WIP: Analyzing and Benchmarking ZK-Rollups

Stefanos Chaliasos [*]
Imperial College London, UK

Itamar Reif [*]
Astria, United States

Adrià Torralba-Agell
Universitat Oberta de Catalunya, Spain

Assimakis Kattis

Benjamin Livshits
Imperial College London, UK
Matter Labs

Abstract

As blockchain technology continues to redefine the landscape of digital transactions, scalability emerges as a paramount challenge. This limitation has led to the development of innovative solutions, notably Layer 2 (L2) scalability solutions, such as rollups. Among these, ZK-Rollups stand out for their use of Zero-Knowledge Proofs (ZKPs) to ensure eager on-chain verification of transactions, thereby enhancing scalability and efficiency without compromising security. However, the inherent complexity of ZK-Rollups has limited comprehensive analysis of their efficiency, economic implications, and overall performance.

This work presents a theoretical and empirical study aimed at understanding and benchmarking ZK-Rollups, with a focus on ZK-EVMs. We undertake a qualitative analysis to dissect the costs associated with ZK-Rollups and scrutinize the design decisions of popular implementations. Addressing the inherent challenges in benchmarking such complex systems, we propose a structured methodology for their evaluation, applying our approach to notable ZK-Rollups: Polygon zkEVM and zkSync Era. Our study yields preliminary results that shed light on the trade-offs and areas for improvement in ZK-Rollup implementations, offering valuable insights for future research, development, and deployment of these systems.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases Zero-Knowledge Proofs, ZK-Rollups, Benchmarking, Blockchain Scalability

1 Introduction

Blockchain technology, with leading examples Bitcoin [21] and Ethereum [35], has introduced novel solutions for finance and various applications, reshaping the landscape of digital transactions by removing the need for centralized entities. However, the surge in their adoption has brought to light a critical challenge: scalability. The inherent limitation in the number of transactions these networks can process per second has prompted an effort within the blockchain community to seek out and develop innovative solutions [38].

Two primary strategies have emerged to address scalability. The first involves the creation of new, modern blockchains designed from the ground up to process transactions more efficiently than their predecessors [36, 5], albeit at the cost of missing the established security and network effects of blockchains like Ethereum. The second strategy revolves around Layer 2 (L2) solutions, or off-chain scalability solutions, with rollups being the most promising and widely adopted in practice [34]. Rollups work by executing transactions on a faster,

[*] These authors contributed equally to this work.

secondary blockchain (L2) and then posting the resulting state root, along with transaction data, back to the main blockchain — Layer 1 (L1). This ensures the integrity of the rollup's state is verifiable and secure, leveraging the underlying blockchain's security.

Among the various rollup approaches, two stand out: optimistic [15] and ZK-Rollups [3]. Optimistic rollups rely on a system of trust and fraud proofs to validate state transitions, which introduces a delay in withdrawals due to the required challenge period. Conversely, ZK-Rollups utilize Zero-Knowledge Proofs (ZKPs) for immediate on-chain verification of state transitions, enhancing both scalability and efficiency without compromising the security of the L1 chain. Despite their advantages, ZK-Rollups introduce additional complexity, and to date, there has been limited research focused on thoroughly evaluating their overall efficiency, limitations, and economics.

Benchmarking ZK-Rollups presents a multifaceted challenge. The deployment of these systems is inherently complex, and their diverse design choices complicate direct comparisons. Additionally, identifying common payloads for benchmarking and establishing appropriate metrics are non-trivial tasks. In response to these challenges, this work embarks on a comprehensive theoretical and empirical analysis of ZK-Rollups. We dissect the operational and per-transaction costs of ZK-Rollups, examine the design decisions of prominent implementations, and propose a methodology for their benchmarking. This includes addressing the challenges inherent in benchmarking these systems, defining key research questions, and developing a reproducible methodology to ensure our findings are publicly accessible and verifiable.

The results of this study aim to illuminate the trade-offs inherent in different ZK-Rollup implementations, offering insights into their advantages and areas in need of improvement. By providing a deeper understanding of the economics underpinning these systems, we hope to inform efforts to decentralize currently centralized systems. Furthermore, as Rollups as a Service continues to grow, our analysis seeks to help users with the knowledge to make informed decisions, enabling them to compare different rollups using our benchmarking infrastructure tailored to their specific needs.

1.1 Contributions

- **Qualitative Analysis of ZK-Rollups:** We conduct a comprehensive theoretical analysis of ZK-Rollups, detailing the costs associated with processing transactions and examining the diverse design choices across different implementations.
- **Towards Benchmarking ZK-Rollups:** Addressing the significant challenges inherent in benchmarking ZK-Rollups, we develop and present a structured methodology for their evaluation. This includes the implementation of our benchmarking approach on prominent implementations such as Polygon ZK-EVM and zksync Era, providing a blueprint for systematic assessment of ZK-Rollups' efficiency and costs.
- **Preliminary Results and Insights:** Offering initial findings from our benchmarking efforts, we aim to contribute to the ongoing discourse on ZK-Rollups by identifying key factors that influence their decentralization and pinpointing areas in need of improvement. These preliminary results are intended to guide future research and development efforts in the field.

2 Background

2.1 Scaling Blockchain and Rollups

Blockchain scalability has been a persistent challenge, particularly for established networks like Ethereum [35], which processes only tens of transactions per second. [1] Efforts to enhance scalability have focused on two primary strategies: base layer scaling and L2 scaling solutions. Base layer scaling, which includes techniques such as sharding and novel consensus protocols, involves either the modification of existing blockchains — a complex and daunting task — or the development of new blockchain architectures. While modern blockchains like Solana [36] and Sui [5] have demonstrated success, they often lack the established security, liquidity, and comprehensive ecosystem found in legacy blockchains like Ethereum. [2]

L2 scaling solutions, on the other hand, offer a promising avenue for scalability without altering the base layer, i.e., L1. Among these solutions, pay,ent channels [1, 2, 18, 33], Plasma [28], and rollups [34] have been the most prevalent solutions. Payment channels enable instant, bi-directional payments between two parties by establishing a network of interconnected channels, exemplified by Bitcoin's Lightning Network. However, they require capital lockup and constant base layer monitoring, making them suitable for specific, long-term use cases. Plasma attempted to solve various issues in different ways. Sguanci et al. [32] provides an overview of the main types of Plasma constructions. However, every attempt at Plasma had some trade-off that resulted in a poor user experience.

