ExpProof : Operationalizing Explanations for Confidential Models with ZKPs

Chhavi Yadav[†*], Evan Monroe Laufer[‡*], Dan Boneh[‡], Kamalika Chaudhuri[†]
[†]*UC San Diego* {*cyadav,kamalika*}*@ucsd.edu*
[‡]*Stanford University* {*emlaufer,dabo*}*@stanford.edu*
[*]*Equal Contribution*

***Abstract*—Explanations are intended as a way to increase trust in machine learning models and are often obligated by regulations. However, many circumstances where these are demanded are adversarial in nature, meaning the involved parties have misaligned interests and are incentivized to manipulate explanations for their purpose. As a result, explainability methods fail to be operational in such settings despite the demand. In this paper, we take a step towards operationalizing explanations in adversarial scenarios with Zero-Knowledge Proofs (ZKPs), a cryptographic primitive. Specifically we explore ZKP-amenable versions of the popular explainability algorithm LIME and evaluate their performance on Neural Networks and Random Forests. Our code is available at : https://github.com/infinite-pursuits/ExpProof.**

1. Introduction

> "Bottom line: Post-hoc explanations are highly problematic in an adversarial context" [5]

Explanations have been seen as a way to enhance trust in machine learning (ML) models by virtue of making them transparent. Although starting out as a debugging tool, they are now also widely proposed to prove fairness and sensibility of ML-based predictions for societal applications, in research studies [15, 16, 20, 23, 24, 31, 32, 33] and regulations alike (Right to Explanation [34]). However, as discussed in detail by [5], many of these use-cases are adversarial in nature where the involved parties have misaligned interests and are incentivized to manipulate explanations to meet their ends. For e.g. a bank which denies loan to an applicant based on an ML model's prediction has an incentive to return an *incontestable* explanation to the applicant rather than reveal the true workings of the model since the explanation can be used by the applicant to prove discrimination in the court of law [5]. In fact, many previous studies show that adversarial manipulations of explanations are possible in realistic settings with systematic and computationally feasible attacks [28, 29, 30, 37]. As such, despite the demand, explanations fail to be operational as a trust-enhancing tool.

A major barrier to using explanations in adversarial contexts is that organizations keep their models confidential due to IP and legal reasons. However, confidentiality aids in manipulating explanations by allowing model swapping – a model owner can use different models for generating predictions vs. explanations, swap the model for specific inputs, or change the model post-audits [30, 36, 38]. This problem demands a technical solution which guarantees that a specific model is used for all inputs, for generating both the prediction and the explanation and prove this to the customer on the receiving end while keeping the model confidential.

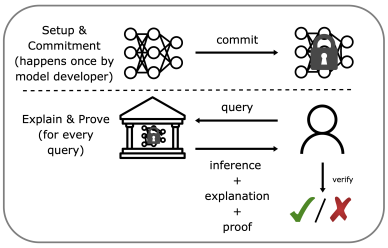

Figure 1. Pictorial Representation of *ExpProof*

Another important barrier to using explanations in adversarial contexts is that many explanation algorithms are not deterministic and have many tunable parameters. An adversary can choose these parameters adversarially to make discriminatory predictions seem benign. Moreover, there is no guarantee that the model developer is following the explanation algorithm correctly to generate explanations. A plausible solution to counter this problem is consistency checks [3, 6]. Apart from being a rather lopsided ask where the onus of proving correctness of explanations lies completely on the customer, these checks require collecting multiple explanation-prediction pairs for different queries and are therefore infeasible for individual customers in the real world. Compounding the issue, it has been shown that auditing local explanations with consistency checks is hard [3]. Note that many of these issues persist even with a perfectly faithful algorithm for generating explanations.

We address the aforementioned challenges by proposing a system called *ExpProof*. *ExpProof* gives a protocol consisting of (1) cryptographic commitments which guarantee that the same model is used for all inputs and (2) Zero-Knowledge Proofs (ZKPs) which guarantee that the explanation was computed correctly using a predefined explanation algorithm, both while maintaining model confidentiality. See Fig. 1 for a pictorial representation of *ExpProof*.

ExpProof ensures uniformity of the model and explanation parameters through cryptographic commitments [4].

Commitments publicly bind the model owner to a fixed set of model weights and explanation parameters while keeping the model confidential. Commitments for ML models are a very popular and widely researched area in cryptography and hence we use standard procedures to do this [17].

Furthermore, we wish to allow the customer to verify that the explanation was indeed computed correctly using the said explanation algorithm, without revealing model weights. To do this, we employ a cryptographic primitive called Succinct Zero-Knowledge Proofs [10, 11]. ZKPs allow a prover (bank) to prove a statement (explanation) about its private data (model weights) to the verifier (customer) without leaking the private data. The prover outputs a cryptographic proof and the verifier on the other end verifies the proof in a computationally feasible way. In our case, if the proof passes the verifier's check, it means that the explanation was correctly computed using the explanation algorithm and commited weights without any manipulation.

How are explanations computed? The explanation algorithm we use in our paper is a popular one called LIME [27], which returns a local explanation for the model decision boundary around an input point. We choose a local explanation rather than a global one since customers are often more interested in the behavior of the model around their input specifically. Additionally, LIME is model-agnostic, meaning that it can be used for any kind of model class.

Traditionally ZKPs are slow and infamous for adding a huge computational overhead for proving even seemingly simple algorithmic steps. Moreover many local explanation algorithms require solving an optimization problem and involve non-linear functions such as exponentials, which makes it infeasible to simply reimplement LIME as-is in a ZKP library. To remedy these issues, we experiment with different variants of LIME exploring the resulting tradeoffs between explanation-fidelity and ZKP-overhead. To make our ZKP system efficient, we also utilize the fact that verification can be easier than re-running the computation – instead of *solving* the optimization problem within ZKP, we verify the optimal solution using duality gap.

Experiments. We evaluate *ExpProof* on fully connected ReLU Neural Networks and Random Forests for three standard datasets on an Ubuntu Server with 64 CPUs of x86_64 architecture and 256 GB of memory without any explicit parallelization. Our results show that *ExpProof* is computationally feasible, with a maximum proof generation time of 1.5 minutes, verification time of 0.12 seconds and proof size of 13KB for NNs and standard LIME.

2. LIME and its variants

Local Interpretable Model-Agnostic Explanations (LIME) [27] explains the prediction for an input point by approximating the local decision boundary around that point with a linear model. Formally, given an input point $x \in \mathcal{X}$, a complex non-interpretable classifier $f : \mathcal{X} \to \mathcal{Y}$ and an interpretable class of models $\mathcal{G}$, LIME explains the prediction $f(x) \in \mathcal{Y}$ with a local interpretable model $g \in \mathcal{G}$. The interpretable model g is found from the class $\mathcal{G}$ via learning, on a set of points randomly sampled around the input point and weighed according to their distance to the input point, as measured with a similarity kernel π. The similarity kernel creates a locality around the input by giving higher weights to samples near input x as compared to those far off. A natural and popular choice for the interpretable class of models $\mathcal{G}$ is linear models such that for any $g \in \mathcal{G}$, $g(z) = w_g \cdot z$ ([9, 27]), where w_g are the coefficients of linear model g. These coefficients highlight the contribution of each feature towards the prediction and therefore serve as the explanation in LIME. Learning the linear model is formulated as a weighted LASSO problem, since sparsity induced by ℓ_1 regularization leads to more interpretable and human-understandable explanations. The similarity kernel is set to be the exponential kernel with ℓ_2 norm as the distance function, $\pi_x(z) = \exp\left(-\ell_2(x,z)^2/\sigma^2\right)$ where σ is the bandwidth parameter of the kernel and controls locality around input x.

Building zero-knowledge proofs of explanations requires the explanation algorithm to be implemented in a ZKP library which is known to introduce a significant computational overhead. Given this, a natural question that comes to mind is if there exist variants of LIME which provide similar quality of explanations but are more ZKP-amenable by design, meaning they introduce a smaller ZKP overhead?

Standard LIME Variants. To create variants of standard LIME (Alg.1), we focus on the two steps which are carried out numerous times and hence create a computational bottleneck in the LIME algorithm – sampling around input x (Step 6 in Alg. 1) and computing distance using exponential kernel (Step 7 in Alg. 1). For sampling, we propose two options as found in the literature : gaussian (G) and uniform (U) [8, 9, 27]. For the kernel we propose to either use the exponential (E) kernel or no (N) kernel. These choices give rise to four variants of LIME, mentioned in Alg. 2. We address each variant by the intials in the brackets, for instance standard LIME with uniform sampling and no kernel is addressed as 'LIME_U+N'.

We propose another variant of LIME called BorderLIME to consider inputs far off from the decision boundary where the LIME algorithm returns trivial explanations. For complete algorithms and more details refer to the Appendix.

3. *ExpProof*: Verification of Explanations

Our system for operationalizing explanations in adversarial settings, *ExpProof*, consists of two phases: (1) a One-time Commitment phase and (2) an Online verification phase which should be executed for every input.

Commitment Phase. To ensure model uniformity, the model owner cryptographically commits to a fixed set of model weights $\mathbf{W}$ belonging to the original model f, resulting in committed weights $\mathsf{com}_{\mathbf{W}}$. Architecture of model f is assumed to be public. Additionally, model owner can also commit to the values of different parameters used in the explanation algorithm or these parameters can be public.

