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 **5**

1.1 Context 5

1.2 Objective 5

2 Related Work **6**

2.1 Zero Knowledge Virtual Machines 6

2.2 Machine Learning in ZKPs 6

3 Solution **7**

3.1 Machine Learning Setup 7

3.2 zkVM Implementation 9

4 Quantatitve Results **10**

4.1 ARM Benchmark 10

4.2 x86 Benchmark 12

4.3 Analysis on Benchmark Results 13

5 Qualitative Results **14**

5.1 Background 14

5.2 Comparison 15

6 Discussion **16**

6.1 Contributions 16

6.2 Limitations & Next Steps 17

7 Conclusion **17**

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 comparable analysis of zkVMs in real-world use cases. To achieve this, we benchmark SP1[1], Risc0[2], and Jolt[3]. They all compile to RISC-V, which is an extremely popular

compile target for common 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 zkVM developer experience and how their performance trade-offs can impact real-world applications.

2 Related Work

2.1 Zero Knowledge Virtual Machines

Our research compares 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. The first zkVM was presented by Bruestle and Gafni[2], who introduced RISC-0, a system designed to produce scalable, transparent proofs for computations based on the RISC-V instruction set. RISC-0 also optimized heavily for proof performance on GPUs. Following RISC-0's success, Uma Roy[1] introduced SP1, a performant zkVM similar to RISC-0, but with slight modifications to the SNARK proof system and the extensive use of hand optimized precompiles that make certain operations (hash functions, signatures, elliptic curve arithmetic, etc.) quicker to prove within the zkVM. It's worth noting, that SP1 does not have a paper. In mid 2024, Arun et al.[3] presented Jolt, a zero-knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) system optimized for VMs, which uses a new lookup technique that makes Jolt faster at generating proofs while maintaining a quick verification time.

2.2 Machine Learning in ZKPs

Machine learning is a highly relevant use case for ZKPs. There are multiple papers 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. Their system optimizes zero-knowledge inference, making it highly applicable to real-world ML applications where data privacy is paramount. Our work also uses inference in a similar 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 focuses on hand-written ZK circuits which requires a high degree of zero-knowledge cryptography experience to write. This is inaccessible to developers and requires orders of magnitude more high development overhead compared to convensional programming techniques. By using zkVMs, most of the ZKP logic is abstracted away, allowing developers to focus on the application itself and privacy-preserving logic (if applicable).

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 required us to export the trained model and test data into Rust, where the inference process is implemented by manually translating the logic from Python to Rust. Finally, we set different test input sizes and benchmarked the proving and verification of the algorithm in the three zkVMs and on different machines.

3.1 Machine Learning Setup

In order to simulate real-world transaction data, historical transaction data was taken from a Kaggle dataset[7]. The dataset comprised of customer transactions, with each record containing information on the transaction date, product price, quantity purchased, and customer ID. The first step in the data preparation process involved determining a cutoff date that would separate the historical data used for training from the future data we wish to predict. We defined this cutoff date as 90 days before the latest transaction in the dataset. This allowed us to segment the data into two parts:

- **Training Data (First Three Quarters):** Transactions occurring before the cutoff date, which we used to derive customer behavior features (recency, frequency, and monetary

values).

- **Target Data (Fourth Quarter):** Transactions occurring after the cutoff date, which were used to calculate the target variable, customer spending in the last quarter (`actual_spend_90_days`).

We aimed to predict customer spending for the next 90-day period (i.e., the last quarter) based on historical transaction data. We used these defined fiscal quarters to organize the data, aggregating customer spending within each quarter to track how spending evolves over time. The ultimate goal was to use the data from Q1, Q2, and Q3 to predict customer spending in Q4.

Next, 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.

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.

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:

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

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 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 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. The MSE value of 21 is acceptable because it aligns well with the scale of the target variable.

