HyperX: Quality-Assurance, Confidentiality, and Transparency in Data Marketplace for Deep Learning

Sainan Li[1] Zhuotao Liu[2] Bihan Wen[3] Qi Li[1] Guoliang Li[1] Qian Wang[4] Chao Shen[5]
[1]Tsinghua University [2]Google [3]Nanyang Technological University [4]Wuhan University [5]Xi'an Jiaotong University

ABSTRACT

As data is increasingly viewed as valuable assets, recruiting desirable datasets through data marketplaces is a promising space that has been explored for various applications. However, regarding machine learning (ML), especially deep learning, applications, existing proposals fall short in two crucial aspects. First, accurately, yet confidentially, assessing data quality is still an open problem since prior proposals either require complete raw datasets for assessment or rely on Trusted Execution Environments or/and Secure Multiparty Computation, which faces practicability and scalability challenges for handling large-scale data exchanges that are essential to deep learning applications. Second, building a secure and transparent underlying infrastructure for the marketplace to ensure the economical fairness of all participants has been largely ignored.

In this paper, we propose HyperX, the first quality-driven, privacy-preserving and economically-fair data trading platform for deep learning applications. At the highest level, HyperX is powered by two innovative designs. First, we propose a novel privacy-preserving data quality assessment design which relies on provably irreversible intermediate data features collected from a small subset of each provider's dataset and an obfuscated feature extractor truncated from a general-purpose model from the collector, yet achieving near-optimal accuracy of identifying the best dataset from all heterogeneous providers. Second, we design a set of security protocols to handle the complexity of realizing real-world data exchanges in a secure, transparent and trust-free manner. We implement a prototype for HyperX using readily-available infrastructural primitives to demonstrate its practicability, and perform extensive evaluations to show the effectiveness of our data quality assessment design.

1 INTRODUCTION

Our information era has been drastically shaped by the advances of Machine Learning (ML). Training data, which is often cited as the *electricity* of ML, is increasingly treated as invaluable assets. For instance, many large organizations, such as Apple [20], Google [13], and Microsoft [16], collect user data while providing their (free) services. *In a world deluged with isolated and heterogeneous data, a quality-aware and secure data-exchange marketplace is power.* With such a marketplace, ML applications have the potential to recruit large-scale datasets which are originally possessed by heterogeneous and distributed sources.

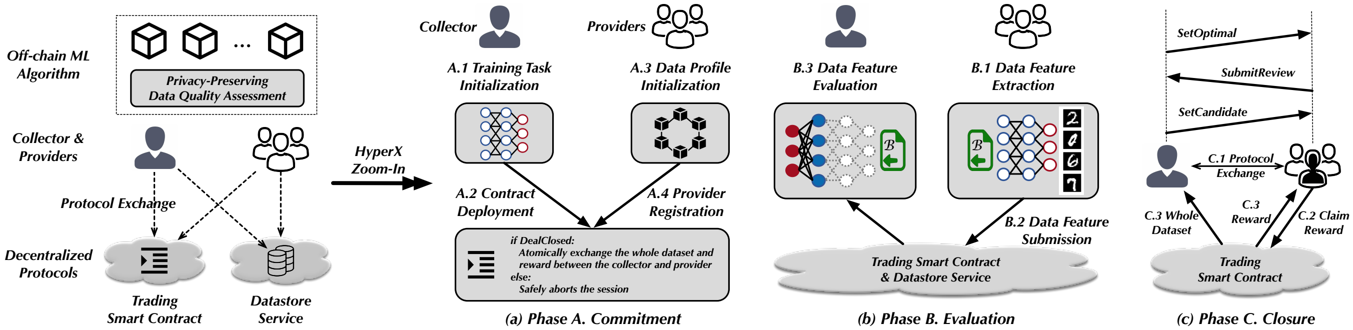
We recognize at least three categories of challenges for building such a data marketplace for ML. First, data quality is known to have significant impact on the performance and generalization of ML, especially deep learning, models [61]. Thus, *assuring data quality* should be the very first requirement for the data marketplace. However, accurately and quantitatively assessing data quality, rather than simply trading data based on their descriptions or tags, has been largely ignored by the prior proposals on data marketplaces, such as [22, 32, 38, 64]. Although several data quality evaluation mechanisms (*e.g.,* [12, 25, 28]) have been proposed, they are not applicable for data marketplaces since they require *the complete dataset* for quality assessment, whereas in practice data providers are unwilling to hand over their complete datasets before reaching any deal with the (untrusted) data collectors.

Second, privacy should be the first-class citizen in a practical data marketplace, such that the datasets of providers and the models of collectors should be kept confidential. Prior work [23] proposes to perform data quality assessment and model training in a Trusted Execution Environment (TEE) (*e.g.,* Intel SGX) to achieve privacy. However, given the memory limitation and bounded CPU capability of a TEE, it faces scalability and efficiency challenges for handling large-scale data exchanges that are essential to deep learning tasks. In addition, heavily relying on TEE on critical computing (such as model training and data quality evaluation) creates a central point of trust on the TEE provider (*e.g.,* Intel), letting alone the concerns raised by various attacks to undermine the security guarantees of the TEE itself (*e.g.,* [36, 37, 53, 54]).

Third, creating an economically fair marketplace is crucial. Such economic fairness should at least cover the following two aspects: (i) *transparent*, the trading process is known to all stakeholders and resilient to misconducts; and (ii) *secure*, regardless of how dishonest parties game the exchange process, the interests of honest parties are protected, and meanwhile malicious parties are held accountable for their activities. Although achieving the economic fairness via a virtual trusted authority is straightforward, in practice, realizing the fairness through decentralized real-world protocols is non-trivial.

Contributions. To meet these challenges, we propose HyperX (the suffix X is inspired by *eXchange*), the first deep learning data marketplace that simultaneously offers *data quality assurance*, *guaranteed dataset and model privacy*, and *fair trading economy.* At the highest level, HyperX is powered by two innovative designs: a data quality assessment mechanism to evaluate each provider's dataset confidentially and accurately, and a set of protocols to securely and transparently execute the trading process. Concretely, we make the following major contributions in this paper.

(i) We propose a novel dataset quality assessment mechanism providing strong privacy protection and high evaluation accuracy. Specifically, HyperX enables collectors to confidentially assess the quality of each provider's dataset only based on provably irreversible intermediate features extracted from a small and random subset of the provider's entire dataset, *i.e.,* no raw data is ever exposed during the quality assessment phase. Further, the model used by a collector to extract intermediate features is obfuscated and any concrete models trained on these features are confidential. Finally, since only a small subset of each provider's dataset is involved in quality assessment, we apply transfer learning to prevent possible


Off-chain ML Algorithm
...
Privacy-Preserving Data Quality Assessment
Collector & Providers
Protocol Exchange
Decentralized Protocols
Trading Smart Contract
Datastore Service
HyperX Zoom-In
Collector
Providers
A.1 Training Task Initialization
A.3 Data Profile Initialization
A.2 Contract Deployment
A.4 Provider Registration
if DealClosed:
Atomically exchange the whole dataset and reward between the collector and provider
else:
Safely aborts the session
B.3 Data Feature Evaluation
B.1 Data Feature Extraction
$\mathcal{B}$
2
0
6
7
B.2 Data Feature Submission
Trading Smart Contract & Datastore Service
SetOptimal
SubmitReview
SetCandidate
C.1 Protocol Exchange
C.3 Whole Dataset
C.3 Reward
C.2 Claim Reward
Trading Smart Contract

(a) Phase A. Commitment

(b) Phase B. Evaluation

(c) Phase C. Closure

Figure 1: The architecture of HyperX. At a very high level, a trading session on HyperX goes through three phases: Commitment, Evaluation and Closure. In the Commitment phase, the collector and providers collectively initialize the trading session by committing the trading task and datasets. In the Evaluation phase, the collector confidentially assesses the quality of each provider's dataset. In the Closure phase, the collector and the optimal provider (decided by the collector) go through a review-and-exchange mechanism to *atomically* close the deal.

overfitting, ensuring that our data quality assessment is accurate and not biased by the small amount of samples.

(ii) We design a set of security protocols to handle the complexity of realizing real-world data exchanges. Taken together, our protocol is (i) *transparent*, allowing data collectors and providers to reach deals with full information and autonomy; (ii) *secure*, the correctness of the exchange is assured such that the deal is either atomically closed (*i.e.,* the full dataset and payment are exchanged simultaneously between a pair of collector and provider) or safely aborted (*i.e.,* the interests of all deal participants are protected despite abortion); and (iii) *trust-free*, the protocol is fully decentralized and realizable without assuming any trusted third parties. We formulate the security properties of our protocol and prove them using the Universal Composability framework [9].

(iii) We demonstrate the practicability of HyperX by fully implementing a prototype of HyperX using readily-available infrastructural primitives. We extensively evaluate the effectiveness of our data quality assessment algorithm using three public datasets, hundreds of synthetic datasets with varying qualities, and thousands of trading sessions with heterogeneous providers. The experimental results show that HyperX achieves the near-optimal accuracy of identifying the best dataset from these providers, compared with a hypothetical design which allows the collector to access the complete raw dataset from all providers a priori.

2 HYPERX OVERVIEW

2.1 Architecture

As depicted in Figure 1, architecturally, HyperX is designed around four components. (i) A data collector using HyperX to recruit high quality training dataset for its deep learning tasks. Each data collection session is associated with a reward that the collector will pay if a data trading deal is eventually closed. (ii) Data providers trading their data on HyperX for the reward. A provider is free to participate in any trading session, given that the provider is willing to deposit sufficient fund required by the session (refundable). The collector and providers rely on the underlying trading infrastructure to realize real-world data exchanges. The infrastructure itself has two building blocks. (iii) A novel data quality assessment design that enables the collector to evaluate the quality of each provider's dataset confidentially (only irreversible intermediate features, rather than the raw data, are disclosed), efficiently (only a small subset of each provider's dataset is used to extract these features), and accurately (evaluation on data samples accurately reflects the overall quality of the whole dataset). (iv) A set of security protocols to realize the trading process in a transparent, secure, and trust-free manner. We prove that our protocols in aggregate achieve strong security properties (such as deal atomicity) as if there were a trusted authority managing all trading sessions.

To build a practical data exchange platform, HyperX relies on two infrastructural primitives. First, to offer transparency, HyperX protocolizes the trading procedure as a smart contract $\mathcal{F}_{\text{SC}}$, an abstracted computation service providing guaranteed availability and correctness, but no privacy. $\mathcal{F}_{\text{SC}}$ serves as a persistent and public protocol exchange medium among all participants, as well as a way of settling payment if a trade is eventually closed. $\mathcal{F}_{\text{SC}}$ can be realized natively on most production-ready blockchains with smart contract support. Second, since HyperX does not (and should not) own any data being traded on the platform, an extra datastore service $\mathcal{F}_{\text{DS}}$ is required. To ensure protocol correctness, $\mathcal{F}_{\text{DS}}$ needs to hold a key invariant: if any storage request, abstracted as a pair of a retrieval key and associated data, is accepted by $\mathcal{F}_{\text{DS}}$, it guarantees to return the same data upon future retrieval with the key, *i.e.,* all storage requests to $\mathcal{F}_{\text{DS}}$ are persistent and their associated data are immutable. We realize $\mathcal{F}_{\text{DS}}$ based on IPFS, a global-scale distributed file system built on a peer-to-peer decentralized network [4].

Key Innovation. We clarify that HyperX is *fundamentally different from an ideal design* assuming that an ideal blockchain and/or an ideal Trusted Execution Environment (TEE), which simultaneously offers private smart contracting, large-scale storage, and unlimited computing capability, were able to step up to claim the responsibility of a versatile yet trust-free dealer in the marketplace. *Instead, practicability and deployability are always the first-class citizens in HyperX.* Thus, we design delicate security protocols, tightly coupled with our novel off-chain privacy-preserving data quality assessment mechanism, to demonstrate a practical system architecture for building a secure, transparent and trust-free marketplace for exchanging large-scale deep learning training data.

2.2 Key Correctness and Security Properties

At the highest level, HyperX achieves three key properties.

Confidentiality. The data quality assessment process is conducted confidentially. (i) Each provider's raw dataset is never disclosed to the collector or other providers. Instead, the collector only learns provably *irreversible* intermediate data features extracted from a small (and random) subset of each provider's dataset. (ii) The training process and any concrete model trained on these collected features are confidential to all providers. Rather, only an obfuscated feature extractor truncated from a general-purpose model (*i.e.,* not specific to the current learning task) is published by the collector.

(Near-)Optimality. Despite the confidentiality and efficiency (*i.e.,* only a small subset of data samples is involved) of our data quality assessment, HyperX still achieves near-optimal accuracy of identifying the best dataset from heterogeneous providers as if the collector were able to access the complete raw datasets from all providers *a priori.* Thus, HyperX does not trade optimality for confidentiality because the two goals are *not fundamentally at odds* by our design.

Atomicity. Each trading session is either closed atomically, where the collector and the optimal provider (where the optimality is decided by the collector) exchange their goods (*i.e.,* the reward and the complete dataset) simultaneously, or safely aborted, where the interests of all participants are protected. The whole trading session, conceptually, is a *commit-reveal* scheme that the collector and all providers first commit their goods confidentially, and later only the collector and the optimal provider reveal their commitments upon mutual agreement. Other providers' commitments remain unrevealed. The commit-reveal scheme is achieved without assuming a trusted third party holding all participants' goods intermediately.

2.3 Security Assumptions and Threat Model

We assume that the cryptographic primitives and the consensus protocol of the blockchain system hosting our trading smart contract are secure so that the blockchain can have the concept of transaction finality and contract publicity. On Nakamoto consensus based blockchains, finality is achieved by assuming that the probability of blockchain reorganizations drops exponentially as new blocks are appended (*i.e.,* the common-prefix property) [17]. On Byzantine tolerance based blockchains, finality is guaranteed by signatures from a quorum of permissioned voting nodes. For any blockchain, if the definition of transaction finality is accepted by all participants, its operation (or trust) model (*e.g.,* permissionless or permissioned) and consensus efficiency (*i.e.,* the latency for a transaction to become final) of the blockchain have no impact on the security guarantees of HyperX. We also assume that the blockchain has a public ledger that allows external parties to examine the public state of its deployed smart contracts.

For the datastore service, we assume all accepted storage requests are persistent and their associated data are immutable. These properties could be achieved, for instance, via content addressing. Similar to the blockchain infrastructure, the operation model of the datastore service, as long as accepted by all participants, does not impact the correctness and security properties of HyperX.

We consider a Byzantine adversary that can interfere with our protocol arbitrarily, including delaying and reordering network messages indefinitely, and compromising protocol participants. For any protocol participant that is not compromised by the adversary, its security properties offered by HyperX are assured.

2.4 Data Exchange Model

General-Purpose Pre-trained Models. HyperX adopts a task-based data exchange model where each trading session is specific to one data recruiting task initiated by a collector. As a prerequisite for our privacy-preserving data quality assessment mechanism, we assume that the collector has a *general-purpose* model pre-trained on datasets (could be public) that are remotely related with the task, and a small set of validation dataset that is closely related with the task. For instance, in a task recruiting robot images, the pre-model could be trained on a generic image dataset such as ImageNet [51], whereas the validation image set should contain robots as objects. Since the pre-model is not specific to the current task, it is impossible for the collector to directly tune the pre-model with its small validation dataset into a high-quality and task-focused deep model (otherwise the collector would not need to recruit additional datasets). Instead, in HyperX, the pre-trained model is used as a feature extractor (actually only the first few layers of the pre-model are used) to power our privacy-preserving data quality assessment design.

We clarify that deep learning tasks are often built upon pre-trained models, to achieve better generalizability and convergence. For instance, tasks over special modalities (*e.g.,* medical image analysis and remote sensing) often have limited training datasets and therefore those deep models are typically pre-trained using natural image datasets [7, 41, 48]. Another category of examples is large-scale networks or complicated tasks. Prior works (*e.g.,* [49, 50, 55]) show that random initialization may end up with poor convergence for these tasks. As a result, using general-purpose pre-trained models is a common and well accepted practice in deep learning.

Data Quality. The definition of data quality is also task-specific. In each trading session, the collector decides a provider's data quality based on how relevant are the intermediate data features submitted by the provider to the collector's current task. To quantify such relevance, the collector trains an evaluation model based on each provider's intermediate features, and then testify the model's performance on the collector's preowned validation dataset. Since the collector uses the same training techniques and validation dataset across all providers, they are evaluated equally and ranked fairly.