Rollups have emerged as hybrid L2 solutions, distinguishing themselves by offloading computation off-chain while retaining data on-chain, thus addressing the data availability issue while inheriting L1's security. Rollups batch and execute transactions on an auxiliary L2 blockchain that either uses the same VM as L1 or a different one. This separation of transaction execution from consensus allows rollups to process significantly more transactions per second than their L1 counterparts. By submitting a summary of the rollup's state — typically, the root of a Merkle tree — to a smart contract in the underlying blockchain, rollups not only ensure data availability but also inherit the security properties of the L1 network. Altering the L2 state recorded on L1 would require breaking substantial security, making it both difficult and costly. This architecture enables rollups to offer an efficient, secure scaling solution for legacy blockchains. Notably, this model, i.e., rollup-centric scaling, [3] has gained traction as the *principal* method for scaling Ethereum, with two predominant variants: optimistic rollups [15] and ZK-Rollups [3]. [4]

Optimistic rollups operate on a principle of trust, where state transitions are accepted without immediate verification, relying instead on fraud proofs to challenge incorrect state updates. This approach, while efficient, necessitates a challenge period, introducing a delay in withdrawals. On the contrary, ZK-Rollups leverage zero-knowledge proofs to verify state transitions on-chain, offering a more immediate and efficient validation process without the need for a challenge period. This method not only enhances scalability and efficiency but also maintains the integrity and security of the L1 chain.

[1] https://l2beat.com/scaling/activity
[2] According to https://defillama.com/chains (accessed: 10/5/2024), Ethereum has 57.85% of the total TVL for all chains, while Solana has only 4.46%.
[3] https://ethereum-magicians.org/t/a-rollup-centric-ethereum-roadmap/4698
[4] As of 18/3/2024, rollups have more than 34B USD TVL according to https://l2beat.com.

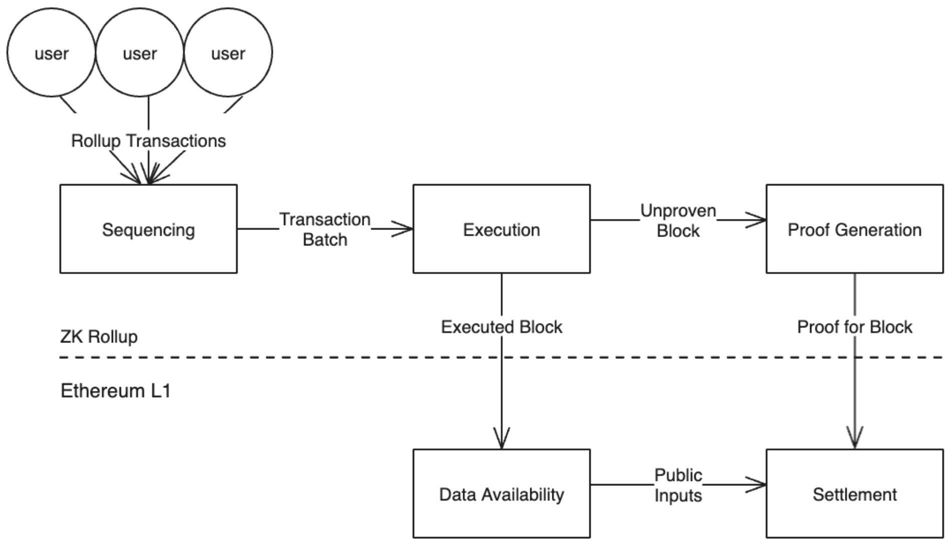

Figure 1 High-level simplified overview of transaction processing in ZK-Rollups. This figure abstracts the core components and workflow of ZK-Rollups, excluding details on bridging and forced L1 transactions.

2.2 ZK-Rollup Components and Transaction Lifecycle

In this section, we outline the main components of a ZK-Rollup. For simplicity, we abstract away certain details, including bridging and forced transactions [13]. Additionally, we do not delve into various sequencing methodologies, such as decentralized or shared sequencers [20]. Figure 1 illustrates the key components involved in processing transactions within a ZK-Rollup. Users initiate the process by signing and submitting transactions to the L2 network. A sequencer then undertakes the tasks of processing, ordering, executing, and batching these transactions. In some architectures, these functions may be distributed across different components. Subsequently, the sequencer forwards these batches to a relayer, which commits them and their resultant state to the L1 rollup contract. Concurrently, the sequencer sends the batch to a coordinator (or aggregator), which, in turn, sends the batch to the prover. In most ZK-Rollups, the coordinator consolidates multiple proofs into a single aggregated proof (i.e., a proof of proofs) and submits this final proof to the rollup contract via the relayer. The contract then verifies the proof, finalizing the state of the L2 as immutable and verified in the L1.

Components.

- **Sequencer** The sequencer provides users with the first confirmation of transaction inclusion and ordering. It is the entity that aggregates user transactions into blocks and batches, either by providing them a transaction submission endpoint or by pulling transactions from the mempool. It provides the canonical sequence of transactions that will be fed into the state transition function. Currently, for the systems discussed in this work, sequencer implementations rely on a trusted operator that provides this service and to which users submit their transactions. Many of the projects discussed are working on decentralizing their sequencer design, but the design space remains nascent.
- **Execution** For most existing rollups, execution is typically done by the same entity as the sequencer. The transaction batch created by the sequencer is taken as the input to

the rollup's state transition function (STF), which is executed to create both the resulting block(s) & related state root(s), as well as the transaction batches, and potentially the intermediate state snapshots (state diffs) that are required for proof generation.

- **Data Availability** Data availability (DA) refers to the ability of clients of the blockchain protocol to retrieve the data required to verify the validity of a given batch. Traditionally provided as part of the consensus algorithm that underlies a blockchain system, the modular architecture used by rollup-based blockchain systems separates the guarantees provided by the L1 blockchain from those provided by the rollup operators. Relying on L1 for the rollup's liveness, the data required to assert safety must be posted on the DA. This involves any execution artifacts required by the settlement logic for verifying the validity proof, such as the transaction data or the state diff of the transactions.
- **Prover** The prover is responsible for generating validity proofs for the executed batches. Given the execution artifacts the prover creates the required witness data and executes a Succinct Non-interactive Argument of Knowledge (SNARK) or Scalable Transparent Argument of Knowledge (STARK) proof to prove the validity of the execution. Note that when a STARK is used, it is typically wrapped into a SNARK to enable efficient verification.
- **Settlement Logic** The generated proofs are then uploaded to the L1 smart contract responsible for settling a given batch. An executed batch needs to be agreed upon as "valid" in order to coordinate between decentralized actors (the blockchain's users). This is done by providing a block's resulting state, proof of that state's validity, and the inputs required for verifying the proof. Approaches to solving this vary between the projects discussed in this work, with some making use of intermediate state snapshots while others require the entire sequenced transaction batch.

Transaction Lifecycle Status.