3.2 zkVM Implementation

Figure 1 illustrates the high-level architecture of the zkVM implementation. In this framework, the ML model and inference data serve as inputs to each of the three zkVMs. Using the provided ML model, the zkVMs predict a specific customer's transaction amount and generate a ZKP that ensures the integrity of the inputs and outputs. Users can access the output by referencing the corresponding ZKP object. To validate the correctness of the output, each zkVM incorporates a verification function that enables users to confirm whether the output was accurately derived from the ML algorithm.

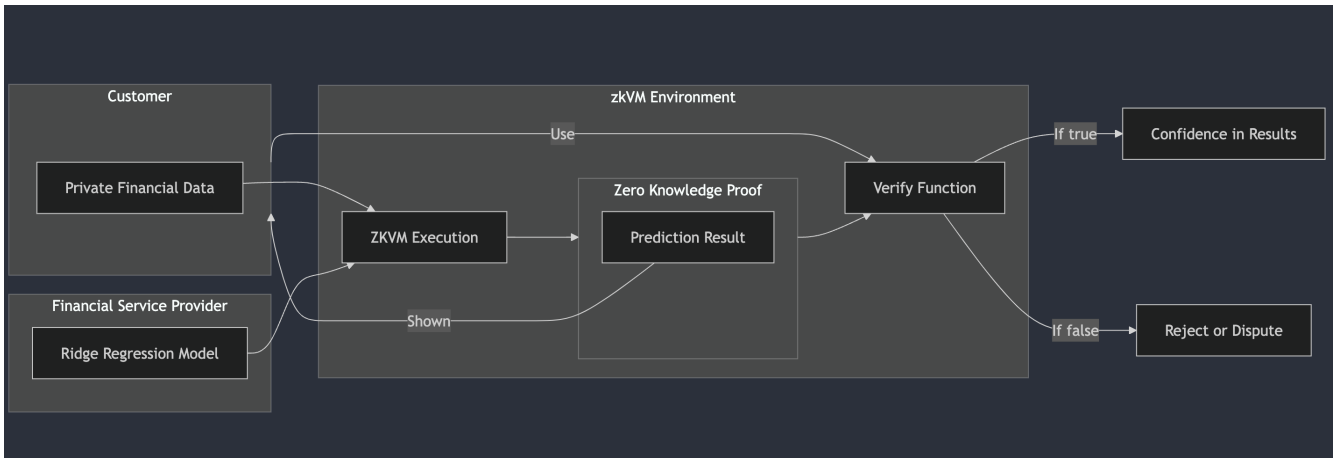

Figure 1: Architecture of our benchmark procedure

The benchmarking process for all three zkVMs involved measuring the time required for proof generation and verification. Timing was recorded by initiating a timer at the start and end of each process. This evaluation was conducted across varying numbers of inference test cases and was performed on both ARM-based and x86-based computing platforms to compare performance across architectures.

4 Quantatitve 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.

The results are categorized based on the architecture of the machines utilized to run the zkVMs.

4.1 ARM Benchmark

The ARM benchmarks were conducted on an M1 Max Apple Mini, a RISC-based system equipped with 10 CPU cores and 64GB of RAM. Figure 2 presents a comparative analysis of the time required to generate the ML proof. The results indicate that SP1 was the fastest for 10 test cases, completing the proof generation in 7.46 seconds. For 1000 test cases, Jolt demonstrated the best performance, achieving proof generation in 15.10 seconds. Conversely, Jolt exhibited the longest runtime for 10 test cases at 13.57 seconds, while Risc0 recorded the slowest performance for 1000 test cases, taking 132.13 seconds to generate the proof.

