Computer Science, Data Science &
Computer Systems Engineering

Capstone Report - Fall 2024

Benchmarking ZK Virtual Machines for Privacy-Preserving Machine Learning Applications

Lawrence Lim
Siddhartha Tuladhar
Brandon Gao

supervised by
Promethee Spathis

Preface

As a team comprising a Computer Systems Engineering major, a Computer Science major, and a Data Science major, we bring diverse perspectives and expertise to address the complex challenges at the intersection of privacy, security, and scalability in technology. This project was inspired by the increasing importance of privacy-preserving computation, particularly in sensitive fields like finance, where secure data handling is paramount. Our collective academic backgrounds have allowed us to explore innovative approaches to these challenges, drawing from distributed systems, cryptography, and data analytics.

Our target audience includes researchers, developers, and industry professionals who are advancing privacy technologies, blockchain systems, and secure data frameworks. By benchmarking zero-knowledge virtual machines (zkVMs) in the context of financial data, this project seeks to provide valuable insights into their capabilities and limitations, contributing to the ongoing development of secure and privacy-centric computational tools.

Acknowledgements

We sincerely thank our advisor, Professor Promethee Spathis, for their guidance and support throughout this project. We are also grateful to the Professor Benedikt Bunz for providing the initial ideation for this project. Lastly, we are grateful to our families and friends for their encouragement and support.

Abstract

This work addresses the challenge of securely processing sensitive data in privacy-critical applications like finance. Zero-knowledge virtual machines (zkVMs) offer a promising solution, but face issues with complexity and proof generation time. We benchmark three zkVMs—SP1, Jolt, and RISC-0—by training a ridge regression model on financial data, evaluating their performance and identifying key bottlenecks. Our findings highlight zkVMs' potential for privacy-preserving computation and provide insights for improving their practical adoption.

Keywords

Capstone; Computer science; Machine Learning; Zero-Knowledge Proofs, Zero-Knowledge Virtual Machines, Jolt, SP1, Risc0, NYU Shanghai

Contents

1 Introduction

1.1 Context

Currently, there exists a large amount of sensitive customer data that is extremely valuable but not being monetized to its fullest extent due to a combination of compliance and ethical concerns. An essential category of this data is personal financial data. Due to the sensitive nature of this data and strict compliance laws, current Fintech platforms require users' explicit permission to access sensitive information like credit history, transaction history, and income. However, **Zero Knowledge Proof (ZKP)** can be utilized to develop services and algorithms that utilize the sensitive data without explicitly revealing it. A ZKP is a cryptographic method of proving a statement is true without revealing any other information besides the fact that the statement is true. ZKPs have three fundamental characteristics:

- **Completeness:** If a statement is true, an honest prover can prove to an honest verifier that they have knowledge of the correct input.
- **Soundness:** If a statement is false, a dishonest prover is unable to convince an honest verifier that they have knowledge of the correct input.
- **Zero-knowledge:** No other information about the input is revealed to the verifier from the prover besides the fact that the statement is true.

Currently, the most-friendly way of generating a ZKP is through the use of zero-knowledge virtual machines (zkVMs). A zkVM, is simply a VM implemented as a circuit for a ZKP system. So, instead of proving the execution of a program, as one would normally do in ZKP systems, you prove the execution of the bytecode of a given Instruction Set Architecture (ISA). However, the zkVM landscape is in early development, and proof generation is bottlenecked by the complexity of the program and hardware limitations. In this paper, we provide a quantatitve and qualitative analysis on the current state of zkVMs on a real-world use case by generating a proof of a machine learning (ML) algorithm on dummy financial data.

1.2 Objective

The objective of this paper is to provide researchers, developers, and industry professionals a comprehensive review on utilizing zkVMs in a real-world use case. In order to achieve this, we benchmark SP1[1], Risc0[2], and Jolt[3], since they all compile to RISC-V, which is an extremely

popular compile target for most programming languages (Rust, C++, LLVM). We provide a quantative analysis on proof generation time and proof verification time. Additionally, we provide a qualitative analysis on the developer experience on developing ML algorithms for these zkVMs. The ML algorithm that we use is the ridge regression model.