Online Verification Phase. This phase is executed every time a customer inputs a query. On receiving the query, the prover (bank) outputs a prediction, an explanation and a zero-knowledge proof of the explanation. Verifier (customer) validates the proof without looking at the model weights. If the proof passes verification, it means that the explanation is correctly computed with the committed model weights and explanation algorithm parameters.

To generate the explanation proof, a ZKP circuit which implements (a variant of) LIME is required. However since ZKPs can be computationally inefficient, instead of reimplementing the algorithm as-is in a ZKP library, we devise some smart strategies for verification, based on the fact that verification can be easier than redoing the computation. We describe verification strategies for these functionalities in Appendix Sec. D.

4. Experiments

Setup. We use three standard fairness benchmarks for experimentation : Adult [2], Credit [39] and German Credit [13]. We train two kinds of models on these datasets, 2-layer fully connected ReLU activated neural networks and random forests. We code *ExpProof* with different variants of LIME in the *ezkl* library [19] (Version 18.1.1) which uses Halo2 [40] as its underlying proof system in the Rust programming language, resulting in $\sim 3.7k$ lines of code.

Research Questions & Metrics. We ask two questions for the different variants of LIME (Q1) How faithful are the explanations generated by the LIME variant? and (Q2) What is the time and memory overhead introduced by implementing the LIME variant in a ZKP library?

To answer Q1, we need a measure of fidelity of the explanation, we use 'Prediction Similarity' defined as the similarity of predictions between the explanation classifier and the original model in a local region around the input. To answer Q2, we will look at the proof generation time taken by the prover to generate the ZK proof, the verification time taken by the verifier to verify the proof and the proof length which measures the size of the generated proof.

4.1. Standard LIME Variants

Fidelity Results. As shown in Fig. 2, we do not find a huge difference between the explanation fidelities of the different variants of LIME as the error bars significantly overlap. This could be due to the small size of the local neighborhoods where the kernel or sampling doesn't matter much. However, for the credit dataset, which has the highest number of input features, gaussian sampling works slightly better than uniform, which could be because of the worsening of uniform sampling with increasing dimension.

ZKP Overhead Results. Across the board, proof generation takes a maximum of ~ 1.5 minutes, verification time takes a maximum of ~ 0.12 seconds and proof size is a maximum of ~ 13KB, as shown in Fig. 3. Note that while proof generation time is on the order of minutes, verification time is on the order of seconds – this is due to the inherent design of ZKPs, requiring much lesser resources at the verifier's end (contrary to consistency-based explanation checks). We also observe that the dataset type does not have much influence on the ZKP overhead; this is due to same ZKP backend parameters needed across datasets.

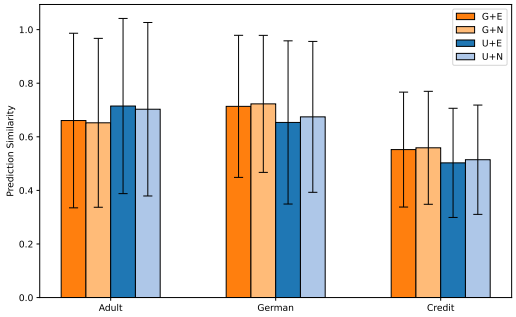

Figure 2. Results for NNs. G/U: gaussian or uniform sampling, E/N: using or not using the exponential kernel. Fidelity of variants of Standard LIME.

Furthermore, we see that gaussian sampling leads to a larger ZKP overhead. This can be attributed to our implementation of gaussian sampling in the ZKP library, wherein we first create uniform samples and then transform them to gaussian samples using the inverse CDF method, leading to an additional step in the gaussian sampling ZKP circuit as compared to that of uniform sampling. Similarly, using the exponential kernel leads to a larger overhead over not using it due to additional steps related to verifying the kernel.

Overall, 'gaussian sampling and no kernel' variant of LIME is likely the most amenable for a practical ZKP system as it produces faithful explanations with a small overhead.

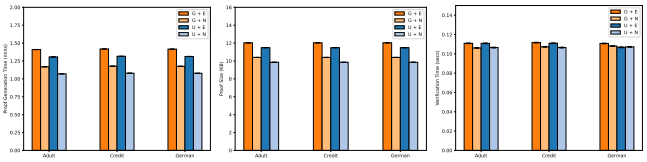

Figure 3. Results for NNs. G/U: gaussian or uniform sampling, E/N: using or not using the exponential kernel. Left: Proof Generation Time (in mins), Mid: Proof Size (in KBs), Right: Verification times (in secs) for different variants of Standard LIME. All configurations use the same number of Halo2 rows, 2^{18}, and lookup tables of size 200k.

All other experiments and all details can be found in the Appendix.

Conclusion & Future Work In this paper we take a step towards operationalizing explanations in adversarial contexts where the involved parties have misaligned interests. We propose a protocol *ExpProof* using Commitments and Zero-Knowledge Proofs, which provides guarantees on the model used and correctness of explanations in the face of confidentiality requirements. We propose ZKP-efficient versions of the popular explainability algorithm LIME and demonstrate the feasibility of *ExpProof* for Neural Networks & Random Forests.

An interesting avenue for future work is the tailored design of explainability algorithms for high ZKP-efficiency and inherent robustness to adversarial manipulations. Another interesting avenue is finding other applications in ML where ZKPs can ensure verifiable computation and provide trust guarantees without revealing sensitive information.

References

[1] U. Aïvodji, H. Arai, O. Fortineau, S. Gambs, S. Hara, and A. Tapp. Fairwashing: the risk of rationalization. In *International Conference on Machine Learning*, pages 161–170. PMLR, 2019.

[2] B. Becker and R. Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[3] R. Bhattacharjee and U. von Luxburg. Auditing local explanations is hard. *arXiv preprint arXiv:2407.13281*, 2024.

[4] M. Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.

[5] S. Bordt, M. Finck, E. Raidl, and U. von Luxburg. Post-hoc explanations fail to achieve their purpose in adversarial contexts. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 891–905, 2022.

[6] S. Dasgupta, N. Frost, and M. Moshkovitz. Framework for evaluating faithfulness of local explanations. In *International Conference on Machine Learning*, pages 4794–4815. PMLR, 2022.

[7] S. Garg, A. Goel, S. Jha, S. Mahloujifar, M. Mahmoody, G.-V. Policharla, and M. Wang. Experimenting with zero-knowledge proofs of training. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1880–1894, 2023.

[8] D. Garreau and U. Luxburg. Explaining the explainer: A first theoretical analysis of lime. In *International conference on artificial intelligence and statistics*, pages 1287–1296. PMLR, 2020.

[9] D. Garreau and U. von Luxburg. Looking deeper into tabular lime. *arXiv preprint arXiv:2008.11092*, 2020.

[10] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, jul 1991.

[11] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[12] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. Poseidon: A new hash function for Zero-Knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535. USENIX Association, Aug. 2021.

[13] H. Hofmann. Statlog (German Credit Data). UCI Machine Learning Repository, 1994. DOI: https://doi.org/10.24432/C5NC77.

[14] M. Jordan, J. Lewis, and A. G. Dimakis. Provable certificates for adversarial examples: Fitting a ball in the union of polytopes. *Advances in neural information processing systems*, 32, 2019.

[15] A.-H. Karimi, G. Barthe, B. Schölkopf, and I. Valera. A survey of algorithmic recourse: definitions, formulations, solutions, and prospects. *arXiv preprint arXiv:2010.04050*, 2020.

[16] L. Kästner, M. Langer, V. Lazar, A. Schomäcker, T. Speith, and S. Sterz. On the relation of trust and explainability: Why to engineer for trustworthiness. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pages 169–175. IEEE, 2021.

[17] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

[18] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):606–617, 2007.

[19] Konduit. ezkl: Efficient zero-knowledge machine learning. https://github.com/zkonduit/ezkl, 2024. Accessed: 2025-01-21.

[20] M. Langer, D. Oster, T. Speith, H. Hermanns, L. Kästner, E. Schmidt, A. Sesing, and K. Baum. What do we want from explainable artificial intelligence (xai)?–a stakeholder perspective on xai and a conceptual model guiding interdisciplinary xai research. *Artificial Intelligence*, 296:103473, 2021.

[21] T. Laugel, M.-J. Lesot, C. Marsala, X. Renard, and M. Detyniecki. Inverse classification for comparison-based interpretability in machine learning. *arXiv preprint arXiv:1712.08443*, 2017.

[22] T. Laugel, X. Renard, M.-J. Lesot, C. Marsala, and M. Detyniecki. Defining locality for surrogates in post-hoc interpretablity. *arXiv preprint arXiv:1806.07498*, 2018.

[23] D. Leben. Explainable ai as evidence of fair decisions. *Frontiers in Psychology*, 14:1069426, 2023.

[24] Q. V. Liao and K. R. Varshney. Human-centered explainable ai (xai): From algorithms to user experiences. *arXiv preprint arXiv:2110.10790*, 2021.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[27] M. T. Ribeiro, S. Singh, and C. 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, pages 1135–1144, 2016.

[28] A. Shahin Shamsabadi, M. Yaghini, N. Dullerud, S. Wyllie, U. Aïvodji, A. Alaagib, S. Gambs, and N. Papernot. Washing the unwashable: On the (im) possibility of fairwashing detection. *Advances in Neural Information Processing Systems*, 35:14170–14182, 2022.

[29] D. Slack, A. Hilgard, H. Lakkaraju, and S. Singh. Counterfactual explanations can be manipulated. *Advances in neural information processing systems*, 34:62–75, 2021.

[30] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, 2020.

