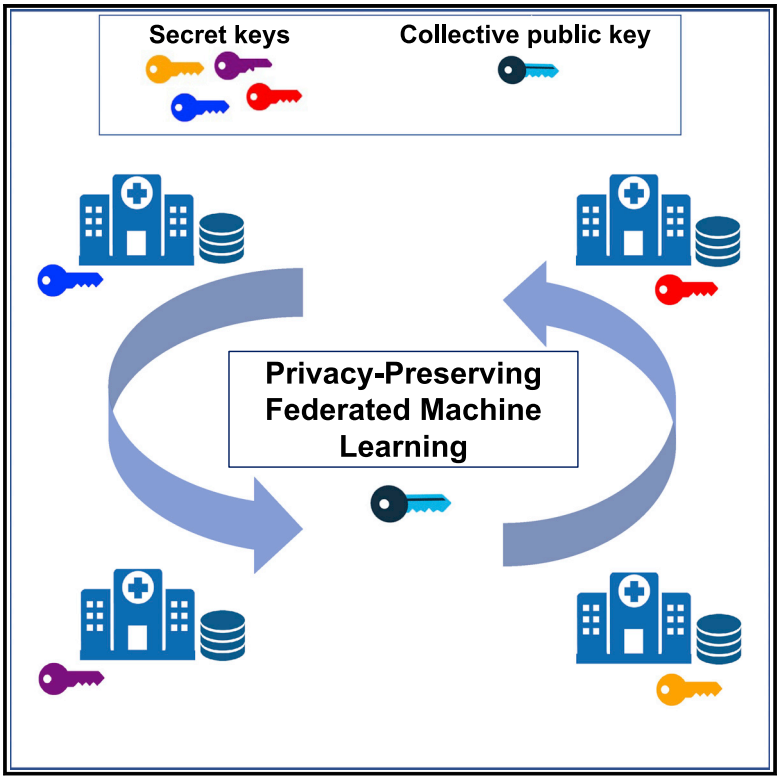
Article

Patterns

Privacy-preserving federated neural network learning for disease-associated cell classification

Graphical abstract



Highlights

- We enable collaborative and privacy-preserving model training between institutions
- Training under encryption does not degrade the utility of the data
- We apply our solution to the single-cell analysis in a federated setting
- Our method is generalizable to other machine learning tasks in the healthcare domain

Authors

Sinem Sav, Jean-Philippe Bossuat, Juan R. Troncoso-Pastoriza, Manfred Claassen, Jean-Pierre Hubaux

Correspondence

sinem.sav@epfl.ch (S.S.), manfred.claassen@med.uni-tuebingen.de (M.C.), jean-pierre.hubaux@epfl.ch (J.-P.H.)

In brief

To enable federated learning with several healthcare institutions in a privacy-preserving way, this work relies on multiparty homomorphic encryption. The paper describes several optimizations for enabling efficient and collaborative machine learning training under encryption. The authors demonstrate the utility and performance of the proposed solution with an application to disease-associated cell classification in a federated setting. The solution preserves the utility while protecting the privacy of the institutions' data, the model, and the exchanged values between the institutions.

Sav et al., 2022, Patterns *3*, 100487
May 13, 2022
https://doi.org/10.1016/j.patter.2022.100487

CellPress

Patterns

CellPress
OPEN ACCESS

Article

Privacy-preserving federated neural network learning for disease-associated cell classification

Sinem Sav,[1,5,*] Jean-Philippe Bossuat,[2] Juan R. Troncoso-Pastoriza,[2] Manfred Claassen,[3,4,*] and Jean-Pierre Hubaux[1,2,*]
[1]Laboratory for Data Security (LDS), EPFL, Lausanne 1015, Switzerland
[2]Tune Insight SA, Lausanne 1015, Switzerland
[3]Internal Medicine I, University Hospital Tübingen, Faculty of Medicine, University of Tübingen, Tübingen 72016, Germany
[4]Department of Computer Science, University of Tübingen, Tübingen 72076, Germany
[5]Lead contact
*Correspondence: sinem.sav@epfl.ch (S.S.), manfred.claassen@med.uni-tuebingen.de (M.C.), jean-pierre.hubaux@epfl.ch (J.-P.H.)
https://doi.org/10.1016/j.patter.2022.100487

THE BIGGER PICTURE High-quality medical machine learning models will benefit greatly from collaboration between health care institutions. Yet, it is usually difficult to transfer data between these institutions due to strict privacy regulations. In this study, we propose a solution, *PriCell*, that relies on multiparty homomorphic encryption to enable privacy-preserving collaborative machine learning while protecting via encryption the institutions' input data, the model, and any value exchanged between the institutions. We show the maturity of our solution by training a published state-of-the-art convolutional neural network in a decentralized and privacy-preserving manner. We compare the accuracy achieved by *PriCell* with the centralized and non-secure solutions and show that *PriCell* guarantees privacy without reducing the utility of the data. The benefits of *PriCell* constitute an important landmark for real-world applications of collaborative training while preserving privacy.

Development/Pre-production: Data science output has been rolled out/validated across multiple domains/problems

SUMMARY

Training accurate and robust machine learning models requires a large amount of data that is usually scattered across data silos. Sharing or centralizing the data of different healthcare institutions is, however, unfeasible or prohibitively difficult due to privacy regulations. In this work, we address this problem by using a privacy-preserving federated learning-based approach, *PriCell*, for complex models such as convolutional neural networks. *PriCell* relies on multiparty homomorphic encryption and enables the collaborative training of encrypted neural networks with multiple healthcare institutions. We preserve the confidentiality of each institutions' input data, of any intermediate values, and of the trained model parameters. We efficiently replicate the training of a published state-of-the-art convolutional neural network architecture in a decentralized and privacy-preserving manner. Our solution achieves an accuracy comparable with the one obtained with the centralized non-secure solution. *PriCell* guarantees patient privacy and ensures data utility for efficient multi-center studies involving complex healthcare data.

INTRODUCTION

Machine learning models, in particular neural networks, extract valuable insights from data and have achieved unprecedented predictive performance in the healthcare domain, e.g., in single-cell analysis,[5] aiding medical diagnosis and treatment,[6,7] or in personalized medicine.[8] Training accurate and unbiased models without overfitting requires access to a large amount of diverse data that is usually isolated and scattered across different healthcare institutions.[9] Sharing or transferring personal healthcare data is, however, often unfeasible or limited due to privacy regulations, such as General Data Protection Regulation (GDPR)or Health Insurance Portability and Accountability Act (HIPAA). Consequently, privacy-preserving collaborative learning solutions play a vital role for researchers, as they enable medical advances without the information about each institution's data being shared or leaked. Collaborative learning solutions play a particularly important role for studies that involve

novel informative, yet not universally established, data modalities, such as high-dimensional single-cell measurements, where the number of examples is typically low at individual study centers and only amounts to critical mass for the successful training of machine learning models across multiple study centers.[10] The ability to satisfy privacy regulations in an efficient and effective manner constitutes a pivotal requirement to carry out translational multi-center studies.

Federated learning (FL) has emerged as a promising distributed learning approach, where the parties keep their raw data on their premises and exchange intermediate model parameters.[11] This approach has enabled collaborative learning for several medical applications, and it has been shown that FL performs comparably with centralized training on medical datasets.[12–14] Recently, the concept of swarm learning (SL) has been proposed; it enables decentralized machine learning for precision medicine. The seminal work of SL[15] is based on edge computing and permissioned blockchains and removes the need for a central server in the FL approach. Despite the advantages of FL and SL for keeping the sensitive data local and for reducing the amount of data transferred/outsourced, the model and the intermediate values exchanged between the parties remain prone to several privacy attacks executed by the other parties or the aggregator (in FL), such as membership inference attacks[16,17] or reconstructing the parties' inputs.[18–20] In this work, we provide a solution that further conceals the global machine learning model from the participants by relying on mathematically secure cryptographic techniques to mitigate these inference attacks.

To mitigate or prevent the leakage in the FL setting, several privacy-preserving mechanisms have been proposed. These mechanisms can be classified under three main categories, depending on the strategy they are based on: differential privacy (DP), secure multiparty computation (SMC), and homomorphic encryption (HE).

DP-based solutions aim to perturb the parties' input data or the intermediate model values exchanged throughout the learning. Several studies in the medical domain keep the data on the local premises and use FL with a differential privacy mechanism on the exchanged model parameters.[21–23] Despite being a pioneering mitigation against privacy attacks, DP-based solutions perturb the model parameters, thus decreasing the utility and making the deployment harder for medical applications, where the accuracy is already constrained by limited data. Quantification of the privacy achieved via DP-based approaches is also very difficult[24] and the implementation of DP, especially in medical imaging applications, is not a trivial task.[9]

Another line of research relies on SMC techniques to ensure privacy and to enable collaborative training of machine learning models.[25–29] SMC techniques rely on secret-sharing the data of the parties and on performing the training on the secret-shared data among multiple computing nodes (usually 2, 3, or 4 nodes). Nevertheless, it is usually hard to deploy these solutions, as they often rely on a trusted third party for the sake of efficiency. Moreover, their scalability with the number of parties is poor due to the large communication overhead.

Finally, several works employ HE to enable secure aggregation or to secure outsourcing of the training to a cloud server.[30,31] These solutions, however, cannot solve the distributed scenario where parties keep their local data in their premises.

The adoption of each of the aforementioned solutions introduces several privacy, utility, and performance trade-offs that need to be carefully balanced for healthcare applications. To balance these trade-offs, several works employ multiparty homomorphic encryption (MHE).[32,33] Although the underlying model in these solutions enables privacy-preserving distributed computations and maintains the local data of the parties on their local premises, the functionality of these works is limited to the execution of simple operations, i.e., basic statistics, counting, or linear regression, and the underlying protocols do not support an efficient execution of neural networks in the FL setting.

Recently, Sav et al. proposed a more versatile solution, Poseidon, for enabling privacy-preserving federated learning for neural networks by relying on MHE[34] to mitigate FL inference attacks by keeping the model and intermediate values encrypted. Their solution, however, does not address the implementation and efficient execution of convolutional neural networks, a widely adopted machine learning model to analyze complex data types, such as single-cell data.

We propose *PriCell*, a solution based on MHE to enable the training of a federated convolutional neural network in a privacy-preserving manner, thus preserving the utility of the data for single-cell analysis. To the best of our knowledge, *PriCell* is the first of its kind in the regime of privacy-preserving multi-center single-cell studies under encryption. By bringing privacy-by-design and by preventing the transfer of patients' data to other institutions, our work contributes to single-cell studies and streamlines the slow and demanding process of the reviewing of independent ethics committees for consent forms and study protocols. To mitigate FL attacks, we keep the model and any value that is exchanged between the parties in an encrypted form, and we rely on the threat model and setting proposed in the work of Sav et al.[34] (detailed in the experimental procedures).

By designing new packing strategies and homomorphic matrix operations, we improve the performance of the protocols for encrypted convolutional neural networks that are predominantly used in the healthcare domain.[35] To evaluate our system within the framework of single-cell analysis, we train a convolutional neural network (CellCnn), designed by Arvaniti and Claassen,[4] within our privacy-preserving system for the disease classification task. We also show the feasibility of our solution with several single-cell datasets utilized for cytomegalovirus infection (CMV)[1] and acute myeloid leukemia (AML)[2] classification, and one dataset for non-inflammatory neurological disease (NIND) and relapsing-remitting multiple sclerosis (RRMS)[3] classification. We compare our classification accuracy in a privacy-preserving FL setting with the centralized and non-encrypted baseline. Our solution converges comparably with the training with centralized data, and we improve on the state-of-the-art decentralized secure solution[34] in terms of training time. For example, in a setting with 10 parties, we improve Poseidon's execution time by at least one order of magnitude.

RESULTS

In this section, we introduce the system overview of our solution, present the neural network architecture that is used for our evaluation, and lay out our experimental findings.

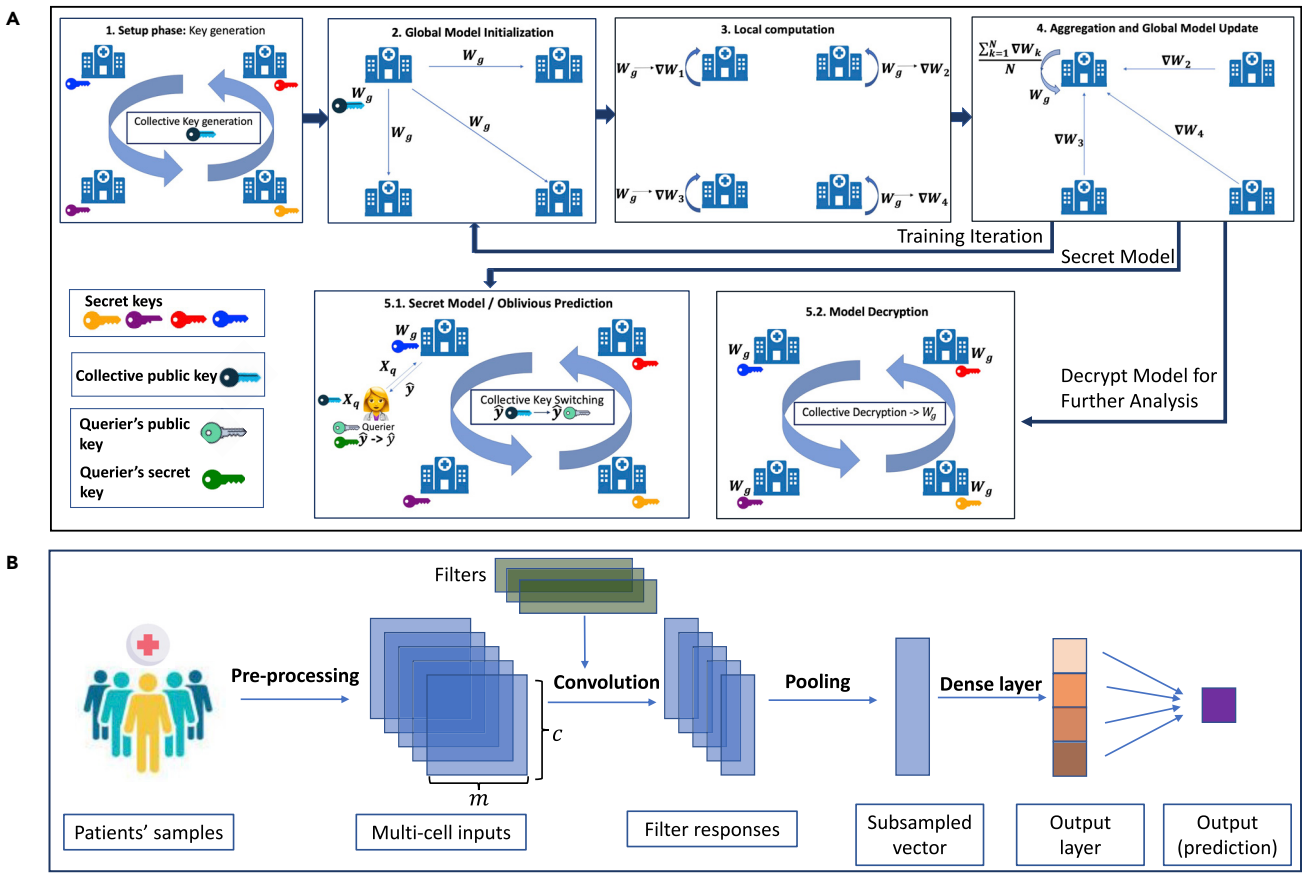


Figure 1. *PriCell*'s system model

(A) *PriCell*'s training and evaluation workflow. Training encapsulates the generation of cryptographic keys and the federated learning iterations on an encrypted model with multiple healthcare institutions. After training, the model is either kept encrypted or decrypted for further analysis.

(B) CellCnn[4] neural network architecture that is used in the local computation phase of (A). The network takes multi-cell samples as an input and applies a 1D convolution with *h* filters followed by a pooling layer. A dense (fully connected) layer then outputs the phenotype prediction.

System overview

We summarize *PriCell*'s system and its workflow for collaborative training and query evaluation (prediction) in Figure 1A. Assuming that there are four healthcare institutions with each holding its respective secret key, the workflow starts with the generation of a *collective public key* and a set of *evaluation keys* that are necessary for the encrypted operations, using each participant's secret key. We refer to this phase as the *setup phase*. In the second phase, the participants agree on the initial random global model weights (W_g) and encrypt them with the collective public key. We denote the encryption of any value with boldface letters, i.e., $\boldsymbol{W_g}$. After encrypting the initial global weights, the *local computation phase* begins; we base this phase on a variant of the FedAvg algorithm[11] to enable collective training. Based on this algorithm, to find the model gradients ($\nabla \boldsymbol{W_k}$) each party performs several encrypted training iterations on their local data (local iteration). The local model gradients are then sent and aggregated at one of the parties that will perform the global model update. The updated model is then broadcast back and the process returns to phase 2. After a fixed number of training iterations, the participants can choose to keep the model confidential (option 5.1 in Figure 1A) or to decrypt it for further analysis (option 5.2 in Figure 1A).