2 Related Work

2.1 Virtual Machine-based ZK Systems

Our research explores the most efficient methods for generating ZKPs for machine learning algorithms by experimenting with different zkVMs. The current papers on zkVMs focus on their architecture and the techniques that are used by the zkVM to achieve optimized proof and verification times. For example, Arun et al.[3] present Jolt, a zero-knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) system optimized for VMs, which uses a new lookup technique to reduce proof size and verification time. Their paper focuses on this optimization that makes Jolt effective in environments where complex computations occur, and proof generation must remain efficient. Similarly, Bruestle and Gafni[2] introduce RISC-0, a system designed to produce scalable, transparent proofs for computations based on the RISC-V instruction set. RISC-0 focuses on architecture-specific applications, such as the RISC-V ecosystem, which contrasts with Jolt's broader application to virtual machines. Currently, there is no literature on the direct comparison between these zkVMs, which is the objective of our paper. Additionally, there is no research or comparisons on SP1.

2.2 Machine Learning in ZKPs

Machine learning is a highly relevant use case for ZKPs. There are multiple articles on the various use cases of ML in ZKPs. Wang et al.[4] developed an efficient ZKP-based pipeline for classical machine learning inference, focusing on privacy-preserving inference that ensures model accuracy without exposing sensitive model information. This system optimizes zero-knowledge inference, making it highly applicable to real-world ML applications where data privacy is paramount. Our paper uses the same inference technique within the zkVM as a real world application. Similarly, Sathe et al.[5] provide a comprehensive survey of ZKP applications in machine learning, covering a range of cryptographic techniques and their applicability to neural networks, decision trees, and other ML models. Their survey offers a broader comparison of ZKP techniques in the ML

space, providing essential context for more specific systems like that of Wang et al. However, their paper does not provide details on the performance or real-world use case. Lastly, Ganescu and Passerat-Palmbach[6] extend the application of ZKPs to generative AI models, by proposing a system that enhances trust in AI by allowing for the verification of generative models without exposing underlying data, which is critical in AI-driven environments where sensitive inputs and outputs must be safeguarded while still proving the model's validity. This paper is similar to Wang et al., where it provides a real-world use case, but it utilizes generating ZKPs through tools that are inaccessible to developers and require high development overhead. By using zkVMs, most of the ZKP logic is abstracted away, which provides developers easier access to converting their programs into ZKPs.

3 Solution

The methodology for achieving our objective is structured into two key phases. The first phase involves developing and training a machine learning algorithm on a given dataset to attain satisfactory accuracy levels. The second phase entails exporting the trained model and test data into Rust, where the inference process is meticulously implemented by manually translating the logic from Python to Rust.

3.1 ML Setup

In order to simulate real-world transaction data, dummy transaction data was taken from a Kaggle dataset[7]. The dataset was processed to extract key features such as recency, frequency, monetary value (RFM), and spending activity, which is depicted in Table 1. RFM serves as a foundation for predictive modeling, where these metrics can be combined with other features to enhance the accuracy of customer spending predictions.

For the machine learning algorithm, we used the Ridge Regression Model due to the model's accuracy and the complexity constraints of the zkVMs. Ridge Regression is an extension of linear regression by adding a penalty term to the loss function to prevent overfitting. Ridge Regression prevents overfitting by penalizing large coefficient values, thereby shrinking them towards zero. The Ridge Regression loss function is formulated as:

$$\text{Loss} = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{n} w_j^2$$

Field	Description
CustomerID	Unique identifier for a customer, used to track and differentiate individuals in the dataset.
Frequency	Number of transactions a customer made during the first three quarters.
Monetary	Total monetary value of a customer's transactions during the first three quarters.
Recency	Number of days since the customer's last transaction during the first three quarters.
Price	Price per unit for each transaction item.
DiscountApplied	Percentage of discount applied to the transaction.
spend_90_flag	Binary flag indicating whether the customer made any transaction in the past 90 days.
actual_spend_90_days	Total amount spent by the customer in the last 90 days(quarter).

Table 1: Description of fields in the customer transaction dataset.

where m is the number of samples, n is the number of features, $\hat{y}_i$ represents the predicted value for the i-th sample, λ is the regularization parameter controlling the strength of the penalty, and w_j are the model coefficients. The second term $\lambda \sum_{j=1}^{n} w_j^2$ is the regularization term, which discourages large coefficients and helps reduce the model's variance. The accuracy of the model is shown in Table 2.

Metric	Value
R^2(R-Squared)	0.8
Mean Square Error (MSE)	21

Table 2: Model performance metrics.