3 SYSTEM DESIGN

Overall, a trading session on HyperX goes through three major phases: Commitment, Evaluation and Closure, as shown in Figure 1. In the Commitment phase, the collector and providers collectively initialize a trading session, where the collector commits its trading task, along with the coded trading protocol, via a smart contract $\mathcal{F}_{\text{SC}}$, and the providers, that decide to join the trading session after reviewing the task, profile their datasets using a datastore service $\mathcal{F}_{\text{DS}}$ and register themselves to $\mathcal{F}_{\text{SC}}$ as participants. The Evaluation phase centers around our privacy-preserving data quality assessment design where providers submit their intermediate data features extracted from a randomly sampled subset of their whole datasets, based on which the collector applies transfer learning to quantify the feature quality. In the Closure phase, if the collector decides to trade with a specific provider, they go through a review-and-exchange mechanism to *atomically* close the deal such that the provider's committed full-dataset and the collector's committed reward are exchanged simultaneously, *i.e.,* cash on delivery, without relying on any trusted entities.

$\text{Prot}_{\text{HyperX}}$ is our umbrella protocol executing all these three phases. $\text{Prot}_{\text{HyperX}}$ is further divided into three preliminary protocols. In particular, Prot_{COL} and Prot_{PRI} define the execution protocols implemented by the collector and providers, respectively. Prot_{SC} is the protocol realization of the trading smart contract $\mathcal{F}_{\text{SC}}$. Finally, $\text{Prot}_{\text{HyperX}}$ also relies on two readily-available infrastructures: a general-purposed blockchain platform $\mathcal{F}_{\text{BC}}$ to host our trading smart contract and a persistent datastore service $\mathcal{F}_{\text{DS}}$ to store data. We specify the protocols of $\text{Prot}_{\text{HyperX}}$ in Figure 3. Next, we explain these protocols as a session goes through each phase.

3.1 Commitment Phase

Part (a) of Figure 1 illustrates the Commitment phase. A trading session is started by the collector Prot_{COL} via the StartSession interface, in which Prot_{COL} initializes the learning task for which the session recruits data and deploys a smart contract to unambiguously protocolize the trading process.

3.1.1 Initialization for Privacy-Preserving Data Evaluation

One of the key challenges of HyperX is to enable the collector to accurately, yet confidentially, assess and rank the data qualities from different providers. A strawman design would be that each provider discloses a fraction of its original data, based on which the collector trains a model and evaluates the model against its validation dataset. However, besides the privacy risk caused by disclosing the data samples, directly training with small-scale data from scratch often results in overfitting, greatly negating the data-quality assessment accuracy (see quantitative results in § 5.1).

Feature Extractor. To tackle this issue, we propose a novel data quality assessment mechanism whose high-level idea is captured in Figure 2. First, the collector is required to pre-train a deep model (denoted as M_{init}) using generic (*i.e.,* not task-specific) datasets. For instance, the collector recruiting data for training a deep robot image classifier may choose to pre-train a neural network using a generic image dataset that does not include robots (*e.g.,* ImageNet [51]). The backbone network of the pre-trained model M_{init} (*i.e.,* the first several layers of M_{init}) tends to capture generic features that can be used as the foundation for further training task-specific models. We truncate the first K layers of the backbone into $\mathsf{M}_{\text{extractor}}$ and use it as a *feature extractor.*

Confidential Feature Submission. Providers use this feature extractor to *confidentially submit their data samples*, *i.e.,* instead of directly disclosing any raw data, the providers only submit *irreversible intermediate features* computed using $\mathsf{M}_{\text{extractor}}$ on *a randomly sampled subset* of their whole dataset. The intermediate features are irreversible because it is infeasible to reliably recover the original data given these features. We validate and prove the property of irreversibility via both empirical experiments (see § 5.2.1) and theoretical analysis (see Lemma 6.1 in § ??). Afterwards, based on the submitted features of a provider, the collector applies transfer learning (TL) to tune the trainable layers of M_{init}, shaping the generic M_{init} into an evaluation model. The performance of the evaluation model on the collector's validation dataset quantifies the data quality of the provider.

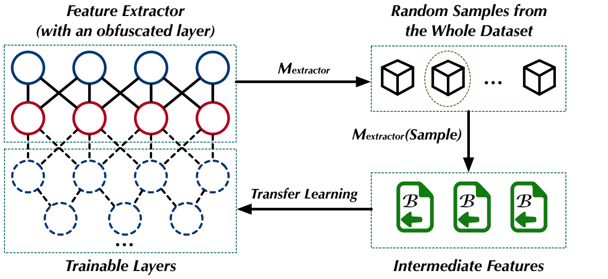


Figure 2: The architecture of HyperX's privacy-preserving data quality assessment design. A feature extractor $\mathsf{M}_{\text{extractor}}$ is applied to ensure that providers only disclose provably irreversible intermediate features extracted from a small fraction of its dataset during the Evaluation phase. Meanwhile, the privacy of $\mathsf{M}_{\text{extractor}}$ can also be protected by obfuscating its last linear layer.

Obfuscating the Feature Extractor. The above approach provides strong privacy guarantee for both the providers and collector. On the one hand, it is clear that the privacy of datasets is preserved since providers only disclose irreversible intermediate data features. On the other hand, the M_{init} and all other concrete models trained by the collector are never disclosed. Meanwhile, the collector can further reclaim the privacy of $\mathsf{M}_{\text{extractor}}$ by mutating its last linear layer. Specifically, denoting the last layer's parameters of $\mathsf{M}_{\text{extractor}}$ as W_k, then W_k is obfuscated as follows

$$\hat{W}_k = W_k \cdot \delta + \sigma, \tag{1}$$

where δ and σ are private multiplicative and additive mutation coefficients, respectively, selected by the collector. We provide both empirical evaluations (see § 5.2.2) and theoretical analysis (see Lemma 6.2) to demonstrate the effectiveness of protecting $\mathsf{M}_{\text{extractor}}$ via obfuscation.

Intermediate Feature Computation. Given the obfuscated $\mathsf{M}_{\text{extractor}}$ (composing of the original layers $f(\cdot)$ and the obfuscated $\hat{W}_k$), the way to compute the intermediate features $\widehat{\mathcal{B}}$ of data $\mathbf{x}$ is as follows (we denote the feature extractor function as $\mathcal{F}_{\text{Extr}}$)

$$\widehat{\mathcal{B}} = \mathcal{F}_{\text{Extr}}(\mathbf{x}) = \hat{W}_k f(\mathbf{x}). \tag{2}$$

The providers use $\mathcal{F}_{\text{Extr}}$ to submit features for their data samples later in the Evaluation phase. Further, since the collector needs to decode $\widehat{\mathcal{B}}$ using its private (δ, σ) in order to perform TL in the Evaluation phase, $\mathcal{F}_{\text{Extr}}(\cdot)$ should not contain any nonlinear activation in the last layer to ensure that $\widehat{\mathcal{B}}$ is decodable. Finally, the collector should publish any required data preprocessing descriptions (*e.g.,* input dimension adjustment) via the configuration of Prot_{SC} (see details below) so that the providers can preprocess their raw data samples into acceptable inputs to $\mathcal{F}_{\text{Extr}}$.

Deep learning in general refers to training multi-layer models with non-linearity. Our proposed quality assessment method is based on such a model structure, and is therefore generalizable to deep-learning tasks. Whether this approach can be generalized to non-DeepLearning tasks is beyond the scope of this paper.

3.1.2 *Trading Contract Deployment*

After acquiring the $\mathsf{M}_{\mathsf{extractor}}$ offline and storing it on $\mathcal{F}_{\mathsf{DS}}$, $\mathsf{Prot}_{\mathsf{COL}}$ deploys and instantiates a trading contract on $\mathcal{F}_{\mathsf{BC}}$ by calling the CreateContract interface of $\mathsf{Prot}_{\mathsf{SC}}$. $\mathsf{Prot}_{\mathsf{SC}}$ also defines multiple other interfaces that shall be called by both the collector and providers throughout the trading process. In this section, we first detail several key designs in the CreateContract interface.

Contract Configuration $\mathbf{SC}_{\mathbf{config}}$. $\mathsf{Prot}_{\mathsf{SC}}$ is initialized with a configuration $\mathsf{SC}_{\mathsf{config}}$, along with the retrieval key $\mathsf{adr}_{\hat{\theta}}$ for $\mathsf{M}_{\mathsf{extractor}}$ on $\mathcal{F}_{\mathsf{DS}}$ (line 74). The $\mathsf{SC}_{\mathsf{config}}$ defines the key properties of $\mathsf{Prot}_{\mathsf{SC}}$, which are *immutable* once $\mathsf{Prot}_{\mathsf{SC}}$ is deployed. We specify several configuration properties in Figure 3 (line 75). $\mathsf{CF}_{\mathsf{reward}}$ is the reward that the collector will pay if the trade is successfully closed; $\mathsf{CF}_{\%}$ is the maximum sampling percentage that the collector can sample from a provider's dataset in order to assess its data quality; $\mathsf{CF}_{\mathsf{deposit}}$ is the minimal required (refundable) deposit from any provider to join the trading session; $\mathsf{CF}_{\mathsf{expiry}}$ is the expiration time of the trading session. $\mathsf{SC}_{\mathsf{config}}$ may contain other properties, such as the data preprocessing requirements, the expected data sharding format for each provider, or anything that should be part of the trading agreement. Providers should review $\mathsf{SC}_{\mathsf{config}}$ to decide whether to join the trading session or not, at their own discretion.

Provider Profile Initialization. In $\mathsf{Prot}_{\mathsf{SC}}$, each provider is profiled with a list of properties (line 77). (i) $\mathsf{adr}^{\mathsf{h}}_{\mathsf{data}}$, provided upon a provider's registration to the session, is the set of *hashed* data retrieval keys on $\mathcal{F}_{\mathsf{DS}}$ to locate the whole dataset that the provider is planning to trade. By containing the hashes of retrieval key, $\mathsf{adr}^{\mathsf{h}}_{\mathsf{data}}$ serves as the provider's privacy-preserving dataset commitment for the trading session. (ii) $\mathsf{adr}^{\mathsf{h}}_{\mathsf{sample}}$, which is later set by the collector via the SampleData interface (line 85), stores the subset of retrieval key hashes for the data sampled by the collector. The provider is expected to only submit features for these sampled data for quality evaluation. The size of the sampled data is bounded by the sampling percentage $\mathsf{CF}_{\%}$, defined in $\mathsf{SC}_{\mathsf{config}}$. Although our data quality assessment method does not disclose any raw data, bounding the sampling size is still desirable to hard-limit the scope of involved data from each provider. (iii) $\mathsf{adr}_{\mathsf{feature}}$ stores the $\mathcal{F}_{\mathsf{DS}}$ retrieval keys for the provider's intermediate data features, which shall be submitted by the provider via the SubmitFeatures interface. (iv) $\mathsf{adr}^{\mathsf{p}}_{\mathsf{sample}}$, $\mathsf{adr}^{\mathsf{p}}_{\mathsf{data}}$ and $\mathsf{K}_{\mathsf{data}}$ are only relevant to the provider from which the collector plans to buy data in the Closure phase. We will explain their functionality in § 3.3. Finally, $\mathsf{V}_{\mathsf{deposit}}$ records the deposit made by the provider which can be reclaimed after the trading session is either closed or expired.

Initialization for Reward Claim. After the trading session ends, only the *optimal* provider $\mathsf{P}_{\mathsf{opt}}$ (if the collector does select one to buy data from) is able to claim the $\mathsf{CF}_{\mathsf{reward}}$ by revealing its committed dataset. To atomically close the deal without trusted authorities, we employ a review-and-exchange mechanism in the Closure phase. In this design, a *candidate* provider, denoted P_{tc}, is required (line 78). Finally, the overall state for the trading session $\mathsf{st}_{\mathsf{trade}}$ (line 79) may be in any of the following states {pending, reviewing, reviewed, closed}. Throughout the trading process, $\mathsf{st}_{\mathsf{trade}}$ must be gradually promoted following the above sequence. Given a $\mathsf{st}_{\mathsf{trade}}$, only a certain subset of the interfaces in $\mathsf{Prot}_{\mathsf{SC}}$ are executable, as described in Figure 3.

3.1.3 *Provider Participation*

Providers should proactively examine $\mathcal{F}_{\mathsf{BC}}$ to screen potentially interested trading contracts. The provider protocol $\mathsf{Prot}_{\mathsf{PRI}}$ accomplishes this by initiating a proactive watching service to $\mathcal{F}_{\mathsf{BC}}$ (line 45). After reviewing the $\mathsf{SC}_{\mathsf{config}}$ of a deployed trading contract $\mathsf{Prot}_{\mathsf{SC}}$, if a provider decides to participate in the trading session, the provider first needs to commit its dataset $\mathcal{D}_{\mathsf{whole}}$ to $\mathcal{F}_{\mathsf{DS}}$. $\mathcal{D}_{\mathsf{whole}}$ should be encrypted by a fresh symmetric key generated for the session. Meanwhile, to facilitate sampling, $\mathcal{D}_{\mathsf{whole}}$ is required to be sharded into disjoint and (ideally) equal-sized chunks, where each of the shards is retrievable from $\mathcal{F}_{\mathsf{DS}}$ via a unique address (or retrieval key). The preferred size of each shard, as well as the number of shards expected from one provider, is part of the trading agreement configured via $\mathsf{SC}_{\mathsf{config}}$. The hashes of all these data-shard retrieval keys are committed to $\mathsf{Prot}_{\mathsf{SC}}$ via the Registration interface (line 80) to disallow any subsequent changes on $\mathcal{D}_{\mathsf{whole}}$.

The collector samples each provider's dataset by selecting a certain percentage of these hashed retrieval keys (see § 3.2). Since the sampling is controlled by the collector, the provider gains only negligible advantages by strategically sharding its dataset on $\mathcal{F}_{\mathsf{DS}}$. Meanwhile, any deliberate manipulation of $\mathcal{D}_{\mathsf{whole}}$, such as inflating the size of $\mathcal{D}_{\mathsf{whole}}$ by duplicating entries or adding irrelevant data entries, may end up disadvantaging the provider if these data chunks are eventually sampled for quality assessment. Therefore, dataset manipulation is at the provider's own risk.

3.2 Evaluation Phase

The Evaluation phase is illustrated in part (b) of Figure 3.

One-time Data Sampling. $\mathsf{Prot}_{\mathsf{COL}}$ further spawns another proactive watching service to $\mathcal{F}_{\mathsf{BC}}$ to examine the state of $\mathsf{Prot}_{\mathsf{SC}}$ in order to decide the proper next step (line 54). Once a provider is successfully registered, $\mathsf{Prot}_{\mathsf{COL}}$ can start the data quality assessment process for the provider by calling the SampleData interface to set the sampling addresses $\mathsf{adr}^{\mathsf{h}}_{\mathsf{sample}}$ for the provider. Since $\mathsf{adr}^{\mathsf{h}}_{\mathsf{sample}}$ is randomly generated by $\mathsf{Prot}_{\mathsf{COL}}$, the subset of $\mathcal{D}_{\mathsf{whole}}$ identified by $\mathsf{adr}^{\mathsf{h}}_{\mathsf{sample}}$ is likely to be a good representative for the whole dataset $\mathcal{D}_{\mathsf{whole}}$, even though the provider fully controls how $\mathcal{D}_{\mathsf{whole}}$ is stored on $\mathcal{F}_{\mathsf{DS}}$. The total amount of sampled data chunks is bounded by $\mathsf{CF}_{\%}$. The collector can only sample each provider once (line 87).