If prediction-as-a-service is offered to a querier (a researcher) and the model is kept encrypted, the querier must encrypt the evaluation data ($\boldsymbol{X_q}$) with the collective public key of the parties. Once the prediction is done, the result ($\tilde{\boldsymbol{y}}$) is collectively switched to the public key of the querier by using the underlying crypto-scheme's collective key-switching functionality. If the model is instead decrypted, the querier encrypts the data with their own key; hence, no key switch is needed after the prediction. As a result, regardless of the model being confidential or not, the evaluation data of the querier and the prediction result always remain protected, as only the querier can decrypt the end result.

CellCnn model overview

CellCnn is a convolutional neural network that enables multi-instance learning and associates a set of observations on cellular population, namely multi-cell inputs, with a phenotype.[4] This architecture is designed for detecting rare cell subsets associated with a disease by using multi-cell inputs generated from high-dimensional single-cell marker measurements. By their nature, these inputs can be used to predict the phenotype of a donor

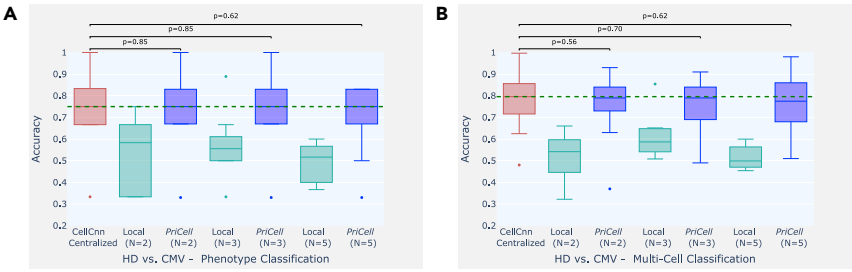


Figure 2. Accuracy boxplots when classifying healthy donor versus cytomegalovirus infection for training multi-cells drawn from the bag of all cells per class

(A and B) Experiments are repeated 10 times with different train and test set splits, the vertical dashed line illustrates the median for the baseline (CellCnn) and the dots represent the outliers. The p values shown at the top of the figure are calculated with a Wilcoxon signed-rank test for the comparison between the corresponding boxplots ($p > 0.05$ indicates that the distributions are not significantly different). Classification accuracy is reported for two datasets: (A) phenotype classification of 6 patients and (B) multi-cell input classification on 4,000 samples. HD, healthy donor; CMV, cytomegalovirus infection.

or the associated phenotype for a given subset of cells. In this scenario, we enable privacy-preserving and distributed multi-instance learning, and we compare our classification performance with the baseline (CellCnn[4] trained on centralized data with no privacy protection). We note here that replicating the full-pipeline of CellCnn[4] for downstream analysis requires either heavy approximations under encryption or the decryption of the trained model. Our solution enables the collective and privacy-preserving training for the classification task, whereas subsequent analyses that require access to the model are beyond the scope of this work. Yet, we show the negligible effect that our encryption would practically have on these analyses in Note S6.

We show the architecture of CellCnn[4] in Figure 1B. The network comprises a 1D convolutional layer followed by a pooling layer and a dense (fully connected) layer. Each multi-cell input sample in Figure 1B is generated using c cells per phenotype with *m* features (markers), and these samples are batched to construct multi-cell inputs. The training set is then generated by choosing z multi-cell inputs per output label or per patient.

We refer to the reader to the work of Arvaniti and Claassen[4] for the details of the neural network architecture. We detail the changes we introduce to this architecture to enable operations under HE in the local neural network operations in the experimental procedures.

Experimental evaluation

We evaluate our proposed solution in terms of model accuracy, runtime performance, scalability with the number of parties, number of data samples, number of features, and communication overhead. In this section, we also provide a comparison with previous work. We give details on the machine learning hyperparameters and security parameters used for our evaluation in Note S3.

Model accuracy

To assess our solution in terms of accuracy, we use the same datasets used in two peer-reviewed biomedical studies.[4,3] We rely on three datasets to perform NIND, RRMS, CMV, and AML classification. We give the details of each dataset in the datasets subsection of the experimental procedures. *Our aim is to show that PriCell achieves a classification performance on par with the centralized non-private baseline.*

As the original studies rely on centralized datasets, we evenly distribute the individual donors in the respective dataset over N parties. We give the classification performance on these datasets in Figures 2, 3, and 4, and provide a tabular version of all the results corresponding to these plots with accuracy, precision, recall, and F score metrics in Tables S3–S7. The x axis shows different training approaches: (1) the data are centralized and the original CellCnn approach[4] is used for training and classification to construct a baseline, (2) each party trains a model with only its local data (Local) without collaborating with other parties, and (3) our solution for privacy-preserving collaboration between parties is used (*PriCell*). For the Local training (2), we average the test accuracy achieved by individual parties. We perform Wilcoxon signed-rank test to prove that the accuracy of the centralized CellCnn and *PriCell* are not significantly different. $p > 0.05$ indicates that there is not enough evidence to reject the null hypothesis and all our findings suggest that there is no significant difference between *PriCell* and the centralized non-private baseline (CellCnn). Note that this is the desired outcome as our aim is to show that the results of *PriCell* and CellCnn are similar.

In our experiments, random multi-cell inputs that are used for training are drawn with replacement from the original training samples. Drawing multi-cell inputs can be done in two ways: using the bag of all cells per class or individually drawing them from each patient. We report the classification performance by using two test datasets: one set is generated using multi-cell inputs with $c = 100 - 200$ cells drawn from all patients in the test set to increase the size of the test set for multi-cell classification, and the second set is generated by drawing 1,000–10,000 cells from each donor separately for phenotype prediction. We give more details about the setting and hyperparameters for each experiment in Note S3.

For CMV classification, we generate the training data by drawing random cell subsets from the cell bags per phenotype. For NIND and RRMS classification, we observe that drawing multi-cells per phenotype varies the accuracy between runs and that the median accuracy over 10 runs increases when distributing the initial dataset among $N = 6$ parties (see Figures 3A, 3B, 4A, and 4B). This suggests that separately drawing multi-cell inputs from each individual performs better for this task, as corroborated by the results obtained with drawing 2,000 cells from each patient with replacement (see Figures 3C, 3D, 4C, and 4D). Finally, in Table S2, we report the median accuracy, precision, recall, and F score of 10 runs (with different train and test set splits) on patient-based sub-sampling for NIND and RRMS,

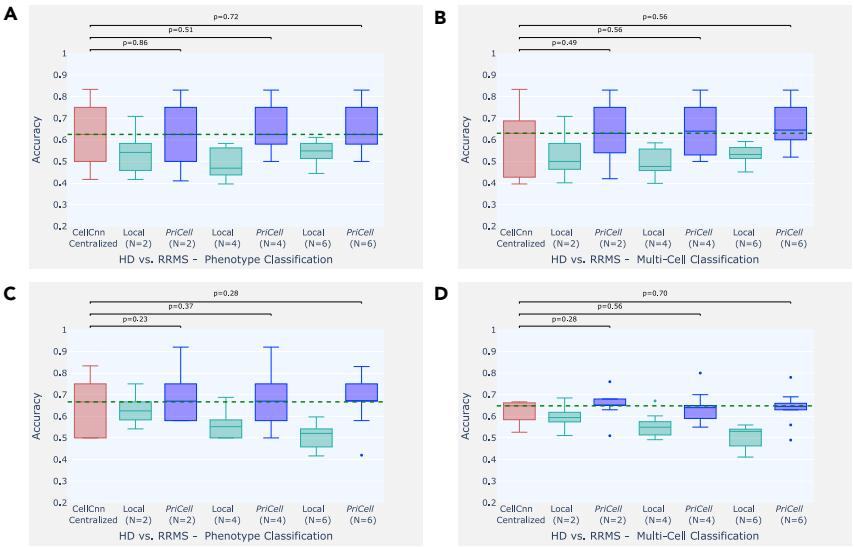


Figure 3. Accuracy boxplots when classifying healthy donor versus relapsing-remitting multiple sclerosis

(A–D) Accuracy boxplots when classifying healthy donor versus relapsing-remitting multiple sclerosis, for training multi-cells drawn from the bag of all cells per class (A and B) and drawn from each patient separately (C and D).

Experiments were repeated 10 times with different train and test set splits, the vertical dashed line illustrates the median for the baseline (CellCnn) and the dots represent the outliers. The p values shown at the top of the figure are calculated with a Wilcoxon signed-rank test for the comparison between the corresponding boxplots ($p > 0.05$ indicates that the distributions are not significantly different). Classification accuracy is reported for 2 datasets: multi-cell input classification on 96 samples and phenotype classification of 12 patients. HD, healthy donor; RRMS, relapsing-remitting multiple sclerosis.

and phenotype-based sub-sampling for CMV and we provide the F score plots for all experiments in Figures S1–S3 to support our findings.

To construct a realistic overall distribution, we limit the number of parties to be lower than the number of donors in the dataset. We observe that, given a sufficient number of samples per party, our distributed secure solution achieves classification performance comparable with the original work, where the data are centralized and the training is done without privacy protection. In the experiments on CMV, for example, the median accuracy achieved by *PriCell* is exactly the same as the centralized baseline for phenotype classification and very close (at most 2% gap) for multi-cell classification. Analogous results are obtained for the other experiments: our privacy-preserving distributed solution achieves almost the same median accuracy with the baseline in RRMS and NIND with patient-based sub-sampling, where the datasets are sufficiently large to be distributed among up to six parties.

Finally, we provide the classification performance on AML in Table S2. As the dataset is relatively small, emulating a distributed setting with more than two parties was not feasible for this task and, as the accuracy does not vary in between different train-test splits, we do not provide the boxplots on the accuracy. However, we observe that, with two parties in *PriCell* training, the accuracy remains exactly the same as the centralized baseline for AML classification.

Most importantly, our evaluation shows that there is always a significant gain in classification performance when switching from local training to privacy-preserving collaboration. The number of donors that each institution has is insufficient for individually training a robust model. In all experimental settings, for a fixed number of N, *PriCell* achieves better performance than the local training while ensuring the confidentiality of the local data.

Runtime

In Table 1, we report the execution times for the training and prediction with $N = 10$ parties and a *ring degree* $\mathcal{N} = 2^{15}$. To be able to compare the runtimes at a larger scale, we use synthetically generated data for this set of experiments and vary the number of features (m). We generate a dataset of 1,000 samples per party with $c = 200$ cells per sample. We use $h = 8$ filters, a local batch size of $n = 100$, and 20 global epochs for training. We report the execution time of the setup phase, of the local computations, and of its communication. We include the execution time of distributed bootstrapping (see the experimental procedures for details) as part of the communication time, which takes 1.2 s per iteration and 122 s over 20 epochs. Hence, the communication column for training comprises the time to perform all communication between parties throughout the training, distributed bootstrapping, and the model update.

We observe that *PriCell* trains, in less than 20 min, a CellCnn model on a training set of 200 cells per sample, 1,000 samples per party, and 32 features across 10 parties, including the setup phase and communication. The training time, when the number of features varies, remains 20–25 min, which is the result of our efficient use of the SIMD (single instruction, multiple data) operations provided by the cryptosystem; this is further discussed in the scalability analysis.

In Table 1, we also report the execution times of an oblivious prediction when both the model and the data are encrypted (phase 5.1 of Figure 1A). We recall that the collective key-switching operation enables us to change the encryption key of a ciphertext from the parties' collective key to the querier's key. The maximum number n of samples that can be batched together for a given ring degree $\mathcal{N}$, number of labels o, and number of features m, is $(\mathcal{N}/2)/(m \cdot o)$ (we also need $m/2$ ciphertexts to batch those samples; see Note S4 for more details). Hence, in our case, the maximum prediction batch size for $\mathcal{N} = 2^{15}$ and $o = 2$ is $n = 2^{13}/m$.

We observe that the local computation for the prediction increases linearly with m, and is linked to the cost of the dominant operation, the convolution, which is, unlike training, carried out between two encrypted matrices (see Note S4). The communication required for prediction includes $m/2$ ciphertexts sent by the querier and one ciphertext (prediction result) sent back by the server. Hence, the communication time also increases

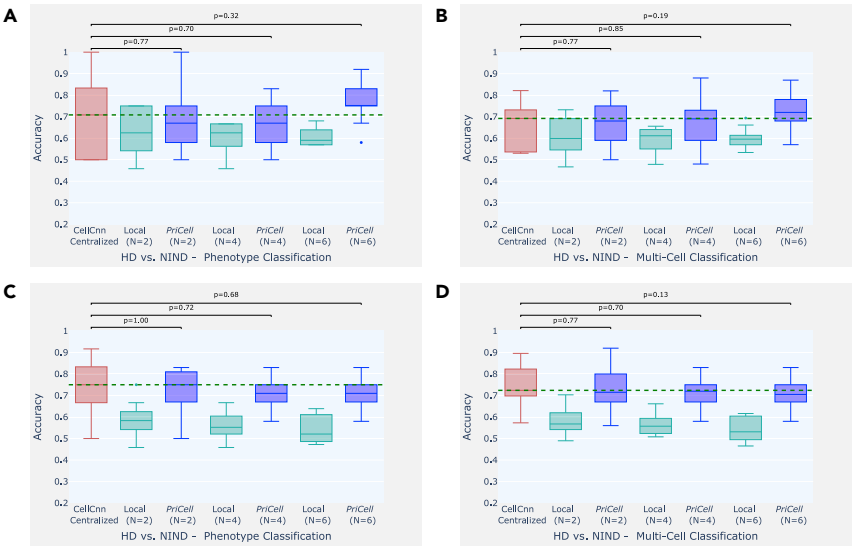


Figure 4. Accuracy boxplots when classifying healthy donor versus non-inflammatory neurological disease

(A–D) Accuracy boxplots when classifying healthy donor versus non-inflammatory neurological disease for training multi-cells drawn from the bag of all cells per class (A and B) and drawn from each patient separately (C and D).

Experiments were repeated 10 times with different train and test set splits, the vertical dashed line illustrates the median for the baseline (CellCnn) and the dots represent the outliers. The p values shown at the top of the figure are calculated with a Wilcoxon signed-rank test for the comparison between the corresponding boxplots ($p > 0.05$ indicates that the distributions are not significantly different). Classification accuracy is reported for 2 datasets: multi-cell input classification on 96 samples and phenotype classification of 12 patients. HD, healthy donor; NIND, non-inflammatory neurological disease.

linearly with m. Finally, the time for the collective key switch remains constant, as it is performed once at the end of the prediction protocol on only one ciphertext.

Scalability analysis

Figure 5 shows the scalability of *PriCell* with the number of parties, the global number of rows (samples), the number of features (markers), and the number of filters for one global training epoch that is to process once all the data of all parties. Unless otherwise stated, we use $c = 200$ cells per sample, a local batch size of $n = 100$, $m = 38$ features, and $h = 8$ filters, for all settings.

We first report the runtime with an increasing number of parties (N) in Figures 5A and 5B when the global number of data samples is fixed to $s = 18{,}000$ and when the number of samples per party is fixed to 500, respectively. As the parties perform local computations in parallel, *PriCell*'s runtime decreases with increasing N when s is fixed (Figure 5A). When the number of data samples is constant per party, *PriCell*'s computation time remains almost constant and only the communication overhead increases when increasing N (Figure 5B).

We further analyze *PriCell*'s scalability for $N = 10$ when varying the number of global samples (s), the number of features (m), and the number of filters (h). In Figure 5C, we show that *PriCell* scales linearly when increasing the number of global samples with $N = 10$. Increasing the number of features and filters has almost no effect on *PriCell*'s runtime due to our efficient packing strategy that enables SIMD operations through features and filters. However, we note that the increase in $h = 64$ in Figure 5E is due to increasing the cryptosystem parameter $\mathcal{N}$ to have a sufficient number of slots to still rely on our one-cipher packing strategy. The increase in runtime is still linear with respect to $\mathcal{N}$ and, as expected, the use of larger ciphertexts also produces a slight increase in the communication

Comparison with previous work

The most recent solutions for privacy-preserving FL in the N-party setting use DP, SMC, or HE.

DP-based solutions[21–23] in the medical domain introduce noise in the intermediate values to mitigate adversarial FL attacks. However, it has been shown that training an accurate model with DP requires a high-privacy budget.[36] Thus, DP-based solutions introduce a privacy-accuracy trade-off by perturbing the model parameters, whereas *PriCell* decouples the accuracy from the privacy and achieves privacy-by-design with a reasonable overhead.

SMC-based solutions[25–29] often require the data providers to communicate their data outside their premises to a limited number of computing nodes, and these solutions assume an honest majority among the computing nodes to protect the data and/or model confidentiality. Comparatively, *PriCell* scales efficiently with N parties and permits them to keep their data on their premises, withstanding collusions of up to $N - 1$ parties.

Finally, HE-based solutions for privacy-preserving analytics in distributed medical settings[32,33] allow for functionalities (e.g., basic statistics, counting, or linear regression) different than those that *PriCell* enables, and they do not enable the efficient execution of neural networks in an FL setting. Due to the fact that the underlying system, the threat model, and the enabled functionalities of all the aforementioned solutions are different from *PriCell*, a quantitative comparison with these works is a challenging task.