[31] N. A. Smuha. The eu approach to ethics guidelines for trustworthy artificial intelligence. *Computer Law Review International*, 20(4):97–106, 2019.

[32] W. J. Von Eschenbach. Transparency and the black box problem: Why we do not trust ai. *Philosophy & Technology*, 34(4):1607–1622, 2021.

[33] S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841, 2017.

[34] Wikipedia contributors. Right to explanation, 2025. Accessed: 2025-01-14.

[35] C. Yadav, A. R. Chowdhury, D. Boneh, and K. Chaudhuri. Fairproof : Confidential and certifiable fairness for neural networks, 2024.

[36] C. Yadav, M. Moshkovitz, and K. Chaudhuri. Xaudit: A theoretical look at auditing with explanations. *arXiv preprint arXiv:2206.04740*, 2022.

[37] C. Yadav, R. Wu, and K. Chaudhuri. Influence-based attributions can be manipulated. *arXiv preprint arXiv:2409.05208*, 2024.

[38] T. Yan and C. Zhang. Active fairness auditing. In *International Conference on Machine Learning*, pages 24929–24962. PMLR, 2022.

[39] I.-C. Yeh. default of credit card clients. UCI Machine Learning Repository, 2016. DOI: https://doi.org/10.24432/C55S3H.

[40] Zcash Foundation. Halo2: A Plonkish zk-SNARK implemented in Rust, 2023. Accessed: 2025-01-27.

Appendix

1. Preliminaries

Cryptographic Primitives. We use two cryptographic primitives in this paper, commitment schemes and Zero-Knowledge Proofs.

Commitment Scheme [4] commits to private inputs w outputting a commitment string com_w. A commitment scheme is *hiding* meaning that com_w does not reveal anything about private input w and *binding* meaning that there cannot exist another input w' which has the commitment com_w, binding commitment com_w to input w.

Zero-Knowledge Proofs (ZKPs) [10, 11] involve a prover holding a private input w, and a verifier who both have access to a circuit P. ZKPs enable the prover to convince the verifier that, for some public input x, it holds a private witness w such that $P(x, w) = 1$ without revealing any additional information about witness w to the verifier. A ZKP protocol is (1) *complete*, meaning that for any inputs (x, w) where $P(x, w) = 1$, an honest prover will always be able to convince an honest verifier that $P(x, w) = 1$ by correctly following the protocol, (2) *sound*, meaning that beyond a negligible probability, a malicious prover cannot convince an honest verifier for any input x, that for some witness w, $P(x, w) = 1$ when infact such a witness w does not exist, even by arbitrarily deviating from the protocol, and (3) *zero-knowledge*, meaning that for any input x and witness w such that $P(x, w) = 1$, a malicious verifier cannot learn any additional information about witness w except that $P(x, w) = 1$ even when arbitrarily deviating from the protocol. A classic result says that any predicate P in the class NP can be verified using ZKPs [10].

LIME. Existing literature has put forward a wide variety of post-hoc (post-training) explainability techniques to make ML models transparent. In this paper, we focus on one of the popular ones, LIME (stands for Local Interpretable Model-Agnostic Explanations) [27].

LIME explains the prediction for an input point by approximating the local decision boundary around that point with a linear model. Formally, given an input point $x \in \mathcal{X}$, a complex non-interpretable classifier $f : \mathcal{X} \to \mathcal{Y}$ and an interpretable class of models $\mathcal{G}$, LIME explains the prediction $f(x) \in \mathcal{Y}$ with a local interpretable model $g \in \mathcal{G}$. The interpretable model g is found from the class $\mathcal{G}$ via learning, on a set of points randomly sampled around the input point and weighed according to their distance to the input point, as measured with a similarity kernel π. The similarity kernel creates a locality around the input by giving higher weights to samples near input x as compared to those far off. A natural and popular choice for the interpretable class of models $\mathcal{G}$ is linear models such that for any $g \in \mathcal{G}$, $g(z) = w_g \cdot z$ ([9, 27]), where w_g are the coefficients of linear model g. These coefficients highlight the contribution of each feature towards the prediction and therefore serve as the explanation in LIME. Learning the linear model is formulated as a weighted LASSO problem, since the sparsity induced by ℓ_1 regularization leads to more interpretable and human-understandable explanations. Following [27], the similarity kernel is set to be the exponential kernel with ℓ_2 norm as the distance function, $\pi_x(z) = \exp\left(-\ell_2(x, z)^2/\sigma^2\right)$ where σ is the bandwidth parameter of the kernel and controls the locality around input x.

For brevity, we will denote the coefficients corresponding to the linear model g as w instead of w_g, unless otherwise noted. For readers familiar with LIME, without loss of generality we consider the transformation of the points into an interpretable feature space to be identity in this paper for simplicity of exposition. The complete LIME algorithm with linear explanations is given in Alg. 1. We will also use 'explanations' to mean post-hoc explanations throughout the rest of the paper.

Algorithm 1 LIME [27]

Input: Input point x, Classifier f
Parameters: Number of points n to be sampled around input point, Length of explanation K, Bandwidth parameter σ for similarity kernel
Output: Explanation e

for $i \in \{1, 2, 3, \ldots, n\}$ **do**
    $z_i \leftarrow$ sample_around(x)
    $\pi_i \leftarrow \exp\left(-\ell_2(x, z_i)^2/\sigma^2\right)$
end for
$\hat{w} \in \arg\min_w \sum_{i=1}^{n} \pi_i \times \left(f\left(z_i\right) - w^\top z_i\right)^2 + \|w\|_1$
$e :=$ top-K$(\hat{w}, K)$ ▷ Sorts the weights according to absolute value & returns these along with corresponding features
Return Explanation e

2. Problem Setting & Desiderata for Solution

To recall, explanations fail as a trust-enhancing tool in adversarial use-cases and can lead to a false sense of security while benefiting adversaries. Motivated by these problems, we investigate if a technical solution can be designed to operationalize explanations in adversarial settings.

Algorithm 2 STANDARD_LIME_VARIANTS

Input: Input point x, Classifier f
Parameters: Number of points n to be sampled around input point, Length of explanation K, Bandwidth parameter σ for similarity kernel, sampling type $smpl\_type$, kernel type $krnl\_type$
Output: Explanation e

for $i \in \{1, 2, 3, \ldots, n\}$ **do**
  if $smpl\_type$==‘uniform’ **then**
    $z_i \leftarrow$ uniformly_sample_around(x)
  else if $smpl\_type$==‘gaussian’ **then**
    $z_i \leftarrow$ gaussian_sample_around(x)
  end if
  if $krnl\_type$==‘exponential’ **then**
    $\pi_i \leftarrow \exp\left(-\ell_2(x, z_i)^2/\sigma^2\right)$
  else
    $\pi_i = 1$
  end if
end for
$\hat{w} \in \arg\min_w \sum_{i=1}^{n} \pi_i \times \left(f\left(z_i\right) - w^\top z_i\right)^2 + \|w\|_1$
$e :=$ top-K$(\hat{w}, K)$
Return Explanation e

Formal Problem Setting. Formally, a model owner confidentially holds a classification model f which is not publicly released due to legal and IP reasons. A customer supplies an input x to the model owner, who responds with a prediction $f(x)$ and an explanation $\mathcal{E}(f, x)$ where $\mathcal{E}$ is the possibly-randomized algorithm generating the explanation.

Solution Desiderata. A technical solution to operationalize explanations in adversarial use-cases should provide the following guarantees.

1) (Model Uniformity) the same model f is used by the model owner for all inputs : our solution is to use cryptographic commitments which force the model owner to commit to a model prior to receiving inputs,
2) (Explanation Correctness) the explanation algorithm $\mathcal{E}$ is run correctly for generating explanations for all inputs : our solution is to use Zero-Knowledge Proofs, wherein the model owner supplies a cryptographic proof of correctness to be verified by the customer in a computationally feasible manner,
3) (Model Consistency) the same model f is used for inference and generating explanations : this is ensured by generating inference and explanations as a part of the same system and by using model commitments,
4) (Model Confidentiality) the model f is kept confidential in the sense that any technique for guaranteeing (1)-(3) does not leak anything else about the hidden model f than is already leaked by predictions $f(x)$ and explanations $\mathcal{E}(f, x)$ without using the technique : this comes as a by-product of using ZKPs and commitments (See Sec. F for the formal theorem and proof),
5) (Technique Reliability) the technique used for guaranteeing (1)-(4) is sound and complete (as in Sec.A): this comes as a by-product of using ZKPs and commitments (See Sec. F for the formal theorem and proof).

Our solution *ExpProof* which provides the above guarantees will be discussed in Sec. D.

3. Variants of LIME

Building zero-knowledge proofs of explanations requires the explanation algorithm to be implemented in a ZKP library[1] which is known to introduce a significant computational overhead. Given this, a natural question that comes to mind is if there exist variants of LIME which provide similar quality of explanations but are more ZKP-amenable by design, meaning they introduce a smaller ZKP overhead?

Standard LIME Variants. To create variants of standard LIME (Alg.1), we focus on the two steps which are carried out numerous times and hence create a computational bottleneck in the LIME algorithm – sampling around input x (Step 6 in Alg. 1) and computing distance using exponential kernel (Step 7 in Alg. 1). For sampling, we propose two options as found in the literature : gaussian (G) and uniform (U) [8, 9, 27]. For the kernel we propose to either use the exponential (E) kernel or no (N) kernel. These choices give rise to four variants of LIME, mentioned in Alg. 2. We address each variant by the intials in the brackets, for instance standard LIME with uniform sampling and no kernel is addressed as ‘LIME_U+N’.

