 2 key differences with respect to the logistic regression example shown in Figure 2. Neural networks are very flexible models. They are capable of perfectly fitting non-linear manifolds, such as moons. In consequence, when these models present *aleatoric uncertainty* it is most often due to the inputs not containing enough information to predict the targets. As little such noise exists in our particular instantiation of moons, our estimates of aleatoric entropy are close to 0. Despite their flexibility, selecting a NN involves adopting some inductive biases [41]. Additionally, unlike logistic regression, the weight space posterior of a BNN is very difficult to characterize. Both of these things are reflected in the BNN predictive posterior's *epistemic uncertainty* only growing in the vertical axis, instead of in all directions.

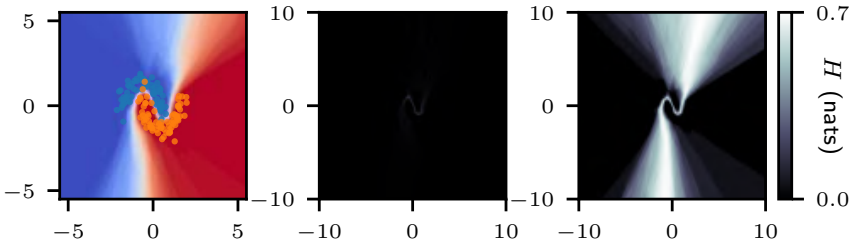


Figure 11: Left: Training points and BNN predictive distribution obtained on the moons dataset with SG-HMC. Center: Aleatoric entropy H_a expressed by the model matches regions of class overlap. Right: Epistemic entropy H_e grows as we move away from the data.

B.3 Architectures and other Network Hyperparameters

For all datasets, our BNNs are fully connected networks with residual connections. Auxiliary VAEs and VAEACs used for tabular data use fully connected encoders and decoders with residual connections and batch normalization at every layer. For MNIST, we employ 6 convolutional bottleneck residual blocks [42] for both encoders and decoders. We use the same architecture for the VAEACs used as ground truth generative models in the computationally grounded evaluation framework put forth in Section 4. Note that the ground truth VAEAC models have slightly larger input spaces due to them modeling inputs and targets jointly. All architectural hyperparameters are provided in Table 3.

In order to improve the artificial sample quality of our "ground truth" VAEACs, we leverage a two-stage VAE configuration [43]. For all datasets, the lower level VAEs use the standard tabular data VAE architecture described above, with 2 hidden layers. We use 300 hidden units for MNIST and 150 for other datasets. Additional details on our use of two-stage VAEs are provided in Appendix I.

We train all generative models with the RAdam optimizer [44] with a learning rate of $1e^{-4}$ for tabular data and $3e^{-4}$ for MNIST. We found RAdam to yield marginally better results than Adam.

We convert categorical inputs to our BNNs into one-hot vectors. When building DGMs, we model continuous inputs with diagonal, unit variance (heteroscedastic) Gaussian distributions. This choice makes these models weigh all input dimensions equally, a desirable trait for explanation generation.

Table 3: Network architecture hyperparameters used in all experiments. Depth refers to number of hidden layers or residual blocks. Latent dimension values marked with a star (*) refer to the second level VAEs for "ground truth" VAEACs.

Dataset	BNN Depth	BNN Width	VAE / VAEAC Depth	VAE Width	VAEAC Width	VAE / VAEAC Latent Dim
LSAT	2	200	3	300	350	4 (*4)
COMPAS	2	200	3	300	350	4 (*4)
Wine	2	200	3	300	350	6 (*6)
Credit	2	200	3	300	350	8 (*8)
MNIST	2	1200	6	-	-	20 (*8)

We place categorical distributions over discrete inputs, expressing them as one-hot vectors. For the LSAT, COMPAS, and Credit datasets, where there are both continuous and discrete features, data likelihood values are obtained as the product of Gaussian likelihoods and categorical likelihoods. During the CLUE optimization procedure, we approximate gradients through one-hot vectors with the softmax function's gradients. This is known as the softmax straight-through estimator [45]. It is biased but works well in practice. For MNIST, we model pixels as the probabilities of a product of Bernoulli distributions. We feed these probabilities directly into our BNNs and DGMs.

We normalize all continuously distributed features such that they have 0 mean and unit variance. This facilitates model training and also ensures that all features are weighed equally under CLUE's pairwise distance metric in (4). For MNIST, this normalization is applied to whole images instead of individual pixels. Categorical variables are not normalized. Changing a categorical variable implies changing two bits in the corresponding one-hot vector. This creates the same ℓ_1 regularization penalty as shifting a continuously distributed variable two standard deviations.

B.4 CLUE Hyperparameters

As mentioned in Section 5, in our experiments we only apply CLUE to points that present uncertainty above a rejection threshold. The rejection thresholds used for each dataset are displayed in Table 4. The same table contains the values of λ_x used in all experiments. In practice we define $\lambda_x^{'} = \lambda_x \cdot d$, where d is the input space dimensionality of a dataset. This makes the strength of CLUE's pairwise input space distance metric agnostic to dimensionality. We choose a significantly larger value of $\lambda_x^{'}$ for MNIST due to there being a large number of pixels that are always black.

Table 4: Values of CLUE's input space similarity weight λ_x and uncertainty rejection thresholds used for all experiments. Next to each dataset's name is the the type of uncertainty quantified: standard deviation (σ) or entropy (H). We report λ_x upscaled by each dataset's input dimensionality d.

Dataset	LSAT (σ)	COMPAS (H)	Wine (σ)	Credit (H)	MNIST (H)
$\lambda_x \cdot d$	1.5	2	2.5	3	25
$\mathcal{H}$ threshold	1	0.2	2	0.5	0.5

C Multiplicity of CLUEs

We exploit the non-convexity of CLUE's objective to generate diverse CLUEs. We initialize CLUE with $\mathbf{z}_0 = \mu_\phi(\mathbf{z}|\mathbf{x}_0) + \epsilon$, where $\epsilon = \mathcal{N}(\mathbf{z}; \mathbf{0}, \sigma_0 \mathbf{I})$, and perform Algorithm 1 multiple times to obtain different CLUEs. We choose $\sigma_0 = 0.15$. In Figure 12, we showcase different CLUEs for the same original MNIST inputs. Different counterfactuals represent digits of different classes. Despite this, all explanations resemble the original datapoints being explained. Being exposed to this multiplicity could potentially inform practitioners about similarities of an original input to multiple classes that lead their model to be uncertain.

Different initializations lead to CLUEs that explain away different amounts of uncertainty. In a few rare cases CLUE fails: the algorithm does not produce a feature configuration which has significantly lower uncertainty than the original input. This is the case for the third CLUE in the bottom 2 rows of Figure 12. We attribute this to a disadvantageous initialization of $\mathbf{z}$.

In Figure 13, we show multiple CLUEs for a single individual from the COMPAS dataset. In this case, uncertainty can be reduced by changing the individual's prior convictions and charge degree, or by changing their sex and age range. Making both sets of changes simultaneously also reduces uncertainty.

Figure 12: We generate 5 possible CLUEs for 11 MNIST digits score above the uncertainty rejection threshold. Below each digit or counterfactual is the predictive entropy it is assigned H and the class of maximum probability c.