We build on the system and threat model proposed by Sav et al.[34] for enabling privacy-preserving FL for neural networks by relying on MHE and make a quantitative comparison with Poseidon. *PriCell* improves upon the state-of-the-art solution, Poseidon, by at least one order of magnitude for training times when the same number of epochs and filters is used. This is due to *PriCell*'s design for optimizing the use of SIMD operations, with a packing strategy that enables encrypting all samples of a batch in a single ciphertext; whereas Poseidon packs the samples within a batch in different ciphertexts. For a local batch size of $n = 1$, 8 filters, 38 features, and 200 cells per sample, *PriCell*'s local computation time is 1.7 s, whereas, Poseidon's is 15.4 s. Increasing the batch size to 100 results in a 100× slower local execution for Poseidon, whereas it remains constant for *PriCell*, as all samples are packed in one ciphertext. In summary, increasing the batch size or the number of filters yields a linear increase in the advantage of our solution, in terms of local computation time.

Table 1. *PriCell*'s execution times for training and prediction with a varying number of features (m), 10 parties, and ring degree $\mathcal{N} = 2^{15}$ (2^{14} *ciphertext slots*)

	Training execution time (s)			Prediction execution time (s)		
m	Setup	Local computation	Communication	Local computation querier + server	Communication	Collective key-switch
8	17.8	753.4	370.1	0.2 + 0.1	0.3	0.3
16	18.1	778.7	387.0	0.3 + 0.2	0.6	0.3
32	19.3	836.1	393.7	0.3 + 0.4	1.0	0.3
64	21.9	951.1	373.1	0.6 + 0.6	2.2	0.3
128	24.5	1,135.9	374.8	1.6 + 1.5	4.2	0.3

The computation is single-threaded in a virtual network with an average network delay of 0.17 ms and 1 Gbps bandwidth on 10 Linux servers with an Intel Xeon E5-2680 v.3 CPUs running at 2.5 GHz with 24 threads and 12 cores and 256 GB RAM.

Downstream analysis

The training in the original CellCnn study aims at detecting rare disease-associated cell subsets via further analysis.[4] Assuming the end model is decrypted upon pre-agreement to conduct these analyses, we further investigate how the changes that we introduce in the CellCnn architecture (see local neural network operations in the experimental procedures) affect the detection capability. To be able to make a comparison with the original study in terms of detection capability, we introduce these changes in the original implementation of CellCnn, simulate our encryption, and evaluate the impact in the subsequent analyses. We report our results and how our changes to the circuit and training affect the detection capability on rare CMV infection in Note S6 (see Figures S4–S6).

DISCUSSION

In this work, we present *PriCell*, a system that enables privacy-preserving federated neural network learning for healthcare institutions, in the framework of an increasingly relevant single-cell analysis, by relying on MHE. To the best of our knowledge, *PriCell* is the first solution to enable the training of convolutional neural networks with N parties under encryption on single-cell data. Using MHE, our solution enables the parties to keep their data on their local premises and to keep the model and any FL intermediate values encrypted end-to-end throughout the training. As such, *PriCell* provides security against inference attacks to FL.[16–20] *PriCell* also protects the querier's (researcher's) evaluation data by using oblivious prediction. The underlying encryption scheme of *PriCell* provides post-quantum security and does not degrade the utility of the data, contrarily to differential privacy-based solutions.[21,37]

In this work, we demonstrate the flexibility of *PriCell* with the different learning parameters (e.g., batch size, number of features, number of filters), different real-world datasets, and a varying number of parties. Our empirical evaluation shows that *PriCell* is able to efficiently train a collective neural network with a large number of parties while protecting the model and the data through HE. We also show that *PriCell*'s computation and communication overhead remains either constant or scales linearly with the number of parties and with the model parameters.

Furthermore, we show that *PriCell* achieves classification accuracy comparable with the centralized and non-encrypted training. Our evaluation demonstrates a substantial accuracy gain by collaboration between the parties when compared with locally training with their data only.

In terms of limitations, *PriCell* relies on the assumption that the parties provide non-tampered data. Although poisoning attacks are beyond the scope of this work, *PriCell* can partially mitigate this kind of threat by integrating several mechanisms, such as statistical checks on parties' input data[38] or by integrating zero-knowledge proofs during training.[39,40] Using such techniques for training under encryption in time-constrained applications remains an open research problem.

While enabling privacy-preserving training over N parties, by default *PriCell* does not monitor the training when the whole process is carried out under encryption. As *PriCell* performs all training operations under encryption, the partial models cannot be assessed. However, relaxing the end-to-end security requirement over training, the parties can collectively decrypt the validation accuracy that is computed on an independent validation set. As for the conditions for *PriCell* to lead to accurate and unbiased models, they remain the same as for standard FL approaches, such as tuning the hyperparameters for client and server. This is a possible future research direction and not a trivial task.[41] While the IID setting is not a prerequisite for *PriCell*, we observe that an increased skewness between the parties' data distribution can decrease the model performance. Similar to poisoning attacks, this limitation can be partially addressed by integrating statistical checks to ensure the skewness is below a certain threshold; this requires a comprehensive study on skew types (i.e., quantity, label, or feature) and the thresholds for each of them for single-cell analysis.

In general, as data sharing in the healthcare domain is usually prevented due to the sensitive nature of data, and due to privacy regulations such as HIPAA or GDPR, *PriCell* brings unprecedented value for the healthcare domain, exemplified in this work for single-cell analysis, where the data are scarce and sparse. These benefits are extensible to federated healthcare scenarios that rely on machine learning, and constitute an important landmark for real-world applications of collaborative training between healthcare institutions while preserving privacy.

EXPERIMENTAL PROCEDURES

Resource availability

Lead contact

Further information and requests for resources should be directed to the lead contact, Sinem Sav (sinem.sav@epfl.ch).

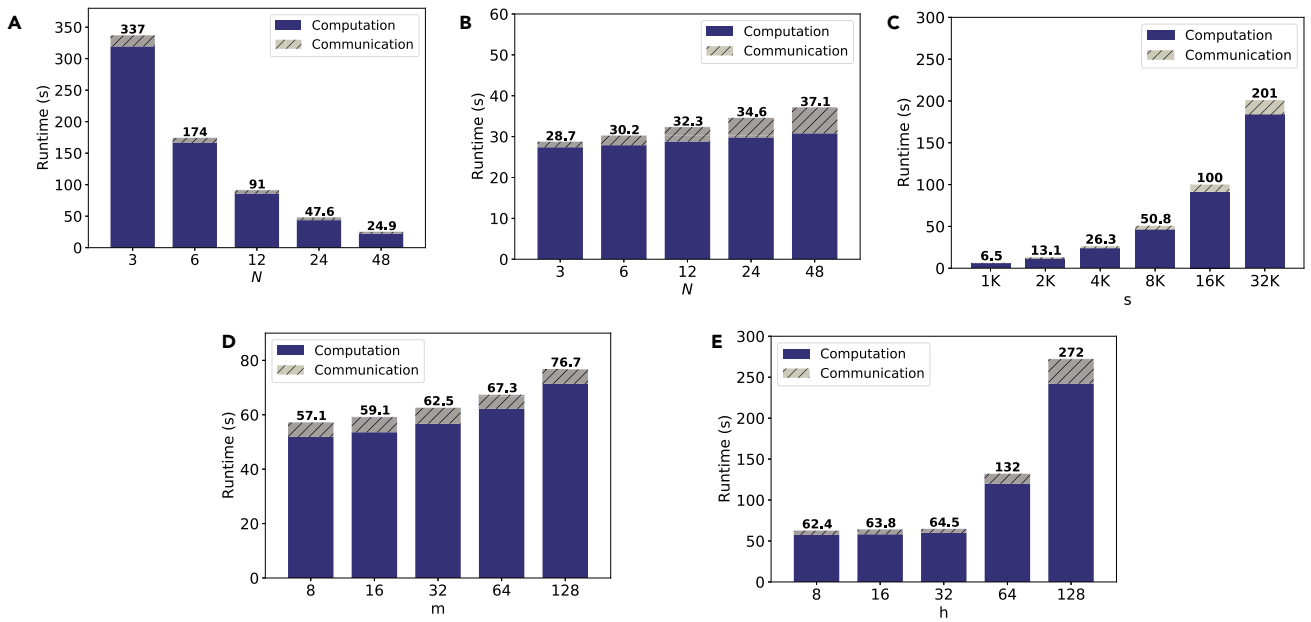


Figure 5. *PriCell*'s training execution time and communication overhead for one training epoch with increasing number of parties, data samples, features, and filters

The computation is single-threaded in a virtual network with an average network delay of 0.17 ms and 1 Gbps bandwidth on 10 Linux servers with an Intel Xeon E5-2680 v.3 CPUs running at 2.5 GHz with 24 threads on 12 cores and 256 GB RAM.

(A) Increasing number of parties (N) when the number of global data samples (s) is fixed to 18,000.

(B) Increasing number of parties (N), each having 500 samples.

(C) Increasing number of data samples (s) when $N = 10$.

(D) Increasing number of features (m) when $N = 10$.

(E) Increasing number of filters (h) when $N = 10$.

Materials availability

No new biological materials were generated by this study.

Data and code availability

This study uses previously published datasets.[1–3] We refer to the reader to Arvaniti and Claassen[4] (https://zenodo.org/record/5597098#.YXbaz9ZBzt0) to obtain the CMV and AML datasets, and to Galli et al.[3] for the repository including the samples from healthy, NIND, and RRMS donors (http://flowrepository.org/experiments/2166/). Our implementation for this study have been deposited at Zenodo (https://doi.org/10.5281/zenodo.6330988).

System and threat model

In this section, we detail *PriCell*'s system and threat model, which is based on the state-of-the-art privacy-preserving framework.[34] In *PriCell*'s scenario, there are N healthcare institutions (parties), each holding its own patient dataset and collectively training a neural network model, without sharing/transferring their local data. Our aim is to preserve the confidentiality of the local data, the intermediate model updates in the FL setting, the querier's evaluation data, and optionally the final model (see Figure 1 for the system overview). We rely on a synchronous learning protocol, assuming that all parties are available throughout the training and evaluation executions. We note here that this assumption can be relaxed by using different HE schemes, such as threshold or multi-key HE,[42,43] but with a relaxed security assumption for the former and an increased computation cost for the latter.

We consider an $N-1$ passive-adversary threat model. Hence, we assume that all parties follow the protocol, and up to $N-1$ colluding parties must not be able to extract any information about the model or other party's input data. We rely on MHE to meet these confidentiality requirements under the posed threat model. In the following section, we briefly introduce the background on the used MHE scheme.

Multiparty Homomorphic Encryption (MHE)

Here, we give the fundamentals of MHE and describe the most important cryptographic details. Note that we use italic fonts for the cryptographic terms when first introduced and these terms are explained in Note S1, cryptography glossary.

We rely on a variant of the Cheon-Kim-Kim-Song (CKKS)[44] cryptographic scheme that is based on the *ring* learning with errors (RLWE) problem (also post-quantum secure[45]) and that provides approximate arithmetic over vectors of complex numbers, i.e., $\mathbb{C}^{\mathcal{N}/2}$ (a complex vector of $\mathcal{N}/2$ slots). Encrypted operations over $\mathbb{C}^{\mathcal{N}/2}$ can be carried out in a SIMD fashion, which allows for excellent amortization. Therefore, adopting an efficient packing strategy that maximizes the usage of all the slots has a significant impact on the overall computation time.

Mouchet et al.[46] show how to construct a threshold variant of RLWE schemes, where the parties have their own secret key and collaborate to establish the collective public keys. In this setting, a model can be collectively trained without having to share the individual secret keys of the parties; this prevents the parties' decryption functionality, without the collaboration of all the parties.

Note that a fresh CKKS ciphertext permits only a limited number of homomorphic operations to be carried out. To enable further homomorphic operations, a ciphertext must be refreshed with a *bootstrapping operation* once it is exhausted. This operation is a costly function and requires communication in our system; hence we optimize the circuit to minimize the number of bootstrapping operations and the number of ciphertexts to be bootstrapped (see detailed neural network circuit in this section).

Datasets

We detail the features of the three used datasets:

Non-inflammatory neurological disease (NIND), relapsing–remitting multiple sclerosis (RRMS)

We rely on a large cohort of peripheral blood mononuclear cells, including 29 healthy donors (HD), 31 NIND, and 31 RRMS donors.[3] The dataset

comprises samples with a varying number of cells for each donor and 35 markers for each cell. We use this dataset for two classification tasks: (1) HD versus NIND and (2) HD versus RRMS, as shown in Figures 3 and 4. For both NIND and RRMS experiments, and in all experimental settings, we use 48 donors (24 HD, 24 NIND/RRMS) for training and 12 donors (5 HD, 7 NIND/RRMS) for testing.

Cytomegalovirus Infection (CMV)

We use a mass cytometry dataset[1] for the classification of CMV. This dataset comprises samples from 20 donors with a varying number of cells for each donor and mass cytometry measurements of 37 markers for each cell, and has 11 CMV− and 9 CMV+ labels. We use 14 donors for training and 6 donors as a test set in all experimental settings.

Acute Myeloid Leukaemia (AML)

We rely on the mass cytometry dataset from Levine et al.[2] for the three-class classification problem for healthy, cytogenetically normal (CN), and core-binding factor translocation (CBF). For each cell, the dataset includes mass cytometry measurements of 16 markers. As in the original work,[4] we use the AML samples with at least 10% $CD34^+$ blast cells with the availability of additional cytogenetic information. The final training dataset comprises three healthy bone marrows (BM1, BM2, and BM3), two CN samples (SJ10 and SJ12), and two CBF samples (SJ1 and SJ2). The test set in all experimental settings comprises two healthy bone marrows (BM4, BM5), one CN (SJ13), and three CBF (SJ3, SJ4, and SJ5) samples.

The individual donors in all aforementioned training sets are then evenly distributed among N parties for *PriCell* collective training. To construct our baselines and to make a fair comparison with the baseline, we use the same data preprocessing for all experiments per setting (centralized CellCnn, Local, or *PriCell*). We give the details of the data preprocessing and parameter selection, in Note S3.

Local neural network operations

In this section, we give a high-level description of the neural network circuit that is evaluated in the encrypted domain (a detailed and step-by-step description can be found in Note S4). We list the frequently used symbols and notations in Note S2 and Table S1.

We first present the changes introduced to the original CellCnn circuit to enable an efficient evaluation under encryption: (1) we approximate the non-polynomial activation functions by polynomials by using least-squares approximation, (2) we replace the max pooling with an average pooling, and (3) we replace the ADAM optimizer with the stochastic gradient descent (SGD) optimizer with mean-squared error and momentum acceleration. Finally, we introduce the packing strategy used in *PriCell* and give a high-level circuit overview. We give more details on these steps and optimizations in Note S4, and we empirically evaluate the effect of these optimizations on the model accuracy in a distributed setting in the results section.

Polynomial approximations

With additions and multiplications, the CKKS scheme can efficiently evaluate polynomials in the encrypted domain. However, these two basic operations are not sufficient for easily evaluating non-linear functions, such as sign or sigmoid. A common strategy to circumvent this problem is to find a polynomial approximation of the desired function. We rely on polynomial least-squares approximations for the non-polynomial activation functions, such as sigmoid, and we use identity function after convolution (instead of ReLU). We show in our results section that these changes only have a negligible effect on the model accuracy.

Pooling

The original CellCnn circuit makes use of both max pooling and average pooling. Max pooling requires the computation of the non-linear sign function which cannot be efficiently done under encryption. We replace the max pooling with the average pooling, which is a linear transformation and brings the following advantages: (1) it is efficient for computing under encryption with only additions and constant multiplication, (2) it simplifies the backward pass under encryption, and (3) it commutes with other linear transformations or functions, such as the convolution and the identity activation, which allows for an efficient preprocessing of the data and reduces the online execution cost. Indeed, we are able to pre-compute the average pooling on the data, which reduces the input size of a batch of samples from $n \times c \times m$ to $n \times m$, i.e., we remove the dependency on c.

Optimizer

The original CellCnn training relies on the ADAM optimizer, which requires the computation of square roots and inverses. Although approximating these operations is possible, a high-precision approximation requires an excessive use of ciphertext levels and significantly reduces the efficiency of the training. To avoid these costly operations, we rely instead on the SGD optimizer with momentum acceleration that, for an equivalent amount of epochs, shows a comparable rate of convergence to the ADAM optimizer.

Packing strategy

The CKKS scheme provides complex arithmetic on $\mathbb{C}^{\mathcal{N}/2}$ in a SIMD fashion. The native operations are addition, multiplication by a constant, multiplication by a plaintext, multiplication by a ciphertext, *slots rotation* (shifting the values in the vector), and complex conjugation. As the rotations are expensive, when considering encrypted matrix operations one of the main challenges is to minimize the number of rotations, which can be done by adopting efficient packing strategies and algorithms. We give more details about the packing and algorithms in Note S4.

With the aforementioned pre-computed pooling, only a $1 \times m$ vector is needed to represent a sample, instead of a $c \times m$ matrix. Hence, we pack an entire batch of n samples in a single ciphertext and compute the forward and backward pass on the whole batch in parallel, which reduces the complexity of the training proportionally to the size of the batch.

