

Efficient Pruning for Machine Learning under Homomorphic Encryption

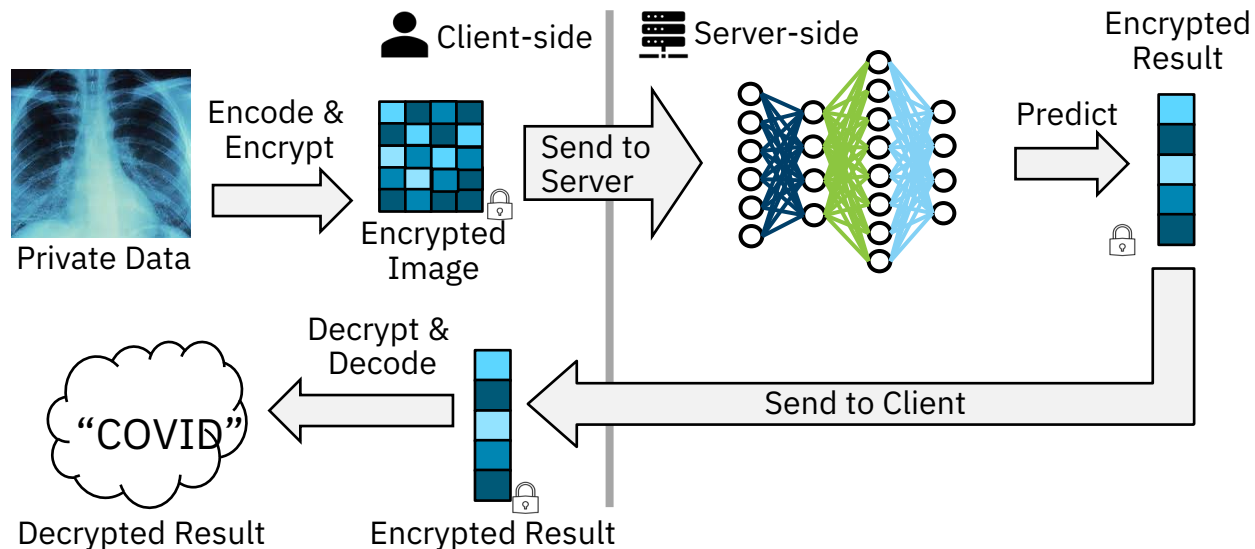
Ehud Aharoni[^], Moran Baruch[^], Pradip Bose^{*}, Alper Buyuktosunoglu^{*}, Nir Drucker[^], **Subhankar Pal**^{*}, Tomer Pelleg[^], Kanthi Sarpatwar^{*}, Hayim Shaul[^], Omri Soceanu[^] and Roman Vaculin^{*}