Original	
Age	Greater than 45
Race	African-American
Sex	Female
Current Charge	Felony
Reoffended Before	No
Prior Convictions	1
Days Served	0

CLUE	
Age	Greater than 45
Race	African-American
Sex	Female
Current Charge	Misdemeanour
Reoffended Before	No
Prior Convictions	0
Days Served	0

CLUE	
Age	25-45
Race	African-American
Sex	Male
Current Charge	Misdemeanour
Reoffended Before	No
Prior Convictions	0
Days Served	0

CLUE	
Age	25-45
Race	African-American
Sex	Male
Current Charge	Felony
Reoffended Before	No
Prior Convictions	1
Days Served	0

CLUE	
Age	25-45
Race	Asian
Sex	Female
Current Charge	Felony
Reoffended Before	No
Prior Convictions	1
Days Served	0

Figure 13: The leftmost entry is an uncertain COMPAS test sample. To its right are four candidate CLUEs. The first three successfully reduce uncertainty past our rejection threshold, while the rightmost does not.

D Sensitivity Analysis in High Dimensional Spaces

In high-dimensional input spaces, $\nabla_{\mathbf{x}}\mathcal{H}$ will often not point in the direction of the data manifold. This can result in meaningless explanations. In Figure 14, we show an example where a step in the direction of $-\nabla_{\mathbf{x}}\mathcal{H}$ leads to a seemingly noisy input configuration for which the predictive entropy is low. An "adversarial examples for uncertainty" is generated. Aggregating these steps for every point in the test set leads to an uncertainty sensitivity analysis explanation that resembles white noise.

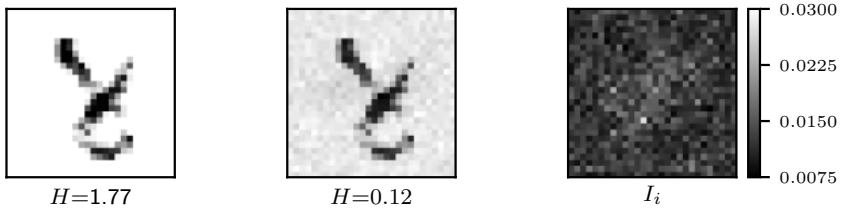


Figure 14: Left: A digit from the MNIST test set with large predictive entropy. Center: The same digit after a step is taken in the direction of $-\nabla_{\mathbf{x}}\mathcal{H}$. Non-zero weight is assigned to pixels that are always zero valued. Right: Uncertainty sensitivity analysis for the entire MNIST test set.

E Visualizing Optimization in Latent Space

Figure 15 shows a 2 dimensional latent space trajectory from $\mathbf{z}_0$ to $\mathbf{z}_{CLUE}$ for a test point from the COMPAS dataset. In practice, we use larger latent spaces to ensure CLUEs are relevant to the original inputs being explained.

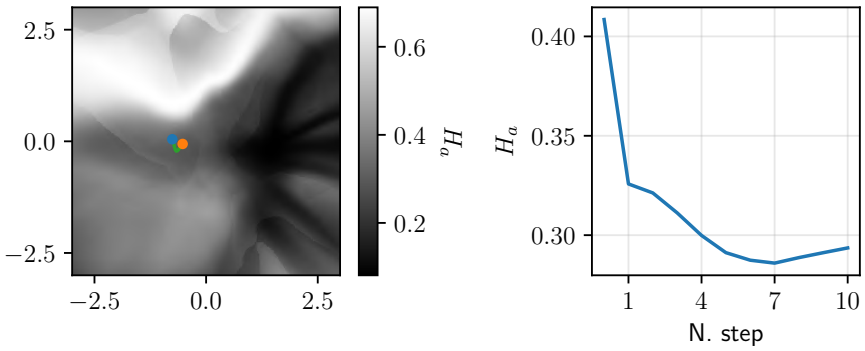


Figure 15: Left: CLUE latent trajectory for a test point from the Credit dataset in a two-dimensional latent space. The blue dot marks the start of the trajectory and the orange one marks the end. Uncertainty levels are displayed in greyscale. Right: Changes in aleatoric entropy for inputs regenerated from latent codes along the trajectory.

F Comparing CLUE to Feature Importance Estimators

Among machine learning practitioners, two of the most popular approaches for determining feature importance from back-box models are LIME and SHAP [3]. LIME locally approximates the back-box model of interest around a specific test point with a surrogate linear model [4]. This surrogate is trained on points sampled from nearby the input of interest. The surrogate model's weights for each class can be interpreted as each feature's contribution towards the prediction of said class. Kernel SHAP extends lime by introducing a kernel such that resulting explanations have desirable properties [33]. For SHAP, a reference input is chosen. It allows importance to be only assigned where the inputs are different from the reference. For MNIST, the reference is an entirely black image. Note that alternative versions of SHAP exist that incorporate information about internal NN dynamics into their explanations. However, they produce very noisy explanations when applied to our BNNs. We conjecture that this high variance might be induced by disagreement among the multiple weight configurations from our BNNs.

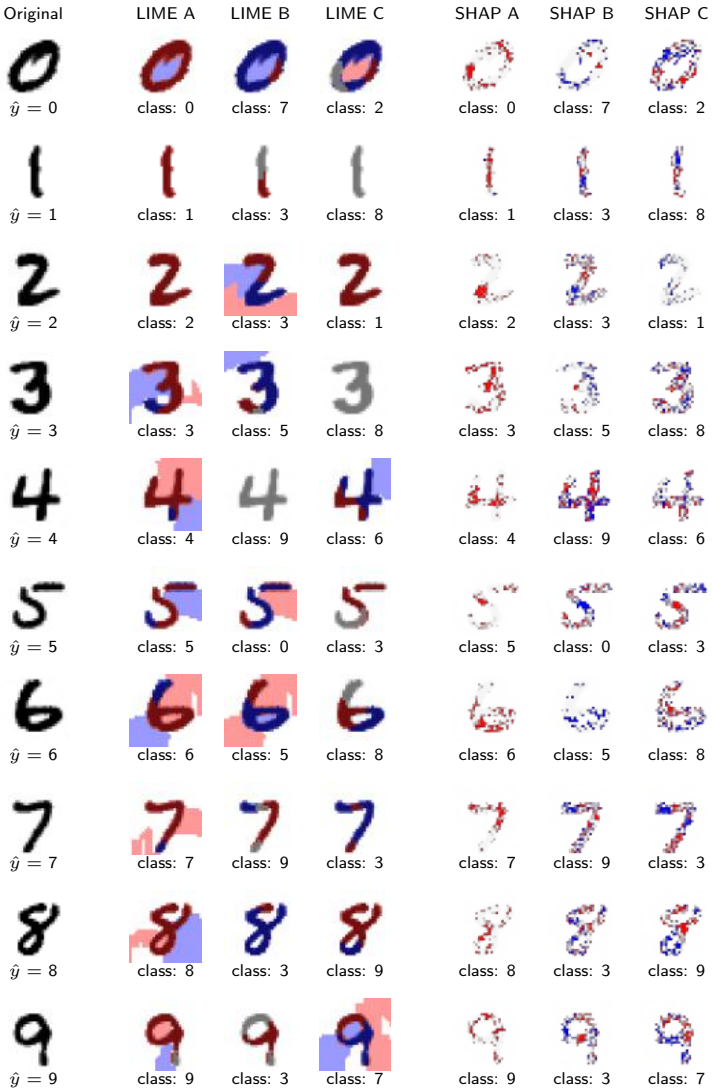


Figure 16: High confidence MNIST test examples together with LIME and SHAP explanations for the top 3 predicted classes. The model being investigated is a BNN with architecture described in Appendix B. The highest probability class is denoted by $\hat{y}$.

Figure 16 shows examples of LIME and Kernel SHAP being applied to a BNN for high confidence MNIST test digits. We use the default LIME hyperparameters for MNIST: the "quickshift" segmenta-

tion algorithm with kernel size 1, maximum distance 5 and a ratio of 0.2. We plot the top 10 segments with weight greater than 0.01. We draw 1000 samples with both methods.