The R^2 value measures the proportion of variance in the target variable that is explained by the independent variables in the model. It ranges from 0 to 1, where higher values indicate a better fit of the model to the data.

The MSE represents the average squared difference between the predicted and actual values. It provides a measure of the model's prediction error, with lower values indicating better performance.

3.2 zkVM Implementation

The zkVM implementation for all three included exporting the model (as a JSON file) and inference dataset. The inference dataset was parsed,

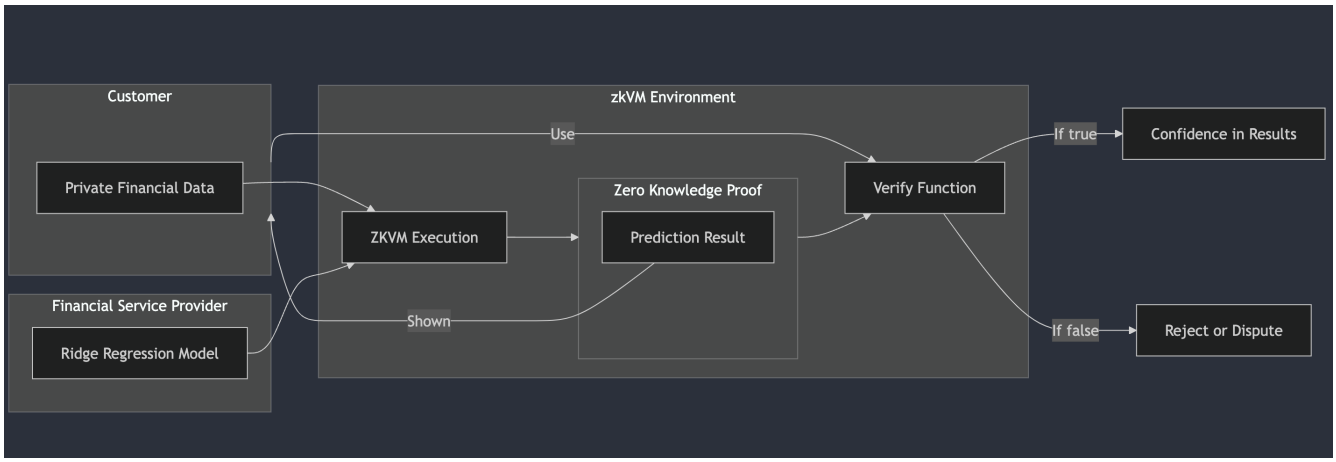

Figure 1: Architecture of our benchmark procedure

4 Results

The results section details your metrics and experiments for the assessment of your solution. It then provides experimental validation for your approach with visual aids such as data tables and graphs. In particular, it allows you to compare your idea with other approaches you've tested, for example solutions you've mentioned in your related work section.

4.1 ARM Benchmark

For example, Figure 2 compares the efficiency of three different service architectures in eliminating adversarial behaviors. Every data point gives the probability that k faulty/malicious nodes managed to participate in a computation that involves 32 nodes. In the absence of at least one reliable node ($k = 32$), the failure will go undetected ; but the results show that this case is extremely unlikely, regardless of the architecture. The most significant result pertains to $k = 16$: the reliable nodes detect the failure, but cannot reach a majority to recover. The graph shows that the `CORPS 5%` architecture is much more resilient than the `DHT 30%` architecture, by a magnitude of 10^{11}.

