Zero Knowledge Proofs towards Verifiable Decentralized AI Pipelines

Nitin Singh, Pankaj Dayama, and Vinayaka Pandit

IBM Research Lab, India
{nitisin1,pankajdayama,pvinayak}@in.ibm.com

Abstract. We are witnessing the emergence of decentralized AI pipelines wherein different organisations are involved in the different steps of the pipeline. In this paper, we introduce a comprehensive framework for verifiable provenance for decentralized AI pipelines with support for confidentiality concerns of the owners of data and model assets. Although some of the past works address different aspects of provenance, verifiability, and confidentiality, none of them address all the aspects under one uniform framework. We present an efficient and scalable approach for verifiable provenance for decentralized AI pipelines with support for confidentiality based on *zero-knowledge proofs* (ZKPs). Our work is of independent interest to the fields of *verifiable computation* (VC) and *verifiable model inference*. We present methods for basic computation primitives like read only memory access and operations on datasets that are an order of magnitude better than the state of the art. In the case of verifiable model inference, we again improve the state of the art for decision tree inference by an order of magnitude. We present an extensive experimental evaluation of our system.

1 Introduction

In this paper we consider a *decentralized* AI pipeline with multiple independent organizations wherein one set of organizations specialize in curating high quality datasets based on independent data sources, another set of organizations specialise in training models from the curated datasets, and another set of organizations deploy the trained models and provide them as a service to the *model consumers*. A typical decentralized AI pipeline is shown in Figure 1. The core *assets* like datasets and models represent significant intellectual property for their respective owners. Therefore, it is essential for the *asset owners* to ensure the confidentiality of their assets beyond the intended usage. On the other hand, since the model consumers are likely to use them for driving major decisions, they would like to ensure *auditability* and *integrity* of the models by (i) verifying the provenance and performance of the models on benchmark datasets [1] and (ii) ensuring that the predictions from the deployed service match with that of the verified model. In summary, decentralized AI pipelines need to provide end to end provenance while ensuring the confidentiality of different assets.

[1] provenance of the model training step is not considered in this paper

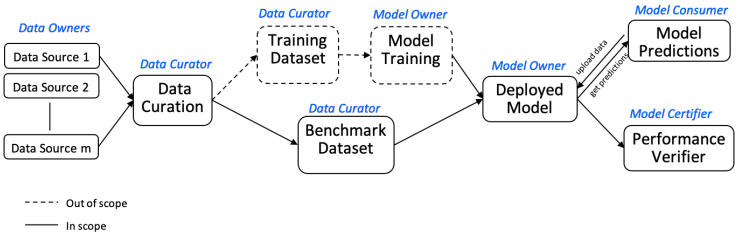

Fig. 1: Typical Decentralized AI Pipeline.

Consider an example of deciding on mortgage applications using an AI service. A data service provider, SP, provides high quality training and benchmark datasets by curating historical mortgage data from reputationally trusted financial institutes. A specialized fintech company, FC, trains and deploys an AI model as a service for the given task. Further, it makes a public claim on the model performance on benchmark dataset. Note that establishing provenance of model training carried out by FC is not addressed in this work. A financial institute, CONS, wanting to use AI in mortgage approval process would want to independently verify the claim made by FC before deciding to subscribe to the service. If CONS is satisfied after the verification process, it might use the deployed service to make decision on mortgage applications. At this time, CONS and individual mortgage applicants should be able to independently verify that the predictions from the deployed service match with that of the verified model. The reputationally trusted data owners and FC would like to protect the confidentiality of their assets except from those actors who are entitled to access them. We would like to highlight a special and important requirement of FC: to prevent model reengineering attacks, the FC would like to ensure that the model verifier does not get to learn the predictions of the models on individual instances during the process of verification.

We present significant progress towards describing efficient and scalable approach to provide public verifiability for common operations in an AI pipeline, while preserving confidentiality of involved data and model assets. In the paper we have highlighted few primitive operations, but more operations on both data and models can be added as state of the art improves. While it is difficult to match the expressiveness of what is possible via plain-text computations, our methods can nevertheless provide provenance over simpler pipelines.

1.1 Related Work

While there is no prior work that addresses all the aspects of verifiable distributed AI pipeline as introduced in this paper, there are past works that address different aspects of the overall requirements. The provenance requirement is addressed in [19, 21], the model verification or certification requirement is addressed in [15, 22], and the verifiable inference from private model requirement is addressed in

[28, 18, 11, 14, 23]. Our work is of independent interest to the field of *Verifiable Computation* (VC) as it provides more efficient methods for useful computational primitives like *Read Only Memory* (ROM) access and operations on datasets. We briefly review and contrast the relevant literature with our work.

Provenance Models for AI: There has recently been considerable interest in the provenance of AI assets. For instance, [19, 21] provide good motivation and DLT based architecture for establishing provenance of AI assets. The provenance is enabled by recording the cryptographic hash of each asset on the tamper-proof ledger, and recording any operations on them as transactions. While this provides auditability and lineage of an asset, its verification necessarily involves revealing the assets, thereby violating the confidentiality requirements in our setting. We build on the tools from verifiable computation to enable verifiability of assets and operations on them while supporting all the stated confidentiality requirements.

Model Certification for AI: Training and testing AI models for fairness and bias is an area of active research. Recently, efforts have been made to leverage methods from *secure multiparty computation* (MPC) to enable fair training and certification of AI models while ensuring privacy of sensitive data of the participants [15, 22]. These methods require a trusted party (e.g. a regulator) to certify the claims on the models and therefore, do not support the public verifiability requirement in our setting.

Verifiable Model Inference: The problem of verifying the predictions from private AI models, with different privacy requirements, has been considered in the literature. For instance, verifiable execution of neural networks has been considered in [14, 18, 23, 27] and verification of predictions from decision trees has been considered in [28]. These works cannot be extended for end to end pipeline verification as they cannot handle verification of operations on datasets. In our work, apart from providing verification for the entire AI pipeline, we improve upon the work of [28] by making the verification of the decision tree inference more scalable as described in Section 1.2.

Reusable Gadgets for VC: On the technical front, our work complements persistent efforts such as [25, 16] to enable more computations efficiently in the VC setting. The problem of efficiently supporting addressable memory inside VC circuits has received considerable attention [3, 5, 25, 16, 31] as many computations are best expressed using the abstraction of memory. Methods in aforementioned efforts support arbitrary *zero knowledge Succinct Arguments of Knowledge* (zkSNARKs). We provide a more efficient variant of prior methods, leveraging a zkSNARK with *commit and prove* capability (see Section 3). However, this is not a major hinderance as many efficient zkSNARKs can be modified to be commit and prove with negligible overhead (see [8]). Our efficient abstractions for read only memory (ROM) and datasets can be incorporated into zkSNARK circuit compilers such as `ZokRates` [10], when suitably targeted for a commit and prove backend. In particular, supporting datasets as first class primitives in zkSNARK compilers will make them more attractive for privacy preserving data science applications. Finally we mention that the work on *Verifiable Outsourced Databases* (e.g. [29], [30]) is not directly applicable here as (i) current implemen-

tations do not address data confidentiality and (ii) they do not support reusable representation of datasets across computations.

1.2 Our Contributions

We present the first efficient and scalable system for decentralized AI pipelines with support for confidentiality concerns of the asset owners (as described in Table 2) and public verifiability. Our work represents major system level innovations in the areas of model certification ([15] - lacks public verifiability, provenance), provenance architectures for AI artifacts ([19, 21] - lack privacy), and confidentiality preserving model inference ([14, 23, 28] - lack provenance). A number of technical contributions enable this system level novelty and they are summarized as follows.

- Improved method for read-only memory access in arithmetic circuits with an order of magnitude gain in efficiency over the existing methods (see Table 3). The improved memory access protocol is crucially used in realizing efficient circuits for data operations (inner-join) and decision tree inference.
- A method for consistent modeling of datasets in arithmetic circuits with complete privacy. In addition, we design efficient circuits to prove common operations on datasets. We make several optimizations over the basic approach of using zkSNARKs resulting in at least an order of magnitude gain in efficiency (see Table 4). On commodity hardware, our implementation scales well to prove operations on datasets with up to 1 million rows in a few minutes. The verification takes few hundred milliseconds.
- We present an improved protocol for privacy preserving verifiable inference from decision tree. Our method yields up to ten times smaller verification circuits by avoiding expensive one-time hashing of the tree used in [28]. Further leveraging our method for read-only memory access, we also incur fewer multiplication gates per prediction (see Section 5 for more details). Comparative performance under different settings is summarized in Table 5.
- We implement our scheme using Adaptive-Pinocchio [24] to experimentally evaluate the efficacy of our scheme. We report the results in Section 6. Our scheme can also be instantiated with other CP-SNARKs.

Our implementation uses pre-processing zkSNARKs [20, 13, 5, 9] which preprocess a circuit description to make subsequent proving and verification more efficient. Our circuits can also be used with generic zkSNARKs such as those in [2, 4, 7], suitably augmented with commit and prove capability.

2 Verifiable provenance in decentralized AI pipelines

A typical AI pipeline consists of different steps, such as accessing raw datasets from multiple sources, performing aggregation and transformations in order to curate training and testing datasets for the AI task on hand, developing the AI model, and deploying it in production. We are interested in settings in which

Operation	Complexity (asymptotic)	Complexity (concrete)	Prov. Time(s)	Ver. Time(ms)
`Aggregation`	$O(N)$	2.1 mil	37	400
`Filter`	$O(N)$	0.7 mil	12	400
`Order-By`	$O(bN)$	3.1 mil	50	400
`Inner-Join`	$O(bN)$	6.5 mil	80	400

Table 1: Performance of our dataset operations. For concrete numbers we took number of rows $N = 100K$ and bit-width of elements $b = 32$.

the AI pipeline is decentralized, i.e, different steps of the pipeline are carried out by different independent actors. We assume five different type of actors: data owners(`DO`), data curators(`DC`), model owners(`MO`), model certifiers(`MCERT`), and model consumers(`MCONS`). For brevity of exposition, we assume that the number of data curators, model owners, model certifiers, and model consumers is just one. However, all the concepts and results extend in a straight forward manner to the general setting involving multiple entities of each type.

We assume that there is a task `T` for which the process of building an AI pipeline is undertaken in a decentralized setting. The salient features of our provenance and certification framework is summarized as follows.

There are m data owners $\mathtt{DO}_1, \mathtt{DO}_2, \ldots, \mathtt{DO}_m$ who share their respective raw datasets $D_1, D_2, \ldots, D_m$ privately with the data curator `DC` and also make a public commitment of the datasets. The data curator curates a dataset $D_b = f(D_1, D_2, \ldots, D_m)$ for the purpose of benchmarking the performance of an AI model for the task `T` and makes a public commitment of D_b. We assume the model owner, `MO`, has a pre-trained AI model M and wants to offer it as a service. `MO` makes a public commitment of the model. `MO` buys the benchmark dataset D_b from `DC`. `MO` wishes to convince potential consumers of the utility of the model M by making performance claim $accuracy = score(M, D_b)$ when M is used for getting predictions on the dataset D_b. The model certifier, `MCERT`, should be able to independently verify the provenance of all the steps and the claimed performance of the model M. `MCERT` also ensures that the timestamp of the public commitment of model M is earlier than the timestamp of public commitment of D_b to ensure that the model M cannot be overfitted to the dataset D_b. `MCERT` certifies the model M only after verifying the correctness of the claim. The model consumer, `MCONS`, subscribes to the model M only upon its successful certification. Suppose `MCONS` supplies a valid input data D' to the service provided by `MO` and gets a prediction Y'. We require that `MCONS` should be able to independently verify that the prediction Y' matches with the prediction of the committed model M on the instance D'.

We observe that the outlined requirements ensure that the decentralized AI pipeline is transparent. The key question we address in this paper is that of providing such a transparency while satisfying the confidentiality requirements of all the actors. We assume that none of the actors in the set up have any incentive to collude with the others, but, can act maliciously. The privacy requirements and security model of different actors is summarized in Table 2.

Participant	Confidentiality Requirement	Security Model
DOs	P1: Only DC can access their plaintext data	S1: Trusted to provide the correct data
DC	P2: Only MO can access curated plaintext data	S2: Not trusted with the correct computation
MO	P3: No one can access the plaintext model P4: During the certification, MCERT cannot get access to prediction of M for any instance in the dataset D_b	S3: Not trusted to make the right performance claim or use the certified model for providing predictions
MCERT	NA	S4: Trusted to certify the model only after end to end provenance is verified
MCONS	P5: No one other than model owner (optional) can access its data in clear	NA

Table 2: Summary of privacy requirements and trust assumptions in our setting.

We present a provenance framework which ensures trust in the AI pipeline by proving each computation step using zero-knowledge proofs, thus meeting all the confidentiality requirements captured in Table 2. Below, we present a concrete example of an AI pipeline for establishing fairness of an AI model, where we clearly highlight involvement of various actors.