Intermediate Data Feature Extraction. Once $\mathsf{Prot}_{\mathsf{PRI}}$ notices that its sample addresses have been set, it can proceed with the feature extraction. Based on the given $\mathsf{adr}^{\mathsf{h}}_{\mathsf{sample}}$, $\mathsf{Prot}_{\mathsf{PRI}}$ is able to identify the corresponding plain retrieval keys and then retrieves the sampled data shards from $\mathcal{F}_{\mathsf{DS}}$. Then it obtains $\mathsf{M}_{\mathsf{extractor}}$ from $\mathcal{F}_{\mathsf{DS}}$ via key $\mathsf{adr}_{\hat{\theta}}$ and computes the intermediate data features $\widehat{\mathcal{B}}$ using Equation (2). Afterwards, $\mathsf{Prot}_{\mathsf{PRI}}$ stores $\widehat{\mathcal{B}}$ on $\mathcal{F}_{\mathsf{DS}}$ with a root retrieval key $\mathsf{adr}_{\mathsf{feature}}$, under which multiple storage chunks can be hierarchically linked if $\widehat{\mathcal{B}}$ does not fit in a single chunk. $\mathsf{Prot}_{\mathsf{PRI}}$ then commits its features by calling the SubmitFeatures interface parameterized with $\mathsf{adr}_{\mathsf{feature}}$. To ensure that $\mathsf{adr}_{\mathsf{feature}}$ is only visible to the collector, $\mathsf{Prot}_{\mathsf{PRI}}$ may encrypt it with the collector's public key (which could be specified via $\mathsf{SC}_{\mathsf{config}}$).

```
$\text{Prot}_{\text{COL}}$: The Execution Protocol for the Collector
Init Data := $\emptyset$
Daemon StartSession():
    Generate the session ID sid := $\{0,1\}^{\lambda}$
    Offline preparation of the feature extractor $\text{M}_{\text{extractor}}$
    Store $\text{M}_{\text{extractor}}$ on $\mathcal{F}_{\text{DS}}$ at address $\text{adr}_{\hat{\theta}} := \mathcal{F}_{\text{DS}}.\text{Store}(\text{M}_{\text{extractor}})$
    Generate the trading session configuration $\text{SC}_{\text{config}}$
    Obtain contract := $\text{Prot}_{\text{SC}}.\text{CreateContract}(\text{SC}_{\text{config}}, \text{adr}_{\hat{\theta}})$
    package contract as a valid transaction $\widehat{contract}$
    call $\mathcal{F}_{\text{BC}}.\text{Execute}(\widehat{contract})$ to deploy the $\widehat{contract}$
    halt until $\widehat{contract}$ is finalized on $\mathcal{F}_{\text{BC}}$
    Initialize Data[sid]:=$\mathcal{S}_{\text{Priv}}$ where $\mathcal{S}_{\text{Priv}}$ is a map storing providers
Daemon Watching(sid, $\mathcal{F}_{\text{BC}}$) private:
  $\mathcal{S}_{\text{Priv}}$ := Data[sid]; abort if not found
  # Triggered when a transaction executing $\text{Prot}_{\text{SC}}$ is finalized on $\mathcal{F}_{\text{BC}}$
  On an interface of $\text{Prot}_{\text{SC}}$ is successfully exec. with parameters Param:
    if interface = $\text{Prot}_{\text{SC}}$.Registration:
      Set the sampling addresses $\text{adr}^{\text{h}}_{\text{sample}} := \text{Sample}(\text{Param.adr}^{\text{h}}_{\text{data}})$
      Initialize $\mathcal{S}_{\text{Priv}}$[Param.sender] := [score:=0, $\text{K}_{\text{data}}$:=null,
                   $\text{adr}^{\text{p}}_{\text{data}}$:=null, $\widehat{\mathcal{B}}$:=null, $\text{adr}^{\text{h}}_{\text{sample}}$:=$\text{adr}^{\text{h}}_{\text{sample}}$]
      Call $\text{Prot}_{\text{SC}}$.SampleData(Param.sender, $\text{adr}^{\text{h}}_{\text{sample}}$)
    if interface = $\text{Prot}_{\text{SC}}$.SubmitFeatures:
      (score, _, _, $\widehat{\mathcal{B}}$, _) := $\mathcal{S}_{\text{Priv}}$[Param.sender]
      Update $\widehat{\mathcal{B}} := \mathcal{F}_{\text{DS}}.\text{Get}(\text{Param.adr}_{\text{feature}})$; abort if not found
      Train an evaluation model $\text{M}_{\text{eval}}$ based on $\widehat{\mathcal{B}}$ using Equation (4)
      Evaluate $\text{M}_{\text{eval}}$ against the collector's preowned validation dataset
      Update the quality score of $\widehat{\mathcal{B}}$ using $\text{M}_{\text{eval}}$'s performance
    if interface = $\text{Prot}_{\text{SC}}$.SubmitReview:
      (_, $\text{K}_{\text{data}}$, _, $\widehat{\mathcal{B}}$, $\text{adr}^{\text{h}}_{\text{sample}}$) := $\mathcal{S}_{\text{Priv}}$[Param.sender]
      Abort if hashes of $\text{Param.adr}^{\text{p}}_{\text{sample}}$ and $\text{adr}^{\text{h}}_{\text{sample}}$ are inconsistent
      Recompute features using the data from $\mathcal{F}_{\text{DS}}.\text{Get}(\text{Param.adr}^{\text{p}}_{\text{sample}})$
      Abort if the recomputed features do not match the committed $\widehat{\mathcal{B}}$
      Update $\text{K}_{\text{data}} := \text{Param.K}_{\text{data}}$
      Call $\text{Prot}_{\text{SC}}$.SetOptimal(Param.sender) to promote the candidate
    if interface = $\text{Prot}_{\text{SC}}$.ClaimReward:
      Update $\mathcal{S}_{\text{Priv}}[\text{Param.sender}].\text{adr}^{\text{p}}_{\text{data}} := \text{Param.adr}^{\text{p}}_{\text{data}}$
      Close the trading session by updating Data.Erase(sid)
Timeout Callback EndEvalution(sid):
    $\mathcal{S}_{\text{Priv}}$ := Data[sid]; abort if not found
    if the highest-score $\text{P}_{\text{opt}}$ in $\mathcal{S}_{\text{Priv}}$ provides a satisfactory dataset:
      Call $\text{Prot}_{\text{SC}}$.SetCandidate($\text{P}_{\text{opt}}$) to select the candidate
    else : Close the trading session by updating Data.Erase(sid)
$\text{Prot}_{\text{PRI}}$: The Execution Protocol for Providers
Init Data := $\emptyset$
Daemon Watching($\mathcal{F}_{\text{BC}}$) private:
    Examine the $\text{SC}_{\text{config}}$ and $\text{M}_{\text{extractor}}$ of a freshly deployed $\text{Prot}_{\text{SC}}$
    Skip if $\text{Prot}_{\text{SC}}$ is not interested
    Generate session ID sid and a fresh data encryption key $\text{K}_{\text{data}}$
    Store the encrypted dataset $\text{adr}^{\text{p}}_{\text{data}} := \mathcal{F}_{\text{DS}}.\text{Store}(\mathcal{D}_{\text{whole}})$ (following
      the data sharding requirement specified in $SC_{config}$)
    Compute the hashed data retrieval keys $\text{adr}^{\text{h}}_{\text{data}} := \text{Hash}\{\text{adr}^{\text{p}}_{\text{data}}\}$
    Call $\text{Prot}_{\text{SC}}$.Registration($\text{adr}^{\text{h}}_{\text{data}}$) to participate in the session
    Initialize Data[sid] := {$\text{adr}^{\text{p}}_{\text{data}}$, $\text{adr}^{\text{h}}_{\text{data}}$, $\text{K}_{\text{data}}$, $\text{adr}^{\text{p}}_{\text{sample}}$:=null}
Daemon Watching(sid, $\mathcal{F}_{\text{BC}}$) private:
  ($\text{adr}^{\text{p}}_{\text{data}}$, $\text{adr}^{\text{h}}_{\text{data}}$, $\text{K}_{\text{data}}$, $\text{adr}^{\text{p}}_{\text{sample}}$) := Data[sid]; abort if not found
  On an interface of $\text{Prot}_{\text{SC}}$ is successfully exec. with parameters Param:
    if interface = $\text{Prot}_{\text{SC}}$.SampleData:
      # Proceed only if the interfaced is executed for the provider
      Abort if Param.pid is not self
      Lookup the plain retrieval keys $\text{adr}^{\text{p}}_{\text{sample}}$ for $\text{Param.adr}^{\text{h}}_{\text{sample}}$
      Compute the intermediate features $\widehat{\mathcal{B}} := \mathcal{F}_{\text{Extr}}(\mathcal{F}_{\text{DS}}.\text{Get}(\text{adr}^{\text{p}}_{\text{sample}}))$
      Call $\text{Prot}_{\text{SC}}$.SubmitFeatures($\mathcal{F}_{\text{DS}}.\text{Store}(\widehat{\mathcal{B}})$) to commit $\widehat{\mathcal{B}}$
    if interface = $\text{Prot}_{\text{SC}}$.SetCandidate:
      Abort if Param.pid is not self (only relevant to $\text{P}_{\text{tc}}$)
      Call $\text{Prot}_{\text{SC}}$.SubmitReview($\text{adr}^{\text{p}}_{\text{sample}}$, $\text{K}_{\text{data}}$) to provide data samples
    if interface = $\text{Prot}_{\text{SC}}$.SetOptimal:
      Abort if Param.pid is not self (only relevant to $\text{P}_{\text{opt}}$)
      Call $\text{Prot}_{\text{SC}}$.ClaimReward($\text{adr}^{\text{p}}_{\text{data}}$) to close the deal
Timeout Callback SessionTimeout(sid):
    (_, _, _, _) := Data[sid]; abort if not found
    Call $\text{Prot}_{\text{SC}}$.ClaimDesposit()
    Close the trading session by updating Data.Erase(sid)
$\text{Prot}_{\text{SC}}$: Protocol Description of the Trading Smart Contract
Upon Receive CreateContract($\text{SC}_{\text{config}}$, $\text{adr}_{\hat{\theta}}$):
    Assert $\text{SC}_{\text{config}} = \{\text{CF}_{\text{reward}}, \text{CF}_{\text{deposit}}, \text{CF}_{\%}, \text{CF}_{\text{expiry}}, \dots\}$
    Initialize a map $\text{P}_{\text{profile}}$ to store the profiles of participating providers
    For pid, profile $\text{P}_{\text{profile}}$[pid] with the following properties: {$\text{adr}^{\text{h}}_{\text{data}}$, $\text{adr}^{\text{h}}_{\text{sample}}$,
    $\text{adr}_{\text{feature}}$, $\text{adr}^{\text{p}}_{\text{data}}$, $\text{adr}^{\text{p}}_{\text{sample}}$, $\text{K}_{\text{data}}$, $\text{V}_{\text{deposit}}$}
    Initialize 3 identifiers $\text{P}_{\text{col}}$ := msg.sender, $\text{P}_{\text{tc}}$ := null, $\text{P}_{\text{opt}}$ := null
    Initialize the state of trading session as $\text{st}_{\text{trade}}$ := pending
Upon Receive Registration($\text{adr}^{\text{h}}_{\text{data}}$):
    Assert msg.value $\geq \text{CF}_{\text{deposit}}$ and assert $\text{st}_{\text{trade}}$ = pending
    Get pid := msg.sender and assert pid is not in $\text{P}_{\text{profile}}$
    Update $\text{P}_{\text{profile}}[\text{pid}].\text{adr}^{\text{h}}_{\text{data}} := \text{adr}^{\text{h}}_{\text{data}}$
    Update $\text{P}_{\text{profile}}[\text{pid}].\text{V}_{\text{deposit}}$ := msg.value
Upon Receive SampleData(pid, $\text{adr}^{\text{h}}_{\text{sample}}$):
    Assert msg.sender = $\text{P}_{\text{col}}$ and assert pid is in $\text{P}_{\text{profile}}$
    Assert $\text{st}_{\text{trade}}$ = pending and assert $\text{P}_{\text{profile}}[\text{pid}].\text{adr}^{\text{h}}_{\text{sample}} = \emptyset$
    if $\text{adr}^{\text{h}}_{\text{sample}} = \emptyset$ :
      Update $\text{adr}^{\text{h}}_{\text{sample}} := \text{RandomSample}(\text{P}_{\text{profile}}[\text{pid}].\text{adr}^{\text{h}}_{\text{data}}, \text{CF}_{\%})$
    Assert $\text{adr}^{\text{h}}_{\text{sample}}.\text{Size}() \leq \text{P}_{\text{profile}}[\text{pid}].\text{adr}^{\text{h}}_{\text{data}}.\text{Size}() \cdot \text{CF}_{\%}$
    Update $\text{P}_{\text{profile}}[\text{pid}].\text{adr}^{\text{h}}_{\text{sample}} := \text{adr}^{\text{h}}_{\text{sample}}$
Upon Receive SubmitFeatures($\text{adr}_{\text{feature}}$):
    Get pid := msg.sender and assert pid is in $\text{P}_{\text{profile}}$
    Assert $\text{st}_{\text{trade}}$ = pending and $\text{P}_{\text{profile}}[\text{pid}].\text{adr}_{\text{feature}} = \emptyset$
    Update $\text{P}_{\text{profile}}[\text{pid}].\text{adr}_{\text{feature}} := \text{adr}_{\text{feature}}$
Upon Receive SetCandidate(pid):
    Assert msg.sender = $\text{P}_{\text{col}}$ and assert $\text{st}_{\text{trade}}$ = pending
    Assert pid is in $\text{P}_{\text{profile}}$ and assert $\text{P}_{\text{profile}}[\text{pid}].\text{adr}^{\text{p}}_{\text{sample}} = \emptyset$
    Update $\text{P}_{\text{tc}}$ := pid and update $\text{st}_{\text{trade}}$ := reviewing
Upon Receive SubmitReview($\text{adr}^{\text{p}}_{\text{sample}}$, $\text{K}_{\text{data}}$):
    Get pid := msg.sender and assert pid = $\text{P}_{\text{tc}}$
    Assert $\text{st}_{\text{trade}}$ = reviewing and assert pid is in $\text{P}_{\text{profile}}$
    Assert Hash({$\text{adr}^{\text{p}}_{\text{sample}}$}) = $\text{P}_{\text{profile}}[\text{pid}].\text{adr}^{\text{h}}_{\text{sample}}$
    Update $\text{P}_{\text{profile}}[\text{pid}].\text{K}_{\text{data}} := \text{K}_{\text{data}}$; $\text{P}_{\text{profile}}[\text{pid}].\text{adr}^{\text{p}}_{\text{sample}} := \text{adr}^{\text{p}}_{\text{sample}}$
Upon Receive SetOptimal(pid):
    Assert msg.sender = $\text{P}_{\text{col}}$ and assert $\text{st}_{\text{trade}}$ = reviewing
    Assert $\text{P}_{\text{opt}}$ = null and assert pid = $\text{P}_{\text{tc}}$
    Update $\text{P}_{\text{opt}}$ := pid and update $\text{st}_{\text{trade}}$ = reviewed
Upon Receive ClaimReward($\text{adr}^{\text{p}}_{\text{data}}$):
    Assert msg.sender = $\text{P}_{\text{opt}}$ and assert $\text{st}_{\text{trade}}$ = reviewed
    Assert Hash({$\text{adr}^{\text{p}}_{\text{data}}$}) = $\text{P}_{\text{profile}}[\text{P}_{\text{opt}}].\text{adr}^{\text{h}}_{\text{data}}$
    Update $\text{P}_{\text{profile}}[\text{P}_{\text{opt}}].\text{adr}^{\text{p}}_{\text{data}} = \text{adr}^{\text{p}}_{\text{data}}$
    transfer(msg.sender, $\text{CF}_{\text{reward}}$)
    Update $\text{st}_{\text{trade}}$ := closed
Upon Receive ClaimDesposit():
    Assert sys.Now() > $\text{CF}_{\text{expiry}}$ or $\text{st}_{\text{trade}}$ := closed
    Get pid := msg.sender and assert pid is in $\text{P}_{\text{profile}}$
    transfer(msg.sender, $\text{P}_{\text{profile}}[\text{pid}].\text{V}_{\text{deposit}}$)
    Update $\text{P}_{\text{profile}}$.Erase(pid) (one provider can claim deposit only once)
```

Figure 3: The protocol specification for Prot_{COL}, Prot_{PRI} and Prot_{SC}. Codes in gray background represent the non-blocking data quality evaluation process that is executed offline. Interfaces with a statement *assert msg.sender = P_{col}* are executable only by the collector (other similar statements have similar meanings).