Encrypted circuit overview

Given a batch size of n samples, each sample being a matrix $L_{c \times m}$, for c the number of cells per sample and m number of features (markers) per cell, we first evaluate the mean pooling across the cells in plaintext. The result is a set of n vectors of size $1 \times m$, which is packed in an $L_{n \times m}$ matrix. The 1D convolution is evaluated with an $L_{n \times m} \times C_{m \times k}$ matrix multiplication. We feed the result to the dense layer $W_{k \times o}$, where o is the number of output labels. Finally, we perform an approximated activation function to the output of the dense layer. Our encrypted circuit with reduced complexity is

$$Y_{n \times o} = \mathrm{Poly}_{\mathrm{act}}((\mathrm{MeanPool}(L_{n \times c \times m}) \times C_{m \times k}) \times W_{k \times o}).$$

Experimental settings

We implemented our solution in Go (Go Programming Language, https://go.dev/) by using the open-source lattice-based cryptography Lattigo (a library for lattice-based HE in Go, https://github.com/tuneinsight/lattigo). We use the implementation of CellCnn[4] to preprocess the data and to construct baselines. We use Onet (Cothority Network Library, https://github.com/dedis/onet) to build a decentralized system and Mininet (http://mininet.org) to evaluate our system in a virtual network with an average network delay of 0.17 ms and 1 Gbps bandwidth on 10 Linux servers with Intel Xeon E5-2680 v.3 CPUs running at 2.5 GHz with 24 threads on 12 cores and 256 GB RAM. The parties communicate over TCP with secure channels (TLS). We choose security parameters that achieve security of at least 128 bits.[47]

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.patter.2022.100487.

ACKNOWLEDGMENTS

We would like to thank Apostolos Pyrgelis, David Froelicher, and Sylvain Chatel who gave valuable feedback on the manuscript. We also thank Shufan Wang and Joao Sa Sousa for their contribution on the experiments and benchmarking. This work was partially supported by grant no. 2017-201 of the Strategic Focal Area "Personalized Health and Related Technologies (PHRT)" of the ETH Domain. M.C. is a member of the Machine Learning Cluster of Excellence, EXC no. 2064/1 – project no. 390727645.

AUTHOR CONTRIBUTIONS

S.S., J.R.T.-P., M.C., and J.-P.H. conceived the study. S.S. and J.-P.B. developed the methods, implemented them, and performed the experiments. All authors contributed to the methodology and wrote the manuscript.

DECLARATION OF INTERESTS

J.R.T.-P. and J.-P.H. are co-founders of the start-up Tune Insight SA (https://tuneinsight.com). The other authors declare no competing interests.

The methods presented in this work are partially covered by the PCT patent applications under publication nos. WO/2022/042848[48] and WO/2021/223873.[49]

Received: December 18, 2021
Revised: February 14, 2022
Accepted: March 14, 2022
Published: April 18, 2022

REFERENCES

1. Horowitz, A., Strauss-Albee, D., Leipold, M., Kubo, J., Nemat-Gorgani, N., Dogan, O., Dekker, C.L., Mackey, S., Maecker, H., Swan, G.E., and Davis, M.M. (2013). Genetic and environmental determinants of human nk cell diversity revealed by mass cytometry. Sci. Transl. Med. *5*, 208ra145. https://doi.org/10.1126/scitranslmed.3006702.
2. Levine, J., Simonds, E., Bendall, S., Davis, K., Amir, E.A., Tadmor, M., et al. (2015). Data-driven phenotypic dissection of aml reveals progenitor-like cells that correlate with prognosis. Cell *162*, 184–197. https://doi.org/10.1016/j.cell.2015.05.047.
3. Galli, E., Hartmann, F.J., Schreiner, B., Ingelfinger, F., Arvaniti, E., Diebold, M., Mrdjen, D., van der Meer, F., Krieg, C., Nimer, F.A., and Sanderson, N. (2019). GM-CSF and CXCR4 define a t helper cell signature in multiple sclerosis. Nat. Med. *25*, 1290–1300. https://doi.org/10.1038/s41591-019-0521-4.
4. Arvaniti, E., and Claassen, M. (2017). Sensitive detection of rare disease-associated cell subsets via representation learning. Nat. Commun. *8*, 14825. https://doi.org/10.1038/ncomms14825.
5. Wang, T., Bai, J., and Nabavi, S. (2021). Single-cell classification using graph convolutional networks. BMC Bioinf. *22*, 364. https://doi.org/10.1186/s12859-021-04278-2.
6. Kirby, S., Eng, P., Danter, W., George, C., Francovic, T., Ruby, R.R., and Ferguson, K.A. (1999). Neural network prediction of obstructive sleep apnea from clinical criteria. Chest *116*, 409–415. https://doi.org/10.1378/chest.116.2.409.
7. Vieira, S., Pinaya, W.H., and Mechelli, A. (2017). Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: methods and applications. Neurosci. Biobehav. Rev. *74*, 58–75. https://doi.org/10.1016/j.neubiorev.2017.01.002.
8. Uddin, M., Wang, Y., and Woodbury-Smith, M. (2019). Artificial intelligence for precision medicine in neurodevelopmental disorders. NPJ Digital Med. *2*. https://doi.org/10.1038/s41746-019-0191-0.
9. Kaissis, G., Makowski, M., Rückert, D., and Braren, R. (2020). Secure, privacy-preserving and federated machine learning in medical imaging. Nat. Machine Intelligence *2*, 305–311. https://doi.org/10.1038/s42256-020-0186-1.
10. Regev, A., Teichmann, S., Lander, E., Amit, I., Benoist, C., Birney, E., Bodenmiller, B., Campbell, P., Carninci, P., Clatworthy, M., et al. (2017). Science forum: the human cell atlas. Elife *6*, e27041. https://doi.org/10.7554/eLife.27041.
11. McMahan, B., Moore, E., Ramage, D., Hampson, S., and Arcas, B.A. (2017). Communication-efficient learning of deep networks from decentralized data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, *54*, A. Singh and J. Zhu, eds. (Proceedings of Machine Learning Research (PMLR)), pp. 1273–1282. https://proceedings.mlr.press/v54/mcmahan17a.html.
12. Sadilek, A., Liu, L., Nguyen, D., Kamruzzaman, M., Serghiou, S., Rader, B., Ingerman, A., Mellem, S., Kairouz, P., Nsoesis, E.O., et al. (2021). Privacy-first health research with federated learning. NPJ Digital Med. *4*, 132. https://doi.org/10.1038/s41746-021-00489-2.
13. Sheller, M., Edwards, B., Reina, G., Martin, J., Pati, S., Kotrotsou, A., Milchenko, M., Xu, W., Marcus, D., Colen, R.R., and Bakas, S. (2020). Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. Sci. Rep. *10*, 12598. https://doi.org/10.1038/s41598-020-69250-1.
14. Gaye, A., Marcon, Y., Kutschke, J., Laflamme, P., Turner, A., Jones, E., Minion, J., Boyd, A.W., Newby, C.J., Nuotio, M.-L., et al. (2014). Datashield: taking the analysis to the data, not the data to the analysis. Int. J. Epidemiol. *43*, 1929–1944. https://doi.org/10.1093/ije/dyu188.
15. Warnat-Herresthal, S., Schultze, H., Shastry, K., Manamohan, S., Mukherjee, S., Garg, V., Sarveswara, R., Händler, K., Pickkers, P., Aziz, N.A., et al. (2021). Swarm learning for decentralized and confidential clinical machine learning. Nature *594*, 265–270. https://doi.org/10.1038/s41586-021-03583-3.
16. Melis, L., Song, C., De Cristofaro, E., and Shmatikov, V. (2019). Exploiting unintended feature leakage in collaborative learning. In 2019 IEEE Symposium on Security and Privacy (SP), pp. 691–706. https://doi.org/10.1109/SP.2019.00029.
17. Nasr, M., Shokri, R., and Houmansadr, A. (2019). Comprehensive privacy analysis of deep learning: passive and active white-box inference attacks against centralized and federated learning. In 2019 IEEE Symposium on Security and Privacy (SP) (IEEE), pp. 739–753. https://doi.org/10.1109/SP.2019.00065.
18. Hitaj, B., Ateniese, G., and Perez-Cruz, F. (2017). Deep models under the gan: information leakage from collaborative deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS '17 (Association for Computing Machinery), pp. 603–618. ISBN 9781450349468. https://doi.org/10.1145/3133956.3134012.
19. Wang, Z., Mengkai, S., Zhang, Z., Song, Y., Wang, Q., and Qi, H. (2019). Beyond Inferring Class Representatives: User-Level Privacy Leakage from Federated Learning (IEEE), pp. 2512–2520. https://doi.org/10.1109/INFOCOM.2019.8737416.
20. Zhu, L., Liu, Z., and Han, S. (2019). Deep Leakage from Gradients (Curran Associates Inc.). http://papers.nips.cc/paper/9617-deep-leakage-from-gradients.
21. Choudhury, O., Gkoulalas-Divanis, A., Salonidis, T., Sylla, I., Park, Y., Hsu, G., and Das, A. (2019). Differential privacy-enabled federated learning for sensitive health data. https://arxiv.org/abs/1910.02578.
22. Kim, M., Lee, J., Ohno-Machado, L., and Jiang, X. (2020). Secure and differentially private logistic regression for horizontally distributed data. IEEE Trans. Inf. Forensics Secur. *15*, 695–710. https://doi.org/10.1109/TIFS.2019.2925496.
23. Li, W., Milletarì, F., Xu, D., Rieke, N., Hancox, J., Zhu, W., Baust, M., Cheng, Y., Ourselin, S., Cardoso, M.J., and Feng, A. (2019). Privacy-preserving federated brain tumour segmentation. In International Workshop in Machine Learning in Medical Imaging (MLMI) (Springer). https://doi.org/10.1007/978-3-030-32692-0_16.
24. Jayaraman, B., and Evans, D. (2019). Evaluating differentially private machine learning in practice. In 28th USENIX Security Symposium (USENIX Security 19) (USENIX Association), pp. 1895–1912, ISBN 978-1-939133-06-9. https://www.usenix.org/conference/usenixsecurity19/presentation/jayaraman.
25. Jagadeesh, K.A., Wu, D.J., Birgmeier, J.A., Boneh, D., and Bejerano, G. (2017). Deriving genomic diagnoses without revealing patient genomes. Science *357*, 692–695. https://doi.org/10.1126/science.aam9710.
26. Cho, H., Wu, D., and Berger, B. (2018). Secure genome-wide association analysis using multiparty computation. Nat. Biotechnol. *36*, 547–551. https://doi.org/10.1038/nbt.4108.
27. Constable, S., Tang, Y., Wang, S., Jiang, X., and Chapin, S. (2015). Privacy-preserving gwas analysis on federated genomic datasets. BMC Med. Inf. Decis. Making *15*, S2. https://doi.org/10.1186/1472-6947-15-S5-S2.
28. Kamm, L., Bogdanov, D., Laur, S., and Vilo, J. (2013). A new way to protect privacy in large-scale genome-wide association studies. Bioinformatics *29*, 886–893. https://doi.org/10.1093/bioinformatics/btt066.

29. Hie, B., Cho, H., and Berger, B. (2018). Realizing private and practical pharmacological collaboration. Science *362*, 347–350. https://doi.org/10.1126/science.aat4807.

30. Kim, M., Song, Y., Wang, S., Yuhou, X., and Jiang, X. (2018a). Secure logistic regression based on homomorphic encryption: design and evaluation. JMIR Med. Inform. *6*, e19. https://doi.org/10.2196/medinform.8805.

31. Bonte, C., and Vercauteren, F. (2018). Privacy-preserving logistic regression training. BMC Med. Genomics *11*, 86. https://doi.org/10.1186/s12920-018-0398-y.

32. Froelicher, D., Troncoso-Pastoriza, J.R., Raisaro, J.L., Cuendet, M.A., Sousa, J.S., Cho, H., Berger, B., Fellay, J., and Hubaux, J.-P. (2021a). Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption. Nat. Commun. *12*, 5910. https://doi.org/10.1038/s41467-021-25972-y.

33. Raisaro, J.L., Troncoso-Pastoriza, J.R., Misbach, M., Sousa, J.S., Pradervand, S., Missiaglia, E., Michielin, O., Ford, B., and Hubaux, J.P. (2019). Medco: enabling secure and privacy-preserving exploration of distributed clinical and genomic data. IEEE/ACM Trans. Comput. Biol. Bioinformatics *16*, 1328–1341. https://doi.org/10.1109/TCBB.2018.2854776.

34. Sav, S., Pyrgelis, A., Troncoso-Pastoriza, J.R., Froelicher, D., Bossuat, J.P., Sousa, J.S., and Hubaux, J.P. (2021). Poseidon: privacy-preserving federated neural network learning. In Network and Distributed System Security Symposium (NDSS). https://doi.org/10.14722/ndss.2021.24119.

35. Rawat, W., and Wang, Z. (2017). Deep convolutional neural networks for image classification: a comprehensive review. Neural Comput. *29*, 1–98. https://doi.org/10.1162/NECO_a_00990.

36. Rahman, M.A., Rahman, T., Laganière, R., and Mohammed, N. (2018). Membership inference attack against differentially private deep learning model. Trans. Data Privacy *11*, 61–79. https://www.tdp.cat/issues16/tdp.a289a17.pdf.

37. Kim, J.W., Jang, B., and Yoo, H. (2018b). Privacy-preserving aggregation of personal health data streams. PLoS One *13*, 1–15. https://doi.org/10.1371/journal.pone.0207639.

38. Chen, W., Sotiraki, K., Chang, I., Kantarcioglu, M., and Popa, R.A. (2021). HOLMES: a platform for detecting malicious inputs in secure collaborative computation. https://eprint.iacr.org/2021/1517.

39. Yang, R., Au, M.H., Zhang, Z., Xu, Q., Yu, Z., and Whyte, W. (2019). Efficient Lattice-Based Zero-Knowledge Arguments with Standard Soundness: Construction and Applications, pp. 147–175. ISBN 978-3-030-26947-0. https://doi.org/10.1007/978-3-030-26948-7_6.

40. Baum, C., and Nof, A. (2020). Concretely-efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography. In Public-Key Cryptography – PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part I (Springer-Verlag), pp. 495–526, ISBN 978-3-030-45373-2.

41. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., and D'Oliveira, R.G. (2021). Advances and open problems in federated learning. Found. Trends Machine Learn. *14*, 1–210. https://doi.org/10.1561/2200000083.

42. López-Alt, A., Tromer, E., and Vaikuntanathan, V. (2012). On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing (STOC '12; Association for Computing Machinery), pp. 1219–1234. ISBN 9781450312455. https://doi.org/10.1145/2213977.2214086.

43. Shamir, A. (1979). How to share a secret. Commun. ACM *22*, 612–613. https://doi.org/10.1145/359168.359176.

44. Cheon, J.H., Kim, A., Kim, M., and Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In Springer International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). https://doi.org/10.1007/978-3-319-70694-8_15.

45. Acar, A., Aksu, H., Uluagac, A.S., and Conti, M. (2018). A survey on homomorphic encryption schemes: theory and implementation. ACM Comput. Surv. *51*, 1–35. https://doi.org/10.1145/3214303.

46. Mouchet, C., Troncoso-pastoriza, J.R., Bossuat, J.P., and Hubaux, J.P. (2021). Multiparty Homomorphic Encryption from Ring-Learning-With-Errors (PETS). https://doi.org/10.2478/popets-2021-0071.

47. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., and Lokam, S. (2018). Homomorphic encryption security standard. Tech. Rep. (Homomorphic Encryption.org).

48. Sav, S., Troncoso-Pastoriza, J.R., Pyrgelis, A., Froelicher, D., Gomes de Sá e Sousa, J.A., Bossuat, J.P., and Hubaux, J.-P. (2022). System and Method for Privacy-Preserving Distributed Training of Neural Network Models on Distributed Datasets (EPFL).

49. Froelicher, D., Troncoso-Pastoriza, J.R., Pyrgelis, A., Sav, S., Gomes de Sá e Sousa, J.A., Hubaux, J.P., and Bossuat, J.P. (2021). System and Method for Privacy-Preserving Distributed Training of Machine Learning Models on Distributed Datasets (EPFL).

Patterns, Volume *3*

Supplemental information

Privacy-preserving federated neural network learning for disease-associated cell classification

Sinem Sav, Jean-Philippe Bossuat, Juan R. Troncoso-Pastoriza, Manfred Claassen, and Jean-Pierre Hubaux

Supplementary Information

Supplementary Note 1: Cryptography Glossary

Here, we provide a summary of the cryptography terms that are frequently used throughout this paper.

Multiparty Homomorphic Encryption: a set of protocols that enable a group of parties to securely compute joint functions over their private inputs by using homomorphic encryption. Compared to LSSS-based (linear secret-sharing scheme) approaches, these protocols scale linearly with the number of parties and do not require private channels.

Ring Learning With Errors (RLWE): a computational problem based on the difficulty of solving linear equations that are perturbed by an error. The security of the cryptographic schemes used in this work is based on this problem.

Ring Degree ($\mathcal{N}$): the degree of the RLWE cyclotomic polynomial $X^{\mathcal{N}}+1$.