Subhankar.Pal@ibm.com

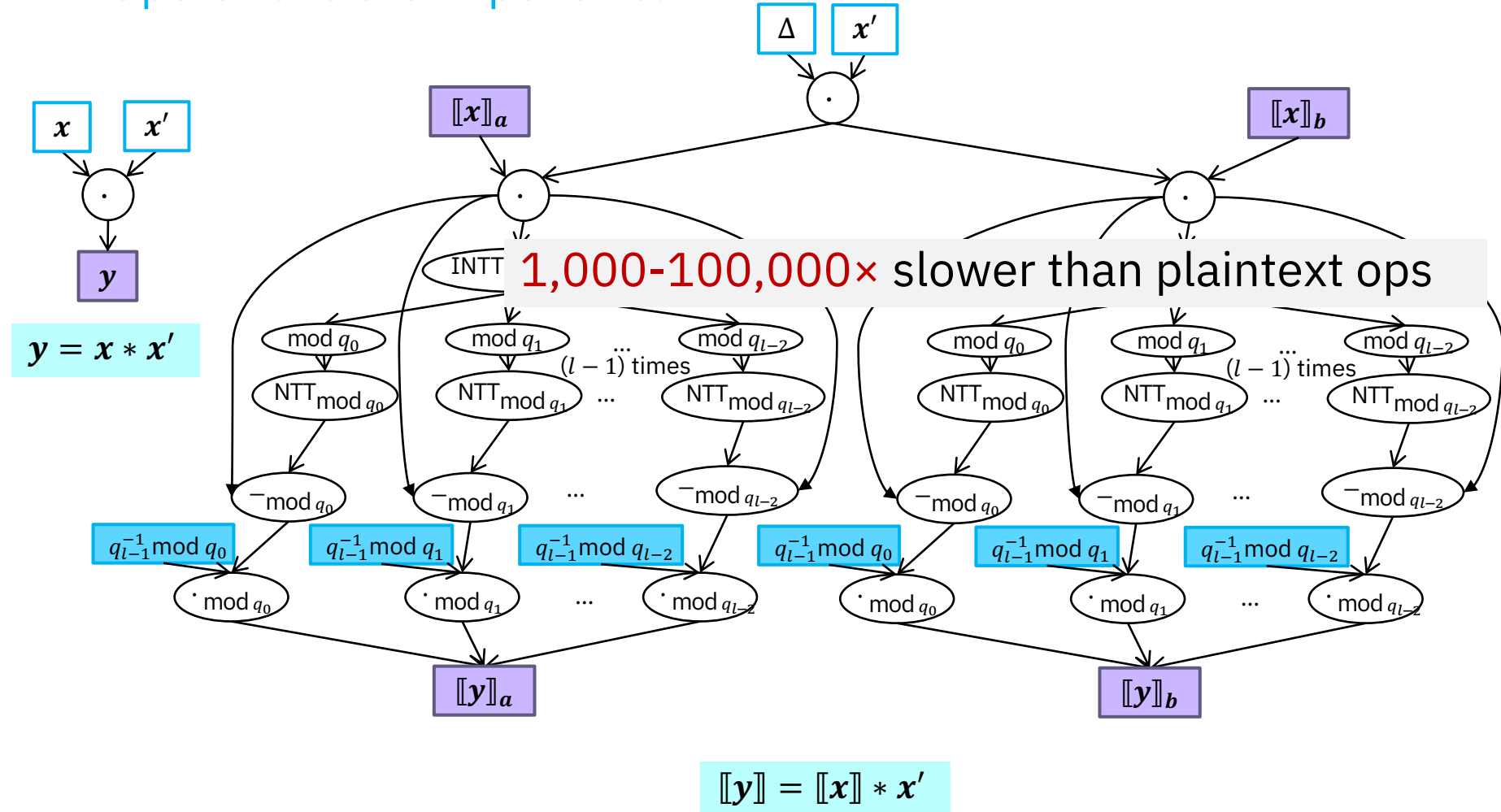
^{*}IBM T.J. Watson Research Center, USA and [^]IBM Research – Israel

Privacy-Preserving Machine Learning using FHE

- Homomorphic encryption (HE) allows computing on encrypted data
 - Useful for privacy-preserving machine learning (PPML): **model** and/or **input** are private
- E.g. a client wants to offload the task of diagnosing COVID-19 from a patient's X-ray image to a server where the server itself may be using a proprietary model trained with encrypted data

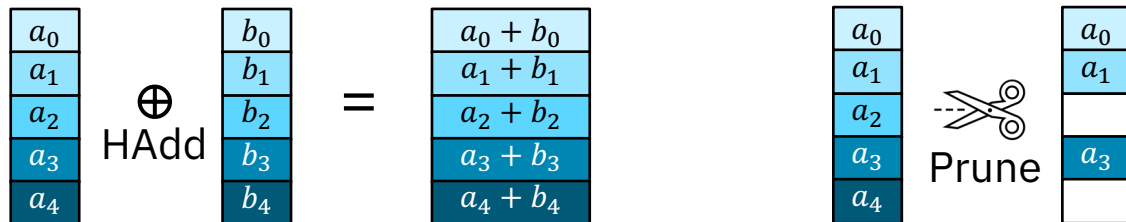


FHE Operations are Expensive!