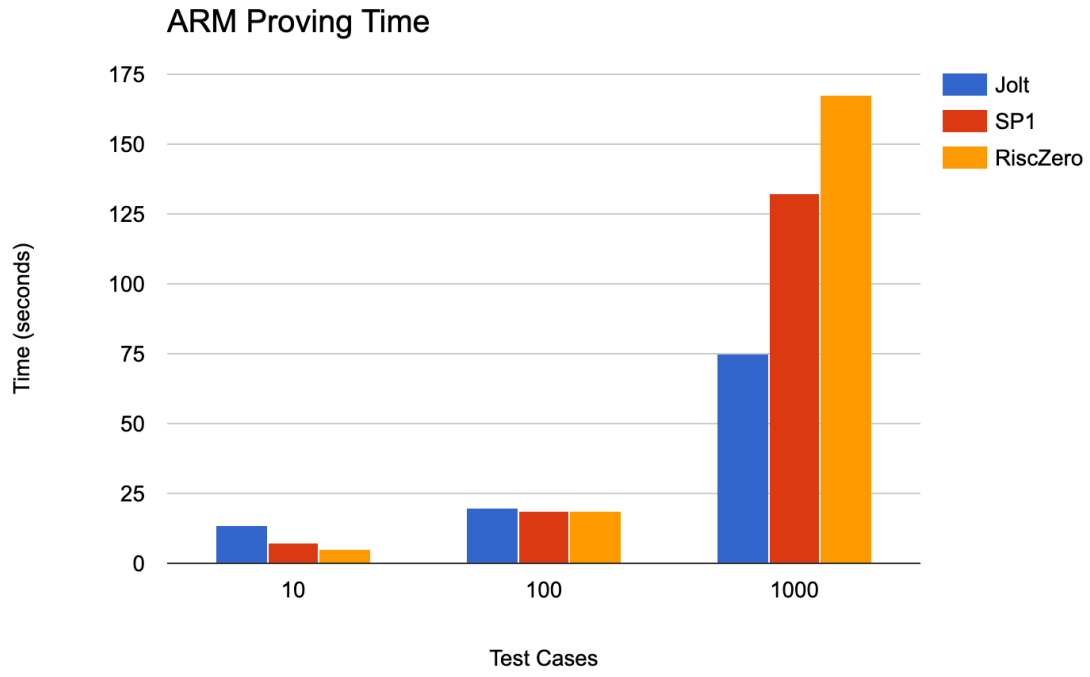

Figure 2: Proof Generation Benchmark for Jolt, SP1, and RISC-0 on ARM chip

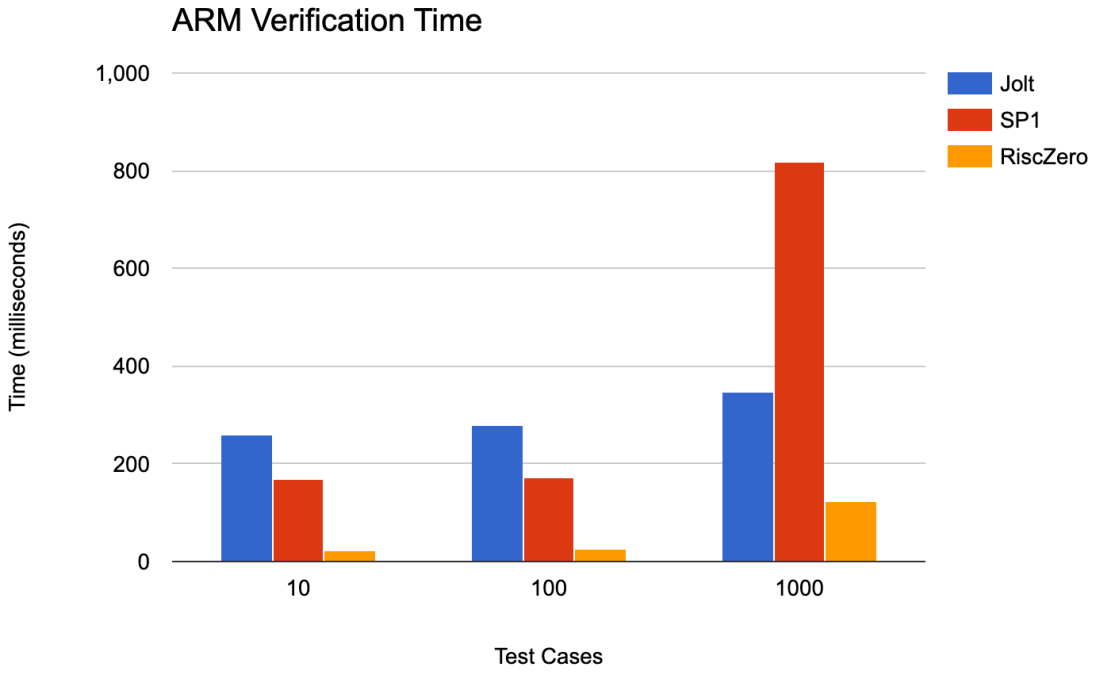

Figure 3: Proof Verification Benchmark for Jolt, SP1, and RISC-0 on ARM chip

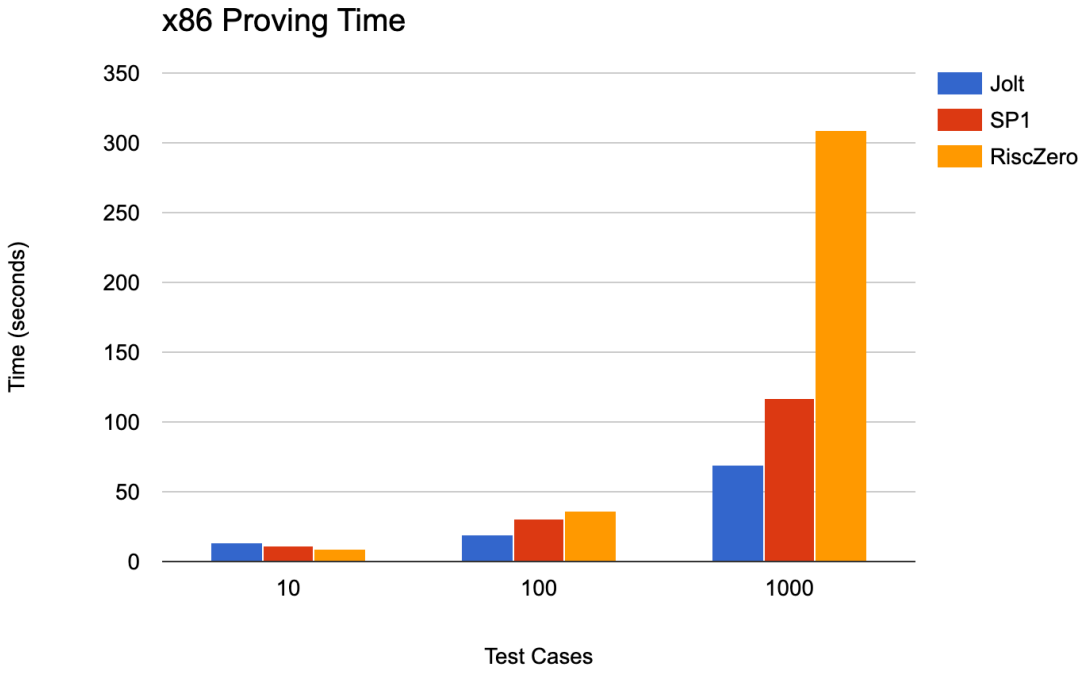

Figure 4: Proof Generation Benchmark for Jolt, SP1, and RISC-0 on x86 chip

Proof generation is typically efficient, and due to the property of succinctness inherent in ZKPs, verification in theory is expected to operate in O(1) time complexity. Figure 3 showcases the proof verification benchmark on ARM architecture. The slowest performance for 10 test cases was observed with Jolt, taking 259.08 ms, while Risc0 achieved the fastest verification time at 22.73 ms. At 1000 test cases, Risc0 maintained its superior performance with a verification time of 124.53 ms, whereas SP1 exhibited the slowest verification time, significantly lagging at 818.74 ms.

4.2 x86 Benchmark

The x86 benchmarks were performed on an Intel Cascade Lake Platinum 8268 processor, a CISC-based architecture featuring 18 CPU cores and 180GB of RAM. Figure 4 illustrates a comparison of the time required to generate machine learning proofs on the x86 architecture. At 10 test cases, proof generation was the slowest on Jolt, taking 13.62 seconds, while Risc0 demonstrated the fastest performance at 9.62 seconds. For 1000 test cases, Jolt exhibited significantly improved performance, completing proof generation in 69.36 seconds, whereas Risc0 required 309.14 seconds, making it over four times slower.