- **Pending:** A user has signed and submitted the transaction to the L2 network.
- **Preconfirmed:** The sequencer has processed the transaction and included it in a block. If users trust the sequencer, they can regard the transaction as processed. Currently, the reliance on centralized sequencers enables near-instant preconfirmation, yet this raises the challenge of maintaining such efficiency without centralization. Preconfirmation significantly enhances user experience, allowing users to treat most transactions as effectively complete. However, it's important to note that for withdrawals from L2, users must await the finalization of transactions.
- **Committed:** The transaction is part of a batch committed to the L1 contract, allowing others to reconstruct the L2 state, including this transaction from L1 data.
- **Verified/Finalized:** The batch containing the transaction has been proven, and the proof verified by the L1 contract, marking the transaction and its batch's state on L2 as immutable.

2.3 Costs of ZK-Rollups

We now delve into the costs associated with processing a transaction within a ZK-Rollup. Specifically, we distinguish between costs that are transaction-specific and those that are constant per batch, meaning they apply to each batch processed regardless of the number of transactions included in it. We largely focus on the costs related to operating these systems without delving into the various fee models they employ. For example, we describe data availability costs in terms of bytes rather than the gas charged for posting those to Ethereum L1.

Batch-Fixed Costs. Each batch carries inherent fixed costs that must be paid regardless of the transactions included in it.

1. **Settlement** which involves (a) calling the Ethereum L1 contract to commit to a specific batch, (b) submitting proofs and executing the verifier logic (e.g., Groth16 verifier) for committed batches.
2. **Proof Compression**, as some constructions involve compressing (or converting) the block's proof from one proof system to another. This typically involves proving the verification of the aggregated proof in a cheaper (with regards to the verification cost) proof system (e.g., Groth16) so that the cost of settlement is lower.

Transaction Marginal Costs. In addition to the batch-specific costs, each transaction included in a rollup's block incurs the following additional costs:

1. **Data Availability**, for the bytes of the transaction's 'calldata'. The transaction's 'calldata' needs to be posted to the data availability provider, e.g., Ethereum L1, so that the rollup's state can be reconstructed.
2. **Proving Costs**, which are split into the following:
   a. Additional witness generation work required
   b. Proof generation for the transaction's execution
   c. Different constructions incorporate a final step of aggregating batches of proofs into a single proof. Proving an additional transaction may require more aggregation work.
3. **L2 Execution Costs**, which require both computing the state transition resulting from the transaction and any long-term storage requirements that it causes. These are the costs of operating the rollup's infrastructure: sequencing, execution, and relaying

3 Qualitative Analysis

In this section, we examine the fundamental components and design choices influencing the performance, efficiency, and complexity of various ZK-Rollups. Figure 2 summarizes a qualitative overview of ZK-Rollups.

3.1 Proof System

Recent advancements in proof systems have led to a significant acceleration of ZK-VMs and ZK-EVMs, with research focusing on developing proof systems to optimize for better performance and efficiency of the proving algorithm. Notable developments include zkASM [27] and PIL [26] ZK languages, developed by Polygon for their ZK-EVM; Boojum [19], developed for zkSync Era; and the halo2 KZG fork [31] implemented by the Scroll team, among others. A common strategy in zkVM design is to also apply recursion, a technique where one ZK proof is verified inside of another, allowing the usage of cheaper verification circuits at the settlement phase while leveraging more complex proof systems in the proving process. For instance, Polygon ZK-EVM leverages a STARK proof to initially prove batch correctness, which is then compressed via recursion before being encapsulated in a SNARK proof for submission to L1. This method benefits from SNARKs' efficient verification and constant proof size. Similar methodologies are employed across various ZK-EVM platforms. However, the choice of a proof system and its specific implementation can lead to different trade-offs, particularly between the speed of proof generation and the computational resources required. This balance is crucial, as it can influence the overall performance and user experience of ZK-Rollups.

3.2 Transaction Data vs. State Diffs

In Ł1 blockchains, all transactions in a block are stored alongside the Merkle root of the final state. This information is disseminated across the network through a “gossiping” protocol [16], and the root of trust is established through re-execution and validation of the state root by the participants. For example, in proof-of-stake networks, validators stake their tokens and vote on the resulting state to ensure consensus [8]. ZK-Rollups, in contrast, establish their root of trust through the verification of a ZKP [3]. The ZKP attests to the correctness of the final state, and its validation is sufficient for participants to accept a batch of blocks as canonical. Verifying the final state requires publicly available inputs, typically either the transaction data included in the batch or intermediate state transition snapshots, known as “state diffs.” Each method has its trade-offs. State diffs are more cost-effective because they omit signatures and publish only the final state changes after multiple transactions, thus allowing for better cost amortization. However, this approach does not preserve a complete transaction history and can complicate the mechanisms of enforcing transactions and reproducing the state through data posted in the Ł1. Currently, ZK-Rollups such as zkSync Era and Starknet utilize state diffs due to their efficiency benefits, while solutions like Polygon ZK-EVM and Scroll opt to publish transaction data to maintain data completeness. Both approaches are exploring innovative compression techniques to further optimize cost efficiency.

3.3 EVM Compatibility

Buterin identifies four main categories of ZK-EVMs [7], which are implementations of ZKP circuits that validate the correctness of Ethereum Virtual Machine (EVM) execution, ranging from fully Ethereum-equivalent to language-compatible. Fully Ethereum-equivalent ZK-EVMs replicate the EVM’s behavior and data structures precisely, ensuring seamless operation for existing Ethereum applications. EVM-equivalent ZK-EVMs maintain core functionalities but introduce slight variations in data structures while ensuring identical behavior when executing EVM-bytecode. EVM-compatible approaches might exclude certain precompiles or slightly modify the gas-metering mechanism, which could affect the execution of specific transactions in edge cases. The most flexible, language-compatible ZK-EVMs, utilize compilers to translate Solidity into different targets, optimizing efficiency and potentially enhancing functionality beyond strict EVM equivalence. This spectrum of compatibility reflects a trade-off between maintaining strict adherence to the EVM and pursuing efficiency gains or advanced features through innovation. For instance, Scroll and Polygon ZK-EVM aim for close EVM equivalence to balance compatibility with performance improvements, while zkSync Era opts for a language-compatible approach with its zksolc compiler, prioritizing efficiency and adaptability.

Another approach, not described in Buterin’s classification, is employed by RISC0’s Zeth [30]. Based on a prover for the RISC-V Instruction Set Architecture (ISA), Zeth leverages Rust and its LLVM-based compiler toolchain to utilize a suite of robust crates, such as revm [5], ethers [6], and alloy [7], enabling the proof of execution for EVM-based transactions and blocks without the need for additional domain-specific implementation circuits by leveraging RISC0's prover. This approach diverges from traditional ZK-EVM designs by proving the correctness of computations at the ISA level rather than focusing exclusively on EVM bytecode. The use of RISC-V as the underlying architecture allows for a high degree of flexibility and the potential to leverage a broader range of programming languages supported by the LLVM ecosystem. This approach results in a fully EVM-equivalent ZK-EVM.