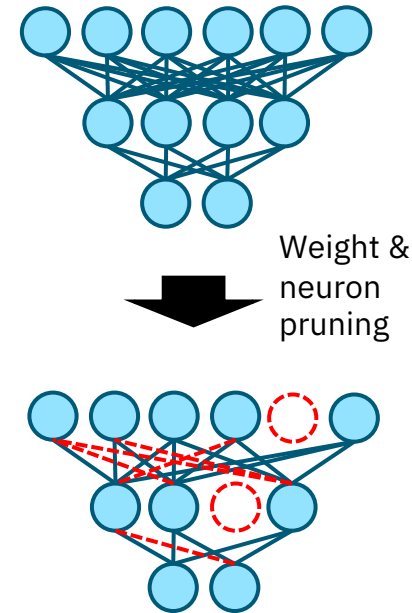


Pruning for Neural Nets under HE

- One technique for speeding up neural network processing is to prune the network parameters, e.g.,
 - Pruning weights based on an absolute threshold (L1 norm)
 - Pruning neurons randomly, etc.
- Commonly used technique in the ML domain
- Improves latency, by reducing the **number of operations** and **storage** of model and/or inputs
- Certain FHE schemes, e.g., CKKS, support operations in a Single-Instruction Multiple Data (SIMD) manner
 - The ciphertext message consists of SIMD slots
 - The same operation is applied to every slot in the ciphertext
- Creates a challenge for pruning

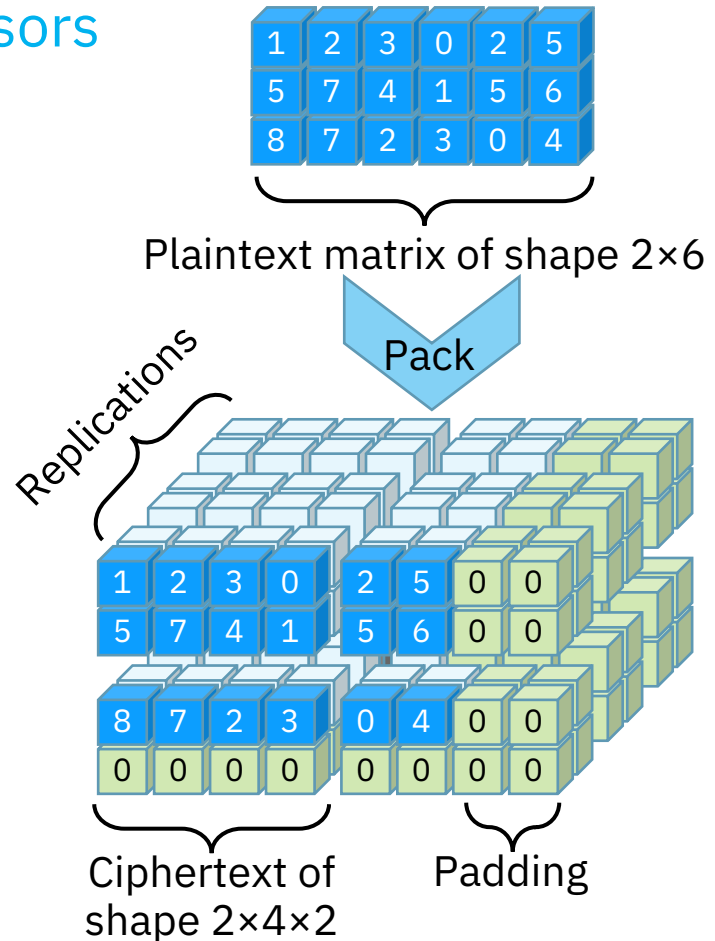


No point! Other values are non-zeros.