Data Quality Evaluation. After a provider submits its data features, $\mathsf{Prot}_{\mathsf{COL}}$ starts the data quality assessment (an offline non-blocking process). $\mathsf{Prot}_{\mathsf{COL}}$ retrieves $\widehat{\mathcal{B}}$ from $\mathcal{F}_{\mathsf{DS}}$, reverts the private coefficients added in Equation (1), and then applies an activation function to extract the final features as follows:

$$\mathcal{B} = f_{\text{actv}}[(\widehat{\mathcal{B}} - \sigma) \cdot \frac{1}{\delta}], \tag{3}$$

where f_{actv} is an activation function. To quantify the quality of $\mathcal{B}$, $\mathsf{Prot}_{\mathsf{COL}}$ uses $\mathcal{B}$ to fine-tune the trainable layers of the M_{init} (see Figure 2) to obtain an evaluation model M_{eval}. The following objective function is used in the fine-tuning process.

$$\min_{\mathsf{M}_{\text{eval}}} \sum \mathcal{L}(y_{\text{sample}}, \mathsf{M}_{\text{eval}}(\mathcal{B})), \tag{4}$$

where $\mathcal{L}$ is the loss function and y_{sample} are data labels that can be stored, for instance, as auxiliaries of $\widehat{\mathcal{B}}$ on $\mathcal{F}_{\mathsf{DS}}$ (omitted in Figure 3 for simplicity). $\mathsf{Prot}_{\mathsf{COL}}$ then tests M_{eval} against its validation dataset prepared a priori and uses the testing accuracy to quantify the provider's dataset quality.

3.3 Closure Phase

The Closure phase is illustrated by the part (c) of Figure 3. It is triggered when the EndEvalution callback in $\mathsf{Prot}_{\mathsf{COL}}$ times out. We explain several crucial designs below.

Review-and-Exchange. After evaluating the intermediate features submitted by all participated providers, the collector selects a provider with the best features, *i.e.,* the optimal provider P_{opt}, to close the deal (the collector may abort if none of the providers have offered sufficiently good features, see Post-review Abortion below). They use the following *review-and-exchange* mechanism to close the deal, which guarantees that the collector and P_{opt} can *simultaneously* exchange their goods (*i.e.,* the collector's reward and P_{opt}'s full dataset), without requiring mutual trusts between them or a trusted third party holding their goods temporally.

Review Dataset Quality. $\mathsf{Prot}_{\mathsf{COL}}$ starts the closure process by setting a candidate provider on $\mathsf{Prot}_{\mathsf{SC}}$ via the SetCandidate interface. If the selected candidate P_{tc} is willing to continue with the exchange, it needs to disclose its sampled data in plaintext to convince the collector that it does faithfully submit the intermediate data features. This disclosure process is done via the SubmitReview interface where P_{tc} submits the data retrieval keys for the sampled data shards in plaintext ($\mathsf{adr}^{\mathsf{p}}_{\text{sample}}$), as well as the session key K_{data} used for encrypting the whole dataset. Note that disclosing K_{data} at this stage is safe because the collector does not know other retrieval keys for P_{tc}'s data shards. As a result, non-sampled shards are still kept private. Again, these data could be encrypted using the collector's public key to hide them from the public.

With $\mathsf{adr}^{\mathsf{p}}_{\text{sample}}$ and K_{data} given by P_{tc}, the collector is able to retrieve the sampled data from $\mathcal{F}_{\mathsf{DS}}$, decrypt it, and recompute intermediate data features. If the recomputed features are consistent with ones submitted by P_{tc}, the collector has high confidence in P_{tc} to faithfully follow the protocol so that the quality of its entire dataset is assured by the sample quality. Otherwise, the collector may choose to abort the session (see how to defend against malicious abortion by the collector below). To continue with the exchange, the collector promotes P_{tc} as the optimal provider P_{opt} via the SetOptimal interface.

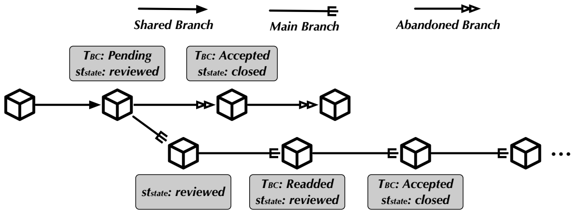


Figure 4: A transaction $\mathcal{T}_{\mathsf{BC}}$ sent to execute ClaimReward interface could be abandoned due to temporary blockchain reorganization. This does not undermine the *eventual exchange atomicity* because the contract state $\mathsf{st}_{\text{trade}}$ remains reviewed on the main branch, allowing P_{opt} to re-add $\mathcal{T}_{\mathsf{BC}}$ to the transaction pool for execution on the main branch.

Atomic Exchange. Finally, P_{opt} is able to claim its reward $\mathsf{CF}_{\text{reward}}$ via the ClaimReward interface. As a prerequisite (line 110) for correctly executing the interface, P_{opt} is required to disclose all its data shards retrieval keys in plaintext (*i.e.,* $\mathsf{adr}^{\mathsf{p}}_{\text{data}}$). Since the collector has already possessed the session key K_{data}, it is able to acquire the unencrypted complete dataset committed by P_{opt}. Because the payment and $\mathsf{adr}^{\mathsf{p}}_{\text{data}}$ are exchanged in the same interface call (*i.e.,* in the same blockchain transaction), the exchange atomicity is assured by the atomicity of blockchain transactions.

In production, there is a subtle gap between the time when P_{opt} adds a blockchain transaction $\mathcal{T}_{\mathsf{BC}}$ executing the ClaimReward interface to the pending transaction pool of $\mathcal{F}_{\mathsf{BC}}$ and the time when $\mathcal{T}_{\mathsf{BC}}$ is finalized on $\mathcal{F}_{\mathsf{BC}}$. If $\mathsf{Prot}_{\mathsf{COL}}$ keeps monitoring the pending transaction pool, the collector is able to obtain $\mathsf{adr}^{\mathsf{p}}_{\text{data}}$ immediately after $\mathcal{T}_{\mathsf{BC}}$ is added to the pool, whereas P_{opt} can secure the reward only after $\mathcal{T}_{\mathsf{BC}}$ is finalized. The timing gap, however, does not undermine *eventual exchange atomicity* because $\mathsf{Prot}_{\mathsf{COL}}$ cannot prevent $\mathcal{T}_{\mathsf{BC}}$ from being eventually finalized as the decentralized consensus proceeds on $\mathcal{F}_{\mathsf{BC}}$.

A temporary blockchain reorganization might occur so that the branch including $\mathcal{T}_{\mathsf{BC}}$ is abandoned, as shown in Figure 4. P_{opt} can address this by adding $\mathcal{T}_{\mathsf{BC}}$ to the transaction pool again. Since the state of $\mathsf{Prot}_{\mathsf{SC}}$ on the main branch remains reviewed, the newly added $\mathcal{T}_{\mathsf{BC}}$ is still valid to execute the ClaimReward interface (*i.e.,* the check on line 110 still passes). Because $\mathcal{T}_{\mathsf{BC}}$ will be eventually final based on the transaction finality assumption in § 2.3, the eventual exchange atomicity is guaranteed.

Post-review Abortion. One concern is that the collector may maliciously abort the session even if the review passes (*i.e.,* P_{tc} is honest). The late abortion allows the collector to obtain a small subset of the raw data samples of P_{tc} without paying P_{tc}. Our design has built-in designs to address this concern. First, the post-review abortion can be effectively compensated by requiring the collector to pay a certain amount of nonrefundable compensation (*e.g.,* $\mathsf{CF}_{\%} \cdot \mathsf{CF}_{\text{reward}}$) to P_{tc} after P_{tc} discloses its raw data samples. Second, the collector can only select P_{tc} once in each session, and *therefore other providers are not subject to this attack.* Initiating many trading-sessions is not a rational strategy because (i) it is more expensive considering the additional cost of deploying many smart-contracts and (ii) the collector at best obtains sporadic samples from heterogeneous providers since the same provider may not participate in all sessions. It is unclear that these sporadic samples would fit together as a usable training dataset. Finally, $\mathsf{CF}_{\%}$ in $\mathsf{SC}_{\mathsf{config}}$ bounds the maximum sample size, which limits the worst case sample disclosure. As evaluated in § 5, $\mathsf{CF}_{\%}$ can be reasonably small (*e.g.,* less than 5%) without affecting the effectiveness of data quality assessment.

Enforcing Accountability with Penalties. Optionally, we could extend $\mathsf{Prot}_{\mathsf{SC}}$ with accountability enforcement on certain misconducts. For instance, the candidate provider P_{tc} should be penalized if it fails to provide data for review (line 103) or provide the entire dataset after being promoted as $\mathsf{P}_{\mathsf{opt}}$ (line 111), meaning that P_{tc} breaks its commitment of having a coherent dataset. This prevents an adversary from abusing the protocol, *e.g.,* submitting features computed from data shards that are not selected by the collector.

3.4 Theorems

We have also worked on theorems for both the protocol $\mathsf{Prot}_{\mathsf{HyperX}}$ and the privacy-preserving data quality assessment design. We formulate the security properties of $\mathsf{Prot}_{\mathsf{HyperX}}$ and prove them using the Universal Composability framework [9]. The theoretical analysis of the data quality assessment design is based on general learning theories. Due to page limit, we defer these to report [**?**].

4 IMPLEMENTATION

In this section, we report the implementation of a HyperX prototype. At the time of writing, the total development effort includes (i) ~3, 600 lines of code, mainly in Solidity and JavaScript, to implement the infrastructural protocols (*i.e.,* $\mathsf{Prot}_{\mathsf{COL}}$, $\mathsf{Prot}_{\mathsf{PRI}}$ and $\mathsf{Prot}_{\mathsf{SC}}$) and (ii) ~2, 880 lines of Python code for implementing our privacy-preserving data quality assessment design. Source code can be provided upon request.

4.1 Trading Platform Implementation

The trading platform of HyperX relies on two infrastructural primitives: a smart contract platform and a datastore service that persists all storage requests. We use Ethereum, the flagship production blockchain network, to host our trading smart contract, and the datastore service is implemented using IPFS. We implement $\mathsf{Prot}_{\mathsf{SC}}$ in Truffle v5.1.17 with Solidity v0.5.16 (solc-js), Node v8.17.0, and Web3.js v1.2.1. The $\mathsf{Prot}_{\mathsf{PRI}}$ and $\mathsf{Prot}_{\mathsf{COL}}$ are implemented based on the Web3.js v1.2.1. We test our implementation on an Ethereum private network with Geth v1.9.11-stable. On IPFS, we shard each provider's dataset into a certain number of shards where each shard can host an arbitrary amount of data by hierarchically constructing a tree of data blocks under the shard root. Both the shard count and shard size are part of the $\mathsf{Prot}_{\mathsf{SC}}$ configuration $\mathsf{SC}_{\mathsf{config}}$.

For completeness, we report the gas cost for running the trading contract on Ethereum. At the time of writing, the gas price was 20 Gwei in all transactions. The contract deployment costs 1,551,635 gas, roughly translating to 0.031 Ether. In Figure 5, we report the gas cost for executing the contract interfaces on the Ethereum Mainnet. The gas cost of many interfaces, such as Registration, SampleData, SubmitReview and ClaimReward, depends on the shard count of each provider and the sampling percentage (we use 10% in this experiment). However, as each shard can hold an arbitrary amount of data, *the total size of dataset being traded does not affect the contract gas cost.* In Table 1, we further report the breakdown of gas cost when each provider has 100 data shards. The gas cost of the optimal provider is the highest, which shall be compensated by the task reward.

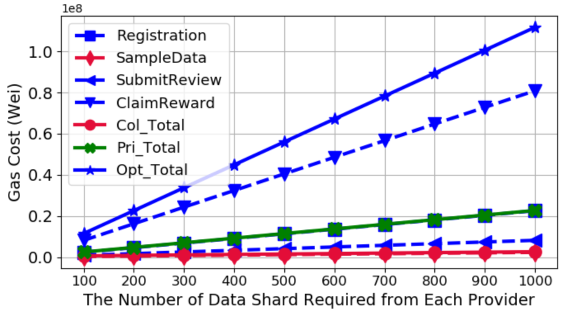


Figure 5: The gas cost of $\mathsf{Prot}_{\mathsf{SC}}$ with different numbers of data shards required from each provider. The legends $\mathsf{Pri}_{\mathsf{Total}}$ and $\mathsf{Opt}_{\mathsf{Total}}$ represent the total cost for non-optimal providers and the optimal provider, respectively.

Table 1: Breakdown of the trading contract gas cost when each provider's dataset is sharded into 100 shards on IPFS. The total size of dataset being traded does not affect the contract gas cost since each shard can host an arbitrary amount of data by hierarchically constructing a tree of data blocks under each shard.

Cost	**Collector**		**Provider**	
	Gas	ETH	Gas	ETH
CreateContract	212,269	0.0042	-	-
Registration	-	-	2,329,764	0.0466
SampleData	276,177	0.0055	-	-
SubmitFeatures	-	-	92,626	0.0019
SetCandidate	64,240	0.0013	-	-
SubmitReview	-	-	875,704*	0.0175*
SetOptimal	49,353	0.0010	-	-
ClaimReward	-	-	8,129,865*	0.1626*
ClaimDeposit	-	-	35,079	0.0007
Total	602,039	0.0120	2,457,469+ 11,463,038*	0.0491+ 0.2293*

+ The cost for regular providers.

* The cost paid by the optimal provider for a successfully closed deal.

Overall, we show that the trading platform in HyperX can be implemented using readily available infrastructural primitives. What are the optimal smart contracting platform and datastore service and how to minimize the service fees are out of scope.

4.2 Data-Quality Assessment Implementation

The privacy-preserving data quality assessment method introduced in § 3.1.1 is generically applicable for all deep learning tasks. Our implementation focuses on deep image classification as a proof-of-concept. We studied three image classification tasks targeting three datasets, *i.e.,* MNIST [5], Cifar10 [2] and MURA [48]. We use the

Keras deep-learning framework to obtain the pre-models (trained on the ImageNet [51]) for these learning tasks.

MNIST Learning Task. The MNIST dataset contains ~70, 000 images of handwritten digits from '0' to '9'. We select the VGG19 model in Keras as the pre-trained $\mathsf{M}_{\mathsf{init}}$ for this task. Data preprocessing is necessary since the VGG19 model only takes in RGB images whereas MNIST images are grayscale. We adopt the opencv library to convert grayscale to RGB, resize the images to 48 × 48, and further transform each pixel to 0~1. The default sampling percentage $\mathsf{CF}_{\%}$ is 0.83%, and the first block of the VGG19 model is selected as the $\mathsf{M}_{\mathsf{extractor}}$. Finally, 1000 randomly selected testing images are reserved as the provider's pre-owned validation dataset.

Cifar10 Learning Task. Cifar10 consists of 60,000 32×32 colorful images in ten classes. We use the Inception-V3 model in Keras as $\mathsf{M}_{\mathsf{init}}$ in this task. Since the Inception-V3 model requires (at least) a 299×299×3 input size, we enlarge the images to this size, and transform their pixel values to 0~1. The default $\mathsf{CF}_{\%}$ is 4%, and the first 156 layers (*conv2d*_30) is used as the $\mathsf{M}_{\mathsf{extractor}}$. All 10,000 testing images are used as the validation dataset for $\mathsf{M}_{\mathsf{eval}}$.

MURA Learning Task. MURA is a large dataset of bone X-rays, containing musculoskeletal radiographic studies including shoulder, humerus, elbow, forearm, and so on. We select elbow images as our target dataset, containing 2,922 normal images and 2,006 abnormal ones. We reserve 235 normal images and 230 abnormal images as the validation dataset for $\mathsf{M}_{\mathsf{eval}}$. We use the DenseNet169 model as $\mathsf{M}_{\mathsf{init}}$ and freeze the first 368 layers (*conv*4) to produce $\mathsf{M}_{\mathsf{extractor}}$. We preprocess the raw data by normalizing each image to the same mean and standard deviation of the ImageNet images used to train $\mathsf{M}_{\mathsf{init}}$, and then scale each image to 224 × 224.

The rationale for selecting proper sampling size and $\mathsf{M}_{\mathsf{extractor}}$ is further discussed in § 5.3.2. The structure and training process for our implemented models are shown in Table 2.

Table 2: Summary of our implemented models in this paper. T_W and S_Sub stands for Train-Whole and Scratch-Subset models, respectively. The architecture of $\mathsf{M}_{\mathsf{eval}}$ only shows the trainable layers (defined in Figure 2).