The landscape of ZK-EVMs has seen a rapid evolvement of innovative approaches in recent years, though not all are "EVM compatible" to the same degree. Several implementations discussed in this section were not included in our direct performance comparison due to differing definitions of compatibility. The term "EVM compatibility" encompasses various properties, including:

- Support for the standard Solidity compilation toolchain, allowing existing Solidity contracts to be ported over without additional work.
- Compliance with Ethereum's exact state transition logic.
- Adherence to Ethereum's gas cost metering mechanism.
- Adherence to Ethereum's JSON-RPC client API.
- Support for Ethereum's smart contract standards (e.g., ERC-20) and precompiles (e.g., `keccak`).
- Support for Ethereum's existing wallet infrastructure.
- Support for Ethereum's existing development infrastructure.

3.4 ZK-Rollups Prover Implementations

At a high level, there are two primary approaches for ZK-Rollup prover design: assembly-based and EVM opcode-based implementations, each offering a distinct approach to handling computations and proofs. Assembly-based VMs implement a specific Instruction Set Architecture (ISA), focusing on proving the correctness of lower-level execution steps that express abstractions around the underlying proof system. This method closely aligns with traditional hardware architectures, where each instruction within the set is designed to perform well-defined, atomic operations [12, 30]. Conversely, opcode-based ZK-EVMs rely on specific circuits that each prove the execution of an EVM opcode or an EVM state transition, with the specific goal of proving the validity of the execution of an EVM transaction.

Figure 3 provides a high-level overview of the compilation and proving process followed by different ZK-Rollups.

The first example of an assembly-based ZK-VM is Starkware's Cairo. Built as an abstraction on top of Algebraic Intermediate Representation (AIR), Cairo assembly generates polynomial constraints over a table of field elements that represent the state throughout the program's execution. This table serves as the trace (or witness) for the STARK-based prover, which then proves whether the trace satisfies Cairo's semantics [12]. While the Cairo framework allows for generating application-specific circuits, in practice, it is used in Starknet to generate circuits for a single set of constraints for the von Neumann architecture-based Cairo CPU. The Cairo CPU is an AIR-generated STARK for a Cairo program that implements a register-based general-purpose VM. This design enables recursive proving,

[5] https://github.com/bluealloy/revm
[6] https://github.com/gakonst/ethers-rs
[7] https://github.com/alloy-rs/alloy

where the verification of one program P_0 can be phrased as another program P_1, and similarly, the verification of another program P_2 can be represented as P_3. All these programs can be executed on the same Cairo CPU, represented as a program P_4 which corresponds to "P_1 AND P_3", thus maintaining consistency [12].

Another example of an assembly-based implementation is RISC0's RISC-V-based ZK-VM. This approach involves compiling a Rust program to RISC-V ISA, referred to as the Guest Program. The Guest Program is executed to produce an execution trace, corresponding to the intermediate states of the RISC-V VM throughout the execution. The trace is then used as a witness by the RISC-V prover, which provides proof that the execution follows RISC-V semantics. Unlike Cairo, which uses a novel ISA tailor-made for STARK, RISC0 builds a prover for the existing RISC-V ISA. Utilizing the LLVM compiler toolchain, RISC0 can execute and prove any language that can be compiled to RISC-V ISA [29]. Using that approach, you can pass an EVM implemented in Rust as the guest program and prove EVM transactions, as demonstrated by Zeth [30] (c.f. Section 3.3).

In contrast to assembly-based ZK-VMs, the most common approach is to directly target EVM bytecode. While the former approaches rely on an intermediate ZK-VM for circuit implementation, many mature ZK-Rollups have chosen to implement specialized circuits that directly prove the execution of EVM bytecode, or close to EVM bytecode. Currently, all major ZK-Rollups beyond Starknet utilize specialized circuit-based ZK-EVM implementations.

For instance, zkSync Era, Polygon's zkEVM, and Scroll implement a ZK-EVM to prove the execution of transactions, but they employ slightly different methods. Polygon and Scroll use Solidity's compiler to allow existing smart contracts written in Solidity (or Yul) to be deployed, executed, and proved on a ZK-Rollup by implementing specialized circuits that validate EVM transaction traces. In contrast, zkSync Era takes a higher-level approach by compiling Solidity directly to ZK-EVM bytecode (using zksolc), which is then executed and proved using circuits designed to verify the ZK-EVM bytecode's correctness instead of EVM bytecodes.

Another approach is Aztec's Abstract Circuit Intermediate Representation (ACIR) and Private Execution Environment (PXE)-based system. Aztec's smart contracts are written in Noir [23], a domain-specific language developed by Aztec Labs for SNARK-based proving systems. Noir compiles to ACIR, an intermediate representation used to generate circuits for proving. Due to additional features provided by Aztec, specifically the support for nullifier-based private transactions, Aztec's execution model separates the handling of private data from the processing of public transactions. Transactions are first executed and proved locally by users inside the PXE, with only the public components of a transaction and a ZKP of the validity of privately executed components propagated to rollup nodes. This ensures the validity of private transactions and the execution of their public components to ensure block validity.

3.5 Target Hardware & Prover Architecture

The hardware configurations required for ZK-VM provers vary significantly across projects, reflecting the diverse computational demands of their proof systems. Our analysis divides prover designs by target hardware and parallelization strategies into the following categories:

1. Single CPU-optimized implementations such as Polygon's ZK-EVM, which demands a high-capacity setup with a 96-core CPU and at least 768GB of RAM
2. Single GPU and CPU proving, such as Scroll's system, which parallelizes the execution of multiple blocks using a single GPU but then aggregates the proofs into a single proof

that is posted on the chain using a CPU.

3. Cluster-based approaches. Both zkSync and Risc0's systems rely on two stages. At first the state transitions are divided into segments and proved in parallel, after which the proofs are aggregated, also in parallel. The key difference with Scroll's approach is that aggregation is also parallelized across a large cluster of GPUs or CPUs.

This highlights an essential consideration: each ZK-VM implementation is meticulously optimized for specific hardware configurations, rendering direct performance comparisons on identical machines less meaningful. Additionally, the ongoing research into hardware acceleration for ZK-EVMs aims to further enhance proving times and system performance.