1. More precisely, arithmetic circuits for the explanation algorithm are implemented in the ZKP library.

BorderLIME. An important consideration for generating meaningful local explanations is that the sampled neighborhood should contain points from different classes [22]. Any reasonable neighborhood for an input far off from the decision boundary will only contain samples from the same class, resulting in vacuous explanations.

To remedy the problem, [21, 22] propose a *radial* search algorithm, which finds the closest point to the input x belonging to a different class, x_{border}, and then uses x_{border} as the input to LIME (instead of original input x). Their algorithm incrementally grows (or shrinks) a search area radially from the input point and relies on random sampling within each 'ring' (or sphere), looking for points with an opposite label. To cryptographically prove this algorithm, we would either have to reimplement the algorithm as-is or would have to give a probabilistic security guarantee (using a concentration inequality), both of which would require many classifier calls and thereby many proofs of inference, becoming inefficient in a ZKP system.

We transform their algorithm into a line search version, called BorderLIME, given in Alg. 3 and 4, using the notion of Stability Radius which is now fed as a parameter to the algorithm. The stability radius for an input x, δ_x, is defined as the largest radius for which the model prediction remains unchanged within a ball of that radius around the input x. The stability radius δ is defined as the minimum stability radius across all inputs x sampled from the data distribution D. Formally, $\delta = \inf_{x \sim \mathcal{D}} \delta_x, \quad \text{where} \quad \delta_x = \sup\{r \geq 0 \mid f(x') = f(x), \forall x' \in \mathcal{B}(x, r)\}$. Here $\mathcal{B}(x, r) = \{x' \mid \|x' - x\| \leq r\}$ denotes a ball of radius r centered at x. Stability radius ensures that for any input from the data distribution, the model's prediction remains stable within at least a radius of δ.

Our algorithm samples m directions and then starting from the original input x, takes δ steps until it finds a point with a different label along all these directions individually. The border point x_{border} is that oppositely labeled point which is closest to the input x. Furthermore, unlike in the algorithm in [21], our algorithm can exploit parallelization by searching along the different directions in parallel since these are independent.

Determining the optimal value of the stability radius is an interesting research question, but it is not the focus of this work. We leave an in-depth exploration of this topic to future work while providing some high-level directions and suggestions next. Stability radius can (and perhaps should) be found *offline* using techniques as proposed in [14, 35] or through an offline empirical evaluation on in-distribution points. A ZK proof for this radius can be generated one-time, in an offline manner and supplied by the model developer (for NNs see [35]). It can also be pre-committed to by the model developer (see Sec. D).

Algorithm 3 BORDERLIME

Input: Input point x, Classifier f
Parameters: Number of points n to be sampled around input point, Length of explanation K, Bandwidth parameter σ for similarity kernel
Output: Explanation e

$x_{border} :=$
    FIND_CLOSEST_POINT_WITH_OPP_LABEL(x, f) ▷ See Alg. 4
$e :=$ LIME(x_{border}, f) ▷ Note that any variant of LIME can be used here
Return Explanation e

4. *ExpProof*: Verification of Explanations

Our system for operationalizing explanations in adversarial settings, *ExpProof*, consists of two phases: (1) a One-time Commitment phase and (2) an Online verification phase which should be executed for every input.

Commitment Phase. To ensure model uniformity, the model owner cryptographically commits to a fixed set of model weights $\mathbf{W}$ belonging to the original model f, resulting in committed weights $\mathsf{com}_{\mathbf{W}}$. Architecture of model f is assumed to be public. Additionally, model owner can also commit to the values of different parameters used in the explanation algorithm or these parameters can be public.

Online Verification Phase. This phase is executed every time a customer inputs a query. On receiving the query, the prover (bank) outputs a prediction, an explanation and a zero-knowledge proof of the explanation. Verifier (customer) validates the proof without looking at the model weights. If the proof passes verification, it means that the explanation is correctly computed with the committed model weights and explanation algorithm parameters.

To generate the explanation proof, a ZKP circuit which implements (a variant of) LIME is required. However since ZKPs can be computationally inefficient, instead of reimplementing the algorithm as-is in a ZKP library, we devise some smart strategies for verification, based on the fact that verification can be easier than redoing the computation. Since all the variants of LIME share some common functionalities, we next describe how the verification strategies for these functionalities. For more details on the verification for each variant, see Appendix Sec. E.2.

Algorithm 4 FIND_CLOSEST_POINT_WITH_OPP_LABEL

Input: Input point x, Classifier f
Parameters: Number of random directions m, Stability radius δ, Iteration Threshold T
Output: Opposite label point x_{border}

$\{\vec{u}_0, \vec{u}_1 \cdots \vec{u}_{m-1}\}$:= Sample m random directions
Initialize $dist_0 \cdots dist_{m-1}$ as inf
for $\vec{u}_i \in \{\vec{u}_0, \vec{u}_1 \cdots \vec{u}_{m-1}\}$ **do**
    $x_{border_i} := x$
    $iter := 0$
    while $f(x_{border_i}) == f(x)$ and $iter \leq T$ **do**
        $x_{border_i} := x_{border_i} + \delta \vec{u}_i$
        $iter := iter + 1$
    end while
    if $f(x_{border_i}) != f(x)$ **then**
        $dist_i := \ell_2(x, x_{border_i})$
    end if
end for
$x_{border} := x_{border_i}$ such that $i := \arg\min dist_i$
Return x_{border}

1. Verifying Sampling (Alg. 7, 12, 13). We use the Poseidon [12] hash function to generate random samples. As part of the setup phase, the prover commits to a random value r_p. When submitting an input for explanation, the verifier sends another random value r_v. Prover generates uniformly sampled points using Poseidon with a key $r_p + r_v$, which is uniformly random in the view of both the prover and the verifier. Then, during the proof generation phase, the prover proves that the sampled points are the correct outputs from Poseidon using *ezkl*'s inbuilt efficient Poseidon verification circuit. We convert the uniform samples into Gaussian samples using the inverse CDF, which is checked in the proof using a look-up table for the inverse CDF.

2. Verifying Exponential Kernel (Alg. 11). ZKP libraries do not support many non-linear functions such as exponential, which is used for the similarity kernel in LIME (Step 5 of Alg.1). To resolve this problem, we implement a look-up table for the exponential function and prove that the exponential value is correct by comparing it with the value from the look-up table.

3. Verifying Inference. Since LIME requires predictions for the sampled points in order to learn the linear explanation, we must verify that the predictions are correct. To generate proofs for correct predictions, we use *ezkl*'s inbuilt inference verification circuit.

4. Verifying LASSO Solution (Alg. 9). ZKP libraries only accept integers and hence all floating points have to be quantized. Consequently, the LASSO solution for Step 7 of Alg. 1 is also quantized in a ZKP library, leading to small scale differences between the exact and quantized solutions. To verify optimality of the quantized LASSO solution, we use the standard concept of duality gap. For a primal objective l and its dual objective g, to prove that the objective value from primal feasible w is close to that from the primal optimal w^*, that is $l(w) - l(w^*) \leq \epsilon$, the duality gap should be smaller than ϵ as well, $l(w) - g(u, v) \leq \epsilon$ where u, v are dual feasible. Since the primal and dual of LASSO have closed forms, as long we input any dual feasible values, we can verify that the quantized LASSO solution is close to the LASSO optimal. The prover provides the dual feasible as part of the witness to the proof. See App. Sec.G for closed forms of the primal and dual functions and for the technique to find dual feasible.

The complete *ExpProof* protocol can be found in Alg. 5; its security guarantee is given as follows.

Theorem A.1. *(Informal) Given a model f and an input point x, ExpProof returns prediction $f(x)$, LIME explanation $\mathcal{E}(f, x)$ and a ZK proof for the correct computation of the explanation, without leaking anything additional about the weights of model f (in the sense described in Sec.B).*

For the complete formal security theorem and proof, refer to App. Sec. F. The proof follows from inherent properties of ZKPs.

5. Experiments

Datasets & Models. We use three standard fairness benchmarks for experimentation : Adult [2], Credit [39] and German Credit [13]. Adult has 14 input features, Credit has 23 input features, and German has 20 input features. All the continuous features in the datasets are standardized. We train two kinds of models on these datasets, neural networks and random forests.

Our neural networks are 2-layer fully connected ReLU activated networks with 16 hidden units in each layer, trained using Stochastic Gradient Descent in PyTorch [25] with a learning rate of 0.001 for 400 epochs. The weights and biases are converted to fixed-point representation with four decimal places for making them compatible with ZKP libraries which do not work with floating points, leading to a $\sim 1\%$ test accuracy drop. Our random forests are trained using Scikit-Learn [26] with 5-6 decision trees in each forest.

ZKP Configuration. We code *ExpProof* with different variants of LIME in the *ezkl* library [19] (Version 18.1.1) which uses Halo2 [40] as its underlying proof system in the Rust programming language, resulting in $\sim 3.7k$ lines of code. Our ZKP experiments are run on an Ubuntu server with 64 CPUs of x86_64 architecture and 256 GB of memory, without any explicit parallelization. We use default configuration for *ezkl*, except for 200k rows for all lookup arguments in *ezkl* and *ExpProof*. We use KZG [17] commitments for our scheme that are built into *ezkl*.

Research Questions & Metrics. We ask the following questions for the different variants of LIME.