Packing: the act of encrypting multiple scalar values in a single ciphertext by using ciphertext slots.

Secret Key: a secret value used to decrypt a ciphertext and to generate the encryption key and evaluation keys.

Collective Public Key: a public key generated with the interaction of a set of parties and that can be used by any party to encrypt. The decryption of a ciphertext that is encrypted with the collective key requires all parties to participate in the decryption protocol.

Evaluation Keys: special public keys used during the homomorphic evaluation of a circuit (e.g., homomorphic slot rotations).

Ciphertext Slots: available space in a ciphertext to encrypt multiple values. In CKKS, the maximum number of *slots* that a ciphertext can have is half of the dimension of the ring degree, i.e. $\mathcal{N}/2$.

Slots Rotation: cyclic shift of the values encrypted in a ciphertext.

Single Instruction, Multiple Data (SIMD) Operations: the ability to carry out operations in parallel on a batch of data that is encrypted in one ciphertext by using ciphertext slots, in the context of this work.

Bootstrapping: the act of homomorphically refreshing a ciphertext to allow for further computations.

Distributed Bootstrapping: bootstrapping that requires interaction between the parties but that is less computationally expensive than its non-interactive variant.

Collective Key Switching: an interactive re-encryption of a ciphertext to a different secret key.

Supplementary Note 2: Symbols and Notations

Table S.1 summarizes the symbols and notation used throughout the paper.

Supplementary Note 3: Data Preprocessing and Parameter Selection

Our data preprocessing is similar to the one used in CellCnn[1]. To address the distributed setting, we split individual donors in the training set to N institutions. Each party then generates the multi-cell inputs similar to CellCnn (Figure 1B) by selecting c cells per sample, and z samples per class or per patient, depending on the experiment. As a result, each multi-cell input sample has a size of $c \times m$, where m is the number of markers; and the total training set per party has $o \times z$ multi-cell inputs, where o is the number of labels when the data is generated in a per class-basis. The total training set has $p \times z$ multi-cell inputs, where p is the number of patients in that institution when the data is generated in a per patient-basis.

For accuracy evaluation, we use two test datasets for each experimental setting: One is generated by multi-cell inputs of c cells and z samples drawn from test set as in training set, and one is generated with the g of cells per individual to predict the phenotype where g is the minimum of all available cells per individual in the test set. Lastly, for a fair comparison, we use the same test set generated in all settings per dataset.

For all experimental settings, we scale and standardize the marker distributions, based on the training data.

Below, we give the parameters for each experimental setting.

RRMS/NIND experiments. For RRMS and NIND experiments in Figures 4 and 5, we generate two training datasets: (i) multi-cell inputs with 100 cells were drawn for each class label to generate a dataset of 30000 samples (phenotype-based) and (ii) multi-cell inputs with 2000 cells were drawn from each patient to generate 480 samples (patient-based). We report the median accuracy in Figures 4 and 5, for patient-based multi-cell input generation in Table S.2.

The size of the test set for multi-cell inputs, is set to 10000 for the phenotype-based multi-cell generation, and to 96 for patient-based multi-cell generation setting. The test set for phenotype classification is 12 donors for all RRMS and NIND experiments.

CMV experiments. We transform the marker measurement with the inverse hyperbolic sine function with a cofactor of 5. The training and test datasets respectively comprise 14 and 6 donors.

For the CellCnn results in Table S.2, 200 cells were drawn for each class label to generate one multi-cell sample and 2000 samples are generated per label. For all distributed settings, we also generate 200 cells per sample and gradually decrease the number of samples bagged in each party for a fair comparison.

The size of the test set including multi-cell inputs is set to 4000 and the size of the test set for phenotype classification is 6 donors.

Notation	Description
P_i	i^{th} Party
X_i	Input matrix of P_i
y_i	True labels of P_i
N	Total number of parties
s	Total number of samples over parties
$L_{n\times c\times m}$	Batch of samples
$C_{m\times h}$	Convolution filters (weights)
$W_{h\times o}$	Weight matrix of dense layer
n	Number of data samples in a batch (batch size)
c	Number of cells in multi-cell input
m	Number of features (markers)
h	Number of filters
o	Number of labels (phenotype)
η	Learning rate
μ	Momentum
$\odot$	Element-wise multiplication
$\times$	Matrix or vector multiplication
$\|\|$	Concatenation
Cryptographic Parameters and Operations	
$\boldsymbol{A}^{\ell}$	Encryption of A (bold-face) at level ℓ
$\mathcal{N}$	Ring degree
R_Q	The ring $\mathbb{Z}_Q[X]/(X^{\mathcal{N}}+1)$, with $\mathcal{N}=2^d$
$\mathsf{MultImag}(\cdot)$	Multiply the slots by the imaginary unit i
$\mathsf{Conjugate}(\cdot)$	Complex conjugate of the slots
$\mathsf{Rotate}_i(\cdot)$	Rotate the slots by i to the left
$\mathsf{InnerSum}_{i,j}(\cdot)$	Sum j batches of i slots
$\mathsf{Replicate}_{i,j}(\cdot)$	Replicate batches of i slots j times

Table S. 1. Frequently Used Symbols and Notations.

AML Experiments. For the AML experiments in Table S.2, we draw 200 cells per class label to generate multi-cell samples and 1000 samples generated per label. Note that there are 3 class labels for this set of experiments. For all distributed settings, we generate 200 cells per sample and gradually decrease the number of samples, as in other experimental settings. The size of the test set including multi-cell inputs, is set to 3000 and the size of the test set for phenotype classification is 6 donors.

Machine Learning Parameters. For all accuracy experiments, we use 8 filters, average pooling, 1 local iteration per party before aggregating local gradients, identity activation after convolution, and an approximated sigmoid activation for the dense layer. For the baseline (CellCnn), we rely on their original optimizer, ADAM, and for *PriCell*, we use SGD with momentum (μ) to enable efficient training of the neural network under encryption. We vary $\mu = 0.5-0.9$ and the learning rate $\eta = 0.0001-0.01$ for the distributed setting.

For the RRMS and NIND experiments, we use a batch size of 64 for the baseline (CellCnn) and gradually decrease the batch size proportionally to the number of parties when data is distributed. For example, when the number of parties is 4, the local batch size is 16. We use the approximated sigmoid activation in $[-1,1]$ with a polynomial degree of 3 for the dense layer,

and 30 epochs.

For the CMV experiments, we use a batch size of 200 for the baseline (CellCnn) and gradually decrease the batch size proportionally to the number of parties when the data is distributed. For example, when the number of parties is 2, the local batch size is 100. We use an approximated sigmoid activation in $[-3,3]$ with a polynomial degree of 3 for the dense layer, and 20 epochs.

For all AML experiments, we use a batch size of 200 for the baseline (CellCnn) and 100 for the local batch size in the distributed setting with 2 parties. We use an approximated sigmoid activation in $[-3,3]$ with a polynomial degree of 3 for the dense layer, and 20 epochs.

Lastly, for the Local training in Figures 2, 3, and 4 or the Local row in Table S.2, we use the original CellCnn architecture, with the same baseline parameters and average the accuracy, precision, and recall over N local parties' models.

Security Parameters. Unless otherwise stated, all experiments use a cyclotomic polynomial ring of dimension $\mathcal{N} = 2^{15}$ and an initial level $L = 10$, which provides 2^{14} slots per ciphertext and allows for a depth-10 circuit before bootstrapping is needed (10 operations to be carried out before bootstrapping). For the inference times given in Table 1 (main text), we start with an initial level of 4 as we do not need the backpropagation. This enables a more efficient forward pass as operations carried out on a ciphertext with a lower level are less expensive. All our cryptographic parameters ensure at least 128-bit security level during the training and up to 256-bit security during the inference.

Supplementary Note 4: Detailed Neural Network Circuit

Notations. We denote a batch of samples, the convolution layer and the dense layer matrices by $L_{n\times c\times m}$, $\boldsymbol{C}_{m\times h}$ and $\boldsymbol{W}_{h\times o}$, respectively, with n the number of samples, c the number of cells per sample per batch, m the number of features (markers), h the number of filters and o the number of output classes (labels). When there is no ambiguity, we eliminate the sub-index of the matrices, e.g., $\boldsymbol{C}_{m\times h}$ is often referred to as $\boldsymbol{C}$. We recall that encrypted matrices are denoted in boldface. We denote the plaintext of binary values as *mask*. When multiplied with a ciphertext, *mask* selects specific slots of the ciphertext by setting the other slots to zero. The terms *row*, *column* and *diagonal* encoding of a matrix denote the mapping of a 2D matrix on a 1D vector by concatenating each row, each column or each diagonal of the matrix respectively.

Convolution With Pre-Pooling. Given a hyper-cube batch of samples $L_{n\times c\times m}$, we first preprocess L by applying the average pooling across the cells. As the convolution, average pooling and the activation of this step are all linear transformations, their order is interchangeable. This preprocessing reduces the size of the hyper-cube from $n\times c\times m$ to only $n\times m$, thus removing its dependency on c. The convolution is computed with a single matrix multiplication $\boldsymbol{P}_{n\times h} = L_{n\times m}\times \boldsymbol{C}_{m\times h}$, with a row-encoded $\boldsymbol{P}_{n\times h}$ matrix where each row stores the result of the convolution layer for one sample. In the rest of this section, we describe how we pack $L_{n\times m}$ and $\boldsymbol{C}_{m\times h}$ in order to enable an efficient convolution through SIMD operations.

We evaluate the convolution with a diagonally-encoded plaintext and row-encoded ciphertext matrix multiplication. As we operate with non-square matrices, we pad the matrix $\boldsymbol{C}$ with the copies of itself until its number of rows reaches n. As such, the result will yield n rows, each of h values. With this approach, the convolution can be evaluated with only m plaintext-ciphertext multiplications and additions, and $m-1$ rotations. If $m\times n$ is not a power of two, cyclic rotations of the ciphertext slots will not result in a cyclic rotation of the flattened matrix. Instead, it requires using the *masking* and rotations, which consumes a level. To overcome this, we pad the flattened matrix with additional copies of itself until it reaches a total of $n+(m-1)$ rows (hence the final size of the flattened matrix is $h(n+m-1)$). This enables us, at the expense of more slots used, to simulate a cyclic rotation. Note that those extra rows are removed by the plaintext multiplication by L that also acts as a masking.

We further reduce the number of operations by making use of complex arithmetic, which is natively provided by the CKKS scheme. Using the following, we compute the dot product of $\langle(a_0,a_1),(b_0,b_1)\rangle$ in a single multiplication:

$$(a_0 - ia_1)\cdot(b_0+ib_1) = (a_0b_0+a_1b_1)+i(a_0b_1-a_1b_0).$$

Hence, the convolution of two half-sized complex matrices $L'_{n,m/2}\times B'_{m/2,h}$ is sufficient to compute the convolution of the real matrices $L_{n\times m}\times \boldsymbol{C}_{m\times h}$:

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \dots & a_{n,m} \end{pmatrix} \times \begin{pmatrix} b_{1,1} & \dots & b_{1,h} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \dots & b_{m,h} \end{pmatrix} \rightarrow \begin{pmatrix} a_{1,1}-ia_{1,2} & \dots & a_{1,m-1}-ia_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1}-ia_{n,2} & \dots & a_{n,m-1}-ia_{n,m} \end{pmatrix} \times \begin{pmatrix} b_{1,1}+ib_{2,1} & \dots & b_{1,m}+ib_{2,m} \\ \vdots & \ddots & \vdots \\ b_{n-1,1}+ib_{n,1} & \dots & b_{n-1,m-1}+ib_{n,m} \end{pmatrix}.$$

The extraction of the real part can be done with complex conjugation and addition. The mapping from $\boldsymbol{C}_{m,h}$ to $\boldsymbol{C}'_{m/2,h}$ is straightforward and can be homomorphically computed with $\boldsymbol{C}' = \boldsymbol{C} + \mathsf{Rotate}_h(\mathsf{MultImag}(\boldsymbol{C}))$. Note that it requires $\boldsymbol{C}$ to be padded with an additional row. The encoding of the plaintext matrix $L_{n,m}$ is done by encoding each diagonal of $L'_{n,m/2}$ in a separate plaintext.

The matrix multiplication $L' \times \boldsymbol{C}'$ is then done with

$$\boldsymbol{P}'_{n\times h} = \sum_{i=0}^{m/2-1} L'_{n\times m/2} \odot \mathsf{Rotate}_{2hi}(\boldsymbol{C}'_{m/2\times h}).$$

The result is a row-encoded $n \times h$ complex matrix. We remove its imaginary part $\boldsymbol{P} = \frac{1}{2}(\boldsymbol{P'} + \mathsf{Conjugate}(\boldsymbol{P'}))$ with the $\frac{1}{2}$ factor being pre-applied to L'. Because the number of rotations is reduced by a factor of two, the number of rows for the padding must also be readjusted:

$$\underbrace{n}_{\text{result}} + \underbrace{(\lceil m/2 \rceil - 1) \cdot 2}_{\text{rotations}} + \underbrace{1}_{i \text{ repacking}},$$

and the total number of slots used to encode $\boldsymbol{C}_{m\times h}$ is $nh + (\lceil m/2 \rceil - 1)2h + h$. We give an overview of how $\boldsymbol{C}_{n\times h}$ and $\boldsymbol{P}_{n\times h}$ are each encoded on a vector:

$$\boldsymbol{C}_{m\times h} = (\underbrace{\boldsymbol{C}_{(1,1)}, \ldots, \boldsymbol{C}_{(1,h)}, \boldsymbol{C}_{(2,1)}, \ldots, \boldsymbol{C}_{(2,h)}, \ldots, \boldsymbol{C}_{m,1}, \ldots, \boldsymbol{C}_{(m,h)}, \boldsymbol{C}_{(1,1)}, \ldots}_{nh+(m/2-1)2h+h}, 0, \ldots, 0),$$

$$\boldsymbol{P}_{n\times h} = \big(\boldsymbol{P}_{(1,1)}, \ldots, \boldsymbol{P}_{(1,h)}, \ldots, \boldsymbol{P}_{(n,1)}, \ldots, \boldsymbol{P}_{(n,h)}, 0, \ldots, 0\big).$$

Dense Layer. The input to the dense layer is a row-encoded $\boldsymbol{P}_{n\times h}$ matrix that is multiplied with the $\boldsymbol{W}_{h\times o}$ matrix. As the matrix $\boldsymbol{P}_{n\times h}$ is row-encoded and requires a homomorphic extraction of its diagonals, the technique used in the convolution step becomes costly for the dense layer. Instead, we use the multiply-then-inner-sum approach, as in POSEIDON[2]. The values of $\boldsymbol{W}_{h\times o}$ are grouped by samples. We first preprocess $\boldsymbol{P}$ by duplicating it o times for each label. This duplication is done with $\log_2(o) + \mathsf{hw}(o) - 1$ rotations. The matrix $\boldsymbol{W}_{h\times o}$ is column-encoded (row-encoding of its transpose), with each of its columns replicated n times (for each sample):

$$\boldsymbol{W}_{h\times o} = \big(\underbrace{(\boldsymbol{W}_{(1,1)}, \ldots, \boldsymbol{W}_{(h,1)}), \ldots, (\boldsymbol{W}_{(1,1)}, \ldots, \boldsymbol{W}_{(h,1)})}_{n\times h}, \ldots, (\boldsymbol{W}_{(1,o)}, \ldots, \boldsymbol{W}_{(h,o)}), \ldots, (\boldsymbol{W}_{(1,o)}, \ldots, \boldsymbol{W}_{(h,o)}), 0, \ldots, 0\big).$$

The multiplication $\boldsymbol{U}_{n\times o} = \boldsymbol{P}_{n\times h} \times \boldsymbol{W}_{h\times o}$ is carried on with a single ciphertext-ciphertext multiplication, followed by an inner-sum of batch n and h ($\log_2(h) + \mathsf{hw}(h) - 1$ rotations). The resulting vector has a size of onh:

$$\boldsymbol{U}_{n\times o} = \big(\underbrace{(\boldsymbol{U}_{(1,1)}, \times, \ldots, \times), \ldots, (\boldsymbol{U}_{(n,1)}, \times, \ldots, \times)}_{n\times h}, \ldots, (\boldsymbol{U}_{(1,o)}, \times, \ldots, \times), \ldots, (\boldsymbol{U}_{(n,o)}, \times, \ldots, \times), 0, \ldots, 0, \underbrace{(\times, \ldots, \times)}_{h-1}\big),$$

with $\times$ denoting unusable by-product values in the ciphertext slots.

Repacking for Bootstrapping. We repack the following elements in a single ciphertext for the optimized bootstrapping:

- $\boldsymbol{U}_{n\times o}$: the result of the dense layer, which uses $onh + h - 1$ slots.
- $\boldsymbol{P}_{n\times h}$: the result of the convolution layer, which uses nh slots.
- $\boldsymbol{W}_{h\times o}$: the dense layer matrix, which uses onh slots.
- $\boldsymbol{\nabla W}^{\mathsf{prev}}_{h\times o}$: the updated dense layer weights of the previous backward pass, which uses onh slots.
- $\boldsymbol{\nabla C}^{\mathsf{prev}}_{m\times h}$: the updated convolution layer weights of the previous backward pass, which uses size $nh + (\lceil m/2 \rceil - 1) \cdot 2h + h$ slots.