4 Benchmarking and Analyzing ZK-Rollups

In this section, we delve into our methodology for benchmarking and analyzing the costs associated with ZK-Rollups. Benchmarking these systems is essential, as it sheds light on the areas most in need of optimization. With many ZK-Rollups projects aiming to decentralize their core components, this analysis offers a timely opportunity to assess the costs involved and explore how these systems can achieve profitability and sustainability. Additionally, we consider projects interested in deploying specialized, application-specific rollups using existing infrastructure, aiming to discern the cost-related trade-offs of each stack. Our analysis is designed to simplify the understanding of how external factors influence the costs and, consequently, the fees associated with ZK-Rollups. For instance, we examine the potential impact of advancements in hardware that could reduce proving costs or the effects of increased prices for blob space. We begin by identifying the primary challenges in benchmarking ZK-Rollups, particularly focusing on their core component, the ZK-EVM. Following this,we highlight the key research questions we aim to answer through our analysis. Lastly, we present our methodology and the decisions made to navigate these challenges and answering the targeted research questions.

4.1 Challenges

In this subsection, we outline the inherent challenges in benchmarking ZK-Rollups, with a specific focus on the prover component, i.e., the ZK-EVM. While assessing the Data Availability or L1 cost per transaction can be relatively straightforward, i.e., by executing transactions without the proving process, understanding per transaction requirements, and leveraging historical L1 pricing data. The benchmarking of the prover presents a more intricate challenge. Here, we outline the primary challenges:

C1 **Metrics**: Determining appropriate metrics is a fundamental challenge. The two principal dimensions are time and cost. Time can be relatively straightforwardly measured as the clock time required to generate a proof, impacting the system's finality. Cost, however, is multifaceted, encompassing not only the financial cost per batch or transaction when utilizing cloud-based proving services (which is the current state-of-practise for running provers) but also factors like energy consumption, computing cycles, memory consumption, and proof size. It is crucial to acknowledge the difficulty in measuring these metrics on a per-transaction basis, as ZK-EVMs typically process transactions in batches, often incurring a fixed cost.

C2 **Configuration**: Choosing the right hardware configuration for benchmarking is another significant challenge. Ideally, different systems should be tested on identical or at least similar hardware specifications. However, this is nearly unfeasible for ZK-EVMs, as they are designed with distinct objectives in mind and optimized for vastly different hardware setups (c.f. Table 2).

C3 **Payloads**: Deciding on the appropriate payloads for benchmarking is also challenging. Given the varying degrees of EVM compatibility across systems, selecting a common payload for comparative analysis is difficult. Moreover, the complexity of these systems further complicates setup and benchmarking efforts with specific payloads.

C4 **Reproducibility**: Beyond the challenges mentioned, ensuring reproducibility is crucial. Regardless of the chosen metrics, configurations, and payloads, it's essential that the benchmarking process is designed in such a way that third parties can validate the results.

4.2 Research Questions

Next, we outline a series of preliminary research questions that will shape our analysis of ZK-Rollups. These questions are designed to uncover critical insights into the performance, cost structure, and overall efficiency and profitability of ZK-Rollups. Our investigation aims to provide a comprehensive understanding of these systems, with more detailed research questions highlighted in future work (Section 6). Our preliminary research questions include:

RQ1 **Proving Time**: How long does it take to generate a proof for a batch of transactions? This question seeks to understand the finality of each ZK-Rollup.

RQ2 **Proving Cost**: What are the costs of generating a proof for a batch? This includes costs in terms of running the specific required machines or energy consumption.

RQ3 **Total Transaction Cost**: What is the total cost, encompassing both Data Availability and proving costs, for executing standard transactions such as transfers and ERC-20 token transfers within each ZK-Rollup?

RQ4 **Aggregation and Verification Costs**: What is the cost of aggregating multiple proofs into a single proof (if applicable) and the subsequent verification of these proofs? This question aims to dissect the constant costs inherent to ZKPs.

RQ5 **Cost Mapping**: How are the costs distributed across different components of a ZK-Rollup transaction, including DA, proof generation, aggregation, L1 posting, and verification? Understanding this distribution is crucial for identifying potential areas for optimization.

RQ6 **Impact of EIP-4844**: How has the introduction of EIP-4844 influenced the cost dynamics of ZK-Rollups? This question explores the effects of EIP-4844 on the cost efficiency and practicality of ZK-Rollups.

4.3 Methodology

In this section, we detail our methodology for addressing the challenges identified in benchmarking ZK-Rollups, as outlined in Section 4.1. Our approach, depicted in Figure 4, provides a high-level overview of the benchmarking process for each ZK-Rollups. Initially, transactions from the selected payload are processed using the sequencer and any auxiliary tools provided by the ZK-Rollup to create a batch. This step can be performed on standard hardware, without the need for specialized, expensive equipment. Subsequently, this batch is fed into the prover pipeline within an instrumented environment designed to capture the necessary metrics. The proving phase generally consists of at least two steps, though this can vary. The final phase involves calculating the total costs, incorporating L1 data posting prices, batch verification costs on L1, and hardware expenses. By analyzing the data required for

L1 posting from each batch and integrating all details, we generate a comprehensive report that breaks down the costs.

Metrics. Our initial focus is on straightforward metrics that facilitate a basic comparison across different systems, making it easier for teams to integrate their ZK-EVMs into our benchmarking framework. In future work, we plan to expand our metrics to include power consumption, RAM usage, and GPU or CPU time. The primary metrics in this preliminary study are:

- **Seconds per Proof**: The time required to generate a proof.
- **USD per Proof**: The cost of generating a proof, calculated by multiplying the clock time by the rate for using a cloud service like AWS, or querying it from another API.
- **USD per Proving a Transaction**: The cost associated with proving a single transaction.

Configuration. Given the unique hardware optimization of each system, as discussed in Section 4.3, we chose to benchmark systems according to the hardware specifications recommended by each team for production use. Our insight is that comparing systems optimized for different hardware on the same setup could be misleading, thus we opted for this approach as the fairest solution. This decision aims to capture the optimal cost/time efficiency based on each system's specific optimizations.

Payloads. Ideally, benchmarking would utilize historical Ethereum blockchain data, but the lack of necessary tooling in most ZK-Rollups complicates this approach. Instead, we focus on benchmarking common smart contract functionalities, including *native transfers*, *ERC-20 transfers*, *token swaps*, and *contract deployment*. These payloads represent typical blockchain operations, though future work may expand this list for more detailed analysis (e.g., native Solidity SHA-256, DAO voting, and NFT mints). This approach ensures a focus on fundamental operations, though it may not fully highlight the advantages of certain solutions, such as state diffs.

Reproducibility. To ensure reproducibility, we publish all configurations, scripts, and instructions used in our benchmarking process. This includes detailed specifications of the machines used, scripts for setting up the environment, and step-by-step instructions for running tests with the specified payloads. Our goal is for others to be able to replicate our findings by following our documentation, thereby reinforcing the validity and reliability of our results.