2.1 Decentralized Model Fairness

Increasingly, AI models are required to be fair (i.e. non-discriminating) with respect to protected attributes (e.g. Gender). There are several metrics which are used to evaluate a model for fairness. For the sake of illustration, we choose the popular metric called *predictive parity*, which requires a model to have similar accuracy for different values of the protected attribute. In our specific example, our goal is to show that for binary classification model M we have:

$$\left|\Pr[M(\boldsymbol{x}) = y \,|\, \mathsf{Gender}(\boldsymbol{x}) = \mathtt{M}] - \Pr[M(\boldsymbol{x}) = y \,|\, \mathsf{Gender}(\boldsymbol{x}) = \mathtt{F}]\right| \leq \varepsilon$$

where $(\boldsymbol{x}, y) \sim \mathcal{D}$ for representative distribution $\mathcal{D}$. We may estimate the above metric emperically on a test data T consisting of samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$. For concreteness, let M be a decision tree model developed by model owner MO to be used by financial institutions for approving home mortgage loan applications. Let D_1 and D_2 be two *private* datasets consisting of loan applications, which are owned by financial instituions $\mathtt{DO}_1$ and $\mathtt{DO}_2$ respectively. A data curator DC curates the dataset T by concatenating (row-wise) datasets D_1, D_2 and further generates datasets T_M, T_F consisting of applications with *male* and *female* applicants respectively. Finally the model owner MO obtains datsets T_M and T_F and computes the accuracy of its model on the respective datasets. In Figure 2, the top left code block shows the operations executed by different actors in the pipeline without verifiability. The remaining code blocks show operations performed by actors in a verifiable pipeline. The asset owners publicly commit their private assets (bottom left) and generate proofs to attest correctness of

their operations on assets (top right). Finally, a verifier (e.g. auditor) uses published commitments and proofs to establish the correctness of steps performed by respective actors in the pipeline (bottom right).

```python
import pandas as pd
from sklearn.metrics import accuracy_score

# ---------- DATA CURATOR ----------
D1 = get_privately_from_DO1()
D2 = get_privately_from_DO2()
T = D1.concat(D2)
TM = T.query('Gender == M')
TF = T.query('Gender == F')
share_privately_with_MO([TM, TF])

# ---------- MODEL OWNER ----------
M = pd.load_model('HMDA-Decision-Tree.pkl')
(TM, TF) = get_privately_from_DC()
accM = accuracy_score(M, TM)
accF = accuracy_score(M, TF)
share_publicly(abs(accM - accF))
```

```python
from VerifiableML import Prover

# ---------- DATA CURATOR ----------
# Compute curated datasets T, TM, TF as before
# Additionally generate ZKPs for verifiability
proof_T = Prover.prove_concat(D1, D2, T)
proof_TM = Prover.prove_filter(T, 'Gender == M', TM)
proof_TF = Prover.prove_filter(T, 'Gender == F', TF)
share_publicly([proof_T, proof_TM, proof_TF])

# ---------- MODEL OWNER ----------
# Compute accuracies accM and accF as before
# Additionally generate ZKPs for verifiability
proof_accM = Prover.prove_dt_accuracy(M, TM, accM)
proof_accF = Prover.prove_dt_accuracy(M, TF, accF)
share_publicly([proof_accM, proof_accF])
```

```python
from VerifiableML import Commit

# Actors commit their assets
cmM = Commit.commit_model(M)  # MO Commits
share_publicly(cmM)
cmD1 = Commit.commit_data(D1) # DO1 commits
share_publicly(cmD1)
cmD2 = Commit.commit_data(D2) # DO2 commits
share_publicly(cmD2)
[cmT, cmTM, cmTF] = Commit.commit_data([T, TM, TF]) # DC commits
share_publicly([cmT, cmTM, cmTF])
```

```python
from VerifiableML import Verifier

# Verify Data Curator Steps
(cmM, ..., cmTF) = get_public_commitments()
(proof_T, ..., proof_accF) = get_public_proofs()
Verifier.verify_concat(cmD1, cmD2, cmT)
Verifier.verify_filter(cmT, 'Gender == M', cmTM)
Verifier.verify_filter(cmT, 'Gender == F', cmTF)

# Verify Model Owner Steps
Verifier.verify_dt_accuracy(cmM, cmTM, accM, proof_accM)
Veifier.verify_dt_accuracy(cmM,cmTF, accF, proof_accF)
```

Fig. 2: Example pipeline for certifying financial model for fairness.

3 Overview

This section provides overview of the technical challenges in instantiating our solution. More detailed technical contributions appear in Sections 4 and 5.

3.1 Building Blocks

Cryptographic Primitives: We use zkSNARKs as the main cryptographic tool to verify correctness of data operations and model inference while maintaining confidentiality of the respective assets. A zkSNARK consists of a triple of algorithms $(\mathsf{G}, \mathsf{P}, \mathsf{V})$ where (i) G takes description of a computation as an arithmetic circuit C and outputs public parameters $\mathsf{pp} \leftarrow \mathsf{G}(1^\lambda, C)$, (ii) P takes pp and a satisfying instance $(\boldsymbol{x}, \boldsymbol{w})$ for C and outputs a proof $\pi \leftarrow \mathsf{P}(\mathsf{pp}, \boldsymbol{x}, \boldsymbol{w})$ while (iii) V takes pp, statement $\boldsymbol{x}$ and a proof π and outputs $b \leftarrow \mathsf{V}(\mathsf{pp}, \boldsymbol{x}, \pi)$. The proof π reveals no knowledge of the witness $\boldsymbol{w}$, while an accepting proof π implies that prover knows a satisfying assignment $(\boldsymbol{x}, \boldsymbol{w})$ with overwhelming probability. A *commit and prove* zkSNARK (CP-SNARK) allows proving knowledge of witness $\boldsymbol{w}$ as before, where part of $\boldsymbol{w}$ additionally opens a public commitment c, i.e. $\boldsymbol{w} = (\boldsymbol{u}, \boldsymbol{z})$ and $\mathsf{Open}(c) = u$. A CP-SNARK specifies a commitment scheme Com and like a zkSNARK, it provides algorithms G, P and V for generating public parameters, generating proofs and verifying proofs respectively. Additionally, a CP-SNARK allows one to generate proofs over data committed using Com with negligible overhead in proof generation and verification.

Notation: We use the notation $[n]$ to denote the set of natural numbers $\{1, \ldots, n\}$. We often use the array notation $\boldsymbol{x}[\,i\,]$ to denote the i^{th} component of the vector $\boldsymbol{x}$, with 1 as the starting index. We will denote the concatenation of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ as $[\![\boldsymbol{x}, \boldsymbol{y}]\!]$. All our arithmetic circuits, vectors and matrices are over a finite field $\mathbb{F}$ of prime order.

Circuits for Dataset Operations: To use zkSNARKs, we express operations on datasets as arithmetic circuits. At a high level, arithmetic circuits representing data operations accept *datasets* as their inputs and outputs. Since establishing provenance of an asset in an AI pipeline requires verifying operations over several related assets, we require *uniform* representation of datasets across arithmetic circuits, which would allow a dataset to be used as inputs/outputs in different circuits. The second design constraint we enforce is that arithmetic circuits to be *universal*, i.e, the same circuit can be used to verify operations on all datasets within a known size bound. We need universal circuits for two primary reasons: (i) the sizes of datasets are considered confidential and must not be inferable from the circuits being used, and (ii) the circuits can be *pre-processed* to yield efficient verification as it is a frequent operation in our applications.

Dataset Representation in Circuits: As we use the same circuit to represent operations over datasets of varying sizes, we first describe a *uniform* representation of datasets which can be used within the arithmetic circuits. Let N denote a known upper bound on the size of input/output datasets. We view a dataset as a collection of its column vectors (of size at most N). We encode a vector of size at most N as $N+1$ size vector $[\![s, \boldsymbol{X}]\!]$ where $\boldsymbol{X} = (\boldsymbol{X}[1], \ldots, \boldsymbol{X}[N])$ In this encoding s denotes the size of the vector, $\boldsymbol{X}[1], \ldots, \boldsymbol{X}[s]$ contain the s entries of the vector, while $\boldsymbol{X}[i]$ for $i > s$ are set to 0[2]. Similarly, a dataset is encoded by encoding each of its columns separately.

Dataset Commitment: Let Com be a vector commitment scheme associated with a CP-SNARK CP. We additionally assume that Com is homomorphic. To commit a vector $\boldsymbol{x}$, we first compute its encoding $\overline{\boldsymbol{x}}$ as a vector of size $N+1$, and then compute $c = \mathsf{Com}(\overline{\boldsymbol{x}}, r)$ as its commitment. Here r denotes the commitment randomness. To commit a dataset $\boldsymbol{D}$ with columns $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M$, we commit each of its columns to obtain $\boldsymbol{c} = (c_1, \ldots, c_M)$, where $c_i = \mathsf{Com}(\boldsymbol{x}_i)$ as the commitment. Using our circuits with the CP-SNARK CP allows us to efficiently prove operations over committed datasets.

3.2 Optimizations

We now highlight optimizations that are pivotal to the scalability of our system:

Mitigating Commitment Overhead: To prove statements over committed values using general zkSNARKs, one generally needs to compute the commitment as part of the arithmetic circuit expressing the computation. This introduces substantial overhead, when the amount of data to be committed is large.

[2] This introduces no ambiguity if 0 is legitimately part of the vector, as s specifies the content of the vector

To avoid this, we use a CP-SNARK and its associated commitment scheme. We instantiate our system using Adaptive-Pinnochio [24], as the CP-SNARK. Adaptive-Pinnochio augments the popular Pinnochio [20] zkSNARK with commit and prove capability. The resulting scheme incurs $\leq 5\%$ overhead in proof generation time over Pinnochio, while verification continues to be efficient ($\leq$ 400ms) in practice. We expect similar savings with other CP-SNARK schemes, and thus our constructs are agnostic to the choice of CP-SNARK.

Circuit Decomposition: For some operations, verification is more efficient when decomposed as two or more circuits, than when encoded as a monolithic circuit. Let $C(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$ be an arithmetic circuit which checks some property on $(\boldsymbol{x}, \boldsymbol{u})$ where $\boldsymbol{u}$ additionally opens the commitment c. Our decomposition takes the form $C(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \equiv C_1(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w_1}) \wedge C_2(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w}_2)$ where $\boldsymbol{w} = (\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{w}_2)$ denotes a suitable partition of witness wires. Using a CP-SNARK we let the prover provide an additional commitment c_0 for the witness wires $\boldsymbol{w}_0$ which are common to both the sub-circuits. In our decompositions, we let C_1 encode relation that is easily verified by an arithmetic circuit and let C_2 encode the relation which has substantially cheaper probabilistic verification circuit, i.e., there exists a circuit $\widetilde{C}_2(\alpha, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w}_2)$ which takes additional random challenge α and has identical output to C_2 with overwhelming probability (over random choices of α). In our constructions, the latter circuit verifies either the *simultaneous permutation* property or *consistent memory access* property which we introduce below. These are inefficient to check deterministically using arithmetic circuits but admit efficient probabilistic circuits.

3.3 Simultaneous Permutation

We say that tuples $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k)$ and $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ of vectors in $\mathbb{F}^N$ satisfy the *simultaneous permutation* relation if there exists a permutation σ of $[N]$ such that $\boldsymbol{v}_i = \sigma(\boldsymbol{u}_i)$ for all $i \in [k]$. We now describe protocol to check the relation over committed vectors: i.e, given commitments $\mathsf{cu}_1, \ldots, \mathsf{cu}_k, \mathsf{cv}_1, \ldots, \mathsf{cv}_k$ the prover shows knowledge of vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ corresponding to the commitments which satisfy the relation. To achieve this, the verifier first sends a challenge $\beta_1, \ldots, \beta_k$ and challenges the prover to show that β-linear combinations of the vectors $\boldsymbol{u} = \sum_{i=1}^{k} \beta_i \boldsymbol{u}_i$, $\boldsymbol{v} = \sum_{i=1}^{k} \beta_i \boldsymbol{v}_i$, corresponding to commitments $\mathsf{cu} = \sum_{i=1}^{k} \beta_i \mathsf{cu}_i$, $\mathsf{cv} = \sum_{i=1}^{k} \beta_i \mathsf{cv}_i$ are permutations of each other. This is accomplished via a further challenge $\alpha \leftarrow \mathbb{F}$ and subsequently chekcing $\prod_{i=1}^{N}(\alpha - \boldsymbol{u}[i]) = \prod_{i=1}^{N}(\alpha - \boldsymbol{v}[i])$. We describe the formal protocol and its analysis in Appendix C.1. The last computation can be expressed in an arithmetic circuit using $O(N)$ multiplication gates which is concretely more efficient compared to deterministic circuits for checking permutation relation using routing networks [6, 26].