Using the same configuration, we generate LIME and SHAP explanations for some MNIST digits to which our BNN assigns predictive entropy above our rejection threshold. The results are displayed in Figure 17.

Figure 17: Ten MNIST test digits for which our BNN's predictive entropy is above the rejection threshold. A single CLUE example is provided for each one. For each digit, the top scoring class is denoted by $\hat{y}$. LIME and SHAP explanations are provided for the three most likely classes.

A positive CLUE attribution means that the addition of that feature will make our model more certain. A positive feature importance attribution means the presence of that feature serves as evidence towards a predicted class. A negative CLUE attribution means that the the absence of that feature will make the model more certain. A negative feature importance attribution means the absence of that feature would serve as evidence for a particular prediction. While CLUE and feature importance techniques solve similar problems and both provide saliency maps, CLUE highlights regions that need to be added or removed to make the input certain to a predictive model. In some cases, we see that feature importance negative attribution aligns with CLUE negative attribution, suggesting the features which negatively contribute to the model's predicted probability are the features that need to be removed to increase the models' certainty. CLUE's ability to suggest the addition of unobserved features (positive CLUE attribution) is unique.

The feature importance methods under consideration are difficult to retrofit for uncertainty. They are unable to add features; they are limited to explaining the contribution of existing features. This may suffice if our input contains all the information needed to make a prediction for a certain class but otherwise results in noisy, potentially meaningless, explanations.

Generative-model based methods methods are counterfactual because they do not assign importance to the observed features but rather propose alternative features based on the data manifold [6]. This is the case for FIDO and CLUE. Generative modeling allows for increased flexibility, which is required when dealing with uncertain inputs. Quantitatively contrasting feature importance and uncertainty explanations under existing evaluation criteria [27] is an interesting direction for future work.

Methods like LIME and SHAP require a choice of class to produce explanations. This complicates their use in scenarios where our model is uncertain and multiple classes have similarly high predictive probability. On the other hand CLUEs are class agnostic.

G Additional CLUE and U-FIDO Examples

We provide additional examples of CLUEs generated for high uncertainty MNIST digits in Figure 18. U-FIDO counterfactuals generated for the same inputs are shown in Figure 19. Both methods often attribute importance to the same features. However, in almost all cases, CLUE is able to reduce the original input's uncertainty significantly more than U-FIDO. The latter method suggests smaller changes. We attribute this to U-FIDO's input masking mechanism being less flexible than CLUE's latent space generation mechanism.

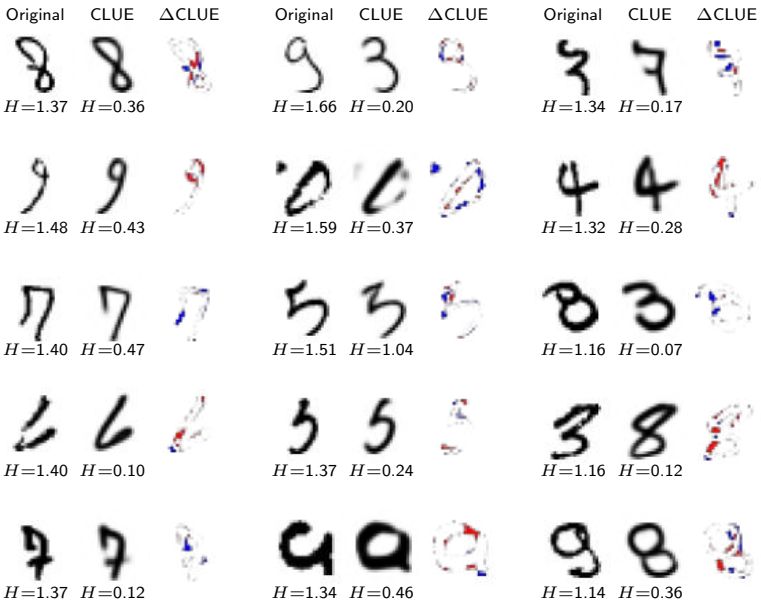


Figure 18: CLUEs generated for MNIST digits for which our BNN's predictive entropy is above the rejection threshold. The BNNs predictive entropy for both original inputs and CLUEs is shown under the corresponding images.

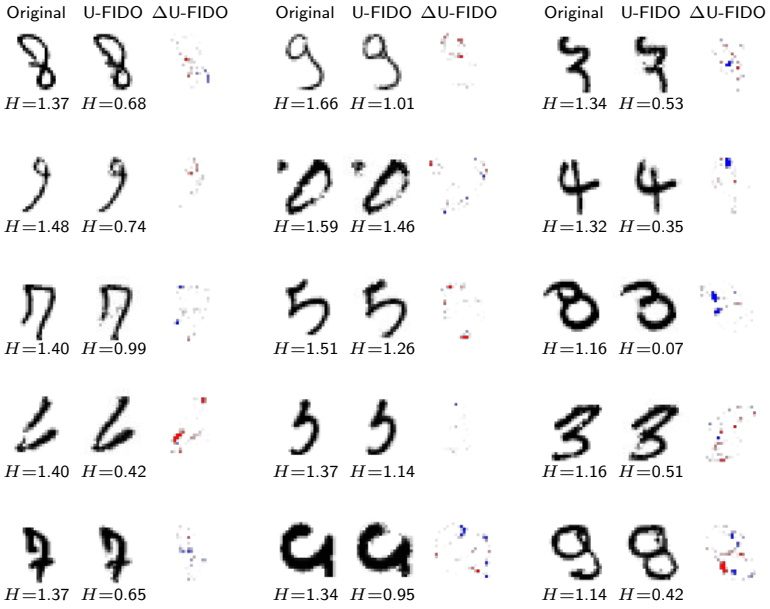


Figure 19: U-FIDO counterfactuals generated for MNIST digits for which our BNN's predictive entropy is above the rejection threshold. The BNNs predictive entropy for both original inputs and counterfactuals is shown under the corresponding images.

H Additional Experimental Results

H.1 Ablation Experiments

In this subsection, we modify some of CLUE's components individually and observe the effects on the procedure's results.

Initialization Strategy: Figure 20 compares Algorithm 1's encoder-based initialization $\mathbf{z}_0 = \mu_\phi(\mathbf{z}|\mathbf{x}_0)$ with $\mathbf{z}_0 = \mathbf{0}$ on all datasets under consideration. For the LSAT, COMPAS and Wine datasets, both approaches produce indistinguishable results. On Credit, our second highest dimensional dataset, using an encoder-based initialization allows for CLUEs to stay slightly closer to original inputs in terms of ℓ_1 distance.

The difference between both approaches is largest on MNIST. We conjecture that this might be due to the higher dimensional nature of the latent space used with this dataset making optimization more difficult. By initializing $\mathbf{z}$ as the VAE encoder's mean, our optimizer starts near a local minima of $d(\mathbf{x}, \mathbf{x}_0)$ and potentially of $\mathcal{L}(\mathbf{z})$. When Algorithm 1 is applied, the magnitude of $\nabla_z H$ might not be large enough to escape this basin of attraction. Thus, CLUE tends to leave most input features unchanged, only addressing those with most potential to reduce uncertainty. This is also desirable behavior for low uncertainty inputs; the closest low uncertainty sample is the input itself.