5 Results

5.1 Candidate Selection & "EVM Compatilibility"

For the purposes of our exploration of measuring rollup performance and costs, our requirements were the following: 1. The ability to use the same contract for measuring costs, as we sought to run the same workload on all systems. 2. Being able to leverage existing libraries and tools for submitting transactions, so that our testing could be reproducible by other members of the community.

The systems we were able to achieve parity on these fronts are:

- Scroll
- Polygon's zkEVM
- zkSync
- Risc0

The systems we analyzed but were not able to construct comparable benchmarks for were:

- **Aztec**, which incorporates privacy features into its execution model and does not support existing EVM-based smart contracts. Both the novel VM design and the Private Execution Environment resulted in a vastly different execution model than the other "EVM-compatible" systems we compared.
- **Starknet**, which at the time of writing did not have support for existing EVM-based smart contracts. Warp, a Solidity-Cairo transpiler that was previously developed and maintained by Nethermind, was retired in July 2023 following Starknet's transition to Cairo 1.0. We elected not to base our analysis on Warp, as it was retired partly due to brittle compiler outputs.

5.2 Overview

6 Open RQs and Future Work

In this section, we present the open questions that remain unanswered in our current work and sketch a roadmap for future research. These questions not only underscore the complexities inherent in ZK-Rollups but also highlight areas that require further exploration.

- **Decentralization's Impact on Costs**: A pivotal question revolves around how decentralizing L2 core components will influence transaction processing costs. Specifically, we seek to understand whether the decentralization will lead to negligible cost implications or if it will significantly alter the economic landscape of ZK-Rollups. Another related open question but beyond the scope of this work is L2 MEV and cross-chain MEV.
- **Batch Size Optimization**: The size of transaction batches is a critical factor affecting proving costs. Future iterations of our work will delve into how variations in batch size influence the cost per transaction and the overall proving time. While current systems may operate with an optimal batch size tailored to their specific needs, emerging forks (e.g., app-chains) may require adjustments to accommodate different priorities. This analysis aims to provide valuable insights for optimizing batch size in response to evolving requirements.
- **Metering Mechanism Evaluation**: Another area of interest is the examination of the metering mechanisms employed by ZK-EVMs, which traditionally mirror those of the EVM [35]. Given that proving certain EVM opcodes might be relatively more costly than their execution, we plan to investigate potential discrepancies in pricing. Through micro-benchmarks, we will explore whether such mismatches are trivial or if they pose significant challenges, such as the underpricing of specific opcodes that could lead to DoS attacks.
- **Proving Market Mechanisms**: We also intend to explore various proving market mechanisms and assess how they might influence the cost dynamics of proof generation. This exploration could shed light on potential economic models conducive to more efficient and cost-effective proving processes.
- **Throughput Limitations**: Identifying the maximum transactions per second (TPS) each rollup can achieve, based solely on proving and DA on Ethereum and excluding other market dynamics, is another critical inquiry. This analysis will help quantify the scalability limits of current ZK-Rollup implementations.

In addition to addressing these open questions, our future work will expand the scope of our benchmarks. We aim to conduct micro-benchmarks at the opcode level to gain a finer-grained understanding of proving costs. Moreover, we plan to introduce macro benchmarks with diverse payloads beyond those presented in this study. Further, by replaying blocks from Ethereum on different ZK-Rollups, we aspire to provide deeper insights into their performance and cost-efficiency. Finally, we plan to analyze and evaluate more ZK-Rollups to get a more holistic overview of the entire ecosystem. Specifically, we also plan to analyze ZK-VM-based ZK-Rollups and compare them with native ZK-Rollups to shed more light on the debate between specialized circuit implementations versus using generic VMs to produce ZKPs.

7 Related Work

Benchmarking Blockchains and EVM Implementations. Benchmarking blockchains has garnered significant attention from both academic and practitioner communities. Gramoli et al. [14] developed a comprehensive benchmark suite for six popular blockchains with smart contract capabilities, conducting an extensive evaluation using five realistic decentralized application payloads across various configurations for each blockchain. Their primary objective was to assess latency in seconds and throughput in terms of transactions per second. In a subsequent study, Nasrulin et al. [22] introduced Gromit, a tool focusing on blockchains with diverse consensus algorithms, with both studies emphasizing the overall performance of the blockchains under examination rather than specific aspects such as the execution node implementations.

In this work, our focus shifts to benchmarking the core components of ZK-Rollups, paralleling efforts such as those by Cortes-Goicoechea et al. [9], who benchmarked the five most prominent Ethereum consensus clients to evaluate their resource consumption. Similarly, Zhang et al. [37] employed simple microbenchmarks to compare the efficiency of WASM EVM nodes against Geth and Openethereum. Our future endeavors include conducting micro-benchmarks to gain deeper insights into ZK-EVM implementations. Furthermore, both academic and industry efforts have explored benchmarking EVM nodes using either straightforward macro benchmarks [10], like ERC-20 transfers, or fuzzy techniques [24] to assess the performance characteristics of EVM implementations comprehensively. Mirroring this approach, we utilized macro benchmarks to evaluate the performance of ZK-EVM implementations, with plans to adapt tools such as flood [24] for future ZK-EVM benchmarking. Busse et al. [6] undertook evaluations of EVMs on various machines to pinpoint any noticeable differences.

While our current analysis does not extend to testing diverse machines for each ZK-Rollup, we aim to conduct such analyses to determine the most cost-effective hardware configurations for running each ZK-EVM and to identify the most optimized setup for each prover. Lastly, Perez and Livshits [25] evaluated the EVM's metering mechanism, identifying potential DoS attack vulnerabilities. Inspired by their findings, we plan to conduct stress tests on ZK-EVM implementations to uncover any mispricing in the proving costs of specific EVM opcodes.

Benchmarking ZKPs. Benchmarking efforts for ZKPs play a crucial role in enhancing the understanding and performance of cryptographic libraries and primitives. Benarroch et al.[4] highlighted the inherent challenges and outlined best practices for implementing benchmarks for ZKP Domain Specific Languages (DSLs). Building upon this foundation, our work outlines the specific challenges of benchmarking ZK-Rollups, particularly focusing on ZK-EVMs, and proposes a comprehensive methodology to tackle these challenges. Ernstberge et al. [11] introduced zk-Bench, a detailed benchmarking framework and estimator tool designed for evaluating the performance of low-level public-key cryptography libraries and SNARK DSLs. This framework enables a thorough examination of the trade-offs in ZKP development, covering the spectrum from low-level arithmetic libraries to advanced SNARK development tools.