The repacking is done solely with additions and rotations, concatenating the empty slots of $\boldsymbol{U}_{n\times o}$:

$$\boldsymbol{D}_{\mathsf{repack}} = \boldsymbol{U}_{n\times o} + \mathsf{Rotate}_{-onh}(\boldsymbol{P}_{n\times h}) + \mathsf{Rotate}_{-2onh}(\boldsymbol{W}_{h\times o}) + \mathsf{Rotate}_{-3onh}(\boldsymbol{\nabla W}^{\mathsf{prev}}_{h\times o}) + \mathsf{Rotate}_{-4onh}(\boldsymbol{\nabla C}^{\mathsf{prev}}_{m\times h}).$$

Bootstrapping and Repacking for Backward Pass. The goal of this step is to refresh the ciphertext $\boldsymbol{D}_{\mathsf{repack}}$ to a higher level, to enable more computation and to re-arrange its slots optimally for the backward pass.

- $\boldsymbol{U}_{n\times o}$: We re-order the slots to arrange them first by samples then by classes, and we duplicate each value h times (replacing the non-zero by-product slots):

$$\boldsymbol{U}_{\mathsf{backW}} = \big((\underbrace{\boldsymbol{U}_{(0,0)},\ldots,\boldsymbol{U}_{(0,0)},\boldsymbol{U}_{(0,1)},\ldots,\boldsymbol{U}_{(0,1)}}_{2h}),\ldots,(\boldsymbol{U}_{(n-1,0)},\ldots,\boldsymbol{U}_{(n-1,0)},\boldsymbol{U}_{(n-1,1)},\ldots,\boldsymbol{U}_{(n-1,1)})\big).$$

We note that the size of this vector remains onh. $\boldsymbol{U}_{\mathsf{backW}}$ will be used to compute the dense layer error for the updated dense layer weights. We pack an additional copy of $\boldsymbol{U}$, $\boldsymbol{U}_{\mathsf{backC}}$, which is pre-formatted for the convolution layer error and clustered by sample. By computing twice the same values in parallel, but packed in two different ways (one for the dense layer, one for the convolution layer), we avoid expensive and level-consuming repacking procedures, at the cost of more slot usage. Hence for each label, we pad the nh values with $(m/2-1)2h+h$ additional copies of the relevant rows. The used size is therefore $(nh+(m/2-1)2h+h)o$.

$$\boldsymbol{U}_{\mathsf{backC}} = \Big(\underbrace{(\underbrace{\boldsymbol{U}_{(1,1)},\ldots,\boldsymbol{U}_{(1,1)}}_{h},\ldots,\boldsymbol{U}_{(n,1)},\ldots,\boldsymbol{U}_{(n-1,0)},\boldsymbol{U}_{(1,1),\ldots})}_{nh+(m/2-1)2h+h},\ldots,\big(\boldsymbol{U}_{(1,o)},\ldots,\boldsymbol{U}_{(1,o)},\ldots,(\boldsymbol{U}_{(n,o)},\ldots,\boldsymbol{U}_{(n,o)},\boldsymbol{U}_{(1,o)},\ldots)\big)\Big).$$

- $\boldsymbol{P}_{n\times h}$: The result of the convolution layer, which is an $n\times h$ row-encoded matrix, is re-arranged by duplicating each of its rows for each class of the dense layer (2 in this example) and multiplied by the learning rate (η).

$$\boldsymbol{P}_{\mathsf{back}} = \eta\big((\underbrace{\boldsymbol{P}_{(1,1)},\ldots,\boldsymbol{P}_{(1,h)},\boldsymbol{P}_{(1,1)},\ldots,\boldsymbol{P}_{(1,h)}}_{2h}),\ldots,(\boldsymbol{P}_{(n,1)},\ldots,\boldsymbol{P}_{(n,h)},\boldsymbol{P}_{(n,1)},\ldots,\boldsymbol{P}_{(n,h)})\big).$$

- $\boldsymbol{W}_{h\times o}$: The dense layer matrix, which is an $h\times o$ column-encoded matrix, is re-arranged by padding each column with itself such that each column has a size of $nh+(m/2-1)2h+h$, for a total size of $o(n\times h+(m/2-1)2h+h)$ and is multiplied by the learning rate (η).

$$\boldsymbol{W}_{\mathsf{back}} = \eta\big((\underbrace{\boldsymbol{W}_{(1,1)},\ldots,\boldsymbol{W}_{(h,1)},\boldsymbol{W}_{(1,1)},\ldots}_{nh+(m/2-1)2h+h}),\ldots,(\boldsymbol{W}_{(1,o)},\ldots,\boldsymbol{W}_{(h,o)},\boldsymbol{W}_{(1,o)},\ldots)\big).$$

- $\nabla\boldsymbol{W}^{\mathsf{prev}}_{h\times o}$: The previous dense layer updated weights, of size nho. The format is preserved (column encoded matrix of size ho with each column replicated n times), but the values are multiplied by the momentum (μ).

$$\nabla\boldsymbol{W}_{\mathsf{back}} = \mu\big(\underbrace{(\nabla\boldsymbol{W}_{(1,1)},\ldots,\nabla\boldsymbol{W}_{(h,1)}),\ldots,(\nabla\boldsymbol{W}_{(1,1)},\ldots,\nabla\boldsymbol{W}_{(h,1)})}_{n\times h},\ldots,(\nabla\boldsymbol{W}_{(1,o)},\ldots,\nabla\boldsymbol{W}_{(h,o)}),\ldots,(\nabla\boldsymbol{W}_{(1,o)},\ldots,\nabla\boldsymbol{W}_{(h,o)})\big).$$

- $\nabla\boldsymbol{C}^{\mathsf{prev}}_{m\times h}$: The previous convolution layer updated weights, of size $n\times h+(m/2-1)2h+h$. The format is preserved (row encoded matrix, padded), but the values are multiplied by the momentum (μ).

$$\nabla\boldsymbol{C}_{\mathsf{back}} = \mu(\underbrace{\nabla\boldsymbol{C}_{(1,1)},\nabla\boldsymbol{C}_{(1,2)},\ldots,\nabla\boldsymbol{C}_{(1,h)},\nabla\boldsymbol{C}_{(2,1)},\ldots,\nabla\boldsymbol{C}_{(2,h)},\ldots,\nabla\boldsymbol{C}_{(m,h)},\nabla\boldsymbol{C}_{(1,1)},\ldots}_{nh+(m/2-1)2h+h}).$$

In summary, the bootstrapped ciphertext contains the following elements:

$$\boldsymbol{D}_{\mathsf{boot}} = \underbrace{U_{\mathsf{backW}}}_{onh} \;||\; \underbrace{U_{\mathsf{backC}}}_{o(nh+(m/2-1)2h+h)} \;||\; \underbrace{P_{\mathsf{back}}}_{onh} \;||\; \underbrace{W_{\mathsf{back}}}_{o(nh+(m/2-1)2h+h)} \;||\; \underbrace{\nabla W_{\mathsf{back}}}_{onh} \;||\; \underbrace{\nabla C_{\mathsf{back}}}_{nh+(m/2-1)2h+h},$$

and the total number of slots used in the ciphertext must respect

$$3onh + (2o+1)(nh + (m/2-1)2h + h) \leq \mathcal{N}/2$$

for the ring degree $\mathcal{N}$. Therefore, a bootstrapped ciphertext can hold up to $n = \lfloor (N/(2h) - (2o+1)(m-1))/(5o+1) \rfloor$ samples. For example, given $N = 2^{15}$, $m = 38$, $h = 8$ and $o = 2$, the ciphertext holds 169 samples. This number is smaller than the number of samples that can be repacked in a single ciphertext before the bootstrapping, hence it sets an upper bound for the number of samples that can be trained in a single batch.

Backward Pass. The backward pass is computed using the ciphertext $\boldsymbol{D}_{\mathsf{boot}}$. The different values contained in $\boldsymbol{D}_{\mathsf{boot}}$ are accessed via rotations and ciphertext duplication. Masking is used only at the very end to minimize the use of levels. We start by computing the error of the dense layer formatted for the dense layer update ($\boldsymbol{E}_1$) and formatted for the convolution layer ($\boldsymbol{E}'_1$) at the same time:

$$(\boldsymbol{E}_1 || \boldsymbol{E}'_1) = \sigma'(\boldsymbol{U}_{\mathsf{backW}} || \boldsymbol{U}_{\mathsf{backC}}) \odot \big(\sigma(\boldsymbol{U}_{\mathsf{backW}} || \boldsymbol{U}_{\mathsf{backC}}) - (Y_{\mathsf{backW}} || Y_{\mathsf{backC}})\big),$$

with $Y_{\mathsf{backW}} || Y_{\mathsf{backC}}$, the plaintext labels, accordingly encoded and formatted. We then compute in parallel the updated weights of each sample of the dense layer and the partial error of the convolution layer by multiplying $\boldsymbol{E}_1 || \boldsymbol{E}'_1$ with $\boldsymbol{P}_{\mathsf{back}} || \boldsymbol{W}_{\mathsf{back}}$. Note that $\boldsymbol{P}_{\mathsf{back}} || \boldsymbol{W}_{\mathsf{back}}$ can be accessed and aligned with a rotation on $\boldsymbol{D}_{\mathsf{boot}}$.

$$\boldsymbol{\nabla W} || \boldsymbol{E}_0 = (\boldsymbol{E}_1 || \boldsymbol{E}'_1) \odot (\boldsymbol{P}_{\mathsf{back}} || \boldsymbol{W}_{\mathsf{back}}).$$

$\boldsymbol{\nabla W}$ is clustered by samples, hence we add a summation across the n samples to obtain the updated dense layer weights of the batch. The output contains only a single copy, column-encoded, of $\boldsymbol{\nabla W}$, and of size oh. An additional step first adds, then masks and extracts, each column of the result and replicates them n times to expand its size back to onh and to match the original encoding format of W (this masking also removes all the unwanted by-product values). $\boldsymbol{\nabla W}_{\mathsf{back}}$ is added to the result to get the final updated weights of the dense layer.

We finalize the computation of E_0 by a summation across the labels, reducing its size to $nh + (m/2-1)2h + h)$. E_0 is already formatted to be multiplied with the plaintext transposed sample matrix $\eta \cdot L^T$ (pre-pooled and multiplied by η). This step is same as the convolution layer matrix multiplication:

$$\boldsymbol{\nabla C} = \eta \cdot L^T \times \boldsymbol{E}_0.$$

The result is of size nh, with no by-product garbage slots due to the plaintext multiplication, but it needs to be extended to a size of $nh + (m/2-1)2h + h$ to comply with the formatting of $\boldsymbol{C}$. This is done by replicating the nh slots until it reaches at least this amount of slots and by masking the overflow of slots. Similarly, $\boldsymbol{\nabla C}_{\mathsf{back}}$ is added to $\boldsymbol{\nabla C}$, and the result is stored as the newly updated weights for the next batch of samples.

We summarize the given steps in Algorithm 1. Note that the algorithm describes the local computations. Then, the parties collectively aggregate and update the global model, which includes the additional step of taking the mean of $\boldsymbol{\nabla W}$ and $\boldsymbol{\nabla C}$ across all the parties.

Algorithm 1: The *local computation* algorithm for encrypted CellCnn training. The exponent of encrypted values (e.g., y for $\boldsymbol{C}^y$) denotes the current ciphertext level. Encryption, encoding, and detailed steps of the repacking during the bootstrapping are omitted for clarity and are described in the Experimental Procedures section. The value csize represents $nh + (\lceil m/2 \rceil - 1)2h + h$. We give the function definitions in Table S.1.

Input: X and Y set of samples and labels, learning rate η, momentum μ, batch size n, number of iterations d, number of features m, number of filters h, number of labels o, $maskW$ a masking vector containing ones in the first onh slots, $maskW_i$ a set of o masking vectors containing ones in the slots inh to $(i+1)nh$ slots for $0 < i < o$, and $maskC$ a masking vector containing ones in the first csize slots.

Output: The encrypted weights $\boldsymbol{C}$ and $\boldsymbol{W}$.

$\boldsymbol{C}^4 \leftarrow \mathsf{Init}(m,h), \boldsymbol{W}^3 \leftarrow \mathsf{Init}(h,o)$ // Initialize convolution and dense weights

$\nabla W^5_{\mathsf{prev}}, \nabla C^4_{\mathsf{prev}} \leftarrow 0$ // Initialize previous updated weights

for $i = 0; i < d; i = i + 1$ **do**

Batch Selection

$X_{\mathsf{batch}} \leftarrow \mathsf{Select}_n(X)$ // Select a batch of random samples

$Y_{\mathsf{batch}} \leftarrow \mathsf{Select}_n(Y)$ // Select the corresponding labels

$L_{\mathsf{pool}} \leftarrow \mathsf{Pre\text{-}pooling}(X_{\mathsf{batch}})$ // Apply the pre-pooling to the batch

Forward Pass

$\boldsymbol{C}^4_{\mathsf{tmp}} \leftarrow \boldsymbol{C}^4 + \mathsf{Rotate}_h(\mathsf{MultImag}(\boldsymbol{C}^4))$ // Preprocessing for complex matrix multiplication

$\boldsymbol{P}^3 \leftarrow \sum_{i=0}^{\lceil m/2 \rceil - 1} L^{\mathsf{diag}[i]}_{\mathsf{pool}} \odot \mathsf{Rotate}_{2hi}(\boldsymbol{C}^4_{\mathsf{tmp}})$ // Convolution

$\boldsymbol{P}^3 \leftarrow \mathsf{Replicate}_{nh,o}(\boldsymbol{P}^3)$ // Replicate the result for each label

$\boldsymbol{U}^2 \leftarrow \mathsf{InnerSum}_{1,h}(\boldsymbol{P}^3 \odot \boldsymbol{W}^3)$ // Dense layer

Bootstrapping

$\boldsymbol{D}^2_{\mathsf{repack}} = \boldsymbol{U}^2 + \mathsf{Rotate}_{-nho}(\boldsymbol{P}^3) + \mathsf{Rotate}_{-2nho}(\boldsymbol{W}^3) + \mathsf{Rotate}_{-3nho}(\nabla \boldsymbol{W}^5_{\mathsf{prev}}) + \mathsf{Rotate}_{-4nho}(\nabla \boldsymbol{C}^4_{\mathsf{prev}})$ // Pack all necessary values in a single ciphertext

$\boldsymbol{D}^9_{\mathsf{boot}} \leftarrow \mathsf{Bootstrapp}_{\eta,\mu}(\boldsymbol{D}^2_{\mathsf{repack}})$ // Refresh the ciphertext and formatting for the backward pass

Backward Pass

$\boldsymbol{U}\boldsymbol{1}^7 \leftarrow \sigma(\boldsymbol{D}^9_{\mathsf{boot}})$ // Activation

$\boldsymbol{U}\boldsymbol{2}^7 \leftarrow \sigma'(\boldsymbol{D}^9_{\mathsf{boot}})$ // Activation derivative

$\boldsymbol{E}\boldsymbol{1}^6 \leftarrow \boldsymbol{U}\boldsymbol{2}^7 \odot (\boldsymbol{U}\boldsymbol{1}^7 - Y_{\mathsf{batch}})$ // Dense layer error

$\boldsymbol{P}^9 \leftarrow \mathsf{Rotate}_{nho+o\cdot\mathsf{csize}}(\boldsymbol{D}^9_{\mathsf{boot}})$ // Access pooling result and dense layer weights

$\nabla \boldsymbol{W}^5 \leftarrow \boldsymbol{P}^9 \odot \boldsymbol{E}\boldsymbol{1}^6$ // Dense layer updated weights and convolution layer error

$\boldsymbol{E}\boldsymbol{0}^5 \leftarrow \mathsf{Rotate}_{nho}(\nabla \boldsymbol{W}^5)$ // Access convolution layer error

$\nabla \boldsymbol{W}^5 \leftarrow \mathsf{InnerSum}_{oh,n}(\nabla \boldsymbol{W}^5)$ // Finish updated weights with summation across the samples

$\boldsymbol{E}\boldsymbol{0}^5 \leftarrow \mathsf{InnerSum}_{\mathsf{csize},o}(\boldsymbol{E}\boldsymbol{0}^5)$ // Finish E1 with summation across the labels

$\nabla \boldsymbol{C}^4 \leftarrow \sum_{i=0}^{\lceil n/2 \rceil - 1} (0.5 \cdot L^{T,\mathsf{diag}[i]}_{\mathsf{pool}}) \odot \mathsf{Rotate}_{2mi}(\boldsymbol{E}\boldsymbol{0}^5)$ // Multiply with the transposed samples

$\nabla \boldsymbol{C}^4 \leftarrow \nabla \boldsymbol{C}^4 + \mathsf{Conjugate}(\nabla \boldsymbol{C}^4)$ // Clean imaginary part

$\nabla \boldsymbol{C}^4 \leftarrow \mathsf{Replicate}_{mh,\lceil \mathsf{csize}/mh \rceil}(\nabla \boldsymbol{C}^4)$ // Format updated weights for convolution layer

$\nabla \boldsymbol{W}^5 \leftarrow \mathsf{Replicate}_{h,n}(\sum_{i=0}^{o-1} \mathsf{Rotate}_{-inh}(maskW_i \odot \nabla \boldsymbol{W}^5))$ // Format updated weights for dense layer

$\nabla \boldsymbol{W}^8_{\mathsf{prev}} \leftarrow maskW \odot \mathsf{Rotate}_{2nho+2o\cdot\mathsf{csize}}(\boldsymbol{D}^9_{\mathsf{boot}})$ // Access and extract the previous updated weights

$\nabla \boldsymbol{C}^8_{\mathsf{prev}} \leftarrow maskC \odot \mathsf{Rotate}_{3nho+2o\cdot\mathsf{csize}}(\boldsymbol{D}^9_{\mathsf{boot}})$ // Access and extract the previous updated weights

Weights Update

$\nabla \boldsymbol{W}^5 \leftarrow \nabla \boldsymbol{W}^5 + \nabla \boldsymbol{W}^8_{\mathsf{prev}}$ // Add previous updated weights with momentum

$\nabla \boldsymbol{C}^4 \leftarrow \nabla \boldsymbol{C}^4 + \nabla \boldsymbol{C}^8_{\mathsf{prev}}$ // Add previous updated weights with momentum

$\boldsymbol{C}^4 \leftarrow \boldsymbol{C}^4 - \nabla \boldsymbol{C}^4$ // Update the weights

$\boldsymbol{W}^3 \leftarrow \boldsymbol{W}^3 - \nabla \boldsymbol{W}^5$ // Update the weights

$\nabla \boldsymbol{C}^4_{\mathsf{prev}} \leftarrow \nabla \boldsymbol{C}^4$ // Store the new updated weights

$\nabla \boldsymbol{W}^5_{\mathsf{prev}} \leftarrow \nabla \boldsymbol{W}^5$ // Store the new updated weights

end

return

Supplementary Note 5: Summary of Experiments

In Table S.2, we show the median accuracy, precision, recall, and F-score values of 10 runs for RRMS and NIND experiments with patient-based sub-sampling shown in Figures 3C, 3D and Figures 4C, 4D, and CMV experiments for phenotype-based sub-sampling shown in Figure 2. We also report these metrics for the AML classification for the centralized and two-party *PriCell* settings.