3.4 Consistent Memory Access

We define *consistent memory access* relation for a triple of vectors $\boldsymbol{L}, \boldsymbol{U}$ and $\boldsymbol{V}$ where $\boldsymbol{L} \in \mathbb{F}^n$ and $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{F}^m$ for some integers m, n. We say that $(\boldsymbol{L}, \boldsymbol{U}, \boldsymbol{V})$

	Circuit Complexity	Circuit Complexity (m=n=10000)	Backend
Linear Scan	$2mn$	200 mill	ZK
Routing Networks [6, 26]	$(m+n)(3\log(m+n)+3\log m)$	5.7 mill	ZK
Buffet [25]	$m(21+2\log n+10\log m)$	1.9 mill	ZK
xJSNARK [16]	$m(2\sqrt{n}+\log n)$	2.1 mill	ZK
Our Work	$5(m+n)$	0.1 mill	CP

Table 3: Comparison of Circuit Complexity for different ROM approaches. ZK and CP denote zkSNARK and CP-SNARK protocols. m and n denote number of reads and memory size respectively.

satisy the relation if $V[\,i\,] = L[\,U[i]\,]$ for all $i \in [m]$. We think of $\boldsymbol{L}$ as *read only memory* (ROM) which is accessed at locations given by $\boldsymbol{U}$ with $\boldsymbol{V}$ being the corresponding values. We adapt the techniques in [3, 5, 25, 31] to take advantage of CP-SNARKs in our construction. Next, we present a protocol to check the relation given commitments to $\boldsymbol{L}, \boldsymbol{U}$ and $\boldsymbol{V}$. The verification proceeds as:

1. First $m+n$ sized vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are computed as follows: For the vector $\boldsymbol{u}$ we require $\boldsymbol{u}[\,i\,] = i$ for $i \in [n]$ and $\boldsymbol{u}[\,i+n\,] = \boldsymbol{U}[\,i\,]$ for $i \in [m]$. For the vector $\boldsymbol{v}$ we require $\boldsymbol{v}[\,i\,] = \boldsymbol{L}[\,i\,]$ for $i \in [n]$ and $\boldsymbol{v}[\,i+n\,] = \boldsymbol{V}[\,i\,]$ for $i \in [m]$ (see Figure 3).
2. The prover also supplies auxiliary vectors $\tilde{\boldsymbol{u}}$ and $\tilde{\boldsymbol{v}}$ of size $m+n$, where $\tilde{\boldsymbol{u}}$ and $\tilde{\boldsymbol{v}}$ are purportedly obtained from $\boldsymbol{u}$ and $\boldsymbol{v}$ via the same permutation.
3. Finally, we ensure that the vector $\tilde{\boldsymbol{u}}$ is sorted and that the vector $\tilde{\boldsymbol{v}}$ differs in adjacent positions only if the same is true for those positions in vector $\tilde{\boldsymbol{u}}$.

The constraints on the first n entries of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ in step (1) can be thought of as "loading" constraints that load the entries of $\boldsymbol{L}$ against corresponding address in memory, while constraints on the last m entries can be thought of as "fetching" constraints that fetch the appropriate value against the specified memory location. The steps (2) and (3) ensure that the value fetched for a given location is same as the value loaded against it during the initial loading steps. We decompose above checks across two circuits. The first arithmetic circuit $\mathsf{C}_{\mathsf{ROM},m,n}$ ensures steps (1) and (3) while the second circuit checks that vectors $\tilde{\boldsymbol{u}}, \tilde{\boldsymbol{v}}$ are obtained by applying the same permutation to vectors $\boldsymbol{u}, \boldsymbol{v}$ respectively. The circuit $\mathsf{C}_{\mathsf{ROM},m,n}$ can be realized using $O(m+n)$ multiplication gates. Generally, verifying that a vector such as $\tilde{\boldsymbol{u}}$ is sorted in step (3) incurs logarithmic overhead due to the need for bit decomposition of each element. However, we can leverage the fact that $\tilde{\boldsymbol{u}}$ is a (sorted) rearrangement of $\boldsymbol{u}$, which includes all elements of $[n]$ by construction. Thus, monotonicity of $\tilde{\boldsymbol{u}}$ is established provided (i) $\tilde{\boldsymbol{u}}[\,n\,] = 1$, (ii) $\tilde{\boldsymbol{u}}[\,m+n\,] = n$ and $\tilde{\boldsymbol{u}}[\,i+1\,] - \tilde{\boldsymbol{u}}[\,i\,] \in \{0,1\}$ for all $1 \le i \le m+n-1$, which together require $O(m+n)$ gates to verify. Finally, we invoke the protocol for "Simultaneous Permutation" property in Section 3.3 to check compliance of step (2). We illustrate the verification circuit and the decomposition in Figure 3. The formal protocol and analysis appears in Appendix C.2. Overall we incur $O(m+n)$ gates, which is more efficient than encoding entire relation in one circuit. In that case one uses routing networks which incur $O((m+n)\log(m+n))$

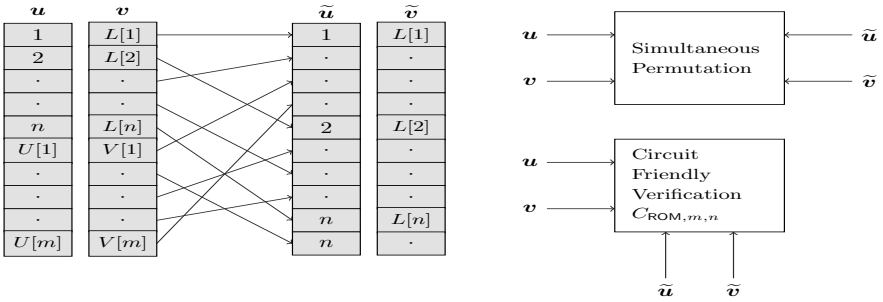

Fig. 3: Consistent Memory Access

gates and are concretely much more expensive. We can optimize further when the same access pattern is used for accessing different ROMs as described below.

Multiplexed Memory Access For access pattern $\boldsymbol{U} \in \mathbb{F}^m$ and ROMs $\boldsymbol{L}_j \in \mathbb{F}^n$ for $j \in [k]$, we can show the correctness of lookup values $\boldsymbol{V}_j[\,i\,] = \boldsymbol{L}_j[\boldsymbol{U}[i]\,]$, $i \in [M], j \in [k]$ using just one instance of protocol discussed in this section. To achieve this, the verifier sends a random challenge $\alpha_1, \ldots, \alpha_k$ to the prover. The prover then shows that $(\boldsymbol{L}, \boldsymbol{U}, \boldsymbol{V})$ satisfy correct memory access where $\boldsymbol{L} = \alpha_1\boldsymbol{L}_1 + \cdots + \alpha_k\boldsymbol{L}_k$ and $\boldsymbol{V} = \alpha_1\boldsymbol{V}_1 + \cdots + \alpha_k\boldsymbol{V}_k$ for uniformly sampled $\alpha_1, \ldots, \alpha_k$. Note that due to the homomorphism of the commitment scheme, both the prover and the verifier can compute the commitments for $\boldsymbol{L}, \boldsymbol{U}$ and $\boldsymbol{V}$.

3.5 Our Techniques in Perspective

Commit and prove functionality in conjunction with zero knowledge proofs has been used in recent works addressing privacy in machine learning, most notably in [18, 27, 28]. In [18] and [28], CP-SNARKs are used to "link" proofs of correctness for different parts of the circuit (similar to Circuit Decomposition in our setting) to prove inference from a private neural network and a decision tree respectively. In [27], public commitments are linked to set of *authenticated inputs* between a prover and a verifier in a two party protocol. Subsequently the prover produces a ZK proof showing correctness of neural network inference over authenticated inputs. In contrast, our usage of CP-SNARKs is more pervasive. We first optimize key relations (simultaneous permutation, consistent memory access) for CP-SNARKs and then design our dataset representation in a way that allows us to represent operations on them in terms of aforementioned relations.

4 Privacy Preserving Dataset Operations

We now describe protocols for common dataset operations such as `aggregation`, `filter`, `order-by`, `inner-join` etc. These operations serve to illustrate our key techniques, which can be further applied to yeild protocols for much more comprehensive list of dataset operations. We use the fact that most of the operations distribute nicely as identical computations over different pairs of columns.

Throughout this section, N denotes the upper bound on the sizes of input/output datasets.

Aggregation: Aggregation operation takes two datasets as inputs and outputs their row-wise concatenation. We first describe arithmetic circuit to verify the concatenation of vectors. The circuit accepts three vectors in their uniform representation as discussed in Section 3.1. Let $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ be three vectors of size at most N represented as $[\![s, \boldsymbol{X}]\!]$, $[\![t, \boldsymbol{Y}]\!]$ and $[\![w, \boldsymbol{Z}]\!]$ respectively where $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ are vectors of size N. The verification involves ensuring that the first w entries of $\boldsymbol{Z}$ contain the first s entries of $\boldsymbol{X}$ and the first t entries of $\boldsymbol{Y}$. Figure 4 illustrates the setting for $s = 3$, $t = 4$, $w = 7$ and $N = 9$. To aid the verification, the prover provides N-length binary vectors $\boldsymbol{\rho}_s, \boldsymbol{\rho}_t$ and $\boldsymbol{\rho}_w$ as auxiliary inputs. The vector $\boldsymbol{\rho}_s$ is 1 in its first s entires, and 0 elsewhere. Similar relation is satisfied by $\boldsymbol{\rho}_t$ and $\boldsymbol{\rho}_w$. The correctness of aggregation now reduces to showing that there is a permutation that simultaneously maps $[\![\boldsymbol{\rho}_s, \boldsymbol{\rho}_t]\!]$ to $[\![\boldsymbol{\rho}_w, \boldsymbol{0}]\!]$ and $[\![\boldsymbol{X}, \boldsymbol{Y}]\!]$ to $[\![\boldsymbol{Z}, \boldsymbol{0}]\!]$. Figure 4 also shows how the verification is decomposed: The first circuit checks that (i) $w = s + t$, (ii) vectors $\boldsymbol{\rho}_s, \boldsymbol{\rho}_t, \boldsymbol{\rho}_t$ are correctly provided and (iii) ensures $\boldsymbol{u}_1 = [\![\boldsymbol{\rho}_s, \boldsymbol{\rho}_t]\!]$, $\boldsymbol{v}_1 = [\![\boldsymbol{X}, \boldsymbol{Y}]\!]$, $\boldsymbol{u}_2 = [\![\boldsymbol{\rho}_w, \boldsymbol{0}]\!]$ and $\boldsymbol{v}_2 = [\![\boldsymbol{Z}, \boldsymbol{0}]\!]$. The second circuit checks the "simultaneous permutation" property on the pairs $(\boldsymbol{u}_1, \boldsymbol{v}_1)$ and $(\boldsymbol{u}_2, \boldsymbol{v}_2)$. Both the circuits can be realized using $O(N)$ multiplication gates. Using a CP-SNARK we can verify the correctness of aggregation of vectors over commitments.

We now leverage the above construction to verify aggregation operation over datasets. Let D_x, D_y and D_z be datasets each with k columns given by $(\boldsymbol{x}_i)_{i=1}^k, (\boldsymbol{y}_i)_{i=1}^k$ and $(\boldsymbol{z}_i)_{i=1}^k$ respectively. The reduction technique involves the verifier sampling random $\alpha_1, \ldots, \alpha_k$ satisfying $\alpha_1 + \cdots + \alpha_k = 1$. Next, we use the above circuit construction with a CP-SNARK to prove that vectors $\boldsymbol{x} = \sum_{i=1}^k \alpha_i \boldsymbol{x}_i$,$\boldsymbol{y} = \sum_{i=1}^k \alpha_i \boldsymbol{y}_i$ and $\boldsymbol{z} = \sum_{i=1}^k \alpha_i \boldsymbol{z}_i$ satisfy the concatenation property. We give complete protocol and proof of the reduction in the Appendix C.3.

Filter: Filter operation involves a dataset and a selection predicate as inputs and subsequently outputs a dataset consisting of subset of rows satisfying the predicate. We divide the computation in two parts (i) Applying selection predicate to rows of the dataset to obtain a binary vector $\boldsymbol{f}$ which we call as *selection vector* and (ii) Applying selection vector to the source dataset to obtain the target dataset. The latter computation can be verified with techniques similar to those used in `aggregation` operation. For the first computation, we describe an efficient circuit for predicates of the form $\wedge_{i=1}^k (\boldsymbol{x}_i == v_i)$ where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ are the columns of the dataset. Once again the verifier chooses random $\alpha_1, \ldots, \alpha_k$ with $\sum_{i=1}^k \alpha_i = 1$ and challenges the prover to show that the selection vector $\boldsymbol{f}$ satisfies $\boldsymbol{f} = (\boldsymbol{x} == v)$ where $\boldsymbol{x} = \sum_{i=1}^k \alpha_i \boldsymbol{x}_i$ and $v = \sum_{i=1}^k \alpha_i v_i$. The relation $\boldsymbol{f} = (\boldsymbol{x} == v)$ can be verified using a circuit with $O(N)$ gates. Due to the homomorphism of the commitment scheme, the verifier can compute the commitment for vector $\boldsymbol{x}$ given the commitments to columns of the dataset. For more general range queries of the form $\wedge_{i=1}^k (\ell_i < \boldsymbol{x}_i \leq r_i)$, we can compute selection vector $\boldsymbol{f}_i$ for each column, and then compute the final selection vector $\boldsymbol{f} = \wedge_{i=1}^k \boldsymbol{f}_i$.