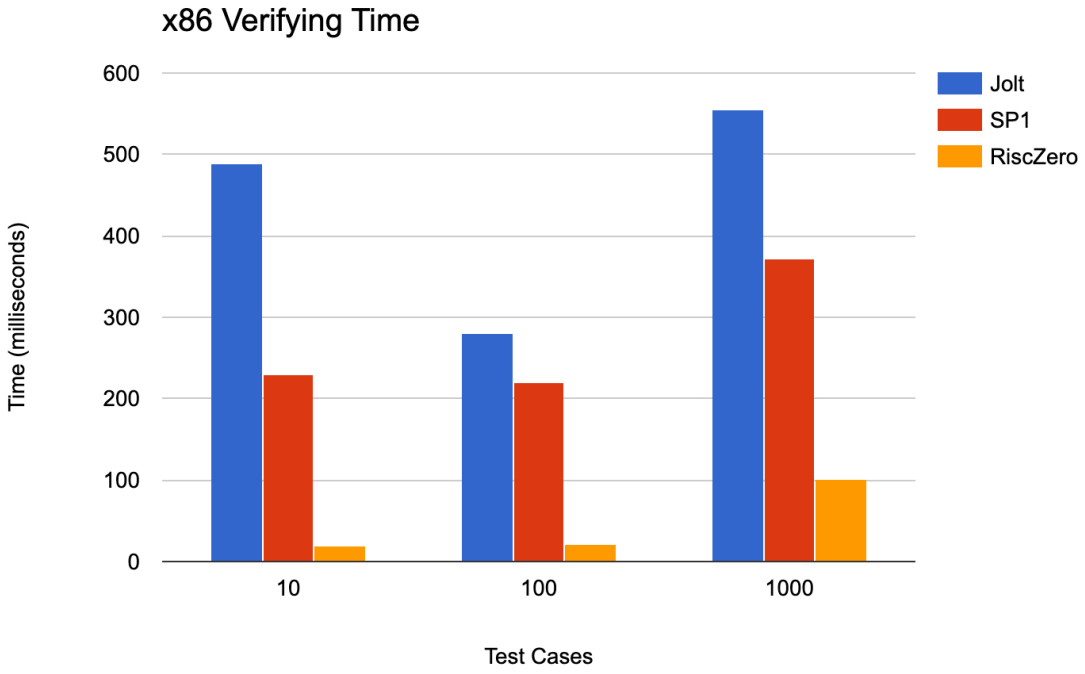

Figure 5: Proof Verification Benchmark for Jolt, SP1, and RISC-0 on x86 chip

The benchmark results for proof verification on the x86 architecture are presented in Figure 5. In this evaluation, Jolt demonstrated the slowest performance across all test cases, requiring 489.86 ms for 10 test cases and 555.56 ms for 1000 test cases. Conversely, Risc0 achieved the fastest verification times, completing the process in 19.86 ms for 10 test cases and 101.21 ms for 1000 test cases.

4.3 Analysis on Benchmark Results

Based on the benchmark results, Jolt demonstrates itself to be the most efficient zkVM in terms of proof generation across both ARM and x86 architectures. Its consistent performance highlights its suitability for real-world applications requiring rapid proof generation. While Risc0 is recognized as the first production-ready zkVM, it exhibits the poorest performance for larger test cases, particularly in proof generation. Notably, Risc0 achieves nearly twice the performance on the ARM architecture compared to x86, likely due to its architectural alignment with ARM's RISC-based design. SP1 consistently performs between Jolt and Risc0 across both architectures, offering predictable and reliable results, with minimal variation between platforms.