Our research complements these efforts by concentrating on ZK-EVMs, which represent some of the most complex systems employing SNARKs. In addition to these efforts from academia, the Celer Network published a blog post[8] that benchmarks the time and memory costs of proving SHA-256 circuits across various ZKP tools. Furthermore, Anoma has introduced a framework[9] for benchmarking different ZKP compilation strategies, ranging from circuit-writing libraries to ZK-VMs capable of executing arbitrary programs, with an emphasis on execution time. Parallel to our efforts, Delendum has developed a framework[10] dedicated to benchmarking ZK-VMs, offering a unique perspective on the evaluation of these technologies. An interesting future work could be to compare the performance of ZK-VM-based ZK-EVMs versus native solutions.

Analyzing Rollups and ZK-Rollups. Thibault et al. [34] have conducted an extensive survey on the use of rollups as a scalability solution for the Ethereum blockchain, discussing the various types, highlighting key implementations, and offering a qualitative comparison between optimistic rollups and ZK-Rollups. Koegl et al. [17] have compiled a comprehensive list of known attacks on rollup systems, shedding light on their potential impacts. In contrast, our work delves into ZK-Rollups, providing a detailed overview and explanation of their characteristics through a qualitative analysis. Moreover, we further focus on the empirical benchmarking of ZK-Rollups and the meticulous analysis of their associated costs. This dual approach not only enriches the understanding of ZK-Rollups as a scalability solution but also underscores the economic viability and technical challenges of implementing ZK-Rollups.

8 Conclusion

In this study, we have undertaken a comprehensive analysis of ZK-Rollups, focusing on their scalability, efficiency, and economic implications. Our theoretical and empirical evaluations of Polygon ZK-EVM and zkSync Era reveal the inherent trade-offs in their design and implementation, providing critical insights into their operational costs and performance. By addressing the challenges of benchmarking ZK-Rollups, we have proposed a structured methodology that facilitates a thorough evaluation of these systems. Our preliminary results highlight areas for improvement and suggest directions for future research. This work not only enhances the understanding of ZK-Rollups but also aims to inform and guide the development of more efficient rollups.

References

1 Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *European Symposium on Research in Computer Security*. Springer, 2018.

2 Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Blitz: Secure multi-hop payments without two-phase commits. In *USENIX Security Symposium*, 2021.

[8] `https://blog.celer.network/2023/07/14/the-pantheon-of-zero-knowledge-proof-development-frameworks/`

[9] `https://github.com/anoma/zkp-compiler-shootout`

[10] `https://github.com/delendum-xyz/zk-benchmarking`

3 Barry Whitehat. Roll up token. https://github.com/barryWhiteHat/roll_up_token, 2018. Accessed: 2024-03-19.

4 Daniel Benarroch, Aurélien Nicolas, Justin Thaler, and Eran Tromer. 'community proposal: A benchmarking framework for (zero-knowledge) proof systems. *QEDIT, Tel Aviv-Yafo, Israel, Tech. Rep*, 2020.

5 Same Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, et al. Sui lutris: A blockchain combining broadcast and consensus. *arXiv preprint arXiv:2310.18042*, 2023.

6 Anselm Busse, Jacob Eberhardt, and Stefan Tai. Evm-perf: high-precision evm performance analysis. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–8. IEEE, 2021.

7 Vitalik Buterin. The different types of zk-evms. https://vitalik.eth.limo/general/2022/08/04/zkevm.html, 2022. Accessed: 2024-03-19.

8 Vitalik Buterin. *Proof of stake: The making of Ethereum and the philosophy of blockchains*. Seven Stories Press, 2022.

9 Mikel Cortes-Goicoechea, Luca Franceschini, and Leonardo Bautista-Gomez. Resource analysis of ethereum 2.0 clients. In *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 1–8. IEEE, 2021.

10 Ziyad Edher. evm-bench. https://github.com/ziyadedher/evm-bench, 2024. Accessed: 2024-03-19.

11 Jens Ernstberger, Stefanos Chaliasos, George Kadianakis, Sebastian Steinhorst, Philipp Jovanovic, Arthur Gervais, Benjamin Livshits, and Michele Orrù. zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks. *Cryptology ePrint Archive*, 2023.

12 Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo–a turing-complete stark-friendly cpu architecture. *Cryptology ePrint Archive*, 2021.

13 Jan Gorzny, Lin Po-An, and Martin Derka. Ideal properties of rollup escape hatches. In *Proceedings of the 3rd International Workshop on Distributed Infrastructure for the Common Good*, pages 7–12, 2022.

14 Vincent Gramoli, Rachid Guerraoui, Andrei Lebedev, Chris Natoli, and Gauthier Voron. Diablo: A benchmark suite for blockchains. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 540–556, 2023.

15 Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1353–1370, 2018.

16 Lucianna Kiffer, Asad Salman, Dave Levin, Alan Mislove, and Cristina Nita-Rotaru. Under the hood of the ethereum gossip protocol. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 437–456. Springer, 2021.

17 Adrian Koegl, Zeeshan Meghji, Donato Pellegrino, Jan Gorzny, and Martin Derka. Attacks on rollups. In *Proceedings of the 4th International Workshop on Distributed Infrastructure for the Common Good*, pages 25–30, 2023.

18 Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 455–471, 2017.

19 MatterLabs. Boojum. https://github.com/matter-labs/era-boojum, 2022. Accessed: 2024-03-19.

20 Shashank Motepalli, Luciano Freitas, and Benjamin Livshits. Sok: Decentralized sequencers for rollups. *arXiv preprint arXiv:2310.03616*, 2023.

21 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

22 Bulat Nasrulin, Martijn De Vos, Georgy Ishmaev, and Johan Pouwelse. Gromit: Benchmarking the performance and scalability of blockchain systems. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 56–63. IEEE, 2022.

23 noir contributors. `noir` zksnark language, 2022. URL: https://aztec.network/aztec-nr/.

24 Paradigm. Flood. https://www.paradigm.xyz/2023/06/flood, 2023. Accessed: 2024-03-19.

25 Daniel Perez and Benjamin Livshits. Broken metre: Attacking resource metering in evm. *arXiv preprint arXiv:1909.07220*, 2019.

26 Polygon. Pil. https://docs.polygon.technology/zkEVM/spec/pil/, 2022. Accessed: 2024-03-19.

27 Polygon. zkasm. https://docs.polygon.technology/zkEVM/spec/zkasm/, 2022. Accessed: 2024-03-19.

28 Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, pages 1–47, 2017.

29 RISC-Zero. Risc zero vm. https://github.com/risc0/risc0, 2022. Accessed: 2024-03-19.

30 RISC-Zero. Zeth. https://www.risczero.com/blog/zeth-release, 2022. Accessed: 2024-03-19.

31 Scroll. halo2 kzg. https://github.com/scroll-tech/halo2, 2022. Accessed: 2024-03-19.

32 Cosimo Sguanci, Roberto Spatafora, and Andrea Mario Vergani. Layer 2 blockchain scaling: A survey. *arXiv preprint arXiv:2107.10881*, 2021.