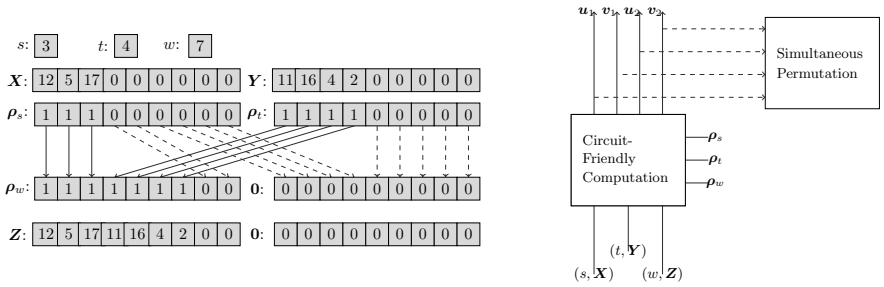

Fig. 4: Circuit for verifying vector concatenation

Order By: Order-By relation involves permuting the rows of the dataset so that a specified column is in sorted order. The verification can be naturally expressed as columns of source and target dataset satisfying simultaneous permutation relation, where additionally the specified column is sorted. We can check the monotonicity of a column using a circuit with $O(bN)$ gates where b is the bit-width of the range of values in the column. We skip the details.

Inner-Join: Inner join operation concatenates pairs of rows of input datasets which have identical value for the designated columns (joining columns). We consider the inner-join operation under the restriction that the joining columns have distinct values. As a first step, we order both the input datasets so that the joining columns are sorted. We can use the verification protocol for `order-by` operation to ensure correctness of this step. We therefore assume that joining columns are sorted, and take distinct values. Let D_1 and D_2 be two datasets which are joined on columns $\boldsymbol{x}$ and $\boldsymbol{y}$ to yield the dataset D. We write D as juxtaposition of columns $[D_1^{'}, \boldsymbol{z}, D_2^{'}]$ where $D_i^{'}$ denotes the columns coming from D_i while $\boldsymbol{z}$ denotes the column obtained as intersection of $\boldsymbol{x}$ and $\boldsymbol{y}$. We first design sub-circuit for *private set intersection* (PSI) to compute the size w of the resulting dataset. We then let the prover provide auxiliary selection vectors $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ of size w. Finally, using the circuit for `filter` relation, we verify that $\boldsymbol{f}_1$ applied to D_1 yields dataset $D_L = [D_1^{'}, \boldsymbol{z}]$ and $\boldsymbol{f}_2$ applied to D_2 yields the dataset $D_R = [D_2^{'}, \boldsymbol{z}]$. The overall circuit complexity is $O(bN)$ where b is the bit-width of the range of values in $\boldsymbol{x}$ and $\boldsymbol{y}$ with set-intersection computation dominating the overall cost.

5 Privacy Preserving Model Inference: Decision Trees

In this section we present a zero knowledge protocol for verifiable inference from decision trees (and random forests). Decision trees are popular models in machine learning due to their interpretability. A decision tree recursively partitions the feature space (arranged as a tree), and finally assigns a label to each leaf segment. The problem of proving correct inference from a decision tree was considered recently in [28], where authors present a privacy preserving method for an adversary to commit to a decision tree and later prove inference from the tree

on public test data. We present a new construction based on *consistent memory access*, which improves upon the prior construction by reducing the number of multiplication gates in the inference circuit. We also provide zero knowledge protocol for establishing the accuracy of a decision tree on test data. We consider variants with test data being public or private. The latter scenario is helpful while verifying performance of a private model on reputationally trusted private dataset.

Decision Tree Representation: We parameterize a binary decision tree with following parameters: the maximum number of nodes (N), the maximum length of a decision path (h) and maximum number of features used as predictors (d). We assume that the nodes in the decision tree have unique identifiers from the set $[N]$, while features are identified using indices in set $[d]$. We naturally represent a decision tree $\mathcal{T}$ as a lookup table with five columns, i.e, $\mathcal{T} = (\boldsymbol{V}, \boldsymbol{T}, \boldsymbol{L}, \boldsymbol{R}, \boldsymbol{C})$, where each column vector is of size N. For a decision tree with $t \leq N$ nodes, we encode as follows: For $i \in [t]$:

- $\boldsymbol{V}[\,i\,]$ denotes the identifier for the splitting feature for i^{th} node.
- $\boldsymbol{T}[\,i\,]$ denotes the threshold value for the splitting feature for i^{th} node.
- $\boldsymbol{L}[\,i\,]$ and $\boldsymbol{R}[\,i\,]$ denote the identifiers for the left and right child of i^{th} node. In case of a leaf node, this value is set to i itself.
- $\boldsymbol{C}[\,i\,]$ denotes the label associated with the i^{th} node, when it is a leaf node. For non-leaf nodes this may be set arbitrarily.

We commit to a decision tree, by committing to each of the vectors. We define $\mathsf{cm}_{\mathcal{T}} = (\mathsf{cm}_V, \mathsf{cm}_T, \mathsf{cm}_L, \mathsf{cm}_R, \mathsf{cm}_C)$ as the commitment to $\mathcal{T}$.

Decision Tree Inference: We model the test data D as $n \times d$ matrix, consisting of n d-dimensional samples. Let $\boldsymbol{D}$ be the vector of size dn obtained by flattening D in row major order. The algorithm below computes decision paths $\boldsymbol{p}_i = (\boldsymbol{p}_i[\,1\,], \ldots, \boldsymbol{p}_i[\,h\,])$ for each sample $i \in [n]$. The prediction vector $\boldsymbol{q}$ contains class labels corresponding to leaf nodes $\boldsymbol{p}_i[\,h\,]$ for $i \in [n]$.

1. For $i = 1, \ldots, n$ do:
   - Set $\boldsymbol{p}_i[\,1\,] = 1$: root is the first node on every decision path.
   - For $j = 1, \ldots, h$ determine next node as follows:
     (a) Compute splitting feature: $\boldsymbol{f}_i[\,j\,] = \boldsymbol{V}[\,\boldsymbol{p}_i[j]\,]$.
     (b) Compute threshold value: $\boldsymbol{t}_i[\,j\,] = \boldsymbol{T}[\,\boldsymbol{p}_i[j]\,]$.
     (c) Compute left and right child id: $\boldsymbol{l}_i[\,j\,] = \boldsymbol{L}[\,\boldsymbol{p}_i[j]\,]$, $\boldsymbol{r}_i[\,j\,] = \boldsymbol{R}[\,\boldsymbol{p}_i[j]\,]$.
     (d) Compute label: $\boldsymbol{c}_i[\,j\,] = \boldsymbol{C}[\,\boldsymbol{p}_i[j]\,]$.
     (e) Compute $\hat{\boldsymbol{f}}_i[\,j\,] = d * i + \boldsymbol{f}_i[\,j\,]$.
     (f) Compute value of splitting feature: $\boldsymbol{v}_i[\,j\,] = D[\,i, \boldsymbol{f}_i[j]\,] = \boldsymbol{D}[\,\hat{\boldsymbol{f}}_i[j]\,]$.
     (g) Compute next node: $\boldsymbol{p}_i[j+1] = \boldsymbol{l}_i[\,j\,]$ if $\boldsymbol{v}_i[\,j\,] \leq \boldsymbol{t}_i[\,j\,]$ and $\boldsymbol{r}_i[\,j\,]$ otherwise.
   - Compute label for the sample: $\boldsymbol{q}[\,i\,] = \boldsymbol{c}_i[\,h\,]$.

Verification of the above algorithm involves verifying (i) hn memory accesses on the tables of $\mathcal{T}$ in steps (a)-(d), which share the access pattern $\boldsymbol{p}_i[\,j\,]$, (ii) verifying hn memory accesses on $\boldsymbol{D}$ (of size dn) in step (f) and (iii) hn comparisons as part of step (g). Using the optimization in Section 3.4, the first verification incurs $O(N + hn)$ multiplication gates, while the second verification incurs $O(dn + hn)$

multiplication gates. Using standard techniques, verification of (iii) can be made using $O(whn)$ multiplication gates, where w is the bit-width of feature values. Thus, overall circuit complexity of our solution is $O(N + n(d + h + wh))$. We compare our solution with the method for zero-knowledge decision tree (zkDT) inference presented in [28]. Broadly, the method in [28] establishes the correctness of inference as three checks:

- Consistency of input decision tree with public commitment: This involves $O(N)$ evaluations of the hash function $\mathcal{H}$ used for commitment and thus incurs $c(\mathcal{H}) \cdot N$ multiplication gates. Here $c(\mathcal{H})$ denotes the size of circuit required to evaluate $\mathcal{H}$.
- Consistency of feature vector with decision path: The verification of this step leverages a "Multiset Check" ([28, Section 4.1]) which costs $O(d \log h)$ multiplication gates per sample.
- Correct evaluation of decision tree function: It involves h comparisons for each sample, which incurs hw mutliplication gates, where w is the bit-width of feature values.

Above steps result in an overall circuit complexity of $c(\mathcal{H})N + n(3d \log h + hw)$ for zkDT. Our solution improves upon the approach in [28] by reducing the cost of the first two checks. Using a CP-SNARK, we avoid the cost of computing the commitment within the verification circuit, while using our optimized protocols for memory access allows us to accomplish the second check with an average cost of $O(h + d)$ gates per sample ($O(dn + hn)$ overall), which compares favorably with per sample cost of $O(d \log h)$ incurred by zkDT for $h = \Theta(d)$. The concrete improvement obtained using our approach depends on which of the three checks dominate the cost for specific parameter settings. We compare the cost of the two approaches for some representative parameter settings in Table 5.

Decision Tree Accuracy: The above circuit for decision tree inference can be easily modified to yield the circuit for proving accuracy of a decision tree on test data. In this case, the prediction vector is kept private, and tallied against ground truth to compute accuracy. Since our system also includes verifiability of model performance (accuracy) on *private* benchmark datasets, we briefly describe the modifications required to achieve the same. Let D be a private dataset with columns $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d)$ with commitments to columns being public. Since, we can no longer compute the flattened vector $\boldsymbol{D}$ as before, we cannot verify the lookup $\boldsymbol{v}_i[\,j\,] = \boldsymbol{D}[\,\hat{\boldsymbol{f}}_i[j]\,]$. Instead we use polynomial interpolation to pre-process D. For i^{th} row $D[i, \cdot]$ of the original data (a vector of size d), we interpolate a polynomial p_i of degree $d-1$ such that $p(j) = D[i, j]$. We obtain the pre-processed dataset D' whose i^{th} row consists of coefficients of p_i. The data owner makes a commitment to D' instead of D. The lookup $\boldsymbol{v}_i[\,j\,] = D'[i, j] = p_i(j)$ now involves evaluating a $d-1$ degree polynomial which incurs d multiplication gates. The overall circuit complexity for accuracy over private datasets is therefore $O(N+hn+hnw+hnd)$.

	No Optimization	Partial Optimization	Full Optimization
`Aggregation`	19.3 mil	1.6 mil	0.21 mil
`Filter`	12.5 mil	0.7 mil	0.07 mil
`Inner-Join`	22.1 mil	4.4 mil	0.65 mil

Table 4: Measuring the efficacy of our optimizations on 100K$\times$ 10 datasets.

Test Data Size (n)	T1=(1000,50,20)		T2=(10000,35,25)	
	Our Work	zkDT [28]	Our Work	zkDT [28]
100	0.11 mil	3.1 mil	0.16 mil	30.1 mil
1000	1 mil	4.3 mil	1.2 mil	31 mil
10000	9.5 mil	16.5 mil	11.5 mil	41 mil

Table 5: Comparison of Circuit Complexity for decision tree inference.

6 Experimental Evaluation

In this section we report the concrete performance of our system primitives. For our implementation, we used Adaptive Pinocchio [24] as the underlying CPSNARK, which we implemented using the `libsnark` [17] library. We also used the `libsnark` library for our circuit descriptions. Our experiments were performed on Ubuntu Linux 18.04 cloud instances with 8 Intel Xeon 2.10 GHz virtual cpus with 32GB of RAM. The experiments were run with finite field arithmetic libraries and FFT libraries compiled to exploit multiple cores. We often use circuit complexity (multiplication gates in the circuit) as the "environment neutral" metric for comparing different approaches (the proving times scale quasi-linearly with circuit complexity).