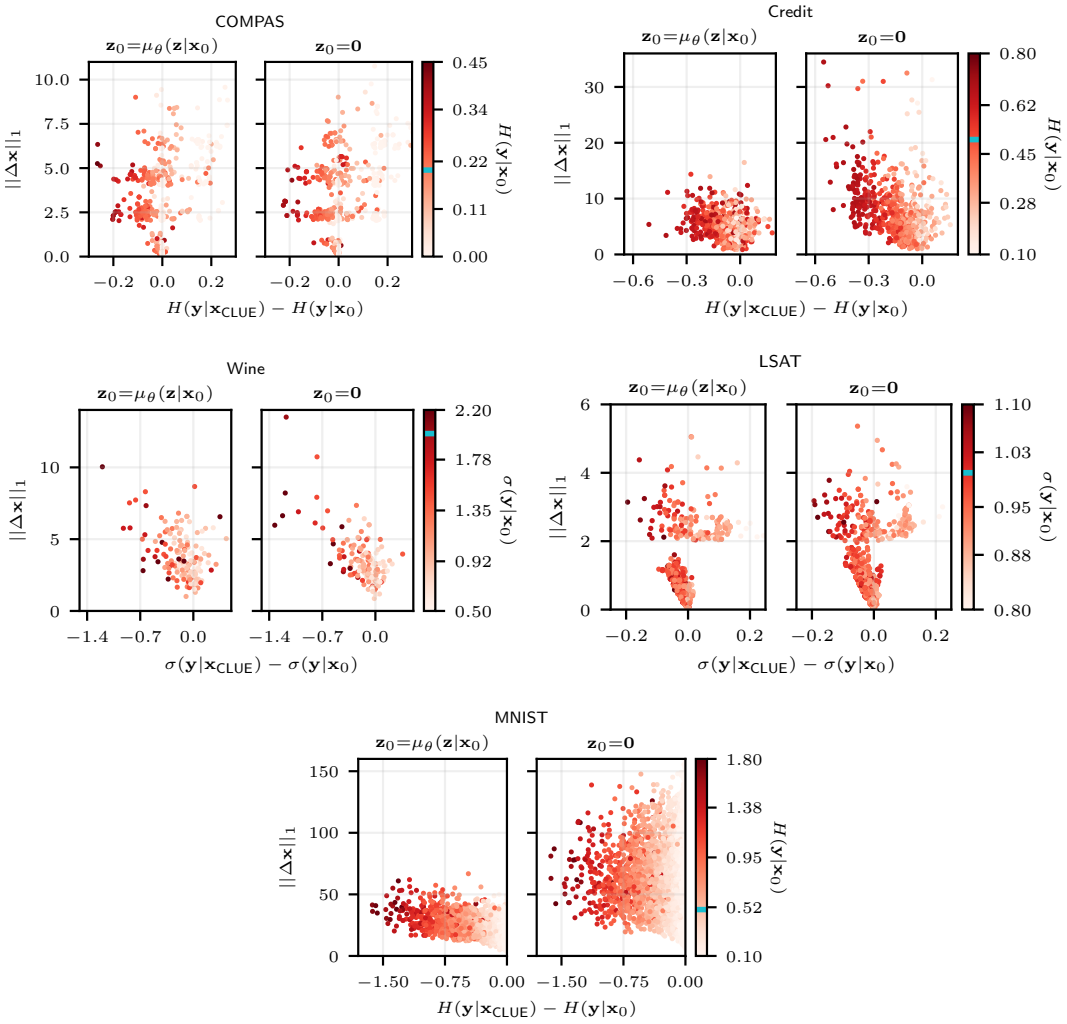


Figure 20: Initialization strategy experiment results for all datasets under consideration. Colorbars' horizontal blue line denotes each dataset's rejection threshold.

Capacity of CLUE's DGM: To capture our predictive model's reasoning, CLUE's DGM must be flexible enough to preserve atypical features in the inputs. As shown in Figure 21, reconstructions from low-capacity VAEs do not preserve the predictive uncertainty of original inputs. The CLUEs generated from these DGMs either leave the inputs unchanged or present large values of $\Delta\mathbf{x}$ while barely reducing H: these degenerate CLUEs simply emphasize regions of large reconstruction error. As our DGM's capacity increases, so does the amount of uncertainty preserved in the auto-encoding operation. The amount of predictive uncertainty explained by CLUEs, which is given by the difference between the autoencoded input uncertainty (orange bars) and CLUE uncertainty (blue bars), increases. We see a clear relationship between dataset dimensionality and size of latent space needed for CLUE to be effective.

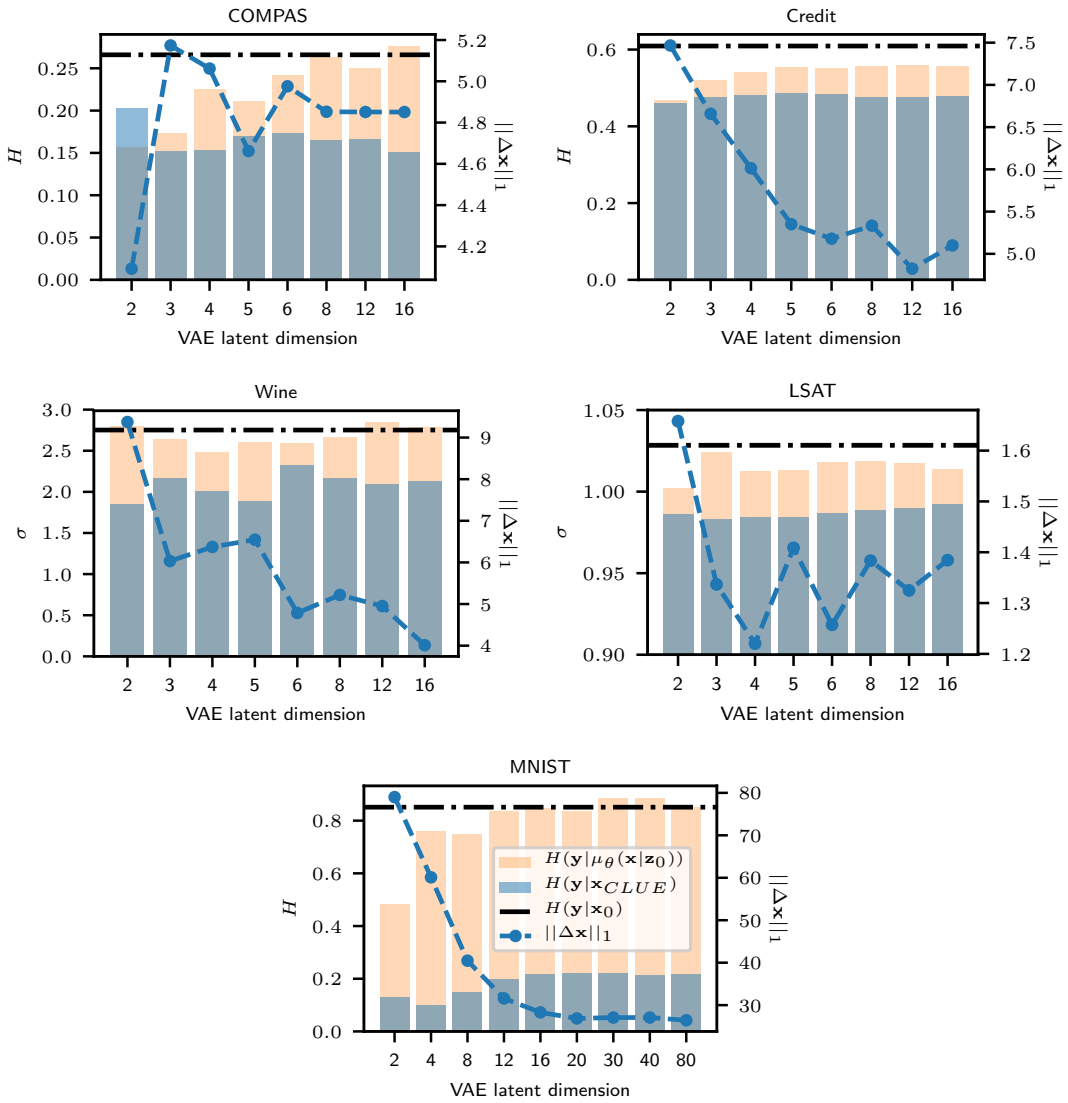


Figure 21: Amount of uncertainty explained away and ℓ_1 distance between original inputs and CLUEs for every dataset under consideration and different capacity VAEs.