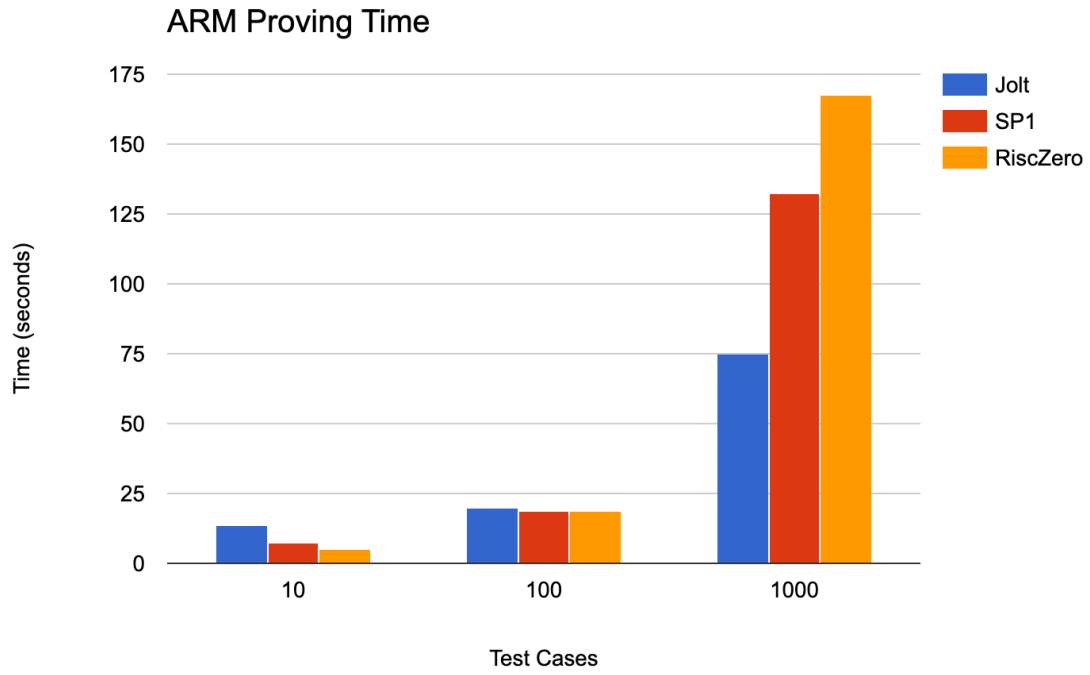

Figure 2: ARM Proving Time

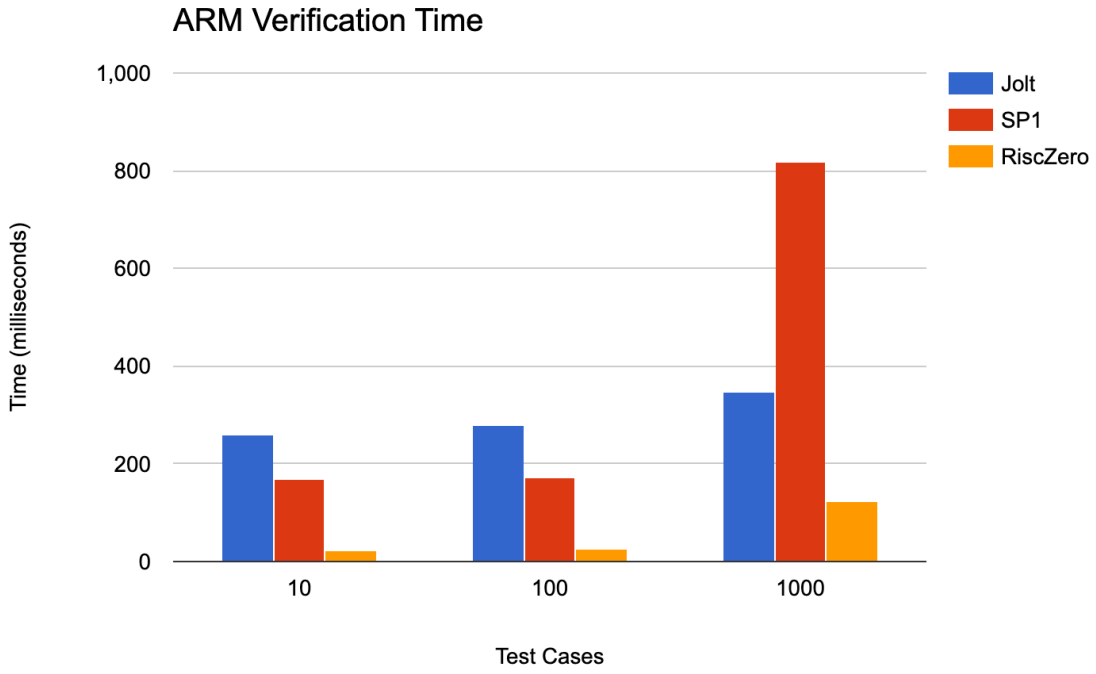

Figure 3: ARM Verification Time

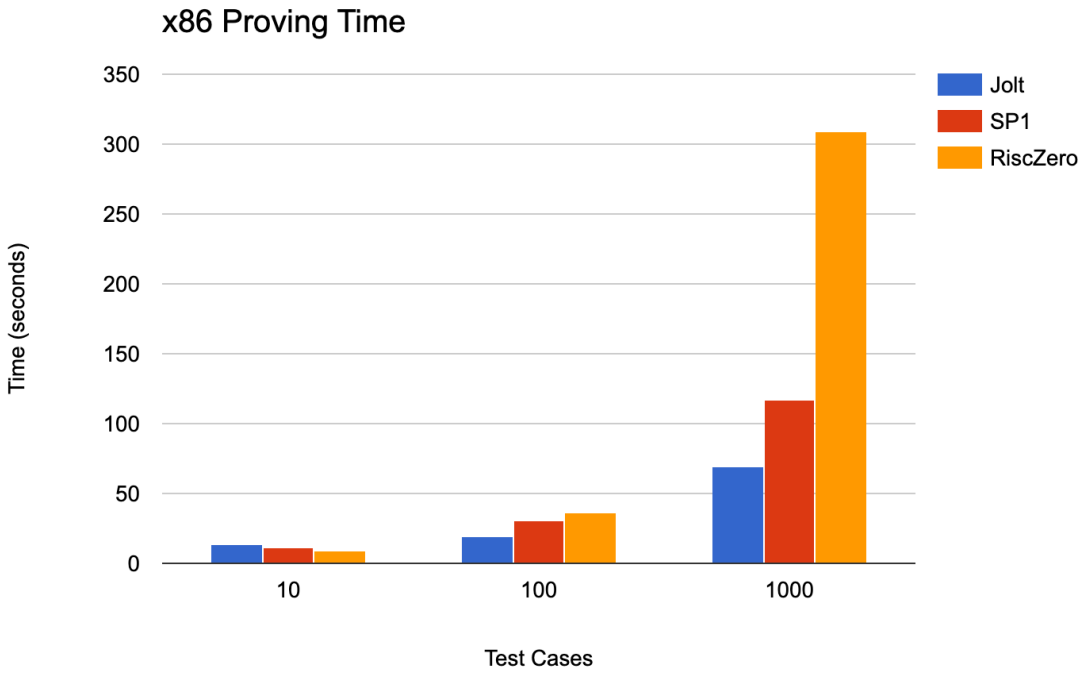

Figure 4: x86 Proving Time

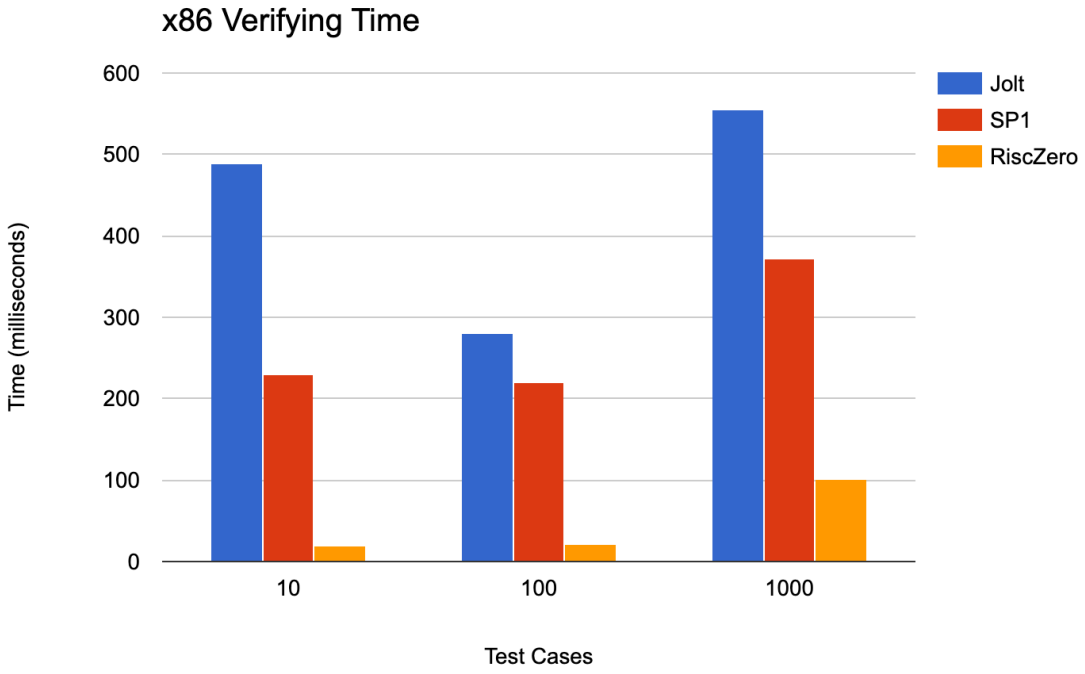

Figure 5: x86 Verification Time

4.2 x86 Benchmark

4.3 Performance Metrics

Analysis

The R^2 value of 0.8 demonstrates that the model explains a substantial portion of the variance in the target variable, effectively capturing the relationships in the data. The MSE value of 21 is acceptable because it aligns well with the scale of the target variable.