We note that for 3-class classification, i.e., AML, we rely on macro-averaging on metrics, and we calculate the F-score in Table S.2 over the averaged precision and recall for the Local-training experimental setting.

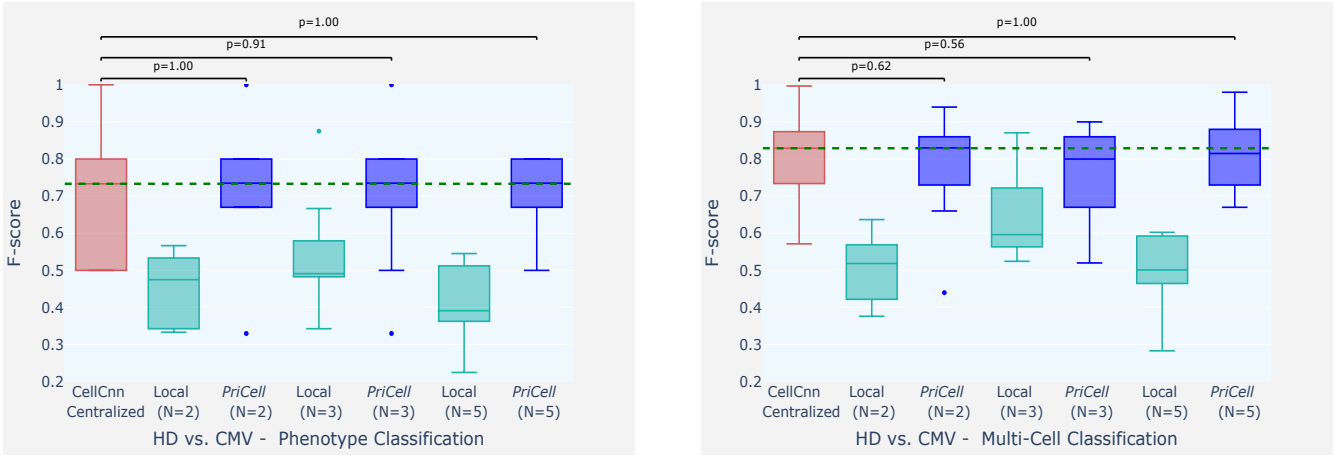
Our results show that *PriCell* achieves an accuracy comparable to the centralized and non-private solutions. The accuracy achieved by *PriCell* remains almost the same as the centralized one, and the slight decrease in phenotype classification in NIND classification is due to the limited number of samples in the test set for this task, i.e., there are only 12 patients in the NIND phenotype test set, which results in accuracy decrease of 4% in the median value when the trained model misses only one patient classification.

We further show the F-score distributions over the experiments in Figure S.1, S.2, and S.3. We perform a Wilcoxon signed-rank test to assess that the CellCnn and *PriCell* paired F-score results come from the same distribution. As all p-values are greater than 0.05, these figures support our interpretation that *PriCell* achieves a classification performance that is comparable to the centralized and non-private solution.

Lastly, we note that the differences in the precision and recall values are due to the nature of the preprocessing and training mechanism: the random selection of multi-cell inputs generates higher or lower precision and recall values depending on the eventual selection, even in the centralized and no privacy-protection solution.

Setting/Metrics	Accuracy	Precision	Recall	F-score
RRMS (multi-cell / phenotype classification)				
CellCnn	0.65 / 0.67	0.62 / 0.71	0.61 / 0.71	0.66 / 0.71
Local (N=2)	0.59 / 0.62	0.59 / 0.68	0.62 / 0.64	0.59 / 0.66
PriCell (N=2)	0.65 / 0.67	0.66 / 0.75	0.62 / 0.64	0.64 / 0.67
Local (N=4)	0.55 / 0.55	0.57 / 0.61	0.60 / 0.61	0.58 / 0.62
PriCell (N=4)	0.64 / 0.67	0.63 / 0.73	0.61 / 0.64	0.61 / 0.69
Local (N=6)	0.53 / 0.52	0.53 / 0.58	0.55 / 0.54	0.53 / 0.57
PriCell (N=6)	0.64 / 0.67	0.68 / 0.80	0.61 / 0.64	0.63 / 0.69
NIND (multi-cell / phenotype classification)				
CellCnn	0.72 / 0.75	0.82 / 0.83	0.75 / 0.71	0.76 / 0.77
Local (N=2)	0.57 / 0.58	0.65 / 0.65	0.59 / 0.52	0.62 / 0.63
PriCell (N=2)	0.72 / 0.75	0.73 / 0.75	0.80 / 0.86	0.78 / 0.80
Local (N=4)	0.55 / 0.55	0.66 / 0.68	0.51 / 0.50	0.59 / 0.58
PriCell (N=4)	0.72 / 0.71	0.75 / 0.73	0.84 / 0.71	0.78 / 0.75
Local (N=6)	0.53 / 0.52	0.63 / 0.64	0.57 / 0.56	0.59 / 0.57
PriCell (N=6)	0.71 / 0.71	0.73 / 0.75	0.76 / 0.71	0.75 / 0.75
CMV (multi-cell / phenotype classification)				
CellCnn	0.80 / 0.75	0.72 / 0.58	0.98 / 1.00	0.83 / 0.73
Local (N=2)	0.54 / 0.58	0.52 / 0.42	0.50 / 0.50	0.52 / 0.47
PriCell (N=2)	0.79 / 0.75	0.71 / 0.58	0.98 / 1.00	0.83 / 0.73
Local (N=3)	0.59 / 0.55	0.56 / 0.40	0.64 / 0.67	0.60 / 0.49
PriCell (N=3)	0.79 / 0.75	0.76 / 0.58	0.84 / 1.00	0.80 / 0.73
Local (N=5)	0.50 / 0.52	0.44 / 0.31	0.57 / 0.55	0.50 / 0.39
PriCell (N=5)	0.78 / 0.75	0.69 / 0.58	0.97 / 1.00	0.82 / 0.73
AML (multi-cell / phenotype classification)				
CellCnn	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00
Local (N=2)	0.98 / 1.00	0.98 / 1.00	0.98 / 1.00	0.98 / 1.00
PriCell (N=2)	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00

Table S. 2. Classification performance (accuracy, precision, recall, and F-score) of the models obtained with original CellCnn, local training without collaboration, and *PriCell* for RRMS, NIND, CMV, and AML classification tasks. All models are tested on two datasets for multi-cell and phenotype classification respectively, separated with '/'.



(a) **(b)**

Figure S. 1. F-score boxplots when classifying healthy donor (HD) vs. cytomegalovirus infection (CMV) for training multi-cells drawn from the bag of all cells per class. Experiments are repeated 10 times with different train and test set splits; the vertical dashed line illustrates the median for the baseline (CellCnn) and the dots represent the outliers. The p-values shown at the top of the figure are calculated with a Wilcoxon signed-rank test for the comparison between the corresponding boxplots ($p > 0.05$ indicates that the distributions are not significantly different). F-score is reported for two datasets: **(a)** phenotype classification of 6 patients and **(b)** multi-cell input classification on 4000 samples.

Supplementary Note 6: Downstream Analysis

The original CellCnn [1] study aims at detecting the rare disease-associated cell subsets via learned filter weights. The final filter weights are used to select phenotype-associated cell subsets via a filter response, i.e., the weighted sum of the abundance profile for each cell. As the cell subset selected by a filter can contain more than one cell type, the authors perform a density-based clustering of the group of cells with high cell-filter responses.

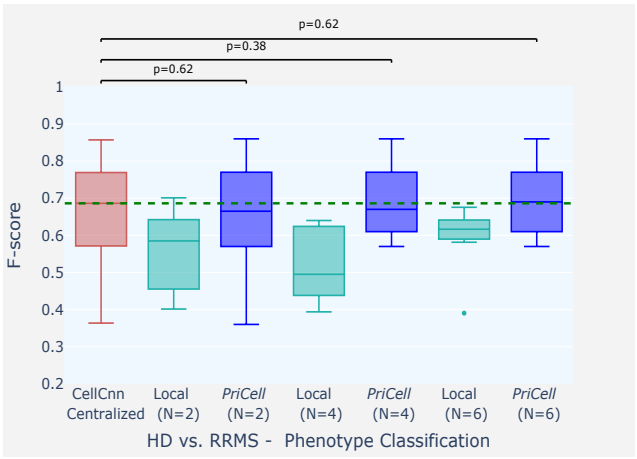
We perform an analogous analysis to evaluate the effect of our introduced changes in the original neural network architecture, namely the average pooling, the approximated activation functions, and the optimizer. We introduce these changes in CellCnn's original implementation, simulate our encryption on centralized data, and conduct further analysis by using their downstream analysis [1]. We use the CMV infection dataset with $c = 200$ cells per multi-cell input and $z = 1000$ samples per phenotype to generate the training dataset. The test set is generated as explained in the Data Preprocessing and Parameter Selection section. We train 20 models for CellCnn and 20 models for *PriCell* simulation and take the best 3 models for each approach based on the validation accuracy, as in the original work [1]. In all model training, we use 20 epochs with early-stopping and varying numbers of filters in each model training.

In Figure S.4, we show the consensus filters, i.e., one representative filter per class (phenotype) that has minimum distance to all other members of the hierarchically clustered filters, based on a threshold of 0.2, found by CellCnn and *PriCell* simulation, respectively. In both Figure S.4a and S.4b, we observe that the filter which is positively associated (second filter) with previous CMV infection gives more weights to the CD16, CD57, NKG2C, and CD94 markers. We note that while the trend of consensus filters is similar, the distribution of the consensus filters, i.e., the final filter weights and the scale of the values, differs between CellCnn and *PriCell*. This is due to our approximated activation function that affects the final values of the filter weights but does not affect the interpretation from the consensus filters. Similar results were found for the repetition of these experiments, which suggests that, as in the original work [1], our encrypted model is able to find natural killer (NK) cell populations associated with prior CMV infection.

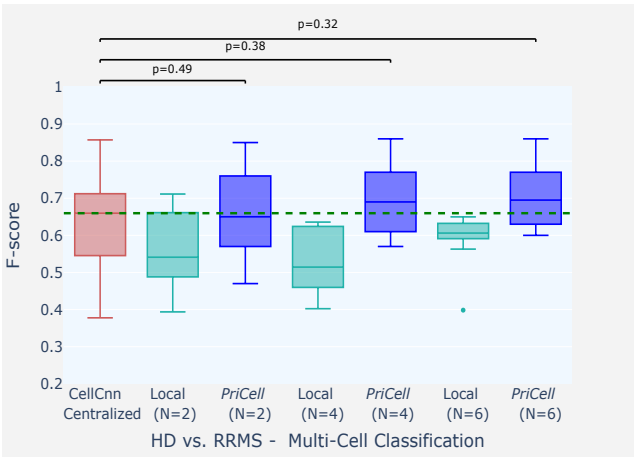
In Figure S.5, we show the boxplot of the selected cell population frequencies from the test samples of the CMV- and CMV+ classes by using the positively associated filter. Although CellCnn has higher discriminative frequencies, *PriCell* simulation is able to select CMV+ cell populations with the positively associated filter.

Finally, we show in Figure S.6 the marker expression profiles for all cells vs. cell population selected by the positively associated filter learned by CellCnn training (Figure S.6a) and by *PriCell* simulation(Figure S.6b). In both CellCnn and *PriCell* training, we again observe that the positively associated filter weighs CD16, CD57, NKG2C, and CD94 markers more than the others.

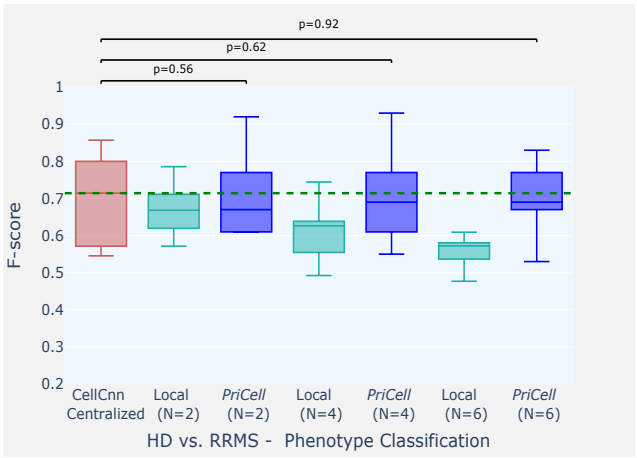
In summary, we show that the *PriCell* training does not affect the further findings of an existing work that performs training on a centralized data without integrating a privacy-preserving mechanism.



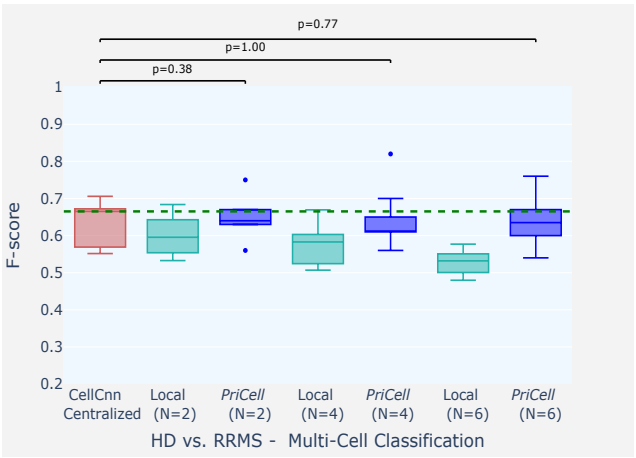
(a)



(b)

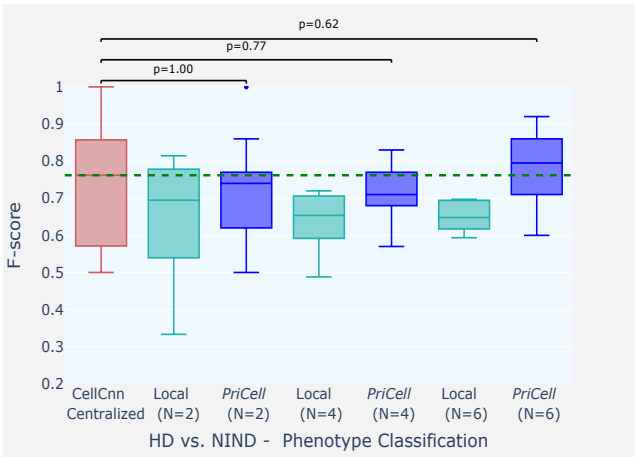


(c)

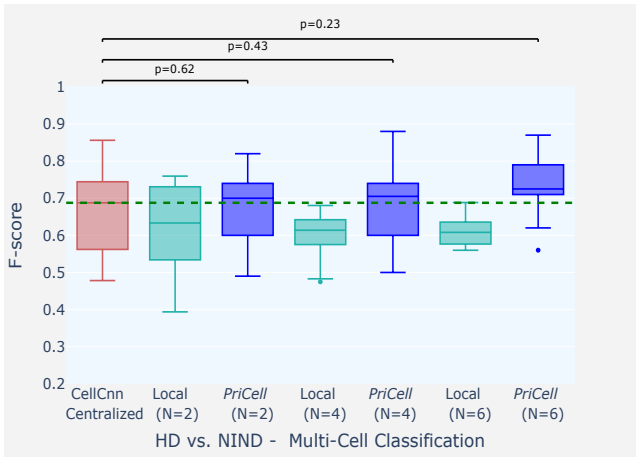


(d)

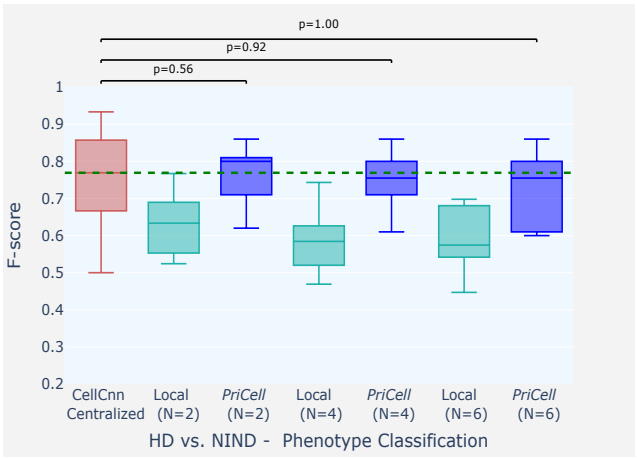
Figure S. 2. F-score boxplots when classifying healthy donor (HD) vs. relapsing–remitting multiple sclerosis (RRMS), for training multi-cells drawn from the bag of all cells per class **(a-b)** and drawn from each patient separately **(c-d)**. Experiments are repeated 10 times with different train and test set splits; the vertical dashed line illustrates the median for the baseline (CellCnn) and the dots represent the outliers. The p-values shown at the top of the figure are calculated with a Wilcoxon signed-rank test for the comparison between the corresponding boxplots ($p > 0.05$ indicates that the distributions are not significantly different). F-score is reported for two datasets: multi-cell input classification on 96 samples, and phenotype classification of 12 patients.