33 Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A 2 l: Anonymous atomic locks for scalability in payment channel hubs. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1834–1851. IEEE, 2021.

34 Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022.

35 Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

36 Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper*, 2018.

37 Yixuan Zhang, Shuyu Zheng, Haoyu Wang, Lei Wu, Gang Huang, and Xuanzhe Liu. Vm matters: A comparison of wasm vms and evms in the performance of blockchain smart contracts. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2024.

38 Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. Solutions to scalability of blockchain: A survey. *Ieee Access*, 8:16440–16455, 2020.

	ZK-Rollups					
	Polygon ZK-EVM	Scroll	zkSync Era	Starknet	RISC-0	Aztec
VM	zkEVM	zkEVM	zkEVM	General Purpose zkVM	General Purpose zkVM	Privacy-Focused zkVM
Proof System	STARK + FFLONK[1]	Halo2-KZG	Boojum + PLONK-KZG[1]	STARK + FRI	STARK + FRI	HONK + Protogalaxy + Goblin PLONK + UltraPlonk
Published Data	TX Data	TX Data	State Diffs	.	.	.
Compatibility	EVM-Compatible[2]	EVM-Compatible[2]	Solidity-Compatible	.	.	.
Hardware	CPU-based / >128-cores / >1TB RAM	GPU-based / 4 GPUs / >48-cores / >192GB RAM	Many GPU-based / 1 GPU / 16-cores / 64GB RAM	.	.	.

Figure 2 High-level comparison of different ZK-Rollups. Note that Monolithic Circuit-based implementations can be parallelized through recursion on a per-batch level. [1]: In those ZK-EVMs, the last step moves from a STARK-based proof system to a SNARK proof. [2]: Eventually being EVM-equivalent. [3]: L2 Execution fees follow the same metering as in Ethereum. [4]: L2 Execution fees do not follow the same metering as in Ethereum.

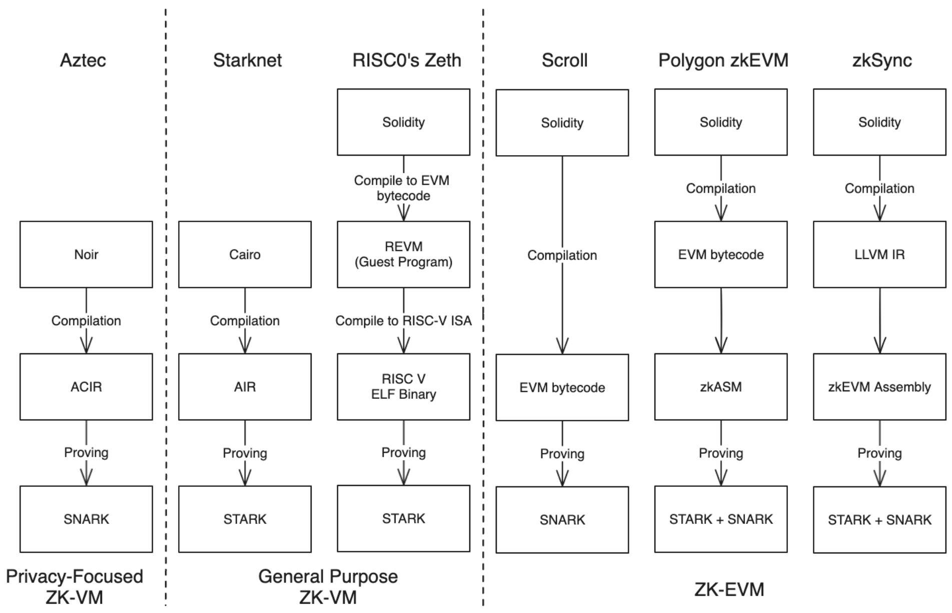

Figure 3 High-level overview of ZK-Rollup compilation pipelines.

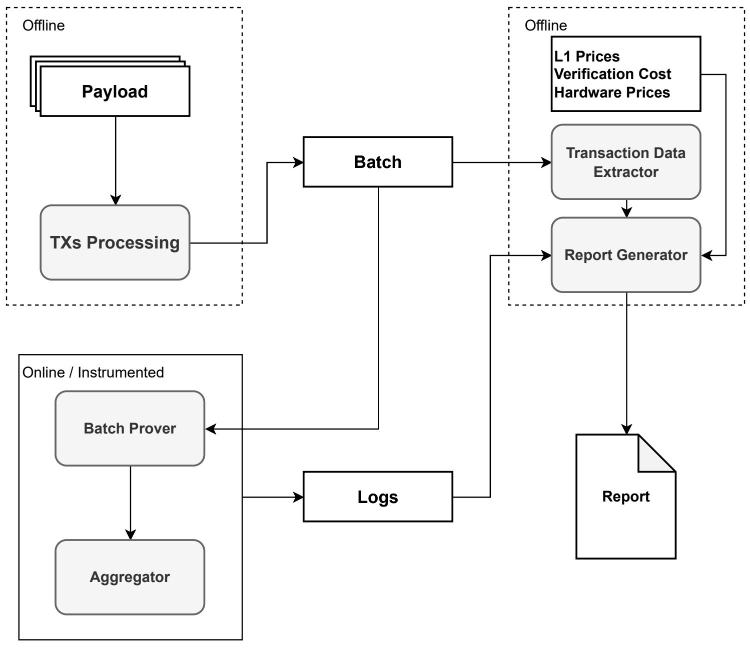

Figure 4 Overview of the benchmarking procedure.

	zkSync Era	Polygon zkEVM
Transactions per batch (median)	4,000	30
Settlement Cost per batch	952,573 GAS ($12.65)	182,343 GAS ($2.4)
Proof Conversion (time/Cloud Cost)	18:00m / $0.56	1:40m / $0.22

Figure 5 Fixed costs of ZK-Rollups. Settlement costs for zkSync Era are: Commit, Prove, and Execute transactions. Settlement costs for Polygon zkEVM are Sequence and Verify. Note that in the zkSync Era, the Execute cost is split between 19 transactions, whereas in Polygon zkEVM, the Sequence is split between 9 transactions, and the Verify cost is between 38. In the settlement cost, we measure the cost per batch. The computations for settlement cost have been made for a gas price of 4.5 Gwei, and ETH at US$2,950. zkSync Era machine cost: US$1.87 per hour. Polygon zkEVM machine cost: US$8.06 per hour. Maybe we should split cloud cost per batch in proof conversion? Maybe we should include proof aggregation cost for polygon conversion. Does Polygon post transaction data in calldata instead of using blobs?

	zkSync Era	Polygon zkEVM
payload 1		

Figure 6 Marginal costs of ZK-Rollups.