Performance of Dataset Operations: Table 1 contains summary of asymptotic as well as concrete efficiency of our dataset operations. All the operations scale linearly with the number of rows (with marginal additive dependence on the number of columns). The numbers for proof generation and verification were generated for representative dataset size of $100K \times 10$. While proof generation is an expensive operation by general standards, it is practical enough for infrequent usage. We also tabulate the efficacy of our optimizations in Table 4. For the unoptimized case, we do not use CP-SNARKs and instead compute commitments using circuit-friendly MiMC hash [1]. For partially optimized case, we use native commitment scheme of CP-SNARK for commitments, but use monolithic circuits to encode the operations. To express permutations in monolithic circuits, we use gadgets for routing networks [6, 26] available in [17]. The fully optimized version delegates permutation checking and memory access check to probabilistic circuits as discussed in Section 3.2. In the first case, hashing dominates the circuit complexity resulting in 50-100 times larger circuits. Decomposing the circuits instead of monolithic circuits also results in an order of magnitude savings.

Performance of Decision Tree Inference: We use two decision trees $T1$ and $T2$ to benchmark performance of our decision tree inference implementation. We also use the same trees to compare our method with the one presented in [28]. We

Test Data Size (n)	T1=(1000,50,20)		T2=(10000,35,25)	
	Prov.Time(s)	Ver.Time(ms)	Prov.Time(s)	Ver.Time(ms)
100	1	400	1	400
1000	5	400	6	400
10000	170	400	200	400

Table 6: Concrete proving and verification time for decision tree inference.

synthetically generate the tree $T1$ with 1000 nodes, 50 features and depth as 20, which roughly corresponds to the largest tree used in [28]. The tree $T2$ is trained on a curated version of dataset [12] for Home Mortgage Approval. We identify 35 features from the dataset to train binary decision tree. We train $T2$ with 10000 nodes and depth 25. We verify the inference from the two trees for batch sizes of 100 (small), 1000 (medium) and 10000 (large). Using our method to generate proof of predictions takes from few seconds (on small data) to few minutes (on large data), as seen in Table 6. The circuit complexity and the proving time scale almost linearly for our method. We also compare the multiplication gates incurred by arithmetic circuits in our method with that in [28] in Table 5. Our efficiency is an order of magnitude better for smaller data sizes, as we do not incur one time cost for hashing the tree. For larger batch sizes, our method is still about 1.5-4× more efficient. As the batch sizes get large, comparisons dominate the circuit complexity in both the approaches. We report the circuit complexity for proving the accuracy for decision trees on private datasets and public datasets. Table 7 shows that the overhead for proving accuracy on private datasets ranges from $50 - 80\%$.

Performance of Memory Access: We also independently benchmark the performance of our memory abstraction technique and compare it to existing methods in Table 3. Leveraging CP-SNARKs and probabilistic reductions we essentially incur *constant* number of gates per access. We compare different approaches both in terms of asymptotic complexity and concrete complexity for parameter settings representative of their usage in our work. Our concrete efficiency is an order of magnitude better than the alternatives considered.

Test Data Size (n)	T1=(1000,50,20)		T2=(10000,35,25)	
	Public	Private	Public	Private
100	0.11 mil	0.18 mil	0.16 mil	0.23 mil
1000	1 mil	1.75 mil	1.2 mil	1.8 mil
10000	9.5 mil	17.4 mil	11.5 mil	18 mil

Table 7: Circuit Complexity for decision tree accuracy for public and private benchmark datasets.

References

1. M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *Proceedings of the 22nd International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 191–219, 2016.
2. S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2087–2104, 2017.
3. E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *Proceedings of the 33rd Annual Cryptology Conference (CRYPTO)*, volume 8043, pages 90–108, 2013.
4. E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, volume 11476, pages 103–128, 2019.
5. E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium*, pages 781–796, 2014.
6. V. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic*. ISSN. Elsevier Science, 1965.
7. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.
8. M. Campanelli, D. Fiore, and A. Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the ACM SI)GSAC Conference on Computer and Communications Security (CCS)*, pages 2075–2092, 2019.
9. A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, volume 12105, pages 769–793, 2020.
10. J. Eberhardt and S. Tai. Zokrates - scalable privacy-preserving off-chain computations. In *Proceedings of the IEEE International Conference on Internet of Things (iThings)*, pages 1084–1091, 2018.
11. B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu. ZEN: efficient zero-knowledge proofs for neural networks. *IACR Cryptology ePrint Archive*, 2021:87, 2021.
12. ffiec. Home mortgage disclosure act. https://ffiec.cfpb.gov/data-publication/snapshot-national-loan-level-dataset/2018. Accessed: 2021-09-14.
13. R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, volume 7881, pages 626–645, 2013.
14. Z. Ghodsi, T. Gu, and S. Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS*, pages 4672–4681, 2017.
15. N. Kilbertus, A. Gascón, M. J. Kusner, M. Veale, K. P. Gummadi, and A. Weller. Blind justice: Fairness with encrypted sensitive attributes. In *Proceedings of the*

35th International Conference on Machine Learning (ICML), pages 2635–2644, 2018.

16. A. E. Kosba, C. Papamanthou, and E. Shi. xjsnark: A framework for efficient verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 944–961, 2018.
17. S. Lab. libsnark: A C++ library for zkSNARK proofs, howpublished = https://github.com/scipr-lab/libsnark, note = Accessed: 2021-09-14.
18. S. Lee, H. Ko, J. Kim, and H. Oh. vcnn: Verifiable convolutional neural network. *IACR Cryptology ePrint Archive*, 2020:584, 2020.
19. P. Lüthi, T. Gagnaux, and M. Gygli. Distributed ledger for provenance tracking of artificial intelligence assets. *CoRR*, abs/2002.11000, 2020.
20. B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 238–252, 2013.
21. K. K. Sarpatwar, R. Vaculín, H. Min, G. Su, T. Heath, G. Ganapavarapu, and D. Dillenberger. Towards enabling trusted artificial intelligence via blockchain. In *Extended papers from the Second International Workshop on Policy-based Autonomic Data Governance*, volume 11550, pages 137–153, 2018.
22. S. Segal, Y. Adi, B. Pinkas, C. Baum, C. Ganesh, and J. Keshet. Fairness in the eyes of the data: Certifying machine-learning models. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, pages 926–935, 2021.
23. F. Tramèr and D. Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.
24. M. Veeningen. Pinocchio-based adaptive zk-snarks and secure/correct adaptive function evaluation. In M. Joye and A. Nitaj, editors, *Proceedings of the 9th International Conference on Cryptology in Africa (AFRICACRYPT)*, volume 10239, pages 21–39, 2017.
25. R. S. Wahby, S. T. V. Setty, Z. Ren, A. J. Blumberg, and M. Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS)*, 2015.
26. A. Waksman. A permutation network. *Journal of the ACM*, 15(1):159–163, 1968.
27. C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang. Mystique: Efficient conversions for Zero-Knowledge proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 501–518, 2021.
28. J. Zhang, Z. Fang, Y. Zhang, and D. Song. Zero knowledge proofs for decision tree predictions and accuracy. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2039–2053, 2020.
29. Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vsql: Verifying arbitrary SQL queries over dynamic outsourced databases. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 863–880, 2017.
30. Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. A zero-knowledge version of vsql. *IACR Cryptology ePrint Archive*, 2017:1146, 2017.
31. Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vram: Faster verifiable RAM with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 908–925, 2018.

A Preliminaries

We briefly summarise some key cryptographic notions that we use throughout the paper. For more details on the notions discussed below, we refer the reader to [8, Section 2].

A.1 Commitment Scheme

Definition 1. *A commitment scheme* $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCommit})$ *is a tuple of algorithms with message space* $\mathcal{D}$*, commitment space* $\mathcal{C}$ *and opening space* $\mathcal{O}$ *which satisfies* correctness*,* hiding *and* binding *as described below:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ *takes security parameter* λ *and outputs commitment key* ck*.*
- $\mathsf{Commit}(\mathsf{ck}, u) \to (c, o)$ *takes commitment key* ck *and* $u \in \mathcal{D}$ *and outputs commitment* $c \in \mathcal{C}$ *and opening* $o \in \mathcal{O}$*.*
- $\mathsf{VerCommit}(\mathsf{ck}, c, u, o) \to b$ *takes commitment key* ck*, commitment* c*, message* u *and opening* o *and outputs* $b \in \{0, 1\}$*.*

Correctness*: A valid commitment always verifies correctly, i.e for* $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(c, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, u)$*, with probability 1, we have* $\mathsf{VerCommit}(\mathsf{ck}, c, u, o) = 1$*.*
Binding*: It is infeasible for a polynomial time adversary to provide two openings to the same commitment.*
Hiding*: Commitments to any two messages are indistinguishable.*

A.2 Zero Knowledge Arguments

We define the notion of pre-processing *zero-knowledge Succinct Arguments of Knowledge* (zkSNARKs).

Definition 2. *A zkSNARK for a family of* NP *relations* $\{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ *is a tuple of algorithms* $(\mathsf{G}, \mathsf{P}, \mathsf{V})$ *where:*

- $\mathsf{G}(1^\lambda, R) \to (\mathsf{pp}, \mathsf{td})$ *takes security parameter and the relation* $R \in \mathcal{R}_\lambda$ *and outputs public parameters* $\mathsf{pp} = (\mathsf{pk}, \mathsf{vk})$ *and a trapdoor* td*. In the above* pk *is called the* evaluation key *and* vk *is called the* verification key*.*
- $\mathsf{P}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{w}) \to \pi$ *takes the evaluation key, public input vector* $\boldsymbol{x}$*, witness vector* $\boldsymbol{w}$ *and outputs a proof* π*.*
- $\mathsf{V}(\mathsf{vk}, \boldsymbol{x}, \pi) \to b$ *takes the verification key, public input vector* $\boldsymbol{x}$*, a proof* π *and outputs* $b = 1$ *(accept) or* $b = 0$ *(reject).*

A zkSNARK $\mathcal{S} = (\mathsf{G}, \mathsf{P}, \mathsf{V})$ *satisfies the following properties:*
Completeness*: For all* $(R, \boldsymbol{x}, \boldsymbol{w})$ *such that* $R \in \mathcal{R}_\lambda$ *and* $R(\boldsymbol{x}, \boldsymbol{w}) = 1$*, the following probability is 1.*

$$\Pr[\pi \leftarrow \mathsf{P}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{w}); \mathsf{V}(\mathsf{vk}, \boldsymbol{x}, \pi) = 1]$$

Knowledge Soundness*: Let* $\mathcal{RG}$ *denote a* relation generator *and* $\mathcal{Z}$ *denote a (benign) auxiliary input generator. Then the zkSNARK* $\mathcal{S}$ *is called* knowledge

sound *for* $(\mathcal{RG}, \mathcal{Z})$ *if for all efficient provers* P^**, there exists an extractor* E^{P^*} *such that the following probability is negligible:*

$$\Pr\left[\begin{array}{c|c} (R, aux_R) \leftarrow \mathcal{RG}, \mathsf{pp} \leftarrow \mathsf{G}(1^\lambda, R) & \\ Z \leftarrow \mathcal{Z}(\mathsf{pp}, R, aux_R) & \mathsf{V}(\mathsf{pp}, \boldsymbol{x}, \pi) \wedge \\ (\boldsymbol{x}, \pi) \leftarrow P^*(R, aux_R, \mathsf{pp}, Z) & \neg R(\boldsymbol{x}, \boldsymbol{w}) \\ \boldsymbol{w} \leftarrow E^{P^*}(R, aux_R, \mathsf{pp}, Z) & \end{array}\right]$$

Zero Knowledge*: We say that* $\mathcal{S}$ *satisfies zero-knowledge for relation generator* $\mathcal{RG}$ *if there exists simulator* $S = (S_1, S_2)$ *such that the following hold:*

– Key Indistinguishability*: For all efficient adversaries* $\mathcal{A}$ *we have:*

$$\begin{aligned} &\Pr\left[(R, aux_R) \leftarrow \mathcal{RG}(1^\lambda), \mathsf{pp} \leftarrow G(1^\lambda, R) \middle| \mathcal{A}(R, aux_R, \mathsf{pp}) = 1\right] \\ &\approx \Pr\left[\begin{array}{l|l} (R, aux_R) \leftarrow \mathcal{RG}(1^\lambda), & \mathcal{A}(R, aux_R, \mathsf{pp}) = 1 \\ (\mathsf{pp}, \mathsf{td}) \leftarrow S_1(R, aux_R) & \end{array}\right] \end{aligned}$$

– Proof Indistinguishability*: For all efficient adversaries* $\mathcal{A}$ *and all* $R \in \mathcal{R}_\lambda$*,* $(\boldsymbol{x}, \boldsymbol{w})$ *such that* $R(\boldsymbol{x}, \boldsymbol{w}) = 1$ *we have:*

$$\begin{aligned} &\Pr\left[\begin{array}{l|l} (R, aux_R) \leftarrow \mathcal{RG}(1^\lambda), & \\ \mathsf{pp} \leftarrow G(R, aux_R), & \mathcal{A}(\mathsf{pp}, aux_R, \pi) = 1 \\ \pi \leftarrow \mathsf{P}(\mathsf{pp}, \boldsymbol{x}, \boldsymbol{w}) & \end{array}\right] \\ &\approx \Pr\left[\begin{array}{l|l} (R, aux_R) \leftarrow \mathcal{RG}(1^\lambda), & \\ (\mathsf{pp}, \mathsf{td}) \leftarrow S_1(R, aux_R), & \mathcal{A}(\mathsf{pp}, aux_R, \pi) = 1 \\ \pi \leftarrow S_2(\mathsf{pp}, \boldsymbol{x}, \mathsf{td}) & \end{array}\right] \end{aligned}$$

A.3 Commit and Prove SNARKs

Informally, a *commit and prove* SNARK (CP-SNARK) is a SNARK that can prove knowledge of *witness* where part of the witness opens a commitment c. In other words, a CP-SNARK for relation R allows one to prove knowledge of $\boldsymbol{w} = (\boldsymbol{u}, \boldsymbol{z})$ such that $R(\boldsymbol{x}, \boldsymbol{w}) = 1$ and c is a commitment for $\boldsymbol{u}$. The commitments can be used in several proofs to prove composite statements. We summarise the formal notion of CP-SNARKs as defined in [8].