Ciphertext SIMD Packing using Tile Tensors

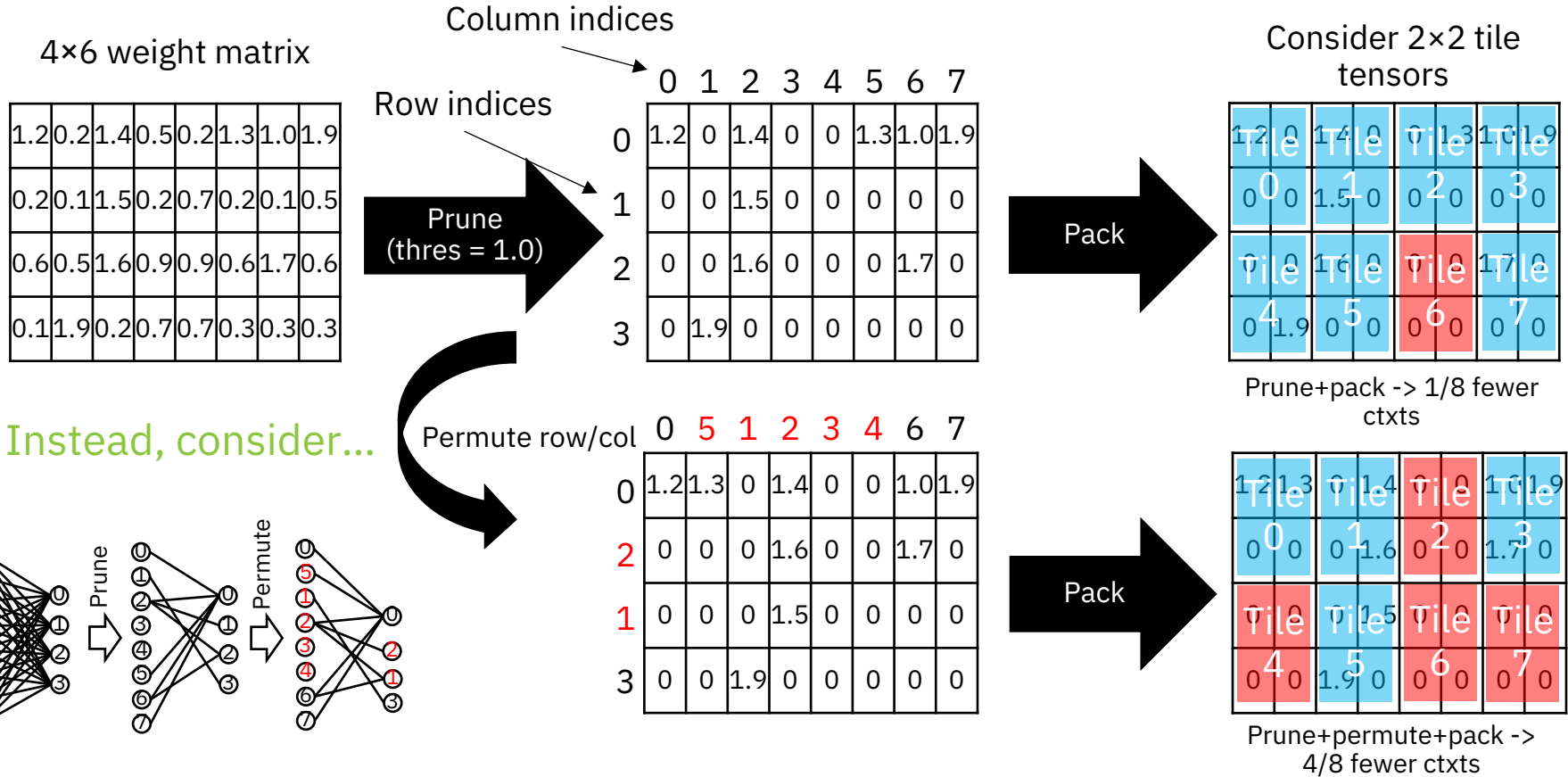
- HE schemes, such as BGV, B/FV, and CKKS
 - Operate on ciphertexts in a SIMD fashion
 - Encrypt a **plaintext vector** into a **ciphertext**
 - Homomorphic ops performed slot-wise on the elements of the plaintext vector
- A recent work by Aharoni et al. [5] proposes a data structure called tile tensor that packs tensors (e.g., vectors, matrices) into fixed-size chunks, i.e., tiles
 - Implemented as part of an offering called HeLayers [6]
- Thus, for NN inference under HE, what matters is not traditional sparsity but **tile sparsity**, i.e., the % of tiles that contain only zero values prior to HE encoding/encrypting



[5] E. Aharoni, et al., "HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data", CoRR abs/2011.0 (2020). 2011.01805