Verification time does not present a significant bottleneck in real-world applications, as it is typ-

ically less critical than proof generation. Interestingly, the verification benchmarks yield almost inverse results compared to proof generation. Jolt consistently exhibits the slowest verification times across both architectures and all test cases. In contrast, Risc0 leads in verification efficiency, maintaining the fastest times regardless of architecture or test size. SP1 again performs between Jolt and Risc0, but with an anomaly in the ARM architecture at 1000 test cases, where its verification time was more than twice as slow as Jolt and nearly eight times slower than Risc0. This unexpected result may indicate a scalability issue specific to SP1 under high verification loads.

5 Qualitative Results

5.1 Background

The section will compare the underlying proof systems of Jolt, SP1, and RISC-0. Before diving into the comparison, we'll briefly discuss the design of zkVMs. We break it up into the frontend and the backend.

5.1.1 Frontend

First, we'll compare the arithmetization schemes of the zkVMs. This is the process of converting VM instructions to circuit format, necessary for ZKPs. In general, arithmetization cannot be done manually except for elementary programs. Besides, the use of naïve arithmetization can lead to significant overhead. To deal with this, dedicated compilers accepting high-level programming languages have been developed.

Next, we'll discuss the use of precompiles. Compared to running a program without any proving overhead, zkVMs are incredibly slow. This poor performance is why all deployed zkVMs today use "precompiles," or hand-optimized protocols for specific computations that come up over and over again, like SHA2 or Keccak evaluations. But over-reliance on precompiles is dangerous: designing pre-complies is exactly the error-prone and time-intensive process that zkVMs are meant to obviate.

5.1.2 Backend

The backend for proof systems is composed of two components: a Polynomial Commitment Scheme (PCS) and a Polynomial Interactive Oracle Proof (PIOP) made non-interactive through

the Fiat-Shamir Transform. Noteably, certain PCS's use a trusted setup whereas others like FRI use an untrusted setup. Something notable about the interaction between the frontend and backend: most SNARKs can be easily tweaked to support both Plonkish (a common SNARK standard) and AIR with the exception of Groth16 which can only support R1CS.

There are additional tradeoffs to consider here such as field size and and FRI expansion rate (or blowup factor) for FRI based SNARKs. Some considerations when choosing a field size is that field operations over a small group are substantially faster than field operations over a large group. It's generally good to have the option to use a small field, but some operations like operating over 256 bit numbers get very annoying over small fields just under 256 bits like Goldilocks. You need to allocate two field elements for one value and it roughly doubles the prover costs. R1CS is constrained to larger fields (for now).

The FRI blowup factor is a tunable parameter that allows you to adjust the cost to be more on the proving side or verifying side. A relatively low blowup factor leads to less prover time with larger proofs and a larger blowup factor leads to high cost to prove with smaller proof size.

So to address this problem, researchers often use a technique known as a "lookup argument". Rather than compute the bitwise instruction directly via additions and multiplications, lookup arguments precompute the outputs of the bitwise instruction on all possible inputs. Then, a zkVM applies a relatively cheap SNARK operation – aka "the lookup" – to verify that the current instruction lives within the pre-computed table. Doing so decreases the cost of the instruction.

5.1.3 Recap

It's important to distinguish between the backend and frontends of SNARKs to make clear assertions of the performance tradeoffs between various arithmetization schemes and SNARK backends. Failure to distinguish between them can results in misconceptions about performance and other characteristics of SNARKs

5.2 Comparison

5.2.1 Jolt

In the frontend, Jolt uses Rank-1 Constraint System (R1CS). The backend employs Spartan and Hyrax, with Hyrax incurring larger prover costs. It operates over an approximately 256-bit field, although efforts are being made to use smaller fields. The system utilizes Spice-based memory

checking and currently does not support recursion for aggregate proofs. For lookups, Jolt relies on Spice.

5.2.2 SP1

In the frontend, SP1 uses Algebraic Intermediate Representation (AIR) which requires expensive Fast Fourier Transforms (FFTs). The backend incorporates Plonky3 STARKs and Hyrax, with Hyrax incurring larger prover costs. It operates over smaller fields, either approximately 31-bit or 64-bit (Baby Bear, Goldilocks). SP1 supports recursion for aggregate proofs. The Fast Reed-Solomon Interactive Oracle Proof (FRI) blowup factor is 2, leading to faster proofs, larger proofs, and more expensive recursion. For lookups, SP1 uses Plookup. Notably, SP1 implements AIRs for each RISC-V instruction compatible with the Plonky3 prover and is optimized for CPU performance.