Q1) How faithful are the explanations generated by the LIME variant?

Q2) What is the time and memory overhead introduced by implementing the LIME variant in a ZKP library?

To answer Q1, we need a measure of fidelity of the explanation, we use 'Prediction Similarity' defined as the similarity of predictions between the explanation classifier and the original model in a local region around the input. We first sample points from a local[2] region around the input point, then classify these according to both the explanation classifier and the original model and report the fraction of matches between the two kinds of predictions as prediction similarity. In our experiments, the local region is created by sampling 1000 points from a Uniform distribution of half-edge length 0.2 or a Gaussian distribution centered at the input point with a standard deviation of 0.2.

To answer Q2, we will look at the proof generation time taken by the prover to generate the ZK proof, the verification time taken by the verifier to verify the proof and the proof length which measures the size of the generated proof.

5.1. Standard LIME Variants. In this section we compare the different variants of Standard LIME, given in Alg. 2 Sec. C, w.r.t. the fidelity of their explanations and ZKP overhead.

Setup. We use the LIME library for experimentation and run the different variants of LIME with number of neighboring samples $n = 300$ and length of explanation $K = 5$. Based on the sampling type, we either sample randomly from a hypercube with half-edge length as 0.2 or from a gaussian distribution centered around the input point with a standard deviation of 0.2. Based on the kernel type we either do not use a kernel or use the exponential kernel with a bandwidth parameter as $\sqrt{\#features} * 0.75$ (default value in the LIME library). Rest of the parameters also keep the default values of the LIME library. Our results are averaged over 50 different input points sampled randomly from the test set.

Results for NNs with 300 neighboring samples and Gaussian sampling for fidelity evaluation are described below. Results for uniform sampling fidelity evaluation, fidelity evaluation with neighborhood $n = 5000$ points and all results for RFs can be found in the Appendix Sec. G.

Fidelity Results. As shown in Fig. 4 left, we do not find a huge difference between the explanation fidelities of the different variants of LIME as the error bars significantly overlap. This could be due to the small size of the local neighborhoods where the kernel or sampling doesn't matter much. However, for the credit dataset, which has the highest number of input features, gaussian sampling works slightly better than uniform, which could be because of the worsening of uniform sampling with increasing dimension.

ZKP Overhead Results. Across the board, proof generation takes a maximum of ~ 1.5 minutes, verification time takes a maximum of ~ 0.12 seconds and proof size is a maximum of ~ 13KB, as shown in Fig. 5. Note that while proof generation time is on the order of minutes, verification time is on the order of seconds – this is due to the inherent design of ZKPs, requiring much lesser resources at the verifier's end (contrary to consistency-based explanation checks). We also observe that the dataset type does not have much influence on the ZKP overhead; this is due to same ZKP backend parameters needed across datasets.

Furthermore, we see that gaussian sampling leads to a larger ZKP overhead. This can be attributed to our implementation of gaussian sampling in the ZKP library, wherein we first create uniform samples and then transform them to gaussian samples using the inverse CDF method, leading to an additional step in the gaussian sampling ZKP circuit as compared to that of uniform sampling. Similarly, using the exponential kernel leads to a larger overhead over not using it due to additional steps related to verifying the kernel.

Overall, 'gaussian sampling and no kernel' variant of LIME is likely the most amenable for a practical ZKP system as it produces faithful explanations with a small overhead.

5.2. BorderLIME. In this section we compare the variants of BorderLIME (Alg. 3, Sec. C) and BorderLIME vs. Standard LIME w.r.t. the fidelity of their explanations and ZKP overheads.

Setup. We implement BorderLIME with all of the Standard LIME variants (Step 6 of Alg. 3). For the purpose of experimentation, we fix the iteration threshold to $T = 250$, number of directions to $m = 5$. Then to *approximate* the

2. Note that this local region is for evaluation and is different from the local neighborhood in LIME.

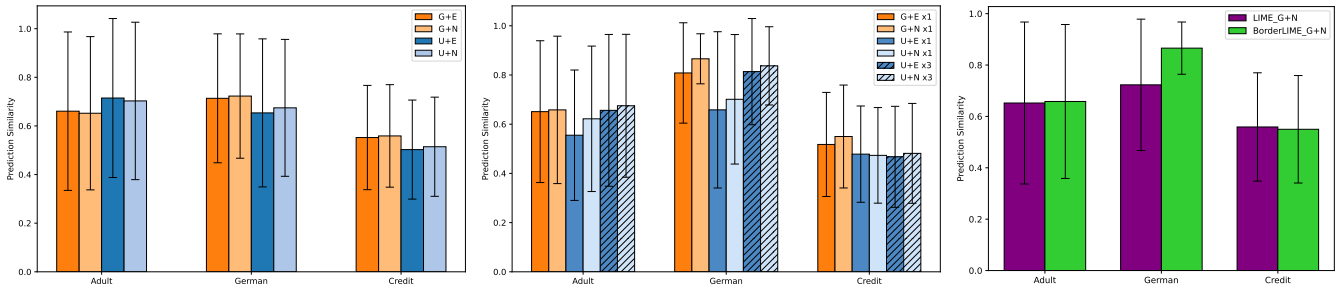

Figure 4. Results for NNs. G/U: gaussian or uniform sampling, E/N: using or not using the exponential kernel. Left: Fidelity of different variants of Standard LIME, Mid: Fidelity of different variants of BorderLIME , Right: Fidelity of Standard vs. BorderLIME.

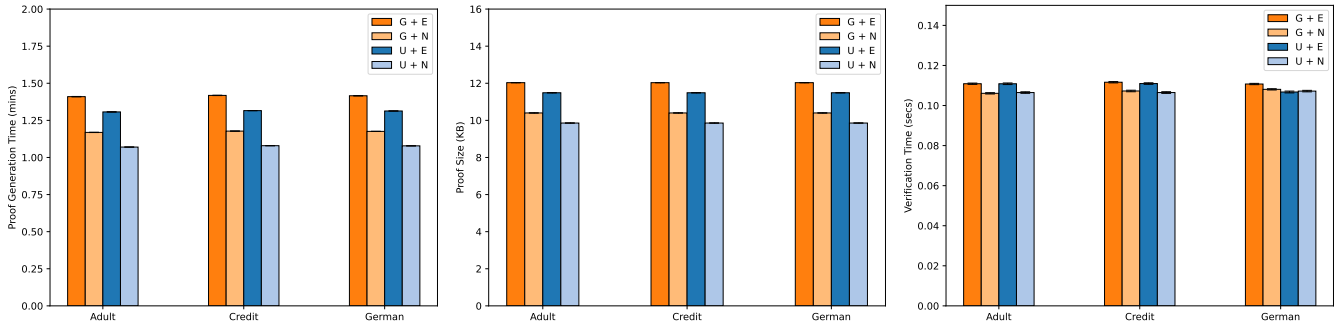

Figure 5. Results for NNs. G/U: gaussian or uniform sampling, E/N: using or not using the exponential kernel. Left: Proof Generation Time (in mins), Mid: Proof Size (in KBs), Right: Verification times (in secs) for different variants of Standard LIME. All configurations use the same number of Halo2 rows, 2^{18}, and lookup tables of size 200k.

stability radius δ, we incrementally go over the set $\{0.01, 0.03, 0.05, 0.07, 0.1, 0.15\}$ and use the smallest value for which an opposite class point is found for all 50 randomly sampled input points. While this is a heuristic approach and does not guarantee the theoretically minimal stability radius, it provides a practical estimate of stability radius for efficient experimentation. Once a suitable value of stability radius is identified, we tighten the number of directions by reducing them while ensuring that at least one opposite-class point exists for each input. Our results are averaged over 50 input points. The exact parameter values used in our final setup can be found in App. Sec. G.

Results for NNs with 300 neighboring samples and Gaussian sampling for fidelity evaluation are described below. Results for uniform sampling fidelity evaluation, fidelity evaluation with neighborhood $n = 5000$ points and all results for RFs can be found in the Appendix Sec. G.

Fidelity Results. Comparing different variants of BorderLIME based on the LIME implementation, we observe that the difference in explanation fidelity between gaussian and uniform sampling becomes more pronounced compared to standard LIME as shown in Fig. 4, reinforcing the importance of gaussian sampling. This gap can sometimes be reduced by using more neighborhood points, i.e. a larger n, when uniformly sampling. As demonstrated in Fig. 4 mid, with three times more points for uniform sampling, we can match the fidelity of gaussian explanations for Adult and German datasets. Comparing the G+N version of BorderLIME and standard LIME in Fig. 4 right, we observe that explanations generated by BorderLIME are atleast as faithful as standard LIME and can sometimes be better hinting to its capability of generating more meaningful explanations.

ZKP Overhead Results. We observe that BorderLIME has a larger ZKP overhead than standard LIME as shown in Table 1; this can be attributed to the additional steps needed in BorderLIME to find the border point with opposite label (Alg. 4) which also have to be proved and verified. Similar to the previous subsection, the overhead is similar across datasets and verification is orders of magnitude cheaper than proof generation.

While *ExpProof* guarantees model and parameter uniformity as well as correctness of explanations for a given model, it cannot prevent the kinds of manipulation where the model itself is corrupted – the model can be trained to create innocuous explanations while giving biased predictions. Here usually a regularization term corresponding to the manipulated explanations is added to the loss function [1, 37]. Preventing such attacks requires a ZK proof of training; this is well-studied in the literature but is outside the scope of this work and we refer an interested reader to [7].