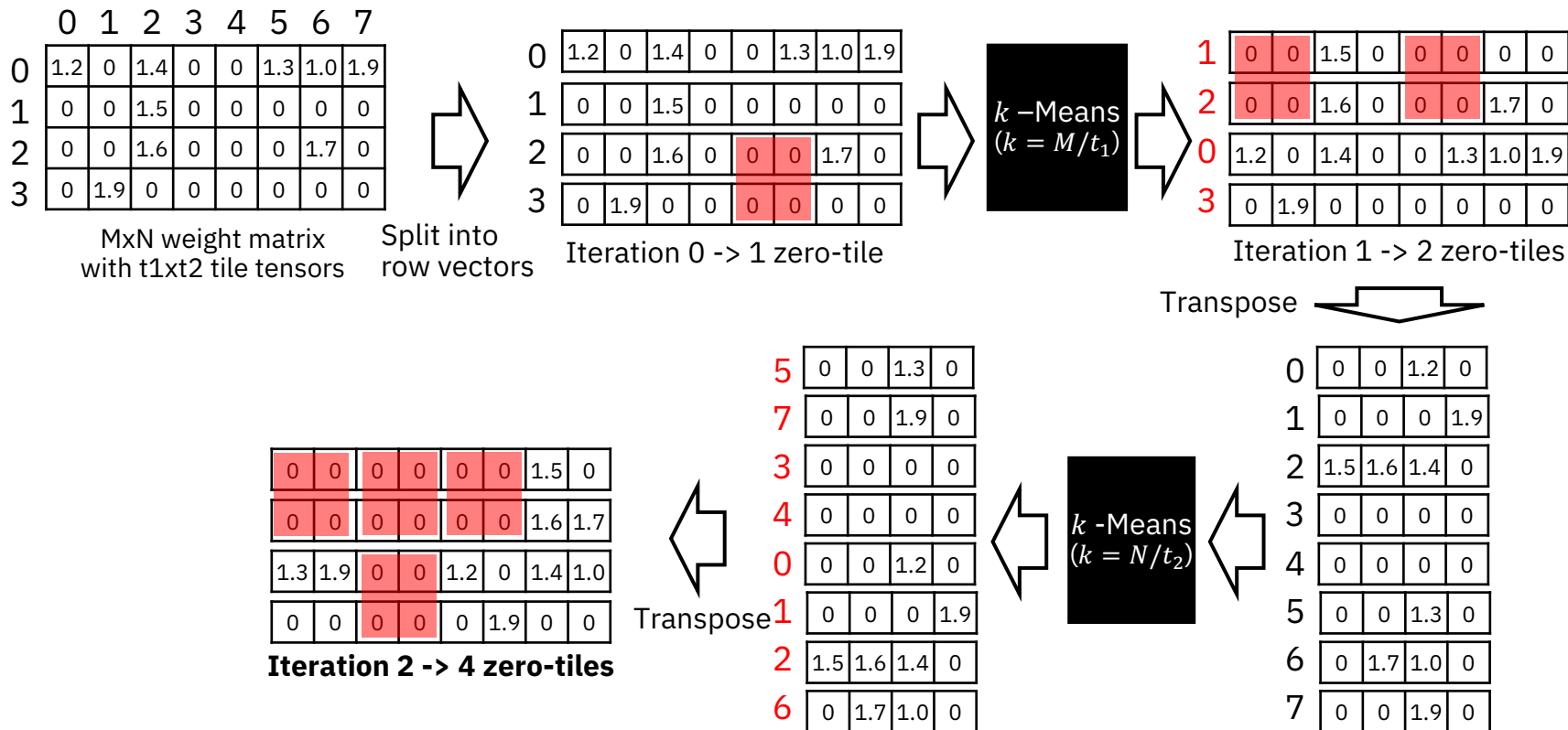
[6] <https://github.ibm.com/bioauth/helayers/>

Problem with NN Pruning+HE Packing and Solution



- Exhaustive search complexity to find best permutation for $M \times N$ matrix = $O(M!N!)$ → prohibitive!
- Instead, we propose an iterative method to arrive at the “best” permuted matrix

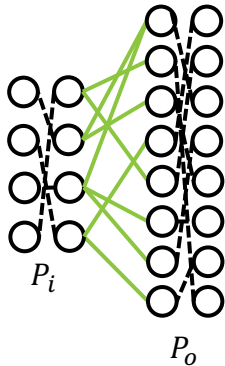
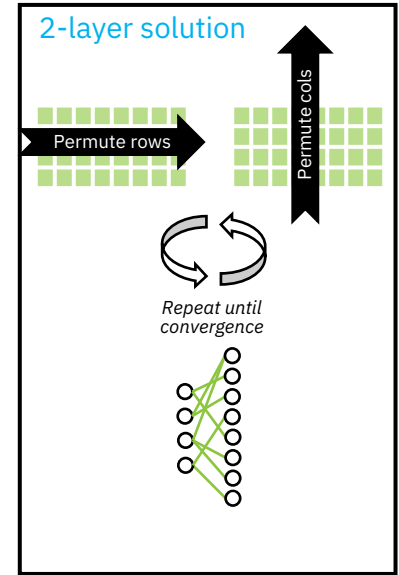
Permute Algorithm Illustration with Weight Pruning



