We will now conclude on the findings of this work, looking at the viablility of trade aggregation, but also of the scaling implications for other dapps or usecases.

0.1 zkSwap

In general, the approach explored in this work seems to be viable. We were able to build a trustless system, that successfully reduces the gas required for trading. The entire system is also non-custodial, mirroring the properties of decentralized exchanges. The data availability problem has also been solved, reaching the goals set for this work. The gas savings also look encouraging. We're able to reduce the cost per trade in this implementation, increasing the batch size can reduce the gas cost per trade even further. While the smart-contract logic used for verifying aggregation batches can still be optimized for gas efficiency, it could process much larger batch sizes, without many changes. It can not be considered a bottleneck in this system, and is only limited by the Ethereums block gas limit. The bottleneck of the system is the aggregator, and the complexity of computing the proofs for each batch. The approaches to scale the aggregation batches can broadly be separated into two groups.

Reducing Constraint Count The main source of constraints in this system originate from hashing functions, that we mainly need to utilize a Merkle tree to store the systems state. The properties of the Merkle tree data structure (compressed state representation and *O(log(N))* inclusion proofs) make it an ideal match for representing state in layer-2, so its a universal problem that all zk-rollup enabled applications face. New hashing functions like Poseidon look promising in reducing the constraint count per hashing operation, but the security of these functions is still being researched. In some cases hashing needs to be done in a zkSNARK circuit, but also on-chain in a smart-contract. Currently no hashing function exists that is efficient in zk-SNARK circuits and also on-chain. In this system we opted to use SHA256 in these cases, as it can be executed cheaply on-chain. This however results in the doubling of the constraints, resulting in longer runtimes of witness computation and proof generation.

Increasing Constraint Throughput The throughput of an aggregation batch can also be increased by speeding up the proving steps. A large speed up could be achived by parallelizing the proving steps, which would make larger batch sizes viable. As shown by Loopring, there seem to be a number of areas where large efficency gains can be achived.

ZK-rollup as a Whole Considering that there is a clear path for improvement, zk-rollup is a strong contender to scale the Ethereum blockchain to process orders of magnitude more transactions. This work has also shown,

that zk-rollup based applications can be integrated with third-party smart-contracts. While this adds complexities and creates a number of inefficiencies, it has proven to be an effective approach to reduce transaction costs for users and strain on the Ethereum blockchain. An important factor for the viablility are rollup to rollup transactions, which have been conceptualized and are in development at the time of writing. Preserving composability of smart-contracts further strengthens the case for zk-rollups. Proven to be a viable scaling approach, zk-rollups are poised to be adopted more in the future. Continouse improvements of zkSNARK are also to be expected in the future, given that a lot of research is ongoing. A particular development to watch are recursive PLONK constructs, as they would allow an entirly new generation of zk-rollup based applications, enabling interoperability between arbitrary circuits.