Learning Task		MNIST	Cifar10	MURA
$\mathsf{M}_{\mathsf{init}}$		VGG19	InceptionV3	DenseNet
$\mathsf{M}_{\mathsf{extractor}}$		1-st Block	conv2d_50	conv4
$\mathsf{M}_{\mathsf{eval}}$	Arch	Dense (128) Dropout(0.5) Dense (10)	Dense (512) Dense (10)	Dense (512) Dense (2)
	Opt	Adam (1e-5)	Adam (1e-5)	Adam (1e-5)
	Batch	8	32	8
	Epoch	50	50	100
T_W/ S_Sub Models	Arch	Conv (32) Conv (32) MaxPooling Dropout (0.25) Dense (128) Dropout (0.5) Dense (10)	Conv (32) Conv (32) MaxPooling Dropout (0.25) Conv (64) Conv (64) MaxPooling Dropout (0.25) Dense (512) Dropout (0.5) Dense (10)	DenseNet
	Opt	Adam (1e-4)	Adam (1e-4)	Adam (1e-5)
	Batch	64	32	8
	Epoch	50	50	100

5 EVALUATION

Since the trading procedural correctness on HyperX is assured by our security protocol $\mathsf{Prot}_{\mathsf{HyperX}}$, we mainly focus on demonstrating the effectiveness of our data quality assessment design in this section. We experiment hundreds of datasets in thousands of trading sessions to show that HyperX achieves near-optimal accuracy of selecting the optimal provider with the highest-quality dataset, *i.e.,* the probability that HyperX identities the best provider is almost the same as a hypothetical situation where the collector were able to access the complete datasets from all providers a priori. Meanwhile HyperX outperforms the strawman design (*i.e.,* direct training on sampled datasets) by non-trivial margins (§ 5.1). We also quantitatively study the confidentiality property of HyperX (§ 5.2), as well as several key design choices (§ 5.3).

5.1 Accuracy of Selecting the Optimal Provider

5.1.1 *Methodology*

We compare HyperX with the following two baseline approaches.

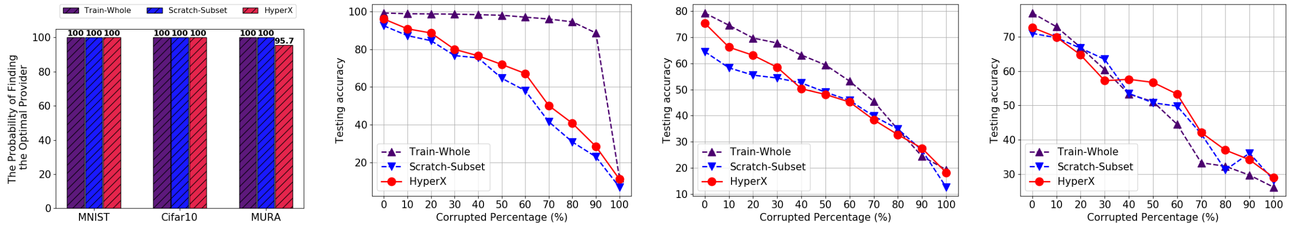
- **Training on the complete dataset.** This is a hypothetical approach assuming the collector could access the complete datasets from providers to perform quality assessment. As a result, the collector can directly train an evaluation model using the provider's full dataset and assess its performance using the validation dataset.

  For each task, we build a training-on-the-whole model using the model structures shown in Table 2. Hereafter, we refer to this method as Train-Whole.
- **Scratch on the data samples.** As introduced in § 3.1.1, a strawman design to assess data quality is training evaluation models directly on data samples, *i.e.,* Scratch-Subset. We build Scratch-Subset models using the same model structures as Train-Whole models, except that the Scratch-Subset models are trained with data samples.

Heterogeneous Providers. To emulate realistic trading sessions, we create a set of providers with heterogeneous datasets. To this end, for each learning task, we apply three methods to mutate the original dataset into various synthetic datasets with different qualities. Our methods include mislabeling the original dataset, degrading image pixels (*e.g.,* by adding noises or removing pixels), and corrupting the purity of original dataset by doping it with irrelevant images. For each setting, we randomly assign the synthetic datasets to a group of providers and evaluate whether HyperX, Train-Whole, and Scratch-Subset can accurately identify the optimal provider within each provider group.

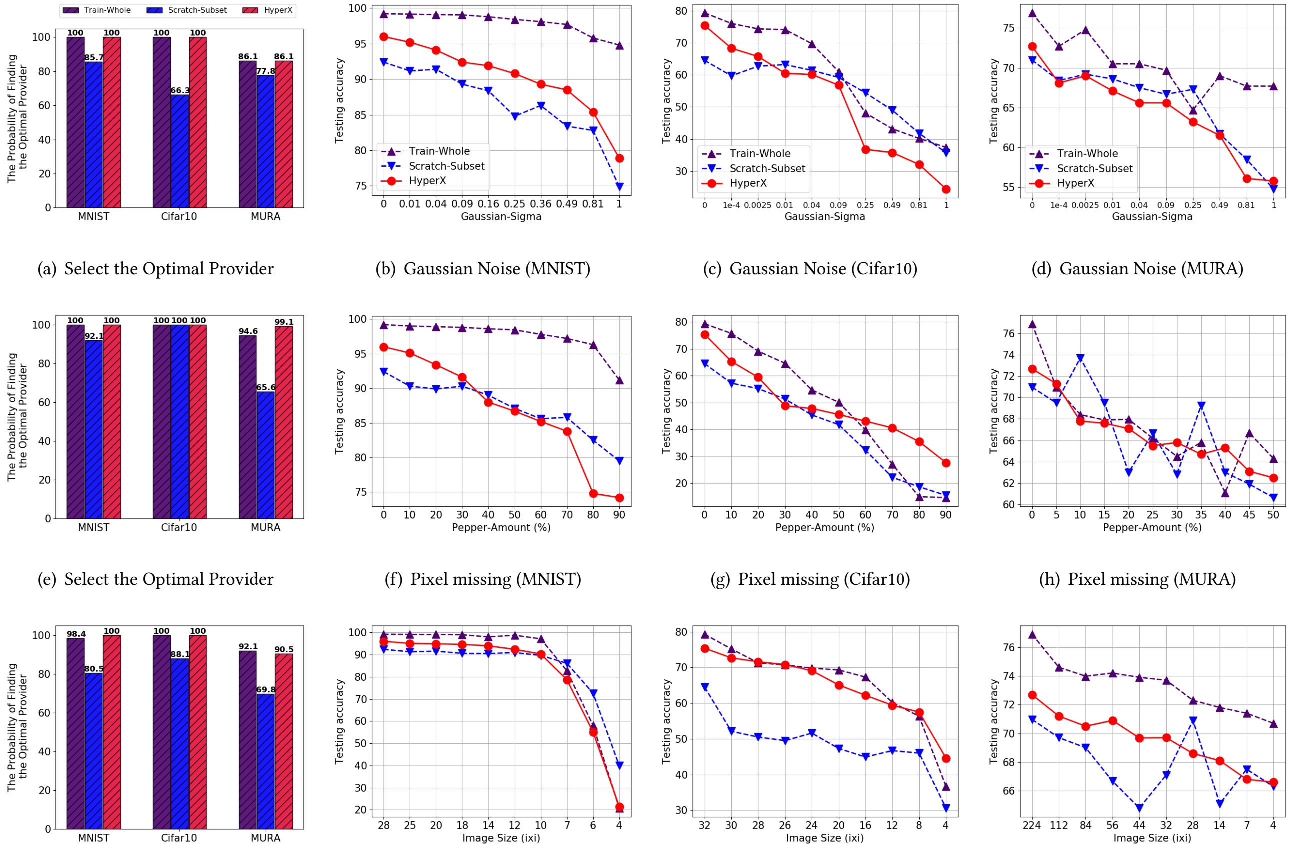
5.1.2 *Label Quality Variance*

Training on datasets with noisy labels (*i.e.,* label flips and outliers) could introduce unsatisfactory performance [24, 63]. In each learning task, to create label quality variance, we select a percentage of the original dataset, ranging from 0 to 100%, to randomly shuffle their labels. In total, we have eleven synthetic datasets (including the original one). Then, we randomly select five synthetic datasets



(a) Select the Optimal Provider (b) Label corruption (MNIST) (c) Label corruption (Cifar10) (d) Label corruption (MURA)

Figure 6: Evaluation of mislabeling-introduced dataset quality variance. We created 11 synthetic datasets and 462 groups of heterogeneous providers. Part (a) is the statistic probability of selecting an optimal provider for all these groups. Part (b) (c) and (d) show the testing accuracy of models trained on each of our synthetic datasets using three methods, representing each method's sensitivity about dataset quality.



(a) Select the Optimal Provider (b) Gaussian Noise (MNIST) (c) Gaussian Noise (Cifar10) (d) Gaussian Noise (MURA)

(e) Select the Optimal Provider (f) Pixel missing (MNIST) (g) Pixel missing (Cifar10) (h) Pixel missing (MURA)

(i) Select the Optimal Provider (j) Image decimation (MNIST) (k) Image decimation (Cifar10) (l) Image decimation (MURA)

Figure 7: Evaluation of image-mutation introduced dataset quality variance. In all cases, HyperX achieves almost the same accuracy rate for selecting the optimal provider as the hypothetical Train-Whole method, and outperforms Scratch-Subset by non-trivial margins (up to over 50% improvement in some cases).

to represent one group of providers so that we have 462 groups of providers with heterogeneous datasets.

The evaluation result on label quality variance is shown in Figure 6. Figure 6(a) shows the statistic probability of selecting the optimal providers among all these 462 provider groups. Figures 6(b), 6(c) and 6(d) plot the testing accuracy of the Train-Whole model, Scratch-Subset model and M_{eval} (our method) trained on these synthetic datasets. This set of accuracy results reflect each method's *sensitivity* about the dataset quality, and therefore it is describable that the testing accuracy monotonically deceases as the corruption percentage increases. Overall, all three methods demonstrate satisfactory performance in this setting. Scratch-Subset and our method have minor sensitivity degradation in the MURA task since their accuracy curves (Figure 6(d)) are not strictly decreasing. Due to the randomness when creating synthetic datasets, the accuracy-inversion causes slightly decreased statistic performance for our method (not in Scratch-Subset) (Figure 6(a)).

5.1.3 *Image Quality Variance*

Image pixel-domain degradation includes noise, corruption or decimation, leading to low signal-to-noise ratio. Noisy data could adversely affect the results of learning tasks [56]. We apply three techniques to create synthetic datasets with varying image qualities. (i) *Adding Noise:* we add Gaussian noise to the original images using the random_noise('Gaussian', σ) method in the skimage library, where σ is the standard deviation. (ii) *Taking Out Pixels:* we randomly set the value of a certain percentage of pixels in the image to zero. (iii) *Decimating Images:* we blur the original images by first downscaling an image to a smaller size, and then upscaling it to its original size. For each image quality degradation method, we create ten synthetic datasets (including the original one), and randomly select five of them to represent a group of providers so that we have 252 groups of providers with heterogeneous datasets.

The evaluation results are shown in Figure 7. For all cases, HyperX achieves almost the same accuracy rate for selecting the optimal provider as the hypothetical Train-Whole method, and outperforms Scratch-Subset by non-trivial margins (up to over 50% improvement in some cases). Meanwhile, HyperX shows greater advantages over Scratch-Subset for relatively complex image datasets, such as Cifar10 and MURA. Finally, in all cases, HyperX is highly sensitive to the dataset quality, outperforming even the Train-Whole method in some cases.

5.1.4 *Matching Quality Variance*

Finally, we consider *matching quality*, *i.e.,* high-quality datasets should be in accord with the collector's learning task. This is to ensure that the collector does not end up with recruiting irrelevant images for its task. To synthesize datasets with different matching qualities, we dope the original datasets with irrelevant images, *i.e.,* doping MNIST dataset with the Fashion-MNIST dataset [3], the Cifar10 dataset with the Category Flower Dataset [1], and the MURA dataset with the Cifar10 images.

Similar to the previous settings, we control the percentage of added irrelevant images to create ten synthetic datasets and 252 groups of heterogeneous providers. The evaluation results are shown in Figure 8. Overall, HyperX achieves similar performance as Train-Whole, and significantly outperforms Scratch-Subset for both the Cifar10 and MURA learning tasks. HyperX also demonstrates strong sensitivity about dataset qualities.

5.2 Empirically Evaluating Privacy Properties

In this section, we empirically evaluate the confidentiality property of our data quality assessment method. Specifically, we show that (i) nearly zero image-specific information can be recovered from the intermediate features submitted by the providers, and (ii) the $M_{\text{extractor}}$ published by the collector is effectively protected via coefficient obfuscation.

Table 3: Settings to empirically evaluate the irreversibility of the intermediate features submitted by the providers. The AE-recovered images lose nearly all semantic meaning when processed by well-trained classifiers.

Task	Target Data	Auxiliary Data to Train AE	Semantics of Recovered Data
MNIST	Digit 5-9 images	Fashion-MNIST	6.02% (10-class task)
		Digit 0-4 images	10.16% (10-class task)
MURA	MURA-Elbow	Cifar-10	50.11% (2-class task)
		MURA-Forearm	53.55% (2-class task)

Table 4: Effectiveness of obfuscating a deep neural model by linearly transforming the parameters on one layer.

Model	δ=1	0.005	0.05	5	50
FC	97.94%	**10.12%**	96.79%	97.90%	97.90%
CNN	71.38%	**10.00%**	**13.73%**	67.87%	66.12%
DenseNet	72.69%	**49.46%**	**49.46%**	52.69%	50.53%

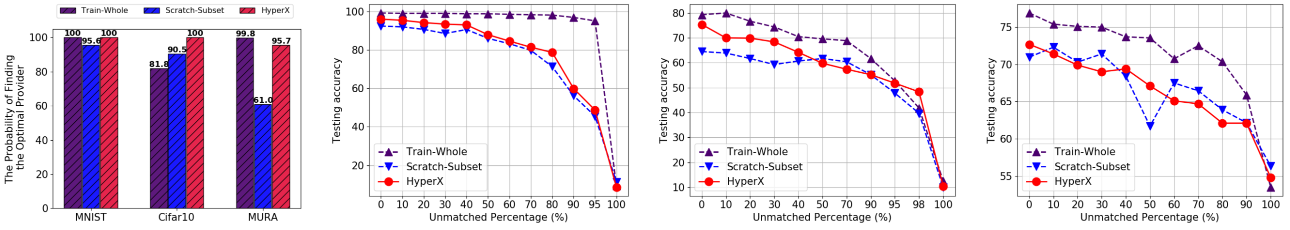
5.2.1 *Irreversibility of the Intermediate Data Features*

We explore what information can be recovered from the providers' intermediate features. In general, a deep neural network is considered to be irreversible due to its nonlinearity and information shrinking. Several recent works have attempted to recover the original inputs of deep neural networks [43, 52, 57, 62]. We study one of the proposals [57] designed based on AutoEncoders. Specifically, a malicious collector trains an AutoEncoder (AE) as the M_{init}, uses the Encoder as $M_{\text{extractor}}$ to extract intermediate features, and then feeds them into the Decoder to recover original images.

We experimentally evaluate the effectiveness of such attacks in HyperX. Table 3 summarizes the setup. For each task, we select two types of auxiliary datasets to train the AE, one with a similar distribution with the target data and one with a different distribution. To demonstrate that attackers cannot reliably recover the original datasets, we show the recovered images have very different semantic meaning as the original ones. To this end, we use well-trained classifiers to process all recovered MNIST and MURA-elbow images. Among all cases, the prediction accuracy rates on these recovered images are almost the same as random guesses, showing that the recovered images have lost nearly all semantic meanings.

5.2.2 *Feature Extractor Privacy*

In § 3.1.1, we propose to obfuscate $M_{\text{extractor}}$ by linearly transforming the model parameters on one layer. In this section, we demonstrate that this obfuscation technique is effective to drastically change the performance of a model. To this end, we implement parameter mutations on three models. One is a fully-connected (FC) model with four layers trained on MNIST; the second one is a CNN model with four convolutional layers and two fully-connected layers trained on Cifar10; and the last one is the DenseNet trained on



(a) Select the Optimal Provider (b) Matching quality (MNIST) (c) Matching quality (Cifar10) (d) Matching quality (MURA)

Figure 8: Evaluation of matching quality variance. HyperX demonstrates similar performance as Train-Whole, and outperforms Scratch-Subset significantly for both the Cifar10 and MURA learning tasks.