Furthermore, to provide end-to-end trust guarantees for fully secure explanations, the explanations should be (1) faithful,

ZKP Overhead Type	BorderLIME	LIME
Proof Generation Time (mins)	4.85 ± 10^{-2}	1.17 ± 10^{-2}
Verification Time (secs)	0.30 ± 10^{-2}	0.11 ± 10^{-2}
Proof Size (KB)	18.30 ± 0	10.40 ± 0

TABLE 1. ZKP OVERHEAD OF BORDERLIME AND STANDARD LIME (BOTH G+N VARIANT) FOR NNS. OVERHEAD FOR BORDERLIME IS LARGER THAN THAT FOR LIME. RESULTS ARE CONSISTENT ACROSS ALL DATASETS.

stable and reliable, (2) robust to realistic adversarial attacks (such as the one mentioned above) and (3) should also be verifiable under confidentiality. This paper looks at the third condition by giving a protocol *ExpProof* and implementing it for verifiable explanations under confidentiality, which has not been studied prior to our work. As such, we view *our work as complementary and necessary* for end-to-end explanation trust guarantees.

Algorithm 5 *ExpProof*: Provable Explanation for Confidential Models

Public Configuration: ZK_LIME configuration $cc = (\ smpl\_type$ sampling type, LIME kernel variant $krnl\_type$, whether to use border LIME $border\_lime$, standard deviation σ, model architecture f, LIME ℓ_1 penalty α, maximum dual gap $\epsilon\ )$
Public Input: Input point x
Private Witness: Model weights $\mathbf{W}$
Output: Output label o, Explanation e, Proof π that $o = f(x)$ and that e is a valid LIME explanation.
Pre-Processing Offline Phase
Sample randomness $r \leftarrow \mathbb{F}$
Commit to the randomness $\mathsf{com}_{\mathbf{r}}$ and release it publicly
Commit to the model weights $\mathsf{com}_{\mathbf{W}}$ and release it publicly
Online Phase
$o = f(\mathbf{W}, x)$
Compute $h_i = \text{Poseidon}(k, i)$
Compute LIME perturbations z from samples s
$y = f(\mathbf{W}, z)$
$(e, \hat{w}) \leftarrow \text{LIME}(f, y, z)$ ▷ Compute a LIME solution using perturbations z with labels y
Compute a feasible dual solution $\hat{v}$ from $\hat{w}$
$\Pi \leftarrow \text{zkLime}(cc, x, o, r_v, C_r, C_{\mathbf{W}}, e; \mathbf{W}, y, h, \hat{w}, \hat{v})$
return (o, e, Π)

Algorithm 6 ZK_LIME

Public Configuration: $smpl\_type$ sampling type, LIME kernel variant $krnl\_type$, whether to use border LIME $border\_lime$, standard deviation σ, model architecture f, LIME ℓ_1 penalty α, maximum dual gap ϵ, sampling bit-width b

Public Instance: input point x, model output o, randomness r_v, commitment to the randomness C_r, commitment to the weights $C_{\mathbf{W}}$, e top-k LIME features

Private Witness: Model weights $\mathbf{W}$, labels of the LIME samples y, hash outputs h, LIME model $\hat{w}$, LIME dual $\hat{v}$

Output: ZK Proof of the computation Π

Check that $C_r = \text{Com}(r_p)$

Check that $C_W = \text{Com}(\mathbf{W})$

$k \leftarrow r_p + r_v$

ZK_CHECK_POSEIDON(h, k) ▷ Check that h is generated from Poseidon using key k

EZKL.CHECK_INFERENCE$(\text{f}, \mathbf{W}, \text{x}, \text{o})$ ▷ Check that $o = f(W, x)$ using EZKL

if $smpl\_type$==‘uniform’ **then**

  $s \leftarrow$ZK_UNIFORM_SAMPLE(h, b) ▷ Check that s is uniform generated from the hashes h

else if $smpl\_type$==‘gaussian’ **then**

  $s \leftarrow$ZK_GAUSSIAN_SAMPLE(h, b) ▷ Check that s is Gaussian generated from the hashes h

else

  return $\perp$

end if

if $border\_lime$ == true **then**

  $x \leftarrow$ZK_FIND_OPP_POINT$(x, s, num_vectors, vector_length)$

  $s \leftarrow s[m \times d...]$ ▷ Skip samples used for opposite point for fresh randomness

end if

for $i \in |z|$ **do**

  $j \leftarrow i \mod |x|$

  $z \leftarrow x_j + s_i - 2^{b-1}$ ▷ Perturb x with samples s

end for

if $krnl\_type$==‘exponential’ **then**

  $\pi \leftarrow$ZK_EXPONENTIAL_KERNEL(x, z, σ, π)

else

  $\pi \leftarrow 1$

end if

for $i \in \{1, 2, 3, \ldots, n\}$ **do**

  EZKL.CHECK_INFERENCE$(\text{f}, \mathbf{W}, \text{z}_\text{i}, \text{y}_\text{i})$ ▷ Check that $y_i = f(W, z_i)$ using EZKL

end for

ZK_LASSO$(z, y, \pi, \hat{w}, \hat{v}, \alpha)$

$e =$ZK_TOP_K$(\hat{w})$

Generate proof Π of the above computation

Algorithm 7 ZK_CHECK_POSEIDON

Input: hashes h, key k

Output: True if each h_i generated from Poseidon with key k and input i

for $h_i \in h$ **do**

  EZKL.CHECK_POSEIDON$(\text{h}_\text{i}, \text{k}, \text{i})$ ▷ Check that $h_i = \text{Poseidon}(k, i)$ using EZKL

end for

return x_{border}

Algorithm 8 ZK_FIND_OPP_POINT

Input: input x, samples s, number of vectors $num\_vectors$, maximum length of each vector $vector\_length$
Output: Border point x_{border} if one exists, otherwise x
$d \leftarrow |x|$
$step \leftarrow z[0..d \times m].reshape(m, d)$ ▷ Get $m \times d$ samples as m randomly sampled vectors
for $i \in \{1, 2, 3, ..., num\_vectors\}$ **do**
    $step_i \leftarrow step_i \times$ LOOKUP_RECIPROCAL_SQRT$(step_i \cdot step_i)$ ▷ Normalize each step vector using a lookup table for $1/\sqrt{\|(step_i)\|_2}$
end for
for $i \in \{1, 2, 3, ..., num\_vectors\}$ **do**
    for $i \in \{1, 2, ..., vector\_length$ **do**
        $v_i \leftarrow i \times step\_size \times step_i$
    end for
end for
$y = f(W, v)$
$x_{border} \leftarrow x$
for $i \in \{vector\_length, vector\_length - 1, ..., 2, 1\}$ **do**
    for $i \in \{1, 2, ..., num\_vectors\}$ **do**
        if $y_i \neq x\_label$ **then**
            $x_{border} \leftarrow v_i$
        end if
    end for
end for
return x_{border}

Algorithm 9 ZK_LASSO

Input: Samples z, labels y, weights π, Lasso solution $\hat{w}$, Lasso dual solution $\hat{v}$, Lasso parameter α, maximum dual gap ϵ
Output: True if the dual solution is feasible and the dual gap is less than ϵ, and False otherwise.
for $i \in \{1, 2, 3, \ldots, n\}$ **do**
    $z'_i \leftarrow \sqrt{\pi_i} \times z_i$
    $y'_i \leftarrow \sqrt{\pi_i} \times y_i$
end for
$p \leftarrow \frac{1}{2n}\|y' - b - w^T z'_i\|^2 + \alpha\|w\|_1$
$d \leftarrow \frac{-n}{2}\|v\|^2 + v^T(y' - b)$
Check $p - d \leq \epsilon$
Let m be the length of each sample z_i
for $i \in \{1, 2, 3, \ldots, m\}$ **do**
    $f_i \leftarrow (X^T)_i v$
    Check $-\alpha \leq f_i \leq \alpha$
end for

Algorithm 10 ZK_TOP_K

Input: Lasso solution $\hat{w}$, top-k values e
Output: True if e contains the top-k values of $\hat{w}$, and False otherwise
$\hat{w}' = \text{Sort}(\hat{w})$
for $i \in \{1, 2, 3, \ldots, k\}$ **do**
    $(v, j) \leftarrow e_i$
    Check $\hat{w}'_i = v$
    Check $\hat{w}_j = v$
end for

Algorithm 11 ZK_EXPONENTIAL_KERNEL

Input: Input point x, LIME samples z, standard deviation σ
Output:
for $i \in \{1, 2, 3, \ldots, N\}$ **do**
    square_distance = $x \cdot z_i$
    $\pi_i \leftarrow$ LOOKUP_EXPONENTIAL$(-\text{square_distance}/\sigma^2)$ ▷ Check exponential function using a lookup table
end for
return π

Algorithm 12 ZK_UNIFORM_SAMPLE

Input: Poseidon hashes h, sampling bit-width b
Output: Uniform samples z
$B = \lfloor W/b \rfloor$
$N = \lceil (|x| * n)/B \rceil$
$j = 0$
for $i \in \{1, 2, 3, \ldots, N\}$ **do**
    Compute z such that $z_j + z_{j+1}2^b + \ldots + z_{j+B}2^{B*b} + rem = h_i$
    Check that $z_j + z_{j+1}2^b + \ldots + z_{j+B}2^{B*b} + rem = h_i$ ▷ Check that the samples are a decomposition of h_i
    for $k \in \{1, 2, 3, \ldots, B\}$ **do**
        Check that $0 \leq z_{i+k} < 2^B$
    end for
    Check that $0 \leq rem < 2^B$
    $j \leftarrow j + B$
end for
return z