Definition 3 (CP-SNARK). *Let* Com *be a commitment scheme with input space* $\mathcal{D}$*, opening space* $\mathcal{O}$ *and commitment space* $\mathcal{C}$*. Let* $\{R_\lambda\}_{\lambda \in \mathbb{N}}$ *be a family of relations* R *over* $\mathcal{D}_x \times \mathcal{D}_u \times \mathcal{D}_w$ *where* $\mathcal{D}_u$ *splits as* $\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell$ *for some* $\ell \geq 1$ *such that* $\mathcal{D}_i \subseteq \mathcal{D}$ *for* $i = 1, \ldots, \ell$*. A commit and prove zkSNARK (*CP*) for* Com *and* $\{R_\lambda\}_{\lambda \in \mathbb{N}}$ *is a zkSNARK for family of relations* $\{R_\lambda^{\mathsf{Com}}\}_{\lambda \in \mathbb{N}}$ *where:*

– *every* $\boldsymbol{R} \in R^{\mathsf{Com}}$ *is represented by* (ck, R) *where* $\mathsf{ck} \in \mathsf{Setup}(1^\lambda)$ *and* $R \in R_\lambda$*.*
– $\boldsymbol{R}$ *is over the pairs* $(\boldsymbol{x}, \boldsymbol{w})$ *where* $\boldsymbol{x} = (x, (c_j)_{j \in [\ell]}) \in \mathcal{D}_x \times \mathcal{C}^\ell$ *is the statement and* $\boldsymbol{w} = ((u_j)_{j \in [\ell]}, (o_j)_{j \in [\ell]}, \omega) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell \times \mathcal{O}^\ell \times \mathcal{D}_\omega$ *is the witness. The relation* $\boldsymbol{R}$ *holds iff:*

$$\bigwedge_{j \in [\ell]} \mathsf{VerCommit}(\mathsf{ck}, c_j, u_j, o_j) = 1 \wedge R(x, (u_j)_{j \in [\ell]}, \omega) = 1$$

Further, we say that CP *is knowledge sound for relation generator* $\mathcal{RG}$ *and auxiliary input generator* $\mathcal{Z}$ *if it satisfies knowledge soundness* $(\mathcal{RG}^{\mathsf{Com}}, \mathcal{Z})$ *where* $\mathcal{RG}^{\mathsf{Com}}$ *denotes the relation generator which samples* (ck, R, aux) *as* $\mathcal{RG}(1^\lambda) \rightarrow (R, aux)$ *and* $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{ck}$.

We elaborate slightly on the intuition behind the above definition. Typically a zkSNARK for relation $R \subseteq \mathcal{D}_x \times \mathcal{D}_\omega$ proves knowledge of $\boldsymbol{w} \in \mathcal{D}_\omega$ for a given statement $\boldsymbol{x} \in \mathcal{D}_x$ such that $R(\boldsymbol{x}, \boldsymbol{w}) = 1$. With a CP-SNARK, we additionally wish to prove that part of the witness $\boldsymbol{w}$ opens a commitment c, i.e $\boldsymbol{w} = (\boldsymbol{u}, z)$ where c is a commitment for $\boldsymbol{u}$. Generalizing this further, we can decompose the committed part of the witness $\boldsymbol{u}$ into ℓ slots, where witness corresponding to each slot opens a specified commitment.

B Security Analysis

We describe our protocols as interactive protocols with (semi) honest verifiers. One can obtain non-interactive arguments of knowledge (SNARKs) in the Random Oracle model from them via Fiat-Shamir heuristic. We first define a secure protocol for proving a relation R under commitments using the commitment scheme Com. We will write a relation R as $R(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$ where $\boldsymbol{x}$ denotes the public input (plain-text), $\boldsymbol{u}$ denotes the committed witness while $\boldsymbol{w}$ denotes the "free" (uncommitted witness). The vector $\boldsymbol{u}$ purportedly opens a public commitment c.

Definition 4 (Secure Protocol). *A* secure protocol *for a relation* R *and commitment scheme* Com *consists of tripe* $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ *consisting of generator algorithm* $\mathcal{G}$*, a* PPT *prover* $\mathcal{P}$ *and a* PPT *verifier* $\mathcal{V}$ *which work as follows:*

1. $\mathcal{G}(\mathsf{ck}, R, 1^\lambda) \longrightarrow \mathsf{pp}$*: Given a commitment key* $ck \leftarrow \mathsf{Com.Setup}(1^\lambda)$ *and* R*,* $\mathcal{G}$ *outputs public parameters* pp.
2. *Given public parameters* pp *for relation* R *and a pair* $(\boldsymbol{x}, c)$ *consisting of statement* $\boldsymbol{x}$ *and a public commitment* c*,* $\mathcal{P}$ *and* $\mathcal{V}$ *interact via an alternating sequence of messages, at the end of which* $\mathcal{V}$ *outputs* $0\,(\texttt{Reject})$ *or* $1\,(\texttt{Accept})$.

Further, a secure protocol Π *satisfies* completeness, soundness *and* zero-knowledge *which we define shortly.*

Let $\Pi(\mathsf{pp}, \boldsymbol{x}, c; \boldsymbol{u}, \boldsymbol{w}, 0)$ denote the output $(0/1)$ of interaction between $\mathcal{P}$ and $\mathcal{V}$ on common input $(\boldsymbol{x}, c)$ and $\mathcal{P}$'s private inputs as $\boldsymbol{u}, \boldsymbol{w}, o$. Similarly, let $\Pi.\mathsf{Vw}(\boldsymbol{x}, c; \boldsymbol{u}, \boldsymbol{w}, o)$ denote $\mathcal{V}$'s view in the interaction. We use $\Pi_{\mathcal{A}}(\mathsf{pp}, \boldsymbol{x}, c)$ to denote the output of interaction between an adversarial prover $\mathcal{A}$ and $\mathcal{V}$ on common input $(\boldsymbol{x}, c)$. Next, we define the security properties satisfied by a secure protocol Π.

Completeness: We call Π to be complete if for all $\mathsf{ck} \in \mathsf{Com.Setup}(1^\lambda)$ and $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \in R$ we have:

$$\Pr\left[\mathsf{pp} \leftarrow \mathcal{G}(\mathsf{ck}, R, 1^\lambda), c = \mathsf{Com.Commit}(\mathsf{ck}, \boldsymbol{u}, o), \Pi(\boldsymbol{x}, c; \boldsymbol{u}, \boldsymbol{w}, o) = 1\right] = 1$$

.

Soundness: We call Π to have soundness if for all PPT adversaries $\mathcal{A}$, there exists and efficient extractor $\mathcal{E}$ such that the following probability is negligible:

$$\Pr\left[\begin{array}{l|c} \mathsf{ck} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{pp} \leftarrow \mathcal{G}(\mathsf{ck}, R, 1^\lambda), & \Pi_{\mathcal{A}}(\mathsf{pp}, \boldsymbol{x}, c) = 1 \\ (\boldsymbol{x}, c) \leftarrow \mathcal{A}(\mathsf{pp}, z), (\boldsymbol{u}, \boldsymbol{w}, o) \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{pp}, z) & \wedge \neg \widetilde{R}(\boldsymbol{x}, c, \boldsymbol{u}, \boldsymbol{w}, o) \end{array}\right]$$

Here $\tilde{R}(\boldsymbol{x}, c, \boldsymbol{u}, \boldsymbol{w}, o) \equiv R(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \wedge \mathsf{Com.VerCommit}(\mathsf{ck}, c, \boldsymbol{u}, o)$.

Zero Knowledge: We say that Π is zero-knowledge if there exists efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for all $\mathsf{ck} \in \mathsf{Com.Setup}(1^\lambda)$, $(\boldsymbol{x}, c, \boldsymbol{u}, \boldsymbol{w}, o)$ such that $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \in R$ and $c = \mathsf{Com.Commit}(\mathsf{ck}, \boldsymbol{u}, o)$, the following are statistically indistinguishable:

$$\begin{aligned} &\left[\mathsf{pp} \leftarrow \mathcal{G}(\mathsf{ck}, R) \mid \left(\mathsf{pp}, \Pi.\mathsf{Vw}(\mathsf{pp}, \boldsymbol{x}, c; \boldsymbol{u}, \boldsymbol{w}, o)\right)\right] \\ &\quad \approx \left[(\mathsf{pp}, \mathsf{td}) \leftarrow \mathcal{S}_1(1^\lambda, R) \mid \left(\mathsf{pp}, \mathcal{S}_2(\mathsf{td}, \mathsf{pp}, \mathsf{ck}, \boldsymbol{x}, c)\right)\right] \end{aligned}$$

First, we exhibit a trivial secure protocol that can be obtained from a CP-SNARK for a relation.

Lemma 1. *Let $\mathsf{CP} = (\mathsf{G}, \mathsf{P}, \mathsf{V})$ be a CP-SNARK for relation R and commitment scheme Com. Then $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ as described below is a secure protocol for relation R and commitment scheme Com.*

- $\mathcal{G}(\mathsf{ck}, R, 1^\lambda) \longrightarrow \mathsf{pp}$ *where* $\mathsf{pp} \leftarrow \mathsf{G}(\mathsf{ck}, R, 1^\lambda)$.
- *On common input $(\boldsymbol{x}, c)$ and $\mathcal{P}$'s input $(\boldsymbol{u}, \boldsymbol{w}, o)$, $\mathcal{P}$ and $\mathcal{V}$ interact as follows:*
  1. *$\mathcal{P}$ computes: $\pi \leftarrow \mathsf{P}(\mathsf{pp}, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}, o)$.*
  2. *$\mathcal{P} \rightarrow \mathcal{V}$: $\mathcal{P}$ sends π to $\mathcal{V}$.*
  3. *$\mathcal{V}$ outputs $\mathsf{V}(\mathsf{pp}, \boldsymbol{x}, c, \pi)$.*

The proof of the above is trivial and follows directly from the properties of CP-SNARK CP. We now formally define the probabilistic relation decomposition and provide a secure protocol for decomposed relation in by gluing the secure protocols for the constituent relations.