Output Space Regularization Parameter λ_y: In Figure 22, we show how increasing λ_y reduces the proportion of samples for which the predicted class differs between original inputs and CLUEs. Interestingly, on LSAT, Wine and COMPAS, a small, but non-zero, value of λ_y results in more uncertainty being explained away by CLUE. However, strongly enforcing similarity of predictions generally comes at the cost of smaller amounts of uncertainty being explained away.

COMPAS predictions stay the same for all values of λ_y. Class predictions only depend on 2 of this dataset's input features (Age and Previous Convictions) [34]. We find that the remaining features can increase or reduce confidence in the prediction given by the two key features, but never change it. CLUEs only change non key features, reinforcing the current classification. On MNIST, we find that, for certain values of λ_y, classifying CLUEs results in a lower error rate than classifying original inputs. This is shown in Figure 23. We did not observe this effect for other datasets.

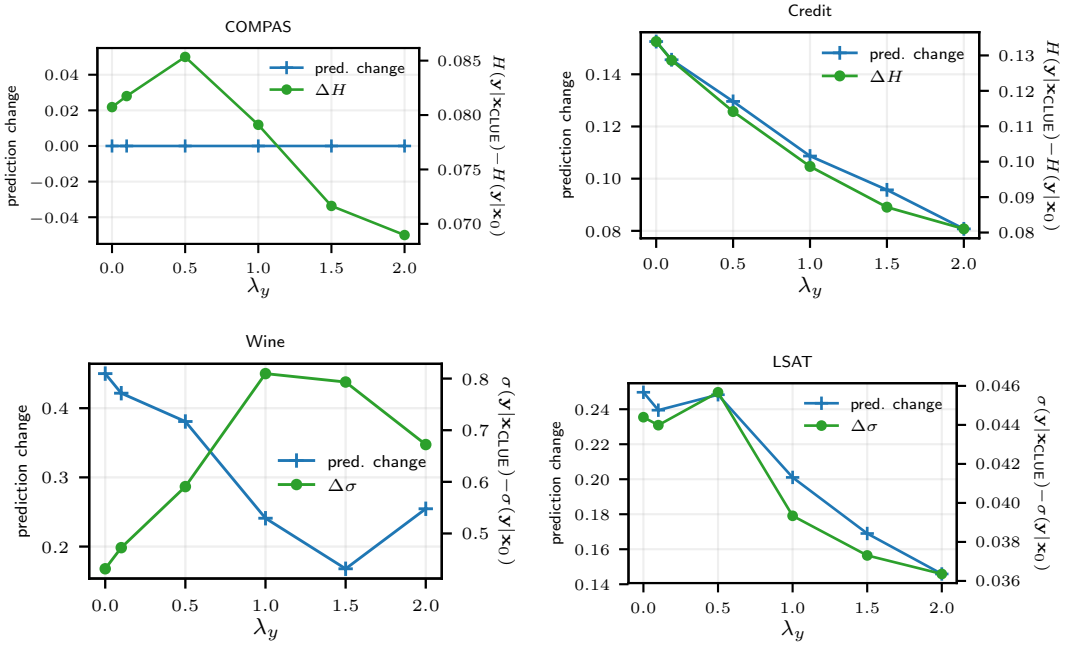


Figure 22: CLUE $\Delta\mathcal{H}$ vs prediction change for all datasets under consideration. Prediction change refers to the proportion of CLUEs classified differently than their corresponding original inputs. All values shown are averages across all testset points above the uncertainty rejection threshold.

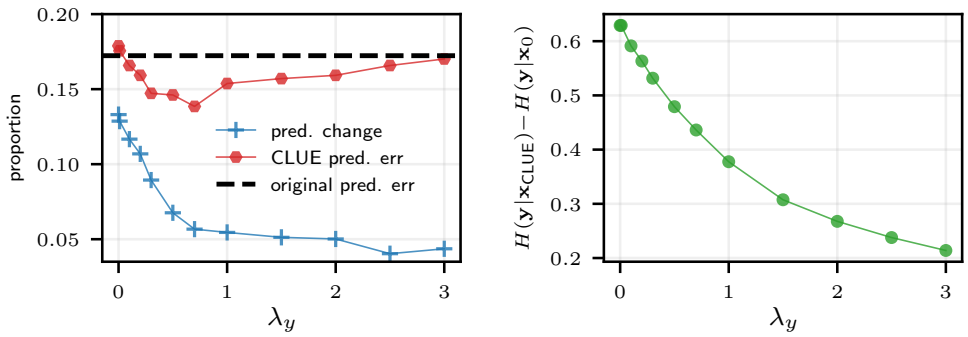


Figure 23: Left: Prediction change refers to the proportion of CLUEs classified differently than their corresponding original inputs. Setting a value of λ_y of around 0.7 results in class predictions for CLUEs being closer to the true labels than the original class predictions. Right: Reduction in predictive entropy achieved by CLUE. All values shown are averages across all testset points above the uncertainty rejection threshold.

Applying CLUE to non-Bayesian NNs: These models are unable to capture model uncertainty. We train deterministic NNs on every dataset under consideration using the architectures described in Appendix B.4. We generate counterfactuals for their noise uncertainty. As shown in Figure 24, CLUE is effective at explaining away noise uncertainty for regular NNs. More uncertain inputs are subject to larger changes in terms of ℓ_1 distance.