6. Security of *ExpProof*

Completeness. $\forall$ ZK_LIME configurations cc, input points x, and model weights $\mathbf{W}$

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \textit{ExpProof}.\mathsf{Setup}(1^k) \\ (pk, vk) \leftarrow \textit{ExpProof}.\mathsf{KeyGen}(pp) \\ (\mathsf{com}_{\mathbf{W}}, \mathsf{com}_r) \leftarrow \textit{ExpProof}.\mathsf{Commit}(\mathsf{pp}, \mathbf{W}, r) \\ (o, e, \pi) \leftarrow \textit{ExpProof}.\mathsf{Prove}(\mathsf{pp}, pk, cc, x, \mathsf{com}_{\mathbf{W}}, \mathbf{W}, \mathsf{com}_r, r_p, r_v) \\ \textit{ExpProof}.\mathsf{Verify}(\mathsf{pp}, vk, cc, x, o, e, \mathsf{com}_{\mathbf{W}}, \mathsf{com}_r, \pi) = 1 \end{array}\right] = 1.$$

Proof Sketch. **Completeness.** The completeness proof mostly follows from the completeness of the underlying proof system (in our case, Halo2). We must also show that for any set of parameters there exists a LIME solution $\hat{w}$ and a feasible dual solution v such that the dual gap between $\hat{w}$ and $\hat{v}$ is less than ϵ. We know from the strong duality of Lasso that there exists a solution $w*$ and $v*$ such that the dual gap is 0 for any input points and labels, therefore such a solution exists. However, we also note that the circuit operates on fixed-point, discrete values (not real numbers), and it is not necessarily true that there are valid solutions in fixed-point. To solve this, the prover can use a larger number of fractional bits until the approximation is precise enough. □

Algorithm 13 ZK_GAUSSIAN_SAMPLE

Input: Poseidon hashes h, sampling bit-width b
Output: Gaussian samples z
$z =$ZK_UNIFORM_SAMPLE(h, b)
$z =$ LOOKUP_GAUSSIAN_INVERSE_CDF(z)
return z

Knowledge-Soundness. We define the relation $\mathcal{R}_{lime}$ as:

$$\mathcal{R}_{lime} = \left\{ (cc, x, o, e, r_v, \mathsf{com}_{\mathbf{W}}, \mathsf{com}_r; \mathbf{W}, r_p, y, h, \hat{w}, \hat{v}) \;\middle|\; \begin{array}{c} \mathsf{com}_{\mathbf{W}} = Com(\mathbf{W}) \\ \mathsf{com}_{\mathbf{r}} = Com(\mathbf{r_p}) \\ o \leftarrow cc.f(\mathbf{W}, x) \\ h_i = \text{Poseidon}(r_p + r_v, i) \\ z \leftarrow \text{SAMPLE\_AROUND}(x, cc.\sigma) \\ y \leftarrow cc.f(\mathbf{W}, z) \\ \pi \leftarrow \text{LIME\_KERNEL}(x) \\ y = cc.f(\mathbf{W}, z) \\ z' \leftarrow \sqrt{\pi} \times z \\ y' \leftarrow \sqrt{\pi} \times y \\ p \leftarrow \frac{1}{2n}\|y' - b - \hat{w}^T z'\|^2 + cc.\alpha\|\hat{w}\|_1 \\ d \leftarrow \frac{-n}{2}\|\hat{v}\|^2 + \hat{v}^T(y' - b) \\ p - d \leq cc.\epsilon \\ f \leftarrow (X^T)v \\ -cc.\alpha \leq f \leq cc.\alpha \end{array} \right\}$$

There exists an extractor $\mathcal{E}$ such that for all probabilistic polynomial time provers $\mathcal{P}*$

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \textit{ExpProof}.\mathsf{Setup}(1^k) \\ (pk, vk) \leftarrow \textit{ExpProof}.\mathsf{KeyGen}(pp) \\ (cc, x, o, e, r_v, \mathsf{com}_{\mathbf{W}}, \mathsf{com}_r, \pi) \leftarrow \mathcal{P}(1^\lambda, \mathsf{pk}) \\ \textit{ExpProof}.\mathsf{Verify}(pp, vk, cc, x, o, e, r_v, \mathsf{com}_{\mathbf{W}}, \mathsf{com}_r) = 1 \\ (\mathbf{W}, r_p, y, h, \hat{w}, \hat{v}) \leftarrow \mathcal{E}^P(...) \\ (cc, x, o, e, r_v, \mathsf{com}_{\mathbf{W}}, \mathsf{com}_r; \mathbf{W}, r_p, y, h, \hat{w}, \hat{v}) \notin \mathcal{R}_{lime} \end{array}\right] \leq negl(\lambda).$$

Proof Sketch. **Knowledge Soundness.** Knowledge-soundness follows directly from the knowledge-soundness of the underlying proof system Halo2. The extractor runs the Halo2 extractor and outputs the Halo2 witness. By the construction of the circuit ZK_LIME, the extracted Halo2 witness satisfies the $\mathcal{R}_{lime}$ relation. □

Zero-Knowledge. We say a protocol Π is *zero-knolwedge* if there exists a polynomial time, randomized simulator $\mathcal{S}$ such that for all $(pk, vk) = \mathsf{Setup}(pp)$, for all $(x, w) \in \mathcal{R}$, for all verifiers V

$$\{P(pk, x, w)\} \approx \{S(pk, x)\}$$

Proof Sketch. **Zero-Knowledge.** Let the simulator $\mathcal{S}$ be the Halo2 simulator. For any $(cc, x, o, e, r_v, \mathsf{com}_{\mathbf{W}}, \mathsf{com}_r, \mathbf{W}, r_p, y, h, \hat{w}, \hat{v}) \in \mathcal{R}_{lime}$, we know that

$$\{\textit{ExpProof}.Prove(cc, x, o, e, r_v; \mathbf{W}, r_p, y, h, \hat{w}, \hat{v})\} \approx \{\mathcal{S}(cc, x, o, e, r_v)\}$$

by zero-knowledge of Halo2. □

7. LASSO Primal and Dual

Notation: Let $X \in \mathbf{R}^{n \times m}$ denote the data inputs, $y \in \mathbf{R}^n$ denote the labels and $\alpha > 0$ denote the regularization parameter or the LASSO constant. Let $w \in \mathbf{R}^m$ denote the primal variable and $v \in \mathbf{R}^n$ denote the dual variable.

The primal LASSO objective is given as, $\frac{1}{2n}\|Xw - y\|_2^2 + \alpha\|w\|_1$ while the dual objective function is given as $-\frac{n}{2}\|v\|_2^2 - v^\top y$ with the feasibility constraint $0 \leq L_\infty\left(X^\top v\right) \leq \alpha$ [18].

From a LASSO primal feasible w, it is possible to compute a dual feasible v as [18]:

$v = 2s(Xw - y)$

$s = \min\left\{\alpha / \mid 2\left(\left(W^T W x\right)_i - 2y_i\right)\mid \; | i = 1, \ldots, n\right\}$

We find that this dual is close enough to the dual optimal to get a good duality gap, however, in the worst case it is possible to apply traditional optimization methods to find a dual feasible with a smaller duality gap.

8. NN results

Next in Fig.6 we show results for using uniform sampling in the 'prediction similarity' evaluation, keeping rest of the parameters same. We observe similar numbers as before with very slight differences from Fig.4.

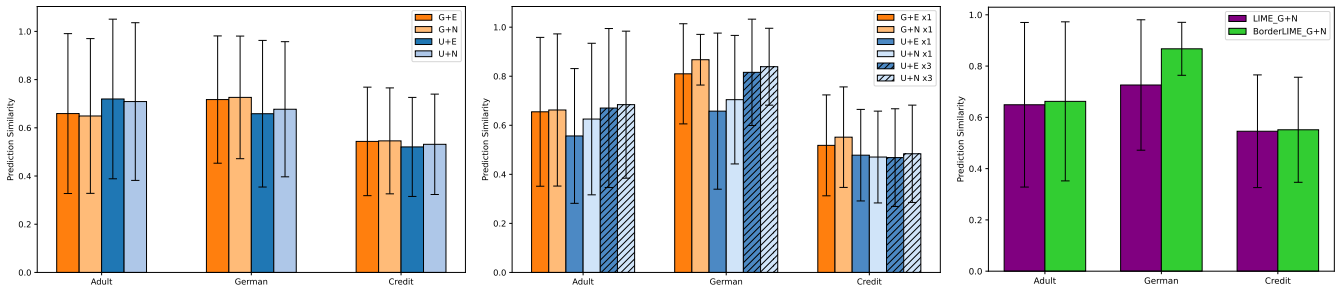

Figure 6. Results for NNs for $n = 300$ neighboring points and uniform sampling in the evaluation. Left: Fidelity of different variants of Standard LIME, Mid: Fidelity of different variants of BorderLIME, Right: Fidelity of Standard vs. BorderLIME.

Next we increase the neighborhood size, n, in LIME from 300 to 5000 samples and present the results in Fig.7. As expected the fidelity increases across the board due to better model fitting with a larger number of points. The size of the error bars only reduces significantly for the German dataset, showing that for more input points the explanations are faithful to the original decision boundaries. Additionally, the German dataset also has the highest fidelity explanations. Both these points hints towards smoother or more well-behaved decision boundaries learnt using the German dataset. Furthermore, BorderLIME consistently outperforms standard LIME pointing to better explanations.