Definition 5 (Probabilistic Relation Decomposition). *Let $R(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$ be a relation. We say that relations (R_1, R_2) are a probabilistic decomposition of R if there exists a canoical partitioning of $\boldsymbol{w}$ as $\boldsymbol{w}_0 || \boldsymbol{w}_1 || \boldsymbol{w}_2$ and a challenge space $\mathcal{C}$ such that for $\alpha \leftarrow \mathcal{C}$:*

$$\begin{aligned} &\Pr\left[R_1(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w}_1) \wedge R_2(\alpha, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w}_2) = 1 \mid R(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) = 1\right] = 1 \\ &\Pr\left[R_1(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w}_1) \wedge R_2(\alpha, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w}_2) = 1 \mid R(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) = 0\right] = \mathsf{negl} \end{aligned}$$

Lemma 2 (Glueing Lemma). *Let (R_1, R_2) be a probabilistic relation decomposition of the relation R and let Π_1 and Π_2 be secure protocols for (R_1, Com) and (R_2, Com) respectively, where Com is a commitment scheme. Then the protocol $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ as described below is a secure protocol for (R, Com).*

- $\mathcal{G}(\mathsf{ck}, R, 1^\lambda) \longrightarrow \mathsf{pp}$*: The algorithm* $\mathcal{P}$ *invokes generator algorithms for the consituent relations as* $\mathsf{pp}_1 \leftarrow \Pi_1.\mathcal{G}(\mathsf{ck}, R_1, 1^\lambda)$, $\mathsf{pp}_2 \leftarrow \Pi_2.\mathcal{G}(\mathsf{ck}, R_2, 1^\lambda)$ *and returns* $\mathsf{pp} = (\mathsf{pp}_1, \mathsf{pp}_2)$.
- *On common input* $(\boldsymbol{x}, c)$ *and private prover inputs* $(\boldsymbol{u}, \boldsymbol{w}, o)$, $\mathcal{P}$ *and* $\mathcal{V}$ *interact as follows:*
  1. $\mathcal{P}$ *computes:* $\mathcal{P}$ *partitions* $\boldsymbol{w}$ *as* $\boldsymbol{w}_0||\boldsymbol{w}_1||\boldsymbol{w}_2$. *Next* $\mathcal{P}$ *samples* $o_w \leftarrow \mathcal{O}$ *and computes* $c_w = \mathsf{Com.Commit}(\mathsf{ck}, \boldsymbol{w}_0, o_w)$.
  2. $\mathcal{P} \rightarrow \mathcal{V}$*:* $\mathcal{P}$ *sends* c_w *to* $\mathcal{V}$.
  3. $\mathcal{P}$ *and* $\mathcal{V}$ *execute the secure protocol* Π_1 *with common input* $(\boldsymbol{x}, (c, c_w))$ *and prover's* $(\Pi_1.\mathcal{P})$ *inputs as* $((\boldsymbol{u}, \boldsymbol{w}_0), \boldsymbol{w}_1, (o, o_w))$. *Let* b_1 *denote the output of the protocol* Π_1.
  4. $\mathcal{V} \rightarrow \mathcal{P}$*:* $\mathcal{V}$ *samples* $\alpha \leftarrow \mathcal{C}$ *and sends* α *to* $\mathcal{P}$.
  5. $\mathcal{P}$ *and* $\mathcal{V}$ *execute the secure protocol* Π_2 *with common input* $((\alpha, \boldsymbol{x}), (c, c_w))$ *and prover's* $(\Pi_2.\mathcal{P})$ *inputs as* $((\boldsymbol{u}, \boldsymbol{w}_0), \boldsymbol{w}_2, (o, o_w))$. *Let* b_2 *denote the output of the protocol* Π_1.
  6. $\mathcal{V}$ *outputs* $b_1 \wedge b_2$.

Proof. We skip the proof of completeness of protocol Π, as it is straightforward to verify. To show soundness, let $\mathcal{A}$ be a PPT adversary such that $\Pi_{\mathcal{A}}(\mathsf{pp}, \boldsymbol{x}, c) = 1$. Let c_w be the first message (commitment) sent by $\mathcal{A}$ to $\mathcal{V}$. From the protocol description of Π, we have:

$$\Pi_{\mathcal{A}}(\mathsf{pp}, \boldsymbol{x}, c) = \Pi_{1,\mathcal{A}}(\mathsf{pp}_1, \boldsymbol{x}, (c, c_w)) \wedge \Pi_{2,\mathcal{A}}(\mathsf{pp}_2, (\alpha, \boldsymbol{x}), (c, c_w))$$

. Thus $\mathcal{A}$ is also an adversary for secure protocols Π_1 and Π_2. Soundness of Π_1 and Π_2 implies existence of extractors $\mathcal{E}_1$ and $\mathcal{E}_2$ such that $((\boldsymbol{u}, \boldsymbol{w}_0), \boldsymbol{w}_1, o) \leftarrow \mathcal{E}_1^{\mathcal{A}}(\mathsf{pp}_1, z)$ and $((\boldsymbol{u}', \boldsymbol{w}_0', \boldsymbol{w}_2, (o', o_w')) \leftarrow \mathcal{E}_2^{\mathcal{A}}(\mathsf{pp}_2, z)$. We define extractor $\mathcal{E}$ which invokes the above extractors and outputs $(\boldsymbol{u}, \boldsymbol{w}, o)$ for $\boldsymbol{w} = \boldsymbol{w}_0||\boldsymbol{w}_1||\boldsymbol{w}_2$. With overwhelming probability we have

$$R_1(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w}_1) \wedge \mathsf{Com.VerCommit}(\mathsf{ck}, (c, c_w), (\boldsymbol{u}, \boldsymbol{w}_0), (o, o_w))$$
$$R_2(\alpha, \boldsymbol{x}, \boldsymbol{w}_0', \boldsymbol{w}_2) \wedge \mathsf{Com.VerCommit}(\mathsf{ck}, (c, c_w), (\boldsymbol{u}', \boldsymbol{w}_0', (o', o_w'))$$

By the binding property of Com, we also have $\boldsymbol{u}' = \boldsymbol{u}$, $\boldsymbol{w}_0' = \boldsymbol{w}_0$, $o' = o$ and $o_w' = o_w$ and $\mathsf{Com.VerCommit}(\mathsf{ck}, (c, c_w), (\boldsymbol{u}, \boldsymbol{w}_0), (o, o_w)) = 1$ with overwhelming probability. Finally, since $R_1(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w}_1) \wedge R_2(\alpha, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}_0, \boldsymbol{w}_2) = 1$, we must have $R(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) = 1$ for $\boldsymbol{w} = \boldsymbol{w}_0||\boldsymbol{w}_1||\boldsymbol{w}_2$ with probability negligibly close to 1. This proves that $\mathcal{E}$ extracts a valid witness with overwhelming proability.

We now show that Π is zero-knowledge. Let $ck \leftarrow \mathsf{Com.Setup}(1^\lambda)$ and let $(\boldsymbol{x}, c, \boldsymbol{u}, \boldsymbol{w}, o)$ be such that $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \in R$ and $c = \mathsf{Com.Commit}(\mathsf{ck}, \boldsymbol{u}, o)$. We show the existence of simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that:

$$\begin{aligned}&\left[\mathsf{pp} \leftarrow \mathcal{G}(\mathsf{ck}, R) \mid \left(\mathsf{pp}, \Pi.\mathsf{Vw}(\mathsf{pp}, \boldsymbol{x}, c; \boldsymbol{u}, \boldsymbol{w}, o)\right)\right] \\ &\quad \approx \left[(\mathsf{pp}, \mathsf{td}) \leftarrow \mathcal{S}_1(1^\lambda, R) \mid \left(\mathsf{pp}, \mathcal{S}_2(\mathsf{td}, \mathsf{pp}, \mathsf{ck}, \boldsymbol{x}, c)\right)\right]\end{aligned}$$

Let $\widetilde{\mathcal{S}} = (\widetilde{\mathcal{S}}_1, \widetilde{\mathcal{S}}_2)$ and $\widehat{\mathcal{S}} = (\widehat{\mathcal{S}}_1, \widehat{\mathcal{S}}_2)$ be the simulators for secure protocols Π_1 and Π_2 respectively. The simulator $\mathcal{S}$ works as follows:

- $\mathcal{S}_1(1^\lambda, R) \longrightarrow (\mathsf{pp}', \mathsf{td}')$: On input R and security parameter, $\mathcal{S}_1$ invokes simulators for R_1, R_2 to obtain $(\mathsf{pp}'_1, \mathsf{td}'_1) \leftarrow \widetilde{\mathcal{S}}_1(1^\lambda, R_1)$, $(\mathsf{pp}'_2, \mathsf{td}'_2) \leftarrow \widehat{\mathcal{S}}_1(1^\lambda, R_2)$ respectively. It sets $\mathsf{pp}' = (\mathsf{pp}'_1, \mathsf{pp}'_2)$ and $\mathsf{td}' = (\mathsf{td}'_1, \mathsf{td}'_2)$.
- $\mathcal{S}_2$ works as follows: It samples $\alpha \leftarrow \mathcal{C}$, $\tilde{o} \leftarrow \mathcal{O}_\lambda$ and computes $\tilde{c}_w = \mathsf{Com.Commit}(\mathsf{ck}, \mathbf{0}, \tilde{o})$. Then it invokes simulators $\widetilde{\mathcal{S}}_2$ and $\widehat{\mathcal{S}}_2$ as:
  - $V'_1 \leftarrow \widetilde{\mathcal{S}}_2(\mathsf{td}'_1, \mathsf{pp}'_1, \boldsymbol{x}, (c, \tilde{c}_w))$,
  - $V'_2 \leftarrow \widehat{\mathcal{S}}_2(\mathsf{td}'_2, \mathsf{pp}'_2, (\alpha, \boldsymbol{x}), (c, \tilde{c}_w))$.
- Finally it outputs $(\alpha, \tilde{c}_w, V'_1, V'_2)$.

The required indistinguishability follows via hybrids shown below. For ease of notation let V_1 denote $\mathit{\Pi}_1(\mathsf{pp}_1, \boldsymbol{x}, (c, c_w); (\boldsymbol{u}, \boldsymbol{w}_0), \boldsymbol{w}_1, (o, o_w))$ and V_2 denote $\mathit{\Pi}_2(\mathsf{pp}_2, (\alpha, \boldsymbol{x}), (c, c_w); (\boldsymbol{u}, \boldsymbol{w}_0), \boldsymbol{w}_2, (o, o_w))$. Then we have:

$$\langle \mathsf{pp}, \mathit{\Pi}.\mathsf{Vw}(\mathsf{pp}, \boldsymbol{x}, c; \boldsymbol{u}, \boldsymbol{w}, o)\rangle \tag{1}$$
$$= \langle \mathsf{pp}_1, \mathsf{pp}_2, \alpha, c_w, V_1, V_2\rangle \tag{2}$$
$$\approx \langle \mathsf{pp}'_1, \mathsf{pp}_2, \alpha, c_w, \widetilde{\mathcal{S}}_2(\mathsf{td}'_1, \mathsf{pp}'_1, \boldsymbol{x}, (c, c_w)), V_2\rangle \tag{3}$$
$$\approx \langle \mathsf{pp}'_1, \mathsf{pp}'_2, \alpha, c_w, \widetilde{\mathcal{S}}_2(\mathsf{td}'_1, \mathsf{pp}'_1, \boldsymbol{x}, (c, c_w)), \widehat{\mathcal{S}}_2(\mathsf{td}'_2, \mathsf{pp}'_2, (\alpha, \boldsymbol{x}), (c, c_w))\rangle \tag{4}$$
$$\approx \langle \mathsf{pp}'_1, \mathsf{pp}'_2, \alpha, \tilde{c}_w, V'_1, V'_2\rangle \tag{5}$$

In the above the indistinguishability of (2) and (3) follows from the zero knowledge property of $\mathit{\Pi}_1$. Similarly zero knowledge of $\mathit{\Pi}_2$ implies indistinguishability of (3) and (4). Finally, the indistinguishability of (4) and (5) follows from the hiding property of Com. This completes the proof.

C Secure Protocols

In this section, we give secure protocols for the different relations discussed in this paper such as simultaneous permutation, consistent memory access, various dataset operations and decision tree inference.

C.1 Simultaneous Permutation

For a fixed N, recall that k-tuples $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k)$ and $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ of vectors in $\mathbb{F}^N$ satisfy simultaneous permutation relation if there exists a permutation σ of $[N]$ such that $\sigma(\boldsymbol{u}_i) = v_i$ for all $i \in [N]$. Let R_σ denote the relation over $(\alpha, \boldsymbol{u}, \boldsymbol{v})$ with $\alpha \in \mathbb{F}$ and $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^N$ such that $\prod_{i=1}^N (\alpha - \boldsymbol{u}[i]) = \prod_{i=1}^N (\alpha - \boldsymbol{v}[i])$. Let $\mathit{\Pi}_\sigma$ denote the trivial secure protocol obtained from CP-SNARK for (R_σ, Com) (using Lemma 1), where we also assume Com is homomorphic.

Lemma 3. *The protocol $\mathit{\Pi}_{\mathrm{perm}} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ in Figure 5 is a secure protocol for simultaneous permutation relation and commitment scheme Com.*

Proof. By standard rewinding technique, with overwhelming probability the extractor $\mathcal{E}$, for an accepting adversarial prover $\mathcal{A}$ can extract vectors $\{\boldsymbol{u}_i, \boldsymbol{v}_i\}_{i=1}^k$ such that $\boldsymbol{u}_i$ opens commitment cu_i and $\boldsymbol{v}_i$ opens commitment cv_i for all $i \in [k]$.