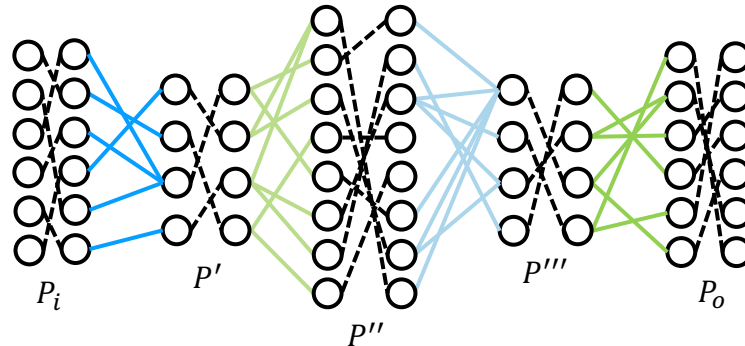
- For clustering using k -means, we use grouped Hamming distance: for two mask vectors a_i and b_i , computes the number of non-zero groups of size $\lfloor \frac{N}{t_2} \rfloor$ (or $\lfloor \frac{M}{t_1} \rfloor$, depending on the iteration) in $(a_i \text{ AND } b_i)$
- Further, we implement a balancing scheme that reassigns points from centroids with $> t_1$ (or, t_2) points to centroids that have the minimum distance to the point

Extending Permutation to Multiple FC Layers

- For a single-layered NN, we can alternately apply our permutation algorithm to the rows and columns of the weight matrix
- For correctness, we need to introduce an input permutation matrix P_i and output permutation matrix P_o , for the input and output activation, resp.
- For a multi-layered NN, if we do the same for each layer, we will need a P_i and P_o for each layer → this is a bad idea



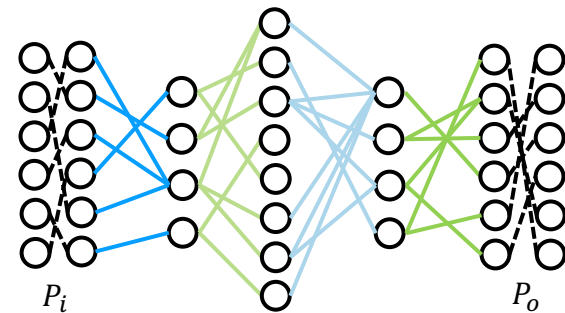
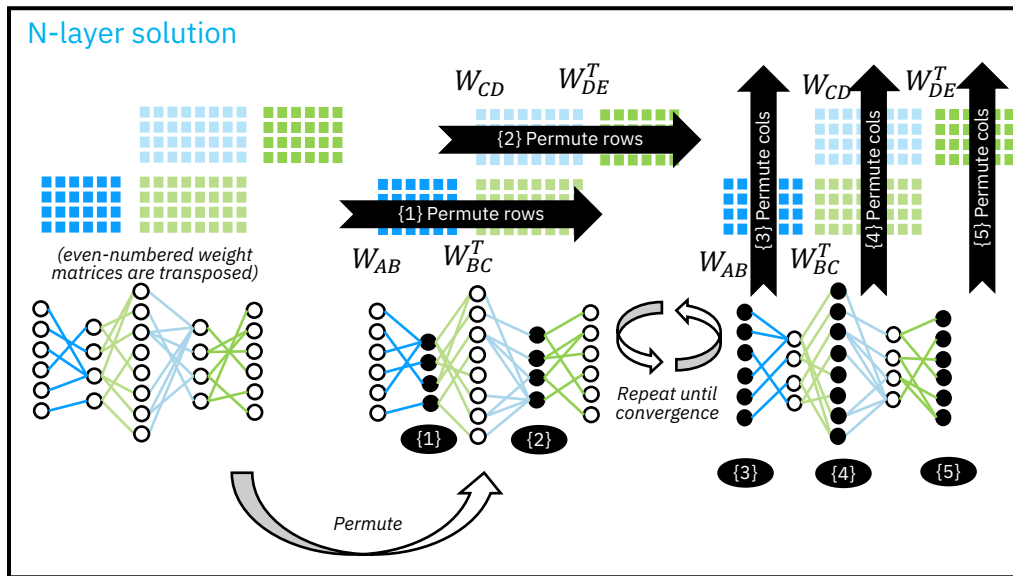
Single-Layered NN



Multi-Layered NN

Extending Permutation to Multiple FC Layers