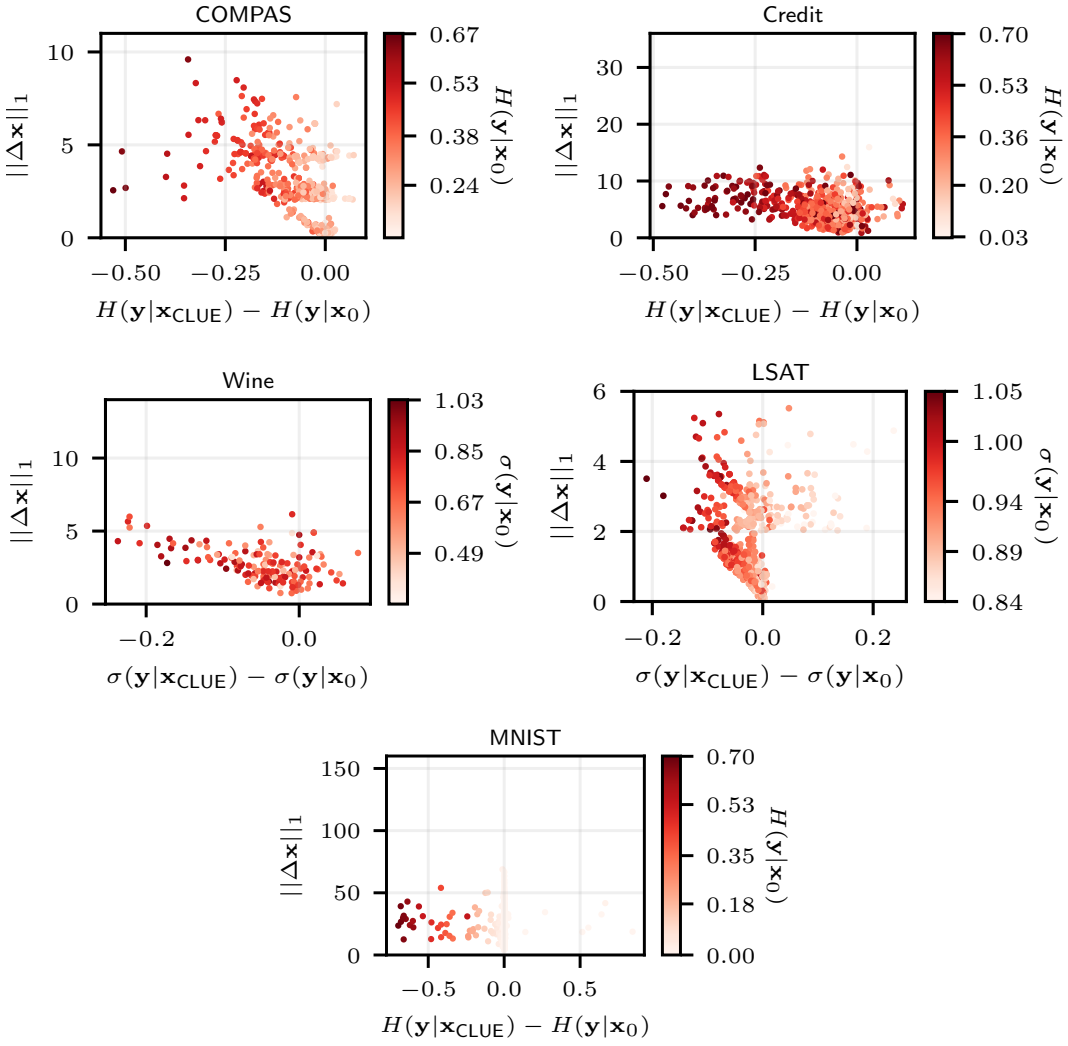


Figure 24: Amount of noise uncertainty explained away vs ℓ_1 shift in input space for all datasets under consideration when applying CLUE to regular NNs. The colorbar indicates the original samples' predictive uncertainty.

Table 5: Relative performance measure obtained by all methods on all datasets under consideration. Lower is better. *e* and *a* indicate results for epistemic (Δerr_{gt}) and aleatoric ($\Delta \mathcal{H}_{\text{gt}}$) uncertainty respectively.

	LSAT		COMPAS		Wine		Credit		MNIST	
Method	e	a	e	a	e	a	e	a	e	a
Sensitivity	0.697	0.672	**0.707**	0.122	0.691	**0.001**	0.623	0.454	0.682	0.698
CLUE	0.419	**0.070**	**0.707**	**0.044**	**0**	0.128	**0**	**0.009**	**0.273**	**0.146**
U-FIDO	**0**	**0**	**0.707**	0.303	0.224	**0**	0.233	0.628	0.450	0.516

H.2 Evaluation with Computationally Grounded Framework

Using the same experimental procedure described in Section 5, we compare local sensitivity analysis, U-FIDO and CLUE in terms of amount of uncertainty explained away $\Delta \mathcal{H}_{\text{gt}}$ vs proximity to the data manifold $\max(0, \log p_{gt}(\bar{\mathbf{x}}_c) - \log p_{gt}(\bar{\mathbf{x}}_0))$. Here, $p_{gt}(\bar{\mathbf{x}}_0)$ refers to the log-likelihood of the artificial data being explained. Again, we normalize both the uncertainty axis and the log-likelihood axis using the largest values obtained by U-FIDO or CLUE. The knee point distances obtained are displayed in Table 5. CLUE performs best in 8 out of 10 tasks. Generating counterfactuals directly from the latent space of a VAE ensures that CLUEs are *relevant*.

H.3 Additional Analysis of User Study

While the main text showed the mean accuracy of CLUE over all tabular questions, we also consider the breakdown of accuracy by dataset and by test point certainty in Table 6. CLUE outperforms all baselines on both datasets. We find that sensitivity does significantly worse in higher dimensions (on COMPAS), lending further credence to the intuition described in Appendix D.

When splitting by the certainty of test points, we immediately notice that accuracy for uncertain test points is quite high for all methods. This similarity is expected since certain context points are the only factor that varies between each method's survey. Survey participants seemed to not use the certain context points to identify uncertain test points. This is probably due to pilot procedure, wherein Participant A carefully paired test points with relevant uncertain context points. Indeed, the random baseline, which controls for the possibility that our task can be solved without access to a relevant counterfactual, performs best on uncertain test points. However, we note a large difference between methods' results when identifying certain test points. CLUE's accuracy almost doubles the second best method's (*Human CLUE*). When generating *Human CLUE*s Participant B had knowledge of the uncertain context point, but not the test point (just like other methods). For this reason, we expect to see dissimilarity in methods' performance on certain test points. CLUE's ability to bring about most relevant contrast is one possible explanation for why it does so much better than baselines for certain context points.

Table 6: Accuracy (%) of participants on the Tabular main survey broken down by dataset and by certainty of test points.

	Combined	LSAT	COMPAS	Certain Test	Uncertain Test
CLUE	**82.22**	**83.33**	**81.11**	**71.00**	96.25
Human CLUE	62.22	61.11	63.33	38.00	92.50
Random	61.67	62.22	61.11	31.00	**100**
Local Sensitivity	52.78	56.67	48.89	20.90	92.50

In addition to the Neymeni Test discussed in the main text, we also run paired Wilcoxon tests of CLUE against each baseline [36]. Under a standard 0.05 acceptance threshold, CLUE significantly outperforms all baselines on the Wilcoxon test as well: CLUE vs. Random ($p = 1.47\text{e}{-5}$), CLUE vs. Sensitivity ($p = 2.60\text{e}{-9}$), and CLUE vs. *Human CLUE* ($p = 2.34\text{e}{-5}$). Statistical significance under the Wilcoxon test holds even if we apply the Bonferroni correction. We conclude that CLUE is useful for practitioners to identify the certainty of test points.

I Additional Details on the Generative Model used in the Proposed Computationally Grounded Evaluation Framework

The framework described in Figure 6 uses a conditional DGM, specifically a VAEAC [28], to both generate artificial data and to evaluate explanations for said data. VAEs are known for generating blurry or overly smoothed data. For our evaluation framework to work well, we require the ground truth DGM to generate sharp data, with atypical characteristic that would to lead to a predictor being uncertain. We can ensure that this is the case by using a large latent dimensionality. However, this brings forth another well-known issue with VAEs: distribution mismatch [43; 46; 47]. The region of latent space where the encoder places probability mass, also known as the aggregate posterior,

$$q_\phi(\mathbf{z}) = \int q_\phi(\mathbf{z}|\mathbf{x})p(\mathbf{x})\, d\mathbf{x}$$

does not match the prior $p(\mathbf{z})$.

To visualize this phenomenon, we train a BNN and a VAE on MNIST. We sample points from the VAE's latent space and evaluate their uncertainty with the BNN. As shown in Figure 25, clusters of same-class digits form in latent space. The aggregate posterior presents low density in the spaces between clusters. Digits generated from these areas are of low-quality, causing our BNN to be uncertain. The outer regions of latent space, where the isotropic Gaussian prior has low density, also generate uncertain digits.

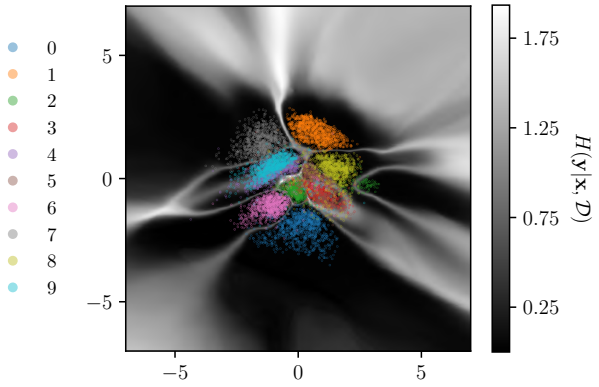


Figure 25: Predictive entropy estimates for artificial MNIST digits generated from a 2-dimensional VAE latent space. The MNIST test set digits have been projected onto the latent space and are displayed with a different color per class.