This is accomplished by running the subprotocol Π_σ for k different linear combinations of commitments given by the challenge $(\beta_1, \ldots, \beta_k)$, and using the extractor for Π_σ to obtain openings for respective linear combinations of vectors. Since the challenges are linearly independent with overwhelming probability, we can solve the system of equations to obtain openings for individual commitments cu_i and cv_i for all $i \in [k]$. By homomorphism of Com, the vectors $\boldsymbol{u} = \sum_{i=1}^{k} \beta_i \boldsymbol{u}_i$ and $\boldsymbol{v} = \sum_{i=1}^{k} \beta_i \boldsymbol{v}_i$ open commitments cu and cv respectively. Again soundness of Π_σ implies with overwhelming probability $(\alpha, \boldsymbol{u}, \boldsymbol{v}) \in R_\sigma$. Since α was drawn uniformly at random, we conclude that there is a permutation π such that $\pi(\boldsymbol{u}) = \boldsymbol{v}$ with probability almost 1. Finally, since $\beta_1, \ldots, \beta_k$ were drawn uniformly at random $\pi(\sum_{i=1}^{k} \beta_i \boldsymbol{u}_i) = \sum_{i=1}^{k} \beta_i \boldsymbol{v}_i$, with overwhelming probability we must have $\pi(\boldsymbol{u}_i) = \boldsymbol{v}_i$ for all $i \in [k]$. This shows the soundness of Π_{perm}. We skip the proof of zero-knowledge for Π_{perm} as it follows from the same property for Π_σ.

$\mathcal{G}(1^\lambda) \longrightarrow \mathsf{pp}$: Obtains pp as $\mathsf{pp} \leftarrow \Pi_\sigma.\mathcal{G}(1^\lambda, R_\sigma)$.
Inputs: On common input $\mathsf{c}_u = (\mathsf{cu}_i)_{i=1}^k$, $\mathsf{c}_v = (\mathsf{cv}_i)_{i=1}^k$ and $\mathcal{P}$'s inputs consisting of $\{\boldsymbol{u}_i, \boldsymbol{v}_i, o_i, \omega_i\}_{i=1}^k$, permutation π of $[N]$; $\mathcal{P}$ and $\mathcal{V}$ interact as follows:

1. $\mathcal{V} \to \mathcal{P}$: $(\alpha, \beta_1, \ldots, \beta_k) \leftarrow \mathbb{F}^{k+1}$.
2. $\mathcal{P}$ and $\mathcal{V}$ compute: $\mathsf{cu} = \sum_{i=1}^{k} \beta_i \mathsf{cu}_i$, $\mathsf{cv} = \sum_{i=1}^{k} \beta_i \mathsf{cv}_i$.
3. $\mathcal{P}$ computes: $\boldsymbol{u} = \sum_{i=1}^{k} \beta_i \boldsymbol{u}_i$, $\boldsymbol{v} = \sum_{i=1}^{k} \beta_i \boldsymbol{v}_i$, $o = \sum_{i=1}^{k} \beta_i o_i$, $\omega = \sum_{i=1}^{k} \beta_i \omega_i$.
4. $\mathcal{P}$ and $\mathcal{V}$ execute the protocol Π_σ with $(\alpha, \mathsf{cu}, \mathsf{cv})$ as the common input and $(\boldsymbol{u}, \boldsymbol{v}, o, \omega)$ as prover's inputs. Let b be the output of the protocol Π_σ.
5. $\mathcal{V}$ outputs b.

Fig. 5: Protocol Π_{perm} for Simultaneous Permutation

C.2 Consistent Memory Access

in this section, we formalize the secure protocol for consistent memory access relation discussed in Section 3.4.

Lemma 4. *There exists a secure protocol Π_{cma} for consistent memory access relation defined in Section 3.4.*

Proof. We consider the relation R_{cma} explained in Section 3.4 for consistent memory access as:

$$R_{\text{cma}}(\cdot, [\![\boldsymbol{L}, \boldsymbol{U}, \boldsymbol{V}]\!], [\![\boldsymbol{u}, \boldsymbol{v}, \tilde{\boldsymbol{u}}, \boldsymbol{v}, \tilde{\boldsymbol{w}_1}, \boldsymbol{w}_2]\!])$$

In the above, there are no public inputs, the committed witness consists of $\boldsymbol{L}, \boldsymbol{U}$ and $\boldsymbol{V}$ which denote the read only memory, access pattern and values

respectively. The uncommitted witness consists of auxiliary inputs $(\boldsymbol{u}, \boldsymbol{v}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{v}})$ and other witness $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ required to prove the relation. The description in Section 3.4 partitions the above as:

$$\mathsf{C}_{\mathsf{ROM},m,n}(\cdot, [\![\boldsymbol{L}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{w}_0]\!], \boldsymbol{w}_1) \wedge R_\sigma(\cdot, \boldsymbol{w}_0, \boldsymbol{w}_2) \tag{6}$$

where $\boldsymbol{w}_0 = [\![\boldsymbol{u}, \boldsymbol{v}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{v}}]\!]$. The secure protocol Π_{ROM} can be obtained using a CP-SNARK for circuit $\mathsf{C}_{\mathsf{ROM},m,n}$ via Lemma 1. Invoking Glueing Lemma (Lemma 2) with Π_{ROM} and protocol Π_{perm} for simultaneous permutation relation, we obtain the secure protocol Π_{cma}.

C.3 Aggregation Operation

We now provide a secure protocol for showing correctness of aggregation operation on datasets as described in Section 4. In Section 4 we described a protocol for checking correct concatenation of vectors under commitments, and then reduced the verification of dataset aggregation to that of verifying concatenation of vectors (obtained via linear combination of columns of dataset). We also justify the aforementioned reduction. We assume Π_{concat} is a secure protocol for checking concatenation of vectors, which we assume is desceribed by the relation R_{concat}. The secure protocol $\Pi_{\mathrm{agg}} = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ for verifying aggregation of datasets appears in Figure 6. Let D_x, D_y and D_z be datasets with columns given by $(\boldsymbol{x}_i)_{i=1}^k$, $(\boldsymbol{y}_i)_{i=1}^k$ and $(\boldsymbol{z}_i)_{i=1}^k$ respectively. Similarly let $(\mathsf{cx}_i)_{i=1}^k$, $(\mathsf{cy}_i)_{i=1}^k$ and $(\mathsf{cz}_i)_{i=1}^k$ denote public commitments to the columns of D_x, D_y and D_z respectively. As in Section 4, let N denote the upper bound on the sizes of datasets and vectors.

$\mathcal{G}(1^\lambda) \longrightarrow \mathsf{pp}$: Obtains pp as $\mathsf{pp} \leftarrow \Pi_{\mathrm{concat}}.\mathcal{G}(1^\lambda, R_{\mathrm{concat}})$.
Inputs: On common input $(\mathsf{cx}_i)_{i=1}^k$, $(\mathsf{cy}_i)_{i=1}^k$ and $(\mathsf{cz}_i)_{i=1}^k$ and $\mathcal{P}$'s inputs consisting of $\{\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i, o_i, \omega_i, \delta_i\}_{i=1}^k$; $\mathcal{P}$ and $\mathcal{V}$ interact as follows:

1. $\mathcal{V} \to \mathcal{P}$: $(\beta_1, \ldots, \beta_k) \leftarrow \mathbb{F}^{k+1}$ satisfying $\sum_{i=1}^k \beta_i = 1$.
2. $\mathcal{P}$ and $\mathcal{V}$ compute: $\mathsf{cx} = \sum_{i=1}^k \beta_i \mathsf{cx}_i$, $\mathsf{cy} = \sum_{i=1}^k \beta_i \mathsf{cy}_i$ and $\mathsf{cz} = \sum_{i=1}^k \beta_i \mathsf{cz}_i$.
3. $\mathcal{P}$ computes: $\boldsymbol{x} = \sum_{i=1}^k \beta_i \boldsymbol{x}_i$, $\boldsymbol{y} = \sum_{i=1}^k \beta_i \boldsymbol{y}_i$, $\boldsymbol{z} = \sum_{i=1}^k \beta_i \boldsymbol{z}_i$. Similarly it also obtains o, ω and δ as β-linear combinations of $\{o_i\}_{i=1}^k$, $\{\omega_i\}_{i=1}^k$, $\{\delta_i\}_{i=1}^k$ respectively.
4. $\mathcal{P}$ and $\mathcal{V}$ execute the protocol Π_{concat} with $(\mathsf{cx}, \mathsf{cy}, \mathsf{cz})$ as the common input and $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, o, \omega, \delta)$ as prover's inputs. Let b be the output of the protocol Π_{concat}.
5. $\mathcal{V}$ outputs b.

Fig. 6: Protocol Π_{agg} for Dataset Aggregation

Lemma 5. *The protocol Π_{agg} in Figure 6 is a secure protocol for aggregation relation on datasets and commitment scheme* Com.

Proof. The completeness and zero-knowledge properties of the protocol are proved in a manner similar to earlier protocols. Here we prove the soundness of the probabilistic reduction from aggregation relation on datasets to concatenation relation on vectors, which implies soundness of the overall protocol. With overwhelming probability, a successful adversary $\mathcal{A}$ knows vectors $(\boldsymbol{x}_i)_{i=1}^k$, $(\boldsymbol{y}_i)_{i=1}^k$ and $(\boldsymbol{z}_i)_{i=1}^k$ such that their respective β-linear combinations $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$ satisfy the concatenation relation. As in Section 4, we write $\boldsymbol{x}_i = [\![s_i, \boldsymbol{X}_i]\!]$, $\boldsymbol{y}_i = [\![t_i, \boldsymbol{Y}_i]\!]$ and $\boldsymbol{z}_i = [\![w_i, \boldsymbol{Z}_i]\!]$ for $i \in [k]$. Similarly, let $\boldsymbol{x} = [\![s, \boldsymbol{X}]\!]$, $\boldsymbol{y} = [\![t, \boldsymbol{Y}]\!]$ and $\boldsymbol{z} = [\![w, \boldsymbol{Z}]\!]$. Note that we must have:

$$s = \sum_{i=1}^{k} \beta_i s_i, \quad t = \sum_{i=1}^{k} \beta_i t_i, \quad w = \sum_{i=1}^{k} \beta_i w_i$$
$$\boldsymbol{X} = \sum_{i=1}^{k} \beta_i \boldsymbol{X}_i, \quad \boldsymbol{Y} = \sum_{i=1}^{k} \beta_i \boldsymbol{Y}_i, \quad \boldsymbol{Z} = \sum_{i=1}^{k} \beta_i \boldsymbol{Z}_i$$

Now, from description in Section 4, the vectors $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$ satisfy the concatenation relation if there exists a permutation of $[2N]$, which we denote by permutation matrix Λ such that $\Lambda \cdot [\![\boldsymbol{\rho}_s, \boldsymbol{\rho}_t]\!] = [\![\boldsymbol{\rho}_w, \boldsymbol{0}]\!]$, $\Lambda \cdot [\![\boldsymbol{X}, \boldsymbol{Y}]\!] = [\![\boldsymbol{Z}, \boldsymbol{0}]\!]$ where vectors $\boldsymbol{\rho}_s, \boldsymbol{\rho}_t$ and $\boldsymbol{\rho}_w$ are in $\{0,1\}^N$ such that $\boldsymbol{\rho}_s$ is 1 in precisely the first s positions, $\boldsymbol{\rho}_t$ is 1 in precisely the first t positions and $\boldsymbol{\rho}_w$ is 1 in precisely the first w positions where further $w = s + t$. The relation thus also implicity requires that $s, t, w \in [N]$. We now claim that $s_i = s$, $t_i = t$ and $w_i = w$ for all $i \in [k]$. Otherwise it is easily seen that s is distributed uniformly in $\mathbb{F}$ (and likewise for t and w) for uniformly sampled $\beta_1, \ldots, \beta_k$ (subject to sum being 1), and thus $s \in [N]$ with negligible probability $N/|\mathbb{F}|$. Similar reasoning also implies that with overwhelming probability we have $\Lambda \cdot [\![\boldsymbol{X}_i, \boldsymbol{Y}_i]\!] = [\![\boldsymbol{Z}_i, \boldsymbol{0}]\!]$ for all $i \in [k]$. Combined with the fact that $\Lambda \cdot [\![\boldsymbol{\rho}_s, \boldsymbol{\rho}_t]\!] = [\![\boldsymbol{\rho}_w, \boldsymbol{0}]\!]$, it implies that the same permutation Λ maps the first s entries of column $\boldsymbol{x}_i$ and first t entries of column $\boldsymbol{y}_i$ to the first $w = s + t$ entries of the column $\boldsymbol{z}_i$ for all $i \in [k]$. Thus D_z corresponds to aggregation of datasets D_x and D_y.

Protcols and Proofs for Other Operations: We have provided circuit descriptions for other operations such as `filter`, `order-by`, `inner-join` and also ML operations such as inference and accuracy from decision trees. These circuits can be used with CP-SNARKs to yeild secure protocols for those operations using techniques similar to presented protocols (essentially using Lemmas 1 and 2), alongwith reduction technique when applicable.