Table 5: Evaluation for using different $M_{extractor}$ in all learning tasks. $M_{extractor}$ has minor impact on the validation accuracy of M_{eval}.

VGG19 (M_{init})		Inception-V3 (M_{init})		DenseNet169 (M_{init})	
MNIST Task		Cifar10 Task		MURA Task	
$M_{extractor}$	M_{eval} Acc	$M_{extractor}$	M_{eval} Acc	$M_{extractor}$	M_{eval} Acc
1_block	**96.0%**	conv2d_12	64.5%	conv2_52	68.3%
2_block	92.9%	conv2d_30	72.4%	conv3_140	68.1%
3_block	94.1%	conv2d_50	**75.4%**	conv4_203	71.6%
4_block	92.7%	conv2d_60	73.5%	conv4_368	**72.7%**
5_block	73.4%	conv2d_94	68.1%	conv5_594	62.4%

Table 6: The performance of M_{eval} with different sampling percentages. The underlined percentages are default values used in each task.

MNIST Task		Cifar10 Task		MURA Task	
Sample %	M_{eval} Acc	Sample %	M_{eval} Acc	Sample %	M_{eval} Acc
0.17%	84.5%	2.0%	70.6%	10.0%	68.1%
0.33%	88.3%	4.0%	75.4%	20.0%	72.7%
0.50%	92.0%	6.0%	77.8%	30.0%	73.1%
0.67%	94.6%	8.0%	79.5%	40.0%	73.3%
0.83%	96.0%	10.0%	81.5%	50.0%	74.5%
100.0%	99.8%	100.0%	94.08%	100.0%	84.3%

MURA. We obfuscate the second layer of the FC and CNN models and the conv4_conv layer of the DenseNet model, by multiplying their parameters with δ. The results are shown in Table 4. When we vary δ from 0.005 to 50, the performance of the mutated models change irregularly. However, for certain selections of δ, the performance of the mutated models degrade drastically. Thus, by properly choosing the transformation coefficients, the collector is able to effectively protect $M_{extractor}$ from external parties.

5.3 Key Design Choices

Our data quality assessment design has two critical moving pieces: (i) how to decide the feature extractor $M_{extractor}$ from the pre-model M_{init}; and (ii) what is the proper sampling percentage. We report experiment results in this regard below.

5.3.1 *Proper Selection of $M_{extractor}$*

For each of the pre-models M_{init} we used in our evaluation, we select different layers to truncate M_{init} to produce five different $M_{extractor}$. Table 5 shows testing accuracy of the M_{eval} (trained on the features extracted from the original, non-mutated dataset) on the validation set when using different $M_{extractor}$ to extract intermediate features. From the results, we did not observe very strong relation between $M_{extractor}$ and the testing accuracy of M_{eval} (using VGG19 5_block as $M_{extractor}$ is an outlier). Thus, the collector has a high flexibility on deciding the $M_{extractor}$. Extended discussion about the impact of fixed layers on transfer learning can be found in [59]. In our experiments, for each learning task, we select the $M_{extractor}$ resulting in the highest testing accuracy (as highlighted in Table 5).

5.3.2 *Impact of the Sampling Percentage*

For a learning task, the collector publishes a sampling percentage as part of $Prot_{SC}$ configuration to decide how many data samples are expected to be used for feature extraction. A smaller percentage is helpful to accelerate the assessment process, and bounds the scope of data involved in quality evaluation. A larger percentage, however, leads to a higher chance that the sampled data faithfully represent the complete dataset. Thus, it is crucial to strike a balance between the two benefits.

In this section, we report our evaluation results on different sampling percentages for all three learning tasks. No dataset mutation is applied. From the results in Table 6, it is clear that the accuracy of M_{eval} does increase as the sample percentage increases. However, our data assessment design does not require the performance of M_{eval} for a provider to be near-optimal in order to select the provider as the optimal one. Instead, only *the relative M_{eval} accuracy rank* among all providers makes a difference. Thus, as long as our dataset assessment method is sensitive about data quality variance (which has been demonstrated in § 5.1), a small sampling percentage is sufficient to differentiate different providers. However, it is clear that the sampling percentage should be task-specific (for instance a relatively complex task like MURA requires a larger sampling percentage), and we leave deeper investigation on how it should be decided to future work.

6 THEOREMS

In this section, we provide theorem analysis for both the security protocol $Prot_{HyperX}$ and the privacy-preserving data quality assessment design.

6.1 Security Theorem of $\text{Prot}_{\text{HyperX}}$

In this section, we present the main theorem for our security protocol $\text{Prot}_{\text{HyperX}}$ described in § 3, and prove the theorem using the UC-framework [9].

6.1.1 *Ideal Functionality $\mathcal{F}_{HyperX}$*

We first present the cryptography abstraction of the $\text{Prot}_{\text{HyperX}}$ protocol in form of an ideal functionality $\mathcal{F}_{\text{HyperX}}$. The ideal functionality articulates the correctness and security properties that HyperX wishes to attain by assuming a trusted entity (as a result, $\mathcal{F}_{\text{HyperX}}$ is often drastically simplified compared with the actual real-world protocol $\text{Prot}_{\text{HyperX}}$). Then we prove that $\text{Prot}_{\text{HyperX}}$ UC-realizes $\mathcal{F}_{\text{HyperX}}$, implicating that $\text{Prot}_{\text{HyperX}}$, without assuming any trusted authorities, achieves the same security properties as $\mathcal{F}_{\text{HyperX}}$. The description of $\mathcal{F}_{\text{HyperX}}$ is given in Figure 9. We provide additional explanations below.

Session Creation. Through this interface, a data collector $\mathcal{P}_{\text{a}}$ requests $\mathcal{F}_{\text{HyperX}}$ to securely realize a data recruiting task $\mathcal{F}_{\text{Task}}$. The provided task $\mathcal{F}_{\text{Task}}$ must specify the essentials for starting a trading session, including the data quality assessment method (*e.g.,* the feature extractor $\text{M}_{\text{extractor}}$ or how it can be trained) and logistics (*e.g.,* similar to the $\text{SC}_{\text{config}}$ discussed in § 3.1.1). As a trusted entity, $\mathcal{F}_{\text{HyperX}}$ generates a utility key for $\mathcal{P}_{\text{a}}$, allowing $\mathcal{F}_{\text{HyperX}}$ to sign blockchain transactions and datastore service requests on behalf of $\mathcal{P}_{\text{a}}$. Next, $\mathcal{F}_{\text{HyperX}}$ deploys the *contract* on $\mathcal{F}_{\text{BC}}$. This is to emulate the real-world *side-effects*, which is necessary to prove $\text{Prot}_{\text{HyperX}}$ UC-realizes $\mathcal{F}_{\text{HyperX}}$. We provide further explanations below.

Provider Participation. A provider $\mathcal{P}_{\text{z}}$ joins the trading session via the ProviderParticipate interface. Due to the assumed trustiness, $\mathcal{P}_{\text{z}}$ can safely hands over its complete dataset $\mathcal{D}_{\text{whole}}$ to $\mathcal{F}_{\text{HyperX}}$. $\mathcal{F}_{\text{HyperX}}$ sends a storage request to $\mathcal{F}_{\text{DS}}$ on behalf of $\mathcal{P}_{\text{z}}$ to store $\mathcal{D}_{\text{whole}}$. Afterwards, $\mathcal{F}_{\text{HyperX}}$ evaluates the quality of $\mathcal{D}_{\text{whole}}$ using the method defined in $\mathcal{F}_{\text{Task}}$ (for instance the method discussed in § 3.1.1). Although $\mathcal{F}_{\text{HyperX}}$ has the full dataset from $\mathcal{P}_{\text{z}}$, its does not rely on this advantage for quality assessment. This is to ensure the parity of evaluation accuracies between $\mathcal{F}_{\text{HyperX}}$ and the real-world protocol $\text{Prot}_{\text{HyperX}}$.

Deal Closure. The collector $\mathcal{P}_{\text{a}}$ calls the DealClose interface to initiate the deal closure process. $\mathcal{F}_{\text{HyperX}}$ proceeds only if the best provider P_{opt}'s dataset is sufficiently good (as defined in $\mathcal{F}_{\text{Task}}$). We explicitly construct $\mathcal{F}_{\text{HyperX}}$ to handle Byzantine parties differently, which is again to achieve the parity between $\mathcal{F}_{\text{HyperX}}$ and $\text{Prot}_{\text{HyperX}}$ when handling the post-review abortion attack discussed in § 3.3. For non-Byzantine parties, the deal is closed atomically.

Verbose Definition of $\mathcal{F}_{\text{HyperX}}$. We *intentionally define $\mathcal{F}_{HyperX}$ verbosely.* For instance, in the SessionCreate interface, $\mathcal{F}_{\text{HyperX}}$ signs a blockchain transaction on behalf of $\mathcal{P}_{\text{a}}$ to simulate the result of deploying a smart contract on $\mathcal{F}_{\text{SC}}$ in the real world. Other examples include the request sent to $\mathcal{F}_{\text{DS}}$ in the ProviderParticipate interface, additional messages sent to the participants to inform actions take by $\mathcal{F}_{\text{HyperX}}$, and the dedicated handling for Byzantine parties. These steps are not essential to ensure the trading correctness due to the assumed trustiness of $\mathcal{F}_{\text{HyperX}}$. However, they are crucial for $\mathcal{F}_{\text{HyperX}}$ to accurately emulate the external side effects of $\text{Prot}_{\text{HyperX}}$ in the real world. As we shall see in § 6.1.5, the emulation is necessary to prove that $\text{Prot}_{\text{HyperX}}$ UC-realizes $\mathcal{F}_{\text{HyperX}}$.

6.1.2 *Security and Correctness Properties of $\mathcal{F}_{HyperX}$*

With the assumed trustiness and simplified construction, it is not difficult to conclude that $\mathcal{F}_{\text{HyperX}}$ offers the following correctness and security properties. First, the data quality assessment process is conducted confidentially so that each provider's dataset is kept secret from $\mathcal{P}_{\text{a}}$, and meanwhile models used for data quality assessment are opaque to providers. Second, the trading deal is closed atomically (*i.e.,* the collector $\mathcal{P}_{\text{a}}$ and the selected optimal P_{opt} exchange their goods simultaneously) if both parties are honest. Otherwise if they are Byzantine-failed, $\mathcal{F}_{\text{HyperX}}$ implements the post-review abortion or penalty policies according to the configuration of $\mathcal{F}_{\text{Task}}$. Detailed discussion on such policies are provided in § 3.3. We list the proportional compensation policy as an example in Figure 9. Finally, the accuracy for $\mathcal{F}_{\text{HyperX}}$ to select the best provider is fully driven by the dataset quality assessment method given by the $\mathcal{P}_{\text{a}}$, *i.e.,* $\mathcal{F}_{\text{HyperX}}$ focuses on ensuring the correctness of the underlying trading infrastructure.

6.1.3 *Security Theorem*

We now present our main security theorem.

Theorem 1. *Assuming that the distributed algorithms used by the underlying blockchains and datastore services are provably secure, the hash function is pre-image resistant, and the digital signature is EU-CMA secure* (i.e., *existentially unforgeable under a chosen message attack), our decentralized real-world protocol Prot_{HyperX} securely UC-realizes the ideal functionality $\mathcal{F}_{HyperX}$ against a malicious adversary in the Byzantine corruption model.*

Theorem 1 also holds for strictly weaker corruption models, such as the passive corruption model where the adversary is able to observe the complete internal state of a corrupted party (participant) whereas the corrupted party is still protocol compliant.

6.1.4 *Proof Overview*

We now present the detail proof of our main security Theorem 1. In the UC framework [9], the model of $\text{Prot}_{\text{HyperX}}$ execution is abstracted as a system of machines $(\mathcal{E}, \mathcal{A}, \pi_1, ..., \pi_n)$ where $\mathcal{E}$ is called the *environment*, $\mathcal{A}$ is the (real-world) adversary, and $(\pi_1, ..., \pi_n)$ are participants (referred to as *parties*) of $\text{Prot}_{\text{HyperX}}$ where each party may execute different parts of $\text{Prot}_{\text{HyperX}}$. Intuitively, the environment $\mathcal{E}$ represents the *external* system that contains other protocols, including ones that provide inputs to, and obtain outputs from, $\text{Prot}_{\text{HyperX}}$. The adversary $\mathcal{A}$ represents adversarial activity against the protocol execution, such as controlling communication channels and sending *corruption* messages to parties. $\mathcal{E}$ and $\mathcal{A}$ can communicate freely.

To prove that $\text{Prot}_{\text{HyperX}}$ UC-realizes the ideal functionality $\mathcal{F}_{\text{HyperX}}$, we need to prove that $\text{Prot}_{\text{HyperX}}$ *UC-emulates* $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$, which is the *ideal protocol* (defined below) of the ideal functionality $\mathcal{F}_{\text{HyperX}}$. That is, for any adversary $\mathcal{A}$, there exists an adversary (often referred to as a *simulator*) $\mathcal{S}$ such that $\mathcal{E}$ cannot distinguish between the ideal world, featured by $(\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}, \mathcal{S})$, and the real world, featured by $(\text{Prot}_{\text{HyperX}}, \mathcal{A})$. Mathematically, on any input, the probability that $\mathcal{E}$ outputs $\overrightarrow{1}$ after interacting with $(\text{Prot}_{\text{HyperX}},$

```
Init: Data := $\emptyset$
Upon Receive SessionCreate($\mathcal{F}_{\text{Task}}$, $\mathcal{P}_{\text{a}}$):
  abort if $\mathcal{F}_{\text{Task}}$ lacks essentials to start a trading session
  generate the session ID sid $\leftarrow \{0,1\}^{\lambda}$ and a utility key for $\mathcal{P}_{\text{a}}$
  generate the trading contract based on $\mathcal{F}_{\text{Task}}.\text{SC}_{\text{config}}$
  compute a blockchain transaction $\mathcal{T}_{\text{BC}}$ on behalf of $\mathcal{P}_{\text{a}}$ to deploy contract on $\mathcal{F}_{\text{BC}}$
  halt until contract is intialized on $\mathcal{F}_{\text{BC}}$
  send $\mathcal{T}_{\text{BC}}$ to $\mathcal{P}_{\text{a}}$ to inform the action take by $\mathcal{F}_{\text{HyperX}}$
  set an expiration timeout timer for the session
  update Data[sid] = $\{\mathcal{F}_{\text{Task}}, \mathcal{P}_{\text{a}}, \mathcal{S}_{\text{pri}} := \emptyset\}$
Upon Receive ProviderParticipate(sid, $\mathcal{D}_{\text{whole}}$, fund, $\mathcal{P}_{\text{z}}$):
  $\{\mathcal{F}_{\text{Task}}, \_, \mathcal{S}_{\text{pri}}\}$ := Data[sid] and abort if not found
  abort if $\mathcal{P}_{\text{z}}$ is already in $\mathcal{S}_{\text{pri}}$ or fund is not sufficient
  generate a utility key for $\mathcal{P}_{\text{z}}$
  send a storage request $\mathcal{T}_{\text{BC}}(\mathcal{D}_{\text{whole}})$ on behalf of $\mathcal{P}_{\text{z}}$ to $\mathcal{F}_{\text{DS}}$
  obtain the data retrieval keys $\text{adr}^{\text{p}}_{\text{data}}$ for $\mathcal{D}_{\text{whole}}$
  send $\text{adr}^{\text{p}}_{\text{data}}$ and $\mathcal{T}_{\text{BC}}(\mathcal{D}_{\text{whole}})$ to $\mathcal{P}_{\text{z}}$ to inform action
  evaluate the quality score of $\mathcal{D}_{\text{whole}}$ based on the method in $\mathcal{F}_{\text{Task}}$
  update $\mathcal{S}_{\text{pri}}[\mathcal{P}_{\text{z}}] = \{\mathcal{D}_{\text{whole}}, \text{adr}^{\text{p}}_{\text{data}}, score\}$
Upon Receive DealClose(sid, $\mathcal{P}$):
  $\{\mathcal{F}_{\text{Task}}, \mathcal{P}_{\text{a}}, \mathcal{S}_{\text{pri}}\}$ := Data[sid] and abort if not found
  assert $\mathcal{P} = \mathcal{P}_{\text{a}}$
  find the provider $\text{P}_{\text{opt}}$ in $\mathcal{S}_{\text{pri}}$ with the highest score
  if $\mathcal{S}_{\text{pri}}[\text{P}_{\text{opt}}]$.score exceeds the quality threshold defined in $\mathcal{F}_{\text{Task}}$ :
    if $\mathcal{P}_{\text{a}}$ or $\text{P}_{\text{opt}}$ is Byzantine failed :
      # An example post-review abortion policy
      disclose randomly sampled data from $\mathcal{S}_{\text{pri}}[\text{P}_{\text{opt}}].\mathcal{D}_{\text{whole}}$ to $\mathcal{P}_{\text{a}}$
      send the compensation defined in $\mathcal{F}_{\text{Task}}.\text{SC}_{\text{config}}$ to $\text{P}_{\text{opt}}$
    else : # close the deal atomically
      disclose the full dataset $\mathcal{S}_{\text{pri}}[\text{P}_{\text{opt}}].\mathcal{D}_{\text{whole}}$ to $\mathcal{P}_{\text{a}}$
      send the reward defined in $\mathcal{F}_{\text{Task}}.\text{SC}_{\text{config}}$ to $\text{P}_{\text{opt}}$
  close the session by Data.Erase(sid)
Timeout Callback DepositClaim(sid):
  abort if sid is not in Data
  return all deposits for participating providers Data[sid].$\mathcal{S}_{\text{pri}}$
  close the session by Data.Erase(sid)
```

Figure 9: The ideal functionality $\mathcal{F}_{\text{HyperX}}$.