Recently, Dai and Wipf [43] have proposed the two-level VAE as a solution to distribution mismatch. After training a standard VAE, a second VAE is trained on samples from the first VAE's latent space. As illustrated in Figure 26, the aggregate posterior over the inner latent variables, which we denote by $q(\mathbf{u})$, more closely resembles the prior. The joint distribution over inputs and latent variables factorizes as: $p(\mathbf{x}, \mathbf{z}, \mathbf{u}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}|\mathbf{u})p(\mathbf{u})$. We refer the reader to [43] for a detailed analysis. Figure 27 shows that, while generating digits from samples of $p(\mathbf{z})$ results in a large amount of low-quality or OOD reconstructions, samples from $p(\mathbf{u})$ map to clean digits. The two-stage mechanism restores the VAE's pivotal ancestral sampling capability, ensuring that our experiments with artificial data will be representative of methods performance on real data.

In order to generate artificial data, we draw samples from the auxiliary latent space, map them back to the VAEAC's latent space and then map them to the input space. This allows for high-quality sample generation. In this way, a single VAEAC can be used for both ancestral sampling and conditional sampling. In addition, it allows us to estimate the log-likelihood of inputs as:

$$\log p_{gt}(\mathbf{x}) = \log \int p_{\theta_1}(\mathbf{x}|\mathbf{z})p_{\theta_2}(\mathbf{z}|\mathbf{u})p(\mathbf{u})\, d\mathbf{z}\, d\mathbf{u} \tag{8}$$

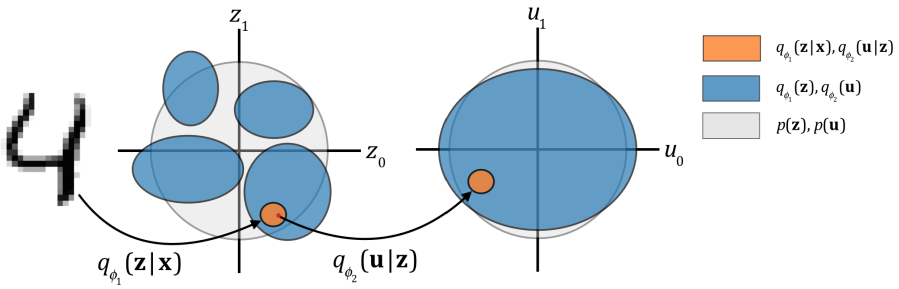


Figure 26: In its first stage, the two-level VAE maps input samples to approximate posteriors in the outer latent space. The aggregate posterior over this latent space need not resemble the isotropic Gaussian prior. The second VAE maps samples from the outer latent space to approximate posteriors in the inner latent space. The aggregate posterior over the inner latent space more closely matches the prior.

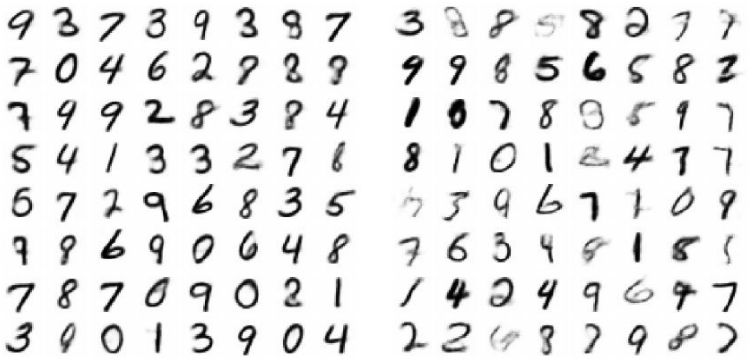


Figure 27: Left: Digits generated from the inner latent space of a VAEAC trained on MNIST with a two-level mechanism. Right: Digits generated from the latent space of a VAEAC trained on MNIST. $\mathbf{u}$ and $\mathbf{z}$ are drawn from $\mathcal{N}(\mathbf{0}, I)$.

In (8) parameter subscripts refer to the outer (1st level) and inner (2nd level) networks. In order to preserve computational tractability, we approximate $p_{\theta_2}(\mathbf{z}|\mathbf{u})$ with a point estimate placed at its mean $p_{\theta_2}(\mathbf{z}|\mathbf{u}) \approx \delta(\mathbf{z} - \mu_{\theta_2}(\mathbf{z}|\mathbf{u}))$. We further approximate (8) with importance sampling:

$$\log p_{gt}(\mathbf{x}) \approx \log \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\theta_1}(\mathbf{x}|\mathbf{z}{=}\mu_{\theta_2}(\mathbf{z}|\mathbf{u}_k))p(\mathbf{u}_k)}{q(\mathbf{u}_k|\mathbf{x})}; \quad \mathbf{u}_k \sim q(\mathbf{u}|\mathbf{x}) \tag{9}$$

I.1 Comparison of Methods under a Ground Truth DGM

The two-level VAEAC setup described above partially addresses the concern that our synthetic data might not be diverse enough to highlight differences among the methods being compared. Indeed, our results from Table 1 and Table 5 show noticeable differences in performance across methods.

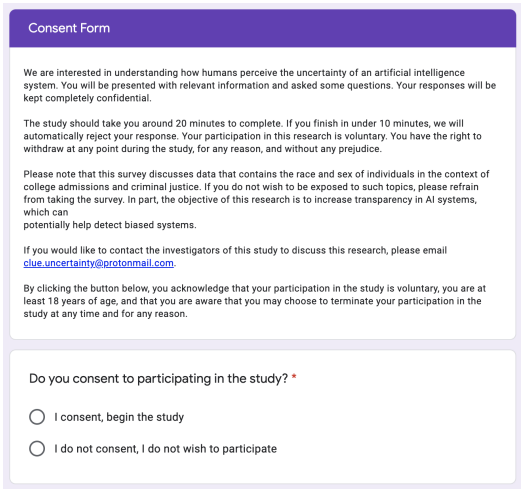
We now address the opposite concern; methods that leverage auxiliary VAEs might be unfairly advantaged under our functionally grounded framework, as the generative process of our synthetic data is also VAE-based. Because VAEs are very flexible neural network based generative models, using them as a ground truth provides relatively little inductive biases for auxiliary DGMs to take advantage of. Additionally, our ground truth VAEAC captures the joint distribution of inputs and targets. The metric of interest, $\Delta\mathcal{H}_{\text{gt}}$, only depends on the conditional distribution over targets $p_{gt}(\mathbf{y}|\mathbf{x})$. Our auxiliary DGMs only model inputs.

Two of the methods we evaluate, U-FIDO and CLUE, leverage auxiliary DGMs. Thus, both would be equally advantaged. The $\Delta\mathcal{H}_{\text{gt}}$ vs $p_{\text{gt}(\bar{\mathbf{x}}_c)}$ metric from Table 5 is the most dependent on the ground truth VAEAC. However, we observe the largest difference between CLUE and U-FIDO on in this metric.