5.2.3 RISC-0

In the frontend, RISC-0 employs AIR, which also requires expensive FFTs. The backend utilizes Plonky3 STARKs and operates over a smaller, approximately 32-bit field (Baby Bear). RISC0 supports recursion for aggregate proofs, with a FRI blowup factor of 4, resulting in slower proofs, smaller proofs, and less expensive recursion. For lookups, RISC0 uses Plookup. This system is optimized for GPU performance.

6 Discussion

6.1 Contributions

Our work differs from related work in that we focus on a holistic comparable analysis of zkVMs for real world applications. We believe our approach is more relevant than referenced papers because us focusing on zkVMs and real world applications is the most relevant for development use cases compared to other work whose results do not give you much practical insight. Additionally, compared to publicly available benchmarks, our work is unbiased and also accompanied by a qualitative architecture comparison; publicly available benchmarks do not provide such analysis.

6.2 Limitations & Next Steps

This study faced several limitations that warrant consideration. First, the benchmarking process was constrained by hardware limitations, as scaling to 10,000 test cases resulted in excessive runtime or timeouts. This issue was particularly pronounced for Jolt, which exhibited high memory demands and would terminate itself when insufficient memory was available. Additionally, the implementation was challenged by the requirement to avoid using Rust's standard library and most external libraries, including essential utilities such as I/O operations and data structures like strings. This was the main reason why we were unable to implement poly ridge regression algorithm. While the zkVMs allow the use of the standard library, its inclusion significantly slows proof generation, rendering benchmarking infeasible in practice.

Furthermore, our study was limited in its scope of metrics, as we could not compare aspects such as memory usage due to the lack of publicly available benchmarks for such evaluations. A deeper analysis of architectural design differences among the zkVMs would also have been valuable, as it could provide more nuanced insights into their performance characteristics. Lastly, benchmarking the zkVMs both with and without precompiles would have offered a clearer understanding of the performance impact of the underlying designs. This approach, suggested by Justin Thaler, Professor at Georgetown University and a leading expert in the field, represents a promising direction for future research. These limitations and potential areas for improvement highlight opportunities for further exploration in subsequent studies.

7 Conclusion

Our paper[8] provides a quantative and qualitative analysis on the performance of zkVMs: Jolt, SP1, and RISC-0 by generating a zero-knowledge proof of a ridge regression algorithm.

- **Jolt:**
  - Demonstrated the fastest performance across both ARM and x86 architectures, particularly excelling in large-scale inference tasks.
  - **10 test cases:** 13.57s (ARM), 13.62s (x86).
  - **1000 test cases:** 15.10s (ARM), 69.36s (x86).
- **Risc0:**

  - Achieved the fastest times for small-scale inference but performed poorly for larger workloads, particularly on x86.
  - **10 test cases:** 7.46s (ARM), 9.62s (x86).
  - **1000 test cases:** 132.13s (ARM), 309.14s (x86).

- **SP1:**
  - Consistently performed between Jolt and Risc0, with minimal variation across architectures.
  - **10 test cases:** 7.46s (ARM), 10.21s (x86).
  - **1000 test cases:** 20.83s (ARM), 108.63s (x86).

The benchmarking conducted in this study did not incorporate GPU acceleration, presenting a potential avenue for future work to explore the impact of hardware acceleration on the performance of zkVMs.

Also, the performance tradeoffs we detailed in the qualitative comparison section are reflected in the benchmark performance. Jolt's proving time was the fastest across all our benchmarks and generally slower for verification. SP1's proving time was also slightly higher than RISC-0's, showing the advantage of precompiles.

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.

[8] L. Lime, "zk-benchmark," https://github.com/larry-lime/zk-benchmark2, 2024, accessed: 2024-12-03.