$\mathcal{A}$) in the real world differs by at most a negligible amount from the probability that $\mathcal{E}$ outputs $\vec{1}$ after interacting with $(\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}, \mathcal{S})$ in the ideal world.

The ideal protocol $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$ is a wrapper around $\mathcal{F}_{\text{HyperX}}$ by a set of dummy parties that have the same interfaces as those of the parties in $\text{Prot}_{\text{HyperX}}$. As a result, $\mathcal{E}$ is able to interact with $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$ in the ideal world the same way it interacts with $\text{Prot}_{\text{HyperX}}$ in the real world. These dummy parties simply pass received inputs from $\mathcal{E}$ to $\mathcal{F}_{\text{HyperX}}$ and relay outputs of $\mathcal{F}_{\text{HyperX}}$ to $\mathcal{E}$, without implementing any other logic. $\mathcal{F}_{\text{HyperX}}$ controls the keys of these dummy parties. For the sake of clear presentation, we abstract the real-world participants of $\text{Prot}_{\text{HyperX}}$ as three types of parties $\{\mathcal{P}_{\text{COL}}, \mathcal{P}_{\text{PRI}}, \mathcal{P}_{\text{SC}}\}$, representing the collector, provider and trading contract. In the ideal world, the corresponding dummy party for $\mathcal{P}_{\text{COL}}$ is denoted as $\mathcal{P}^{\mathcal{I}}_{\text{COL}}$. This annotation mechanism applies for other parties as well.

Based on [9], to prove that $\text{Prot}_{\text{HyperX}}$ UC-emulates $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$ for any adversaries, it is sufficient to construct a simulator $\mathcal{S}$ only for the *dummy adversary* $\mathcal{A}$ that simply relays messages between $\mathcal{E}$ and the real-world parties. The overall proof procedure is that the simulator $\mathcal{S}$ observes the *side effects* of $\text{Prot}_{\text{HyperX}}$ in the real world, such as transitions on the blockchain and requests to the datastore service, and then accurately emulates these effects in the ideal world, with the help from $\mathcal{F}_{\text{HyperX}}$. As a result, $\mathcal{E}$ cannot distinguish the ideal and real worlds.

6.1.5 Indistinguishability of Real and Ideal Worlds

To prove indistinguishability of the real and ideal worlds from the perspective of $\mathcal{E}$, we will go through a sequence of *hybrid arguments*, where each argument is a hybrid construction of $\mathcal{F}_{\text{HyperX}}$, a subset of dummy parties of $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$, and a subset of real-world parties of $\text{Prot}_{\text{HyperX}}$, except that the first argument that is $\text{Prot}_{\text{HyperX}}$ without any ideal parties and the last argument is $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$ without any real world parties. We prove that $\mathcal{E}$ cannot distinguish any two consecutive hybrid arguments. Then based on the transitivity of protocol emulation [9], we prove that the first argument (*i.e.,* $\text{Prot}_{\text{HyperX}}$) UC-emulates the last argument (*i.e.,* $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$).

During the proof process, we will also specify how the simulator $\mathcal{S}$ should be constructed by specifying what actions $\mathcal{S}$ should take upon observing instructions from $\mathcal{E}$. As a distinguisher, $\mathcal{E}$ sends the same instructions to the ideal world dummy parities and real world parties.

Real World. We start with the real world $\text{Prot}_{\text{HyperX}}$ with a dummy adversary that simply passes messages to and from $\mathcal{E}$ to these real-world parties.

Hybrid $\mathbf{A_1}$. Hybrid $\mathbf{A_1}$ is the same as the real world, except that the $\mathcal{P}_{\text{COL}}$ is replaced by the dummy $\mathcal{P}^{\mathcal{I}}_{\text{COL}}$. Upon $\mathcal{E}$ gives an instruction to $\mathcal{P}^{\mathcal{I}}_{\text{COL}}$ to start a data recruiting task $\mathcal{F}_{\text{Task}}$, $\mathcal{S}$ extracts $\mathcal{F}_{\text{Task}}$ from the instruction and constructs a SessionCreate call to $\mathcal{F}_{\text{HyperX}}$ with parameter $(\mathcal{F}_{\text{Task}}, \mathcal{P}^{\mathcal{I}}_{\text{COL}})$. $\mathcal{F}_{\text{HyperX}}$ will then output a blockchain transaction to $\mathcal{P}^{\mathcal{I}}_{\text{COL}}$ to emulate the actions taken by $\mathcal{P}_{\text{COL}}$ in the real world. $\mathcal{S}$ then dispatch the blockchain transaction on $\mathcal{F}_{\text{BC}}$ to deploy the trading contract $\mathcal{P}_{\text{SC}}$ in the Hybrid $\mathbf{A_1}$.

Upon observing an instruction from $\mathcal{E}$ to execute a branch defined in the Watching service of $\mathcal{P}_{\text{COL}}$ (or $\mathcal{P}^{\mathcal{I}}_{\text{COL}}$) (see definitions in Figure 3), $\mathcal{S}$ is able to collect any necessary information from $\mathcal{P}_{\text{SC}}$ in the Hybrid $\mathbf{A_1}$ to handle the instruction. For instance, if $\mathcal{E}$ instructs $\mathcal{P}^{\mathcal{I}}_{\text{COL}}$ to sample addresses for a provider pid, $\mathcal{E}$ retrieves the $\text{adr}^{\text{h}}_{\text{data}}$ for pid from $\mathcal{P}_{\text{SC}}$ (abort if not found) and randomly samples a set of addresses from $\text{adr}^{\text{h}}_{\text{data}}$. Then with the help of $\mathcal{F}_{\text{HyperX}}$, $\mathcal{S}$ creates a blockchain transaction on behalf of $\mathcal{P}^{\mathcal{I}}_{\text{COL}}$ to call the SampleData interface of $\mathcal{P}_{\text{SC}}$ with the sampled address. For other instructions, such as reviewing a provider's submitted features, $\mathcal{S}$ may also rely on $\mathcal{F}_{\text{HyperX}}$ to evaluate these features.

If $\mathcal{P}_{\text{COL}}$ is corrupted by any Byzantine corruption messages from $\mathcal{E}$, it may not follow the predefined protocol. However, this does not prevent $\mathcal{S}$ from emulating a corrupted $\mathcal{P}_{\text{COL}}$ in Hybrid A_1 since every protocol execution by $\mathcal{P}_{\text{COL}}$ in the real world is publicly visible on the trading contract $\mathcal{P}_{\text{SC}}$. Therefore, $\mathcal{S}$ can reproduce these executions in Hybrid A_1.

Finally, in the real world, the trading session is automatic in the sense that it can continuously proceed even without additional instructions from $\mathcal{E}$ after successful session setup. In the Hybrid $\mathbf{A_1}$, although $\mathcal{P}_{\text{COL}}$ has been replaced by the dummy party $\mathcal{P}^{\mathcal{I}}_{\text{COL}}$

without any internal logic, $\mathcal{S}$, with the public information from $\mathcal{P}_{\text{SC}}$ (in the Hybrid $\mathbf{A_1}$) and the help of $\mathcal{F}_{\text{HyperX}}$, is still able to drive the process so that from $\mathcal{E}$'s perspective, the trading session is executed automatically. Finally, since $\mathcal{P}_{\text{SC}}$ still lives in the Hybrid $\mathbf{A_1}$, $\mathcal{S}$ should not trigger the DealClose interface of $\mathcal{F}_{\text{HyperX}}$ to avoid double execution on the same contract terms.

Fact 1. *With the aforementioned construction of $\mathcal{S}$ and $\mathcal{F}_{HyperX}$, it is immediately clear that the outputs of the dummy $\mathcal{P}^{\mathcal{I}}_{COL}$ in the Hybrid A_1 are exactly the same as the outputs of the actual $\mathcal{P}_{COL}$ in the real world, and all side effects (*i.e., *blockchain transactions and datastore requests) in the real world are accurately emulated by $\mathcal{S}$ in the Hybrid A_1. Thus, $\mathcal{E}$ cannot distinguish with the real world and the Hybrid A_1.*

Hybrid $\mathbf{A_2}$. Hybrid A_2 is the same as the Hybrid A_1, expect that $\mathcal{P}_{\text{PRI}}$ is further replaced by the dummy $\mathcal{P}^{\mathcal{I}}_{\text{PRI}}$. When $\mathcal{E}$ instructs $\mathcal{P}^{\mathcal{I}}_{\text{PRI}}$ to participate a trading session with a dataset $\mathcal{D}_{\text{whole}}$, $\mathcal{S}$ extracts $\mathcal{D}_{\text{whole}}$ and constructs a call to the ProviderParticipate interface of $\mathcal{F}_{\text{HyperX}}$ with parameters $(\mathcal{P}^{\mathcal{I}}_{\text{PRI}}, \mathcal{D}_{\text{whole}})$. Then $\mathcal{S}$ creates a blockchain transaction on behalf of $\mathcal{P}^{\mathcal{I}}_{\text{PRI}}$ to register the provider on $\mathcal{P}_{\text{SC}}$ in Hybrid $\mathbf{A_2}$. Afterwards, for any instruction from $\mathcal{E}$ to execute a branch in Watching interface of $\mathcal{P}_{\text{PRI}}$, $\mathcal{S}$ has all required information from $\mathcal{P}_{\text{SC}}$ and $\mathcal{F}_{\text{HyperX}}$ to handle the instruction on behalf of $\mathcal{P}^{\mathcal{I}}_{\text{PRI}}$. Thus, Hybrid $\mathbf{A_2}$ is identically distributed as Hybrid $\mathbf{A_1}$ from the view of $\mathcal{E}$.

Hybrid $\mathbf{A_3}$, *i.e.,* the ideal world. Hybrid $\mathbf{A_3}$ is the same as the Hybrid $\mathbf{A_2}$, expect that $\mathcal{P}_{\text{SC}}$ (the last real-world party) is further replaced by the dummy $\mathcal{P}^{\mathcal{I}}_{\text{SC}}$. Thus, the Hybrid $\mathbf{A_3}$ is essentially $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$. In $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$, $\mathcal{S}$ is required to resume the same responsibility of $\mathcal{P}_{\text{SC}}$ in the Hybrid $\mathbf{A_2}$. Emulating a public smart contract is trivial. In particular, for any instruction from $\mathcal{E}$ to invoke *contract*, $\mathcal{S}$ locally executes *contract* with the same input and then publishes the updated *contract* to $\mathcal{P}^{\mathcal{I}}_{\text{SC}}$ via $\mathcal{F}_{\text{HyperX}}$. Therefore, $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$ is indistinguishable with the Hybrid $\mathbf{A_2}$ from $\mathcal{E}$'s perspective.

Then given the transitivity of protocol emulation, we show that $\text{Prot}_{\text{HyperX}}$ UC-emulates $\mathcal{I}_{\mathcal{F}_{\text{HyperX}}}$, and therefore prove that $\text{Prot}_{\text{HyperX}}$ UC-realizes $\mathcal{F}_{\text{HyperX}}$, which implies that $\text{Prot}_{\text{HyperX}}$ achieves the same security properties as $\mathcal{F}_{\text{HyperX}}$. Throughout the simulation, we maintain a key invariant: $\mathcal{S}$ and $\mathcal{F}_{\text{HyperX}}$ together can always accurately simulate the desired outputs and side effects on all (dummy and real) parties in all Hybrid worlds. Thus, from $\mathcal{E}$'s view, the indistinguishability between the real and ideal worlds naturally follows. This concludes our proof for Theorem 1.

6.2 Learning Theories for Data Quality Assessment

The confidentiality property of HyperX's data quality assessment design includes both dataset privacy for providers and model privacy for the collector (detailed in § 2.2). We justify both of them using learning theories.

Irreversibility of Immediate Data Features. Due to the "irreversibility" of deep neural networks, the collector can not reliably recover the image-domain input based on the intermediate data features. In a typical multi-layer neural network, the output vector of neurons in layer k is $y_k = f(W_k\, y_{k-1})$, where $f(\cdot)$ is a nonlinear activation function and W_k is the weight matrix for the k-th linear layer. Here the commonly-used $f(\cdot)$ include the tangent function $f(z) = (e^{2z} - 1)(e^{2z} + 1)^{-1}$, sigmoid $f(z) = (1 + e^{-z})^{-1}$ (ill-conditioned and not commonly used in mainstream deep learning models), Rectified Linear Unit (ReLU) $f(z) = \max(0, z)$, and softplus $f(z) = \log(1 + e^z)$. We take ReLU as the example of $f(\cdot)$, and prove the irreversibility in Lemma 6.1. The result can be generalized to other non-linear activation functions as well.

LEMMA 6.1. *For any input vector y that follows i.i.d. normal distribution, i.e., $y_i \sim \mathcal{N}(0, 1)$, i.i.d. , the output of any fully connected layer $z = f(Wy + b)$ with finite weights $\{W, b\}$ will always have information loss using ReLU activation $f(\cdot)$.*

PROOF. Denote the input vector of the ReLU as $\tilde{z} \triangleq Wy + b$. After linear transformation, $\tilde{z}$ follows the multivariate normal distribution as $\tilde{z} \sim \mathcal{N}(\mu, \Sigma)$, where $\mu_i = b_i$, $\forall i$ and $\Sigma = W\,W^T$. The ReLU activation $f(\cdot)$ is a non-linear operator for each element of $\tilde{z}$ independently, *i.e.,* $f(\tilde{z}_i) = \tilde{z}_i$ if $\tilde{z}_i \geq 0$; $f(\tilde{z}_i) = 0$ if $\tilde{z}_i < 0$. Thus, the elements $\tilde{z}_i < 0$ will be reset to zero after ReLU, and the expected information loss will be $\Phi(\tilde{z}_i < 0) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-\frac{b_i}{\Sigma_{i,i}}} \exp\left\{-\frac{u^2}{2}\right\} du > 0$ $\forall i$ with finite W and b. □

Lemma 6.1 guarantees that the extracted intermediate features have information loss. Therefore, the collector cannot reliably infer the original input data of the deep neural networks based on any intermediate feature vector $\mathcal{B}^i$ $(y_k, k \geq 2)$.

Feature Extractor Privacy. To extract features, the collector is required to publish $\mathsf{M}_{\text{extractor}}$, which is obtained by truncating the first K layers from the pre-model M_{init} and obfuscating the parameters of the last linear layer in $\mathsf{M}_{\text{extractor}}$. In general, since K is very small compared to the depth of M_{init}, little information of M_{init} is actually exposed by $\mathsf{M}_{\text{extractor}}$. Further, without knowing the oracle obfuscation factor δ (privately set by the collector), external parties can hardly infer the initial weights of M_{init}. However, directly using the obfuscated weights for certain task leads to degraded or unreliable performance. We quantify such degradation for an image reconstruction task using a simple 2-layer network. In Lemma 6.2, we show that the obfuscated feature will generate reconstruction error with its approximate upper bound monotonically increasing with δ, *i.e.,* $\delta = (\Delta z + z)/z$. For more complicated tasks with generic model architecture, we have provided experimental evaluation of the model privacy by reporting the classification accuracy degradation in § 5.2.2. Finally, as the published weights are only the initialization, the eventually trained model by the collector will be significantly different. Compared with prior proposals that directly evaluate well-trained models [10, 35], HyperX is much less privacy-invasive.