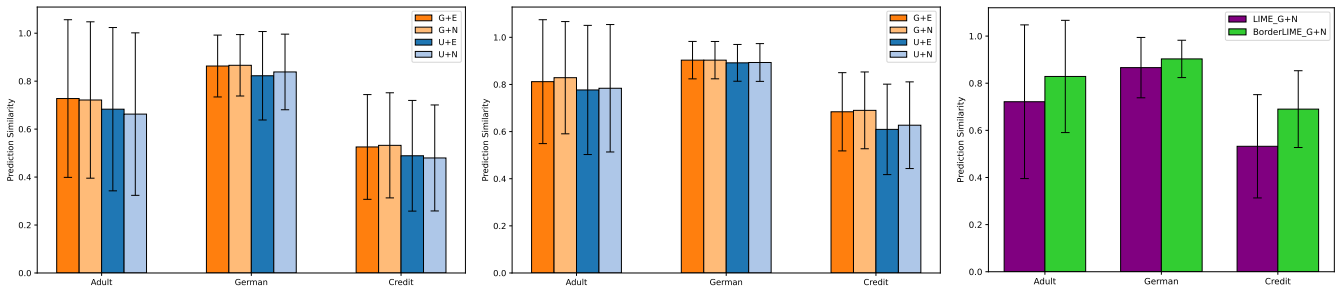

Figure 7. Results for NNs for $n = 5000$ neighboring points and gaussian sampling in the evaluation. Left: Fidelity of different variants of Standard LIME, Mid: Fidelity of different variants of BorderLIME , Right: Fidelity of Standard vs. BorderLIME.

9. RF results

Next in Fig.8 and Fig.9 we show results for fidelity of explanations for Random Forests using gaussian and uniform sampling in the 'prediction similarity' evaluation respectively, keeping rest of the parameters same. We observe results as for NNs.

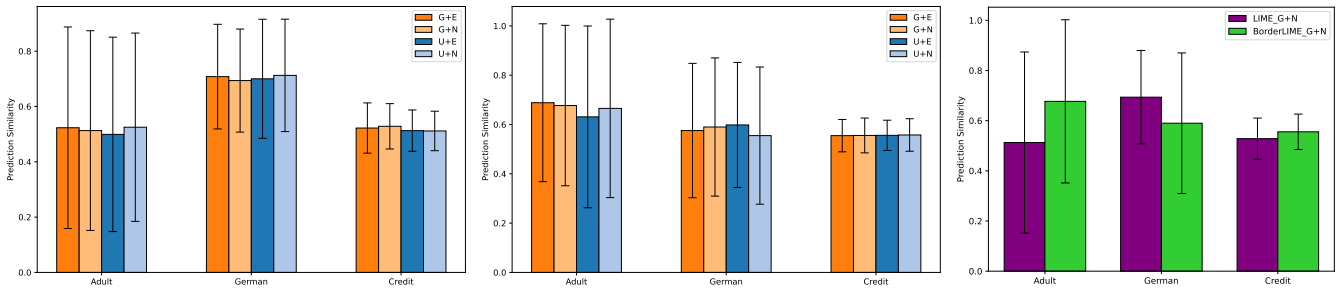

Figure 8. Results for RFs for $n = 300$ neighboring points and gaussian sampling in the evaluation. Left: Fidelity of different variants of Standard LIME, Mid: Fidelity of different variants of BorderLIME , Right: Fidelity of Standard vs. BorderLIME.

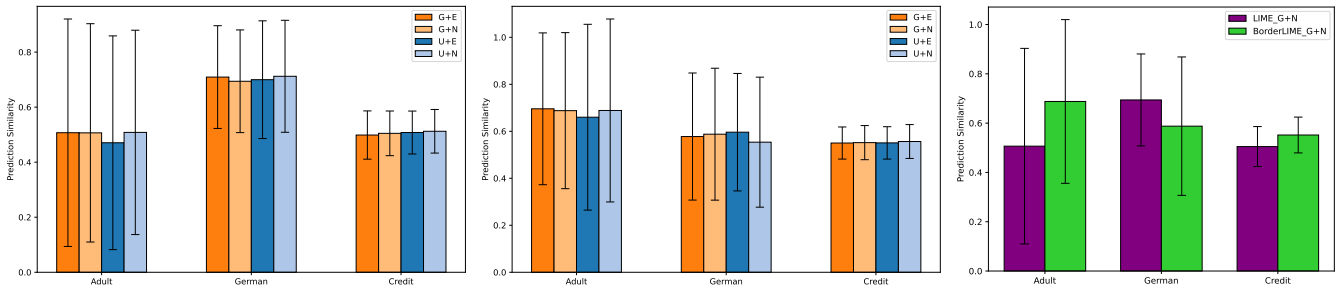

Figure 9. Results for RFs for $n = 300$ neighboring points and uniform sampling in the evaluation. Left: Fidelity of different variants of Standard LIME, Mid: Fidelity of different variants of BorderLIME , Right: Fidelity of Standard vs. BorderLIME.

Next we show the ZKP overheads for RFs in Fig.10 and Table 2. Trends and observations are similar as for NNs.

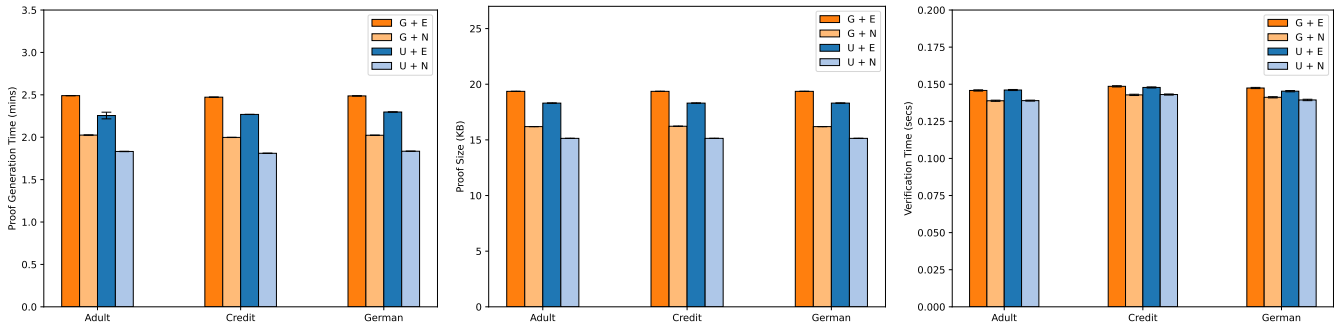

Figure 10. Results for RFs for $n = 300$ neighboring points. Left: Proof Generation Time (in mins), Mid: Proof Size (in KBs), Right: Verification times (in secs) for different variants of Standard LIME. All configurations use the same number of Halo2 rows, 2^{18}, and lookup tables of size 200k.

ZKP Overhead Type	BorderLIME	LIME
Proof Generation Time (mins)	8.46 ± 10^{-1}	2.02 ± 10^{-2}
Verification Time (secs)	0.44 ± 10^{-2}	0.14 ± 10^{-3}
Proof Size (KB)	17.25 ± 0	16.20 ± 10^{-2}

TABLE 2. ZKP OVERHEAD OF BORDERLIME AND STANDARD LIME (BOTH G+N VARIANT) FOR RFS FOR 300 NEIGHBORING POINTS. OVERHEAD FOR BORDERLIME IS LARGER THAN THAT FOR LIME. RESULTS ARE CONSISTENT ACROSS ALL DATASETS.

Next we increase the neighborhood size, n, in LIME from 300 to 5000 samples and present the results in Fig.11. As expected the fidelity increases across the board due to better model fitting with a larger number of points. We see slight reduction in error bars for German dataset. BorderLIME equals or outperforms standard LIME.

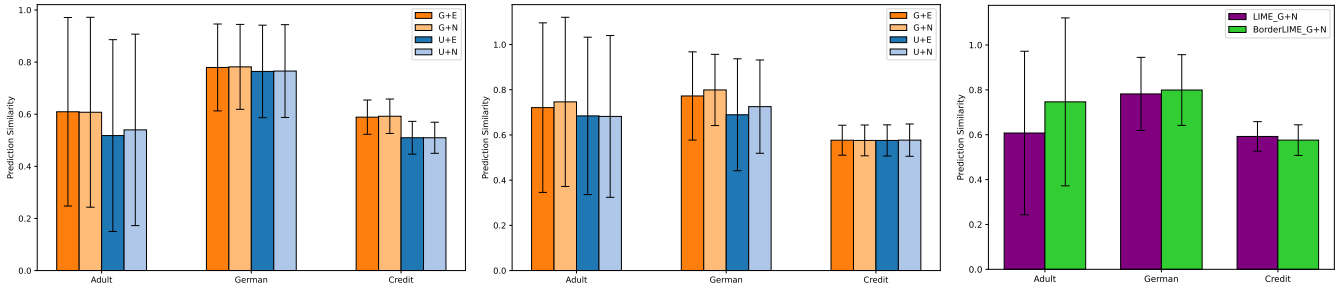

Figure 11. Results for RFs for neighboring points $n = 5000$ and gaussian sampling in the evaluation. Left: Fidelity of different variants of Standard LIME, Mid: Fidelity of different variants of BorderLIME , Right: Fidelity of Standard vs. BorderLIME.