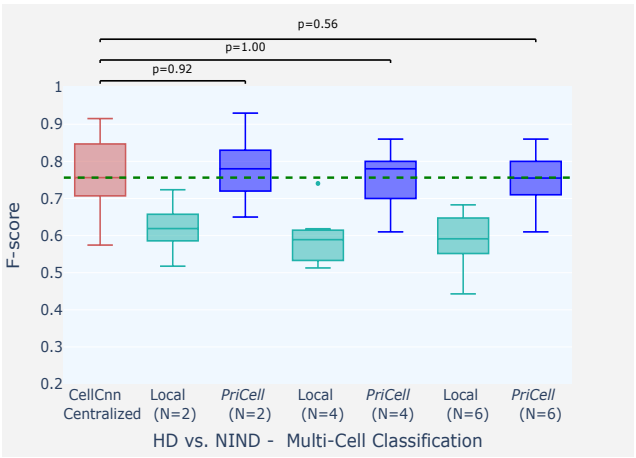
(a)



(b)

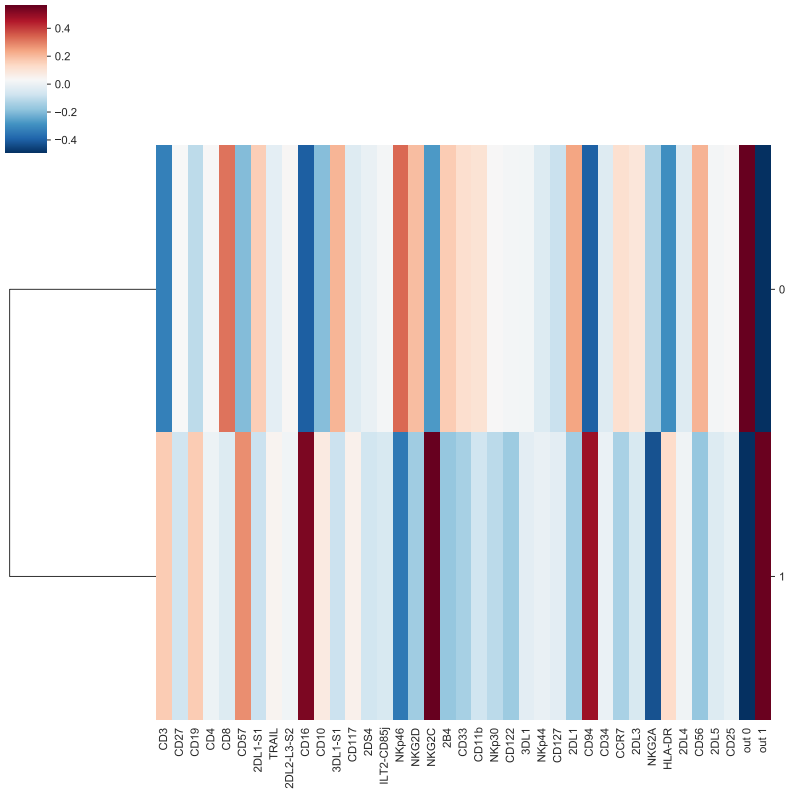


(c)

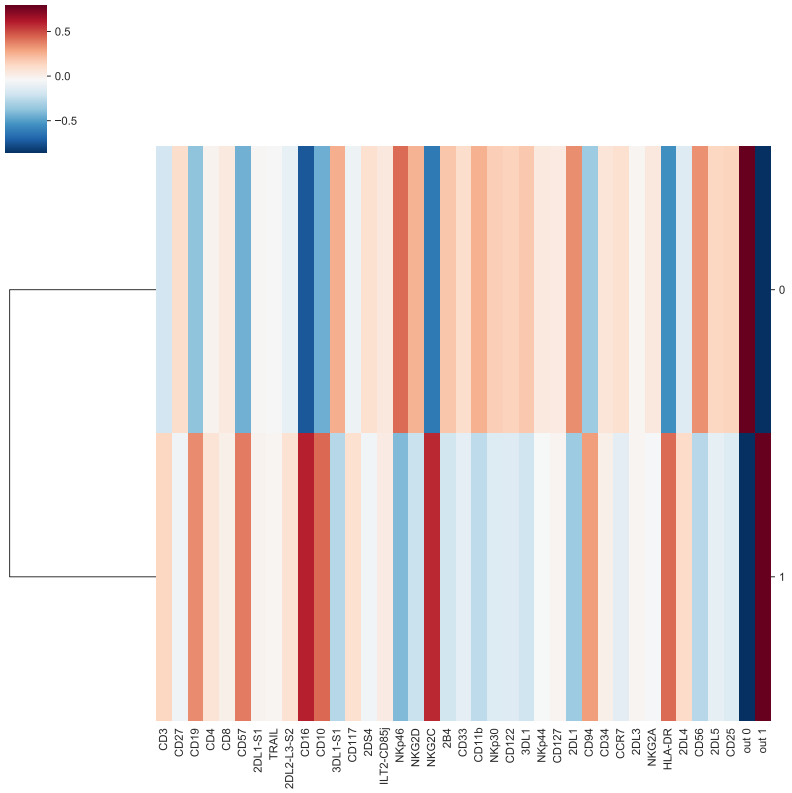


(d)

Figure S. 3. F-score boxplots when classifying healthy donor (HD) vs. non-inflammatory neurological disease (NIND), for training multi-cells drawn from the bag of all cells per class **(a-b)** and drawn from each patient separately **(c-d)**. Experiments are repeated 10 times with different train and test set splits; the vertical dashed line illustrates the median for the baseline (CellCnn) and the dots represent the outliers. The p-values shown at the top of the figure are calculated with a Wilcoxon signed-rank test for the comparison between the corresponding boxplots ($p > 0.05$ indicates that the distributions are not significantly different). F-score is reported for two datasets: multi-cell input classification on 96 samples and phenotype classification of 12 patients.

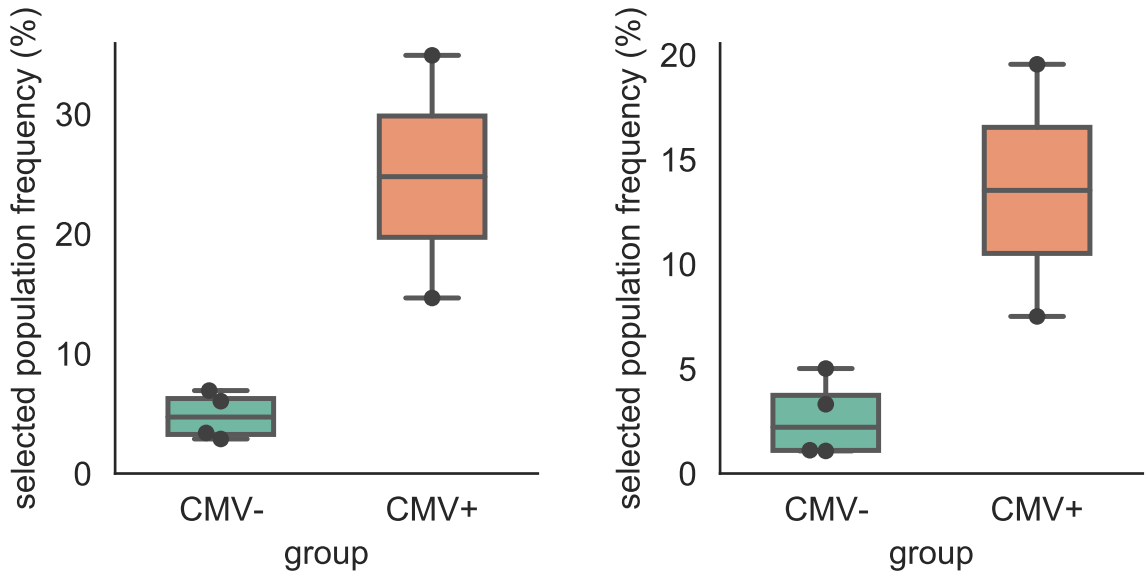


(a) Consensus filters found by CellCnn



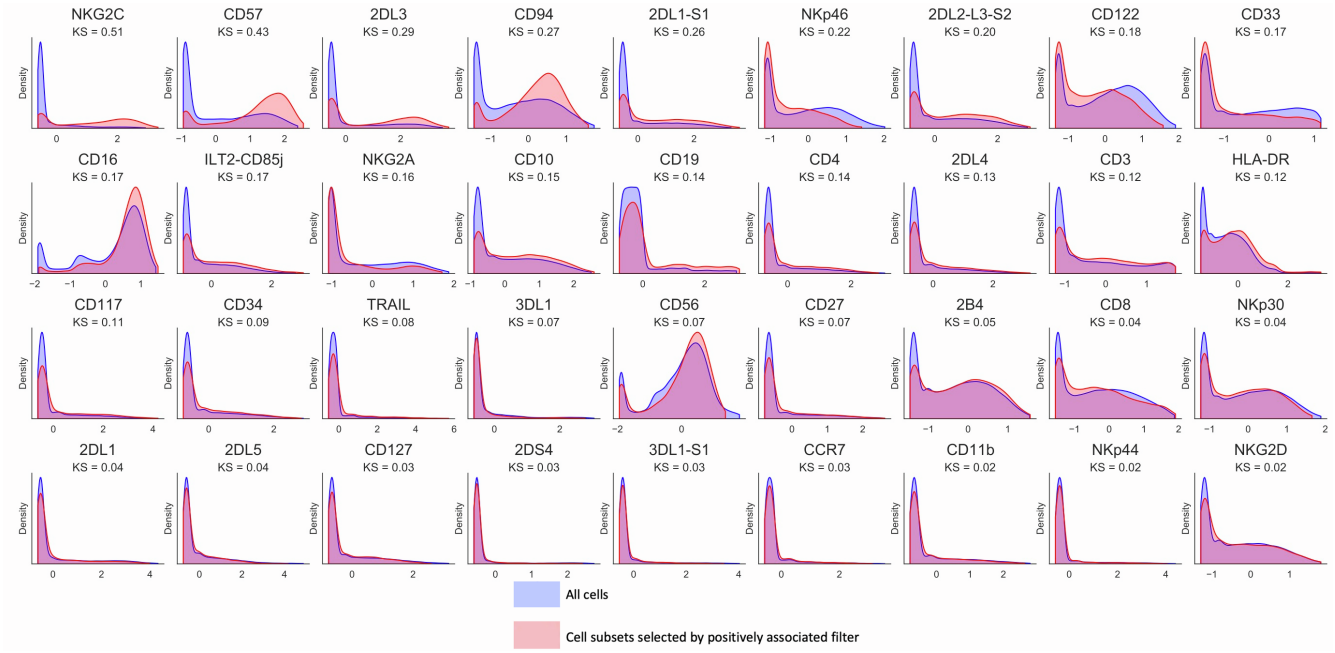
(b) Consensus filters found by *PriCell* simulation

Figure S. 4. Comparison of the consensus filters (one representative filter per class label) learned by (a) CellCnn original architecture, and by (b) *PriCell*'s adapted architecture for encrypted training on CMV dataset.

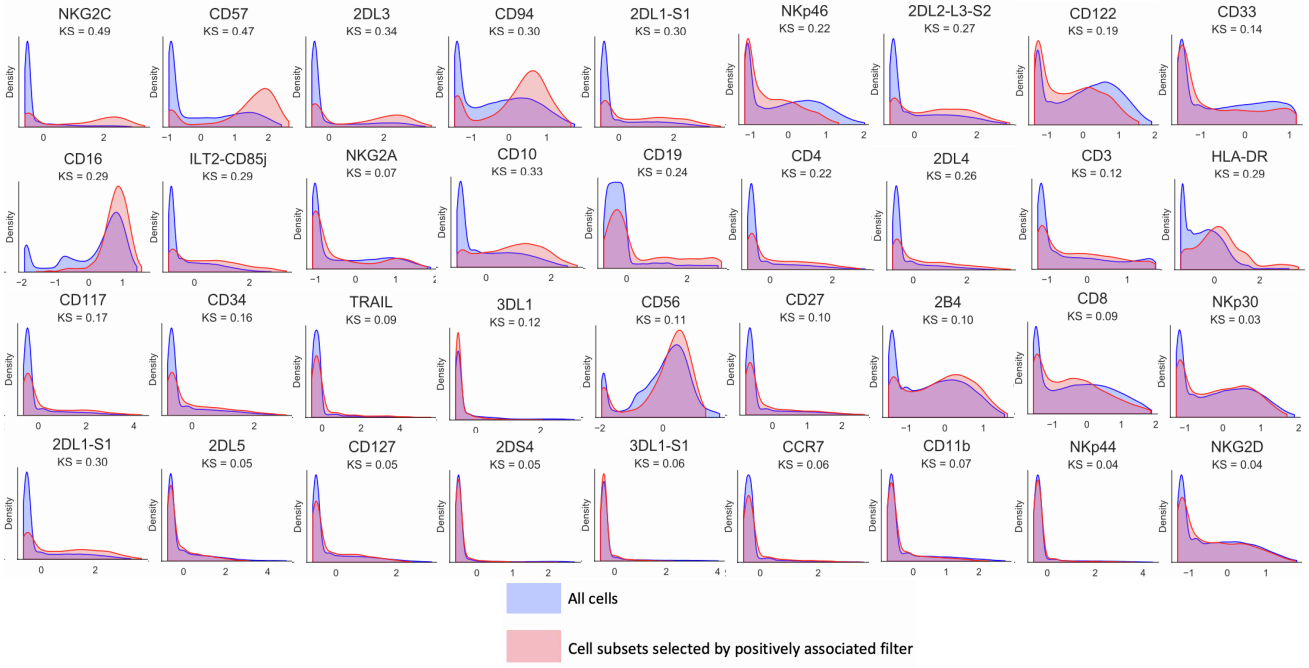


(a) CellCnn **(b)** *PriCell* simulation

Figure S. 5. Comparison of the selected cell population frequencies from the test samples of the CMV- and CMV+ classes by using the positively associated filter learned by (a) CellCnn original architecture, and by (b) *PriCell*'s adapted architecture for encrypted training.



(a) CellCnn



(b) *PriCell* simulation

Figure S. 6. Comparison of the histograms of univariate z-transformed marker expression profiles for all cells and for the cell population selected by the positively associated filter learned by (a) CellCnn original architecture, and by (b) *PriCell*'s adapted architecture for encrypted training on CMV dataset. The distributions show that *PriCell* training does not affect the findings of the non-privacy preserving training.

References

1. Arvaniti, E. & Claassen, M. (2017). Sensitive detection of rare disease-associated cell subsets via representation learning. *Nature Communications*, 8(1), 14825. https://doi.org/10.1038/ncomms14825

2. Sav, S., Pyrgelis, A., Troncoso-Pastoriza, J. R., Froelicher, D., Bossuat, J.-P., Sousa, J. S., & Hubaux, J.-P. (2021). Poseidon: Privacy-preserving federated neural network learning. *Network and Distributed System Security Symposium (NDSS)*. https://doi.org/10.14722/ndss.2021.24119