LEMMA 6.2. *With a linear transform layer $z = Wy$, to reconstruct y using feature z, the least square solution is $\hat{y} = W^{-1}z$. When the feature z is obfuscated as $z + \Delta z$, the learned reconstruction transform is perturbed accordingly as $W + \Delta W$. The reconstruction $y + \Delta y$ will have error that is approximately upper-bounded as*

$$\frac{\|\Delta y\|_2}{\|y\|_2} \lesssim \kappa(W) \left\{ \frac{\|\Delta z\|_2}{\|z\|_2} + \frac{\|\Delta W\|_2}{\|W\|_2} \right\} \qquad (5)$$

where $\kappa(W)$ denotes the condition number of the square matrix W.

Proof. Based on the exact linear relationship $z = Wy$, the perturbed equation becomes

$$z + \Delta z = (W + \Delta W)(\Delta y + y) \tag{6}$$

Assuming that both ΔW and Δy are sufficiently small, we ignore the second-order term, *i.e.,* $\Delta W \Delta y \approx 0$. The linear equation is approximated as $\Delta z = W\Delta y + \Delta W y$. We solve the inverse problem via least square, and the estimated reconstruction error is

$$\Delta y = W^{-1}(\Delta z - \Delta W y) \tag{7}$$

Taking the norm of both sides, and using the triangle inequality, we have the error following bound

$$\frac{\|\Delta y\|_2}{\|y\|_2} \lesssim \frac{1}{\sigma_n} \left\{ \frac{\|\Delta z\|_2}{\|y\|_2} + \|\Delta W\|_2 \right\} \tag{8}$$

where σ_k denotes the k-th sigular value of W, $1 \leq k \leq n$. On the right hand side of the inequality, the first term $\frac{\|\Delta z\|_2}{\|y\|_2} \leq \sigma_1 \frac{\|\Delta z\|_2}{\|z\|_2}$ since $\|y\|_2 \geq \frac{1}{\sigma_1} \|z\|_2$; The second term $\|\Delta W\|_2 = \sigma_1 \frac{\|\Delta W\|_2}{\|W\|_2}$. Therefore, with $\kappa(W) \triangleq \frac{\sigma_1}{\sigma_n}$, we obtain the claimed approximate error upper bound

$$\frac{\|\Delta y\|_2}{\|y\|_2} \lesssim \kappa(W) \left\{ \frac{\|\Delta z\|_2}{\|z\|_2} + \frac{\|\Delta W\|_2}{\|W\|_2} \right\} \tag{9}$$

□

7 RELATED WORK

Data Marketplace. The concept of data marketplace was introduced in [8] where Balazinska et al. foresaw great potential for cloud-based data marketplaces. Since then, the research community has proposed to recruit useful data for various applications and use cases through marketplaces. For instance, proposals (*e.g.,* [45–47, 64]), focusing on Internet of Things (IoT) datasets, rely on high-level data descriptions such as vendor or sensor types to filter datasets. Other proposals (*e.g.,* [21, 38]), rely on a trusted third party to handle consumers' query requests and data providers' answers throughout the trading session. Ye et al. [58] propose a mechanism to collect key-value data based on local differential privacy, without a well-established notion of data quality.

Prior designs [19, 29] have proposed to use multi-party computation (MPC) to enable computation over encrypted data. However, it is very challenging to apply MPC to implement nonlinear deep learning algorithms, even if certain transformation techniques [26, 39, 44] are applied on these deep models. Hynes et al. [23] proposed a ML data marketplace design that performs data-quality evaluation and model training inside a TEE. However, given the memory limitation and bounded CPU capability of a TEE, it faces scalability and efficiency challenges for deep learning tasks. Further, TEE itself is not a silver bullet with potential vulnerabilities (*e.g.,* [36, 37, 53, 54]).

A few other proposals studied direct model exchanges (*e.g.,* [10, 35]), which are orthogonal to HyperX.

Data Value or Quality Assessment. A wide range of prior works have studied data value or quality assessment. For instance, proposals [12, 18, 25, 28] evaluate data value based on Shapley Value using a game theoretic approach, which typically requires access to the full datasets. Some other literatures (*e.g.,* [8, 31, 32]) propose a pricing framework that allows the data providers to set explicit prices to based on ad-hoc relational queries, which is only suitable for exchanging relational or structured datasets.

Decentralized Protocol-Powered System. The enthusiasm for Web 3.0 has driven a growing number of literatures on powering real-world systems via decentralized (or trust-free) protocols, including various secure multiparty computation proposals (*e.g.,* [6, 33, 34, 42], verifiable cloud computing (*e.g.,* [14]), decentralized digital good exchanges (*e.g.,* [15]), blockchain interoperability (*e.g.,* [40, 60]), and various proposals to improve the Blockchain systems themselves (such as adding support for private smart contracting [11, 27, 30]). Overall, practicability and deployability are two the primary challenges for designing decentralized protocols to power real-world systems. Therefore, HyperX fully excludes *ideal infrastructural primitives*, but enabling our design via delicate security protocols and a novel data quality evaluation mechanism.

8 CONCLUSION

In this paper, we proposed HyperX, the first quality-driven, privacy-preserving and economically-fair data exchange platform for deep learning applications. At the highest level, HyperX is powered by two innovative designs: (i) a novel data quality assessment mechanism simultaneously offering data and model confidentiality and high evaluation accuracy, (ii) and a set of security protocols to realize real-world data tradings in a secure, transparent and trust-free manner. We implemented a prototype of HyperX using readily-deployable infrastructure primitives to demonstrates its practicality, and extensively evaluated the prototype with hundreds of datasets and thousands of data trading sessions. The results showed that HyperX achieved near-optimal accuracy of identifying the best dataset from heterogeneous providers comparing with a hypercritical design that allows the collector to access the complete datasets from all providers a priori), and outperformed the strawman design (directly using the raw data samples for quality assessment) by non-trivial margins.

REFERENCES

[1] Category Flower Dataset. http://www.robots.ox.ac.uk/~vgg/data/flowers/17/.

[2] Cifar10 Dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[3] Fashion-MNIST Dataset. https://github.com/zalandoresearch/fashion-mnist.

[4] InterPlanetary File System. https://ipfs.io.

[5] MNIST Dataset. http://yann.lecun.com/exdb/mnist/.

[6] Andrychowicz, M., Dziembowski, S., Malinowski, D., and Mazurek, L. Secure Multiparty Computations on Bitcoin. In *IEEE Symposium on Security and Privacy (SP)* (2014).

[7] Antony, J., McGuinness, K., O'Connor, N. E., and Moran, K. Quantifying radiographic knee osteoarthritis severity using deep convolutional neural networks. In *IEEE ICPR* (2016).

[8] Balazinska, M., Howe, B., and Suciu, D. Data markets in the cloud: An opportunity for the database community. *VLDB Endowment* (2011).

[9] Canetti, R. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science* (Accessed on 2020).

[10] Chen, L., Koutris, P., and Kumar, A. Towards Model-based Pricing for Machine Learning in a Data Marketplace. In *ACM SIGMOD* (2019).

[11] Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., and Song, D. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *IEEE European S&P* (2019).

[12] Chessa, M., and Loiseau, P. A cooperative game-theoretic approach to quantify the value of personal data in networks. In *Proceedings of the 12th workshop on the Economics of Networks, Systems and Computation* (2017), ACM.

[13] Ding, B., Kulkarni, J., and Yekhanin, S. Collecting Telemetry Data Privately. In *Advances in Neural Information Processing Systems* (2017).

[14] Dong, C., Wang, Y., Aldweesh, A., McCorry, P., and van Moorsel, A. Betrayal, Distrust, and Rationality: Smart Counter-Collusion Contracts for Verifiable Cloud Computing. In *ACM CCS* (2017).

[15] Dziembowski, S., Eckey, L., and Faust, S. Fairswap: How to fairly exchange digital goods. In *ACM CCS* (2018).

[16] Erlingsson, Ú., Pihur, V., and Korolova, A. Rappor: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *ACM CCS* (2014).

[17] Garay, J., Kiayias, A., and Leonardos, N. The Bitcoin Backbone Protocol with Chains of Variable Difficulty. In *Crypto* (2017).

[18] Ghorbani, A., and Zou, J. Data Shapley: Equitable Valuation of Data for Machine Learning. In *International Conference on Machine Learning (ICML)* (2019).

[19] Gilad-Bachrach, R., Laine, K., Lauter, K. E., Rindal, P., and Rosulek, M. Secure Data Exchange: A Marketplace in the Cloud. *IACR Cryptology ePrint Archive* (2016).

[20] Greenberg, A. Apple's 'differential privacy'is about collecting your data–but not your data. *Wired Magazine* (2016).

[21] Grubenmann, T., Bernstein, A., Moor, D., and Seuken, S. Fedmark: A marketplace for federated data on the web. *arXiv: Databases* (2018).

[22] Gupta, P., Kanhere, S. S., and Jurdak, R. A decentralized iot data marketplace. *arXiv: Networking and Internet Architecture* (2019).

[23] Hynes, N., Dao, D., Yan, D., Cheng, R., and Song, D. A demonstration of sterling: a privacy-preserving data marketplace. *VLDB Endowment* (2018).

[24] Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., and Li, B. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *IEEE Symposium on Security and Privacy (SP)* (2018).

[25] Jia, R., Dao, D., Wang, B., Hubis, F. A., Hynes, N., Gurel, N. M., Li, B., Zhang, C., Song, D., and Spanos, C. Towards Efficient Data Valuation Based on the Shapley Value. In *AISTATS* (2019).

[26] Juvekar, C., Vaikuntanathan, V., and Chandrakasan, A. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security Symposium* (2018).

[27] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S. M., and Felten, E. W. Arbitrum: Scalable, Private Smart Contracts. In *USENIX Security Symposium* (2018).

[28] Kleinberg, J., Papadimitriou, C. H., and Raghavan, P. On the value of private information. In *Proceedings of the 8th conference on Theoretical aspects of rationality and knowledge* (2001), Morgan Kaufmann Publishers Inc.

[29] Kong, Y., Ma, Y., and Wu, Y. Securely trading unverifiable information without trust. *arXiv: Computer Science and Game Theory* (2019).

[30] Kosba, A., Miller, A., Shi, E., Wen, Z., and Papamanthou, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE Symposium on Security and Privacy (SP)* (2016).

[31] Koutris, P., Upadhyaya, P., Balazinska, M., Howe, B., and Suciu, D. Query-market demonstration: Pricing for online data markets. *VLDB Endowment* (2012).

[32] Koutris, P., Upadhyaya, P., Balazinska, M., Howe, B., and Suciu, D. Toward practical query pricing with querymarket. In *ACM SIGMOD* (2013).

[33] Kumaresan, R., and Bentov, I. Amortizing Secure Computation with Penalties. In *ACM CCS* (2016).

[34] Kumaresan, R., Vaikuntanathan, V., and Vasudevan, P. N. Improvements to Secure Computation with Penalties. In *ACM CCS* (2016).

[35] Kurtulmus, A. B., and Daniel, K. Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain. *arXiv: Cryptography and Security* (2018).

[36] Lee, J., Jang, J., Jang, Y., Kwak, N., Choi, Y., Choi, C., Kim, T., Peinado, M., and Kang, B. B. Hacking in Darkness: Return-oriented Programming against Secure Enclaves. In *USENIX Security Symposium* (2017).

[37] Lee, S., Shih, M.-W., Gera, P., Kim, T., Kim, H., and Peinado, M. Inferring fine-grained control flow inside *sgx* enclaves with branch shadowing. In *USENIX Security Symposium* (2017).

[38] Li, Y., Sun, H., Dong, B., and Wang, H. W. Cost-efficient Data Acquisition on Online Data Marketplaces for Correlation Analysis. *VLDB Endowment* (2018).

[39] Liu, J., Juuti, M., Lu, Y., and Asokan, N. Oblivious neural network predictions via minionn transformations. In *ACM CCS* (2017).

[40] Liu, Z., Xiang, Y., Shi, J., Gao, P., Wang, H., Xiao, X., Wen, B., and Hu, Y.-C. Hyperservice: Interoperability and Programmability across Heterogeneous Blockchains. In *ACM CCS* (2019).

[41] Ma, J., Wu, F., Zhu, J., Xu, D., and Kong, D. A Pre-trained Convolutional Neural Network based Method for Thyroid Nodule Diagnosis. *Ultrasonics* (2017).

[42] Malkhi, D., Nisan, N., Pinkas, B., Sella, Y., et al. Fairplay-Secure Two-Party Computation System. In *USENIX Security Symposium* (2004).

[43] Melis, L., Song, C., De Cristofaro, E., and Shmatikov, V. Exploiting Unintended Feature Leakage in Collaborative Learning. In *IEEE Symposium on Security and Privacy (SP)* (2019).

[44] Mohassel, P., and Zhang, Y. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy (SP)* (2017).

[45] Oh, H., Park, S., Lee, G. M., Heo, H., and Choi, J. K. Personal data trading scheme for data brokers in iot data marketplaces. *IEEE Access* (2019).

[46] Özyilmaz, K. R., Doğan, M., and Yurdakul, A. IDMoB: IoT Data Marketplace on Blockchain. In *IEEE CVCBT* (2018).

[47] Park, J., Youn, T., Kim, H., Rhee, K., and Shin, S. Smart contract-based review system for an iot data marketplace. *Sensors* (2018).

[48] Rajpurkar, P., Irvin, J., Bagul, A., Ding, D., Duan, T., Mehta, H., Yang, B., Zhu, K., Laird, D., Ball, R. L., et al. Mura: Large dataset for abnormality detection in musculoskeletal radiographs. *arXiv preprint arXiv:1712.06957* (2017).

[49] Redmon, J., and Farhadi, A. YOLO9000: better, faster, stronger. In *IEEE CVPR* (2017).

[50] Ren, S., He, K., Girshick, R., and Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems* (2015).

[51] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* (2015).

[52] Salem, A., Bhattacharyya, A., Backes, M., Fritz, M., and Zhang, Y. Updates-leak: Data set inference and reconstruction attacks in online learning. In *USENIX Security Symposium* (2020).

[53] Seo, J., Lee, B., Kim, S. M., Shih, M.-W., Shin, I., Han, D., and Kim, T. Sgx-shield: Enabling address space layout randomization for sgx programs. In *NDSS* (2017).

[54] Shih, M.-W., Lee, S., Kim, T., and Peinado, M. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *NDSS* (2017).

[55] Tan, M., and Le, Q. V. EfficientNet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946* (2019).

[56] Wang, F., Chen, L., Li, C., Huang, S., Chen, Y., Qian, C., and Change Loy, C. The devil of face recognition is in the noise. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018).

[57] Yang, Z., Zhang, J., Chang, E., and Liang, Z. Neural Network Inversion in Adversarial Setting via Background Knowledge Alignment. In *ACM CCS* (2019).

[58] Ye, Q., Hu, H., Meng, X., and Zheng, H. Privkv: Key-value data collection with local differential privacy. In *IEEE Symposium on Security and Privacy (SP)* (2019).

[59] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems* (2014).

[60] Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., and Knottenbelt, W. Xclaim: Trustless, Interoperable, Cryptocurrency-Backed Assets. In *IEEE Symposium on Security and Privacy (SP)* (2019).

[61] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530* (2016).

[62] Zhang, Y., Jia, R., Pei, H., Wang, W., Li, B., and Song, D. The secret revealer: Generative model inversion attacks against deep neural networks. In *IEEE CVPR* (2020).

[63] Zhao, M., An, B., Gao, W., and Zhang, T. Efficient label contamination attacks against black-box learning models. In *IJCAI* (2017).

[64] Zheng, Z., Peng, Y., Wu, F., Tang, S., and Chen, G. An online pricing mechanism for mobile crowdsensing data markets. In *ACM MobiHoc* (2017).