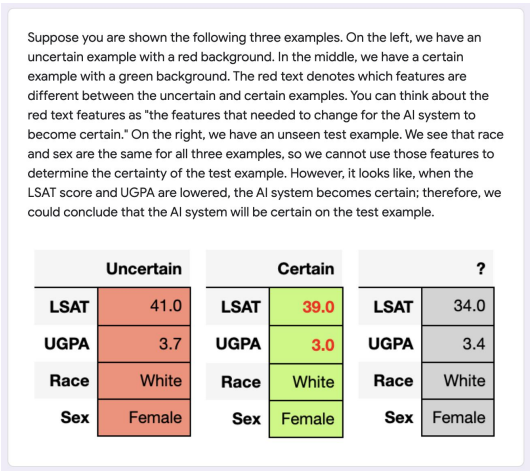
J Details on User Study

J.1 Additional Details on Tabular User Study

For our pilot point selection procedure, we take points from each dataset's test set that score above the uncertainty rejection thresholds described in Appendix B.4 as uncertain points. Points below the thresholds are labeled as certain points. Pilot procedure participants, referred to as participant A in the main text, were not informed that the pools were split up by the points' certainty with respect to the BNN being explained.



(a) Consent Form for the Tabular Main Survey



(b) Attention Check for the Tabular Main Survey

Figure 28: Setup of tabular user studies.

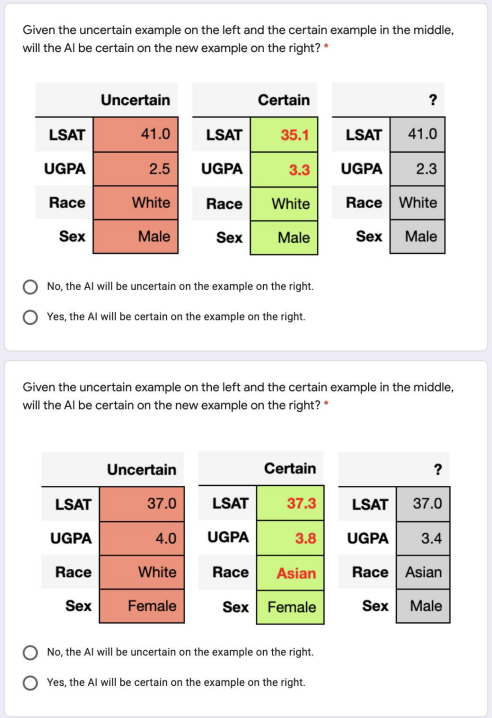
We now go through the various sections of the main survey. In Figure 28a, we include the consent form used in our user studies. This user study was performed with the approval of the University of Cambridge's Department of Engineering Research Ethics Committee. Only three participants who were asked to take the survey did not provide consent and thus exited the form. We still ensured that at least ten participants took each of the four survey variants.

We then include an example question for each dataset, called an "attention check." An example is shown in Figure 28b. Note that the answer to this example question is provided in line. Later in the survey, we ask participants this exact same question. We ask one attention check per dataset. If participants get the attention check wrong for both datasets, we void their results. We only had to void one result. This did not affect our criteria of ten completed surveys per variant. The consent form and attention check questions were the same for all survey variants. The main survey participants were first asked the ten LSAT questions followed by the ten COMPAS questions: we made this design decision since the dimensionality of LSAT is lower than that of COMPAS, easing participants into the task. Examples of questions from the CLUE survey variant are shown in Figure 29.

J.2 MNIST User Study

In order to validate CLUE on image data, we create a modified MNIST dataset with clear failure modes for practitioners to identify. We first discard all classes except four, seven, and nine. We then manually identify forty sevens from the training set which have dashes crossing their stems. Using K-nearest-neighbors, we identify the twelve sevens closest to each of the ones manually selected. We delete these 520 sevens from our dataset. We repeat the same procedure for fours which have a closed, triangle-shaped top. We do not delete any digits from the test set. We train a BNN on this new dataset. Our BNN presents high epistemic uncertainty when tested on dashed sevens and closed fours as a consequence of the sparsity of these features in the train set.

We evaluate the test set of fours, sevens, and nines with our BNN. Datapoints that surpass our uncertainty threshold are selected as candidates to be shown in our user study as uncertain context



(a) Two LSAT questions with certain points generated by CLUE

(b) Two COMPAS questions with certain points generated by CLUE

Figure 29: Example Tabular Main Survey questions

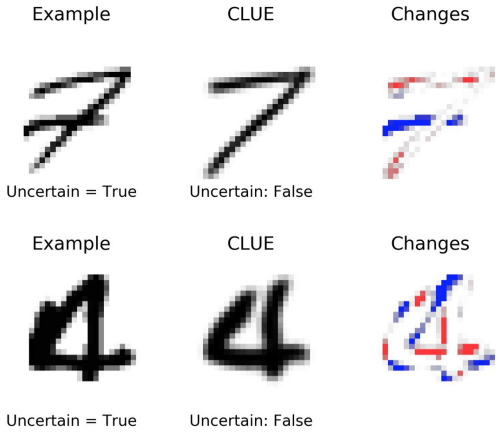


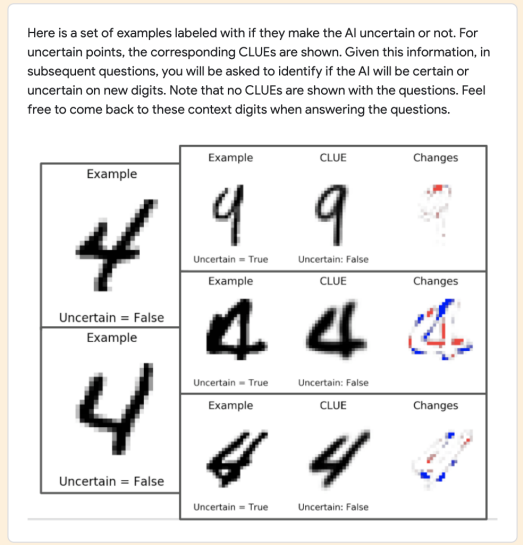
Figure 30: Examples of high uncertainty digits containing characteristics that are uncommon in our modified MNIST dataset. Their corresponding CLUEs and ΔCLUEs are displayed beside them.

examples or test questions. We show example CLUEs for a four and a seven that display the characteristics of interest in Figure 30.

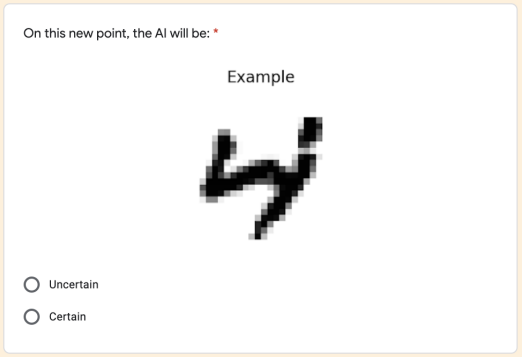
Leveraging the modified MNIST dataset, we run another user study with 10 questions and two variants. Unlike our tabular experiments, we show practitioners a set of five *context points* to start, as opposed to a pair. This set of *context points* is chosen at random from the training set. The first variant involves showing users the set of *context points*, labeled with if their uncertainty surpasses our predefined threshold. We then ask users to predict if new test points will be certain or uncertain to

the BNN. The second variant contains the same labeled context points and test datapoints. However, together with uncertain context points, practitioners are shown CLUEs of how the input features can be changed such that the BNN's uncertainty falls below the rejection threshold. The practitioners are then asked to decide if new points' predictions will be certain or not. If CLUE works as intended, practitioners taking the second variant should be able to identify points on which the BNN will be uncertain more accurately.

The first variant was shown to 5 graduate students with machine learning expertise who only received context points and rejection labels (uncertain or not). This group was able to correctly classify 67% of the new test points as high or low uncertainty. The second variant was shown to 5 other graduate students with machine learning expertise who received context points together with CLUEs in cases of high uncertainty. This group was able to reach an accuracy of 88% on new test points. This user study suggests CLUEs are useful for practitioners in image-based settings as well.



(a) Example Context Set with CLUEs



(b) Example question

Figure 31: MNIST User Study Setup