5 Discussion

The discussion section focuses on the main challenges/issues you had to overcome during the project. Outline what your approach does better than the ones you mentioned in your related work, and explain why. Do the same with issues where other solutions outperform your own. Are there limitations to your approach? If so, what would you recommend towards removing/mitigating them? Given the experience you've gathered working on this project, are there other approaches that you feel are worth exploring?

5.1 Limitations of the Approach

One major limitation of the current implementation is its reliance on Rust without the Standard Library (stdlib). While this choice allows for reduced build and runtime, it significantly limits the ability to implement more complex machine learning models. This low-level implementation, while efficient for simpler models like Ridge Regression, imposes constraints on usability and feature expansion. For instance, without stdlib, basic functionalities like I/O and access to additional data structures are unavailable, reducing the flexibility of the implementation.

5.2 Recommendations for Mitigation

To overcome these limitations, it is recommended to:

- Explore ways to selectively incorporate necessary parts of stdlib without significantly impacting performance.
- Investigate low-level optimization techniques to support complex models while maintaining efficiency.

- Utilize external libraries or frameworks designed for Rust to enhance capabilities without reintroducing excessive overhead.

5.3 Next Steps

Based on the current implementation and findings, the following steps are proposed for future work:

- Perform benchmarking without precompiles to evaluate raw performance.
- Compare CPU vs. GPU performance to understand hardware utilization.
- Conduct memory usage benchmarking to optimize resource management.
- Carry out a qualitative analysis of the trade-offs between low-level implementation and stdlib inclusion.

These steps will provide deeper insights into the strengths and limitations of the approach and inform potential improvements in future iterations.

6 Personal Contributions

6.1 Siddhartha Tuladhar

6.2 Lawrence Lim

6.3 Brandon Gao

7 Conclusion

Give a clear, short, and informative summary of all your important results. Answer the initial question(s) or respond to what you wanted to do, as stated in your introduction. It can be a short table or a list, and possibly one or two short comments or explanations.

Target a reader who may not have time to read the whole report yet, but needs the results or the conclusions immediately. This is a typical situation in real life. Some readers will read your introduction and skip to your conclusion first, and read the whole report only later (if at all).

You may also draw perspectives. What's missing? In what directions could your work be extended?

References

[1] U. Roy. Introducing sp1: A performant, 100% open-source, contributor-friendly zkvm. [Online]. Available: https://blog.succinct.xyz/introducing-sp1/

[2] J. Bruestle and P. Gafni, "Risc zero zkvm: Scalable, transparent arguments of RISC-V integrity," RiscZero Team, Tech. Rep., 2023. [Online]. Available: https://www.risczero.com/proof-system-in-detail.pdf

[3] A. Arun, S. Setty, and J. Thaler, "Jolt: Snarks for virtual machines via lookups," in *Advances in Cryptology – EUROCRYPT 2024*. Springer Nature Switzerland, 2024, pp. 3–33.

[4] H. Wang, R. Bie, and T. Hoang, "An efficient and zero-knowledge classical machine learning inference pipeline," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, 2024.

[5] A. Sathe, V. Saxena, P. A. Bharadwaj, and S. Sandosh, "State of the art in zero-knowledge machine learning: A comprehensive survey," in *Advancements in Smart Computing and Information Security*. Springer Nature Switzerland, 2024, pp. 98–110.

[6] B.-M. Ganescu and J. Passerat-Palmbach, "Trust the process: Zero-knowledge machine learning to enhance trust in generative AI interactions," *arXiv*, 2024. [Online]. Available: https://arxiv.org/

[7] F. Rehman, "Retail transaction dataset," https://www.kaggle.com/datasets/fahadrehman07/retail-transaction-dataset, 2023, accessed: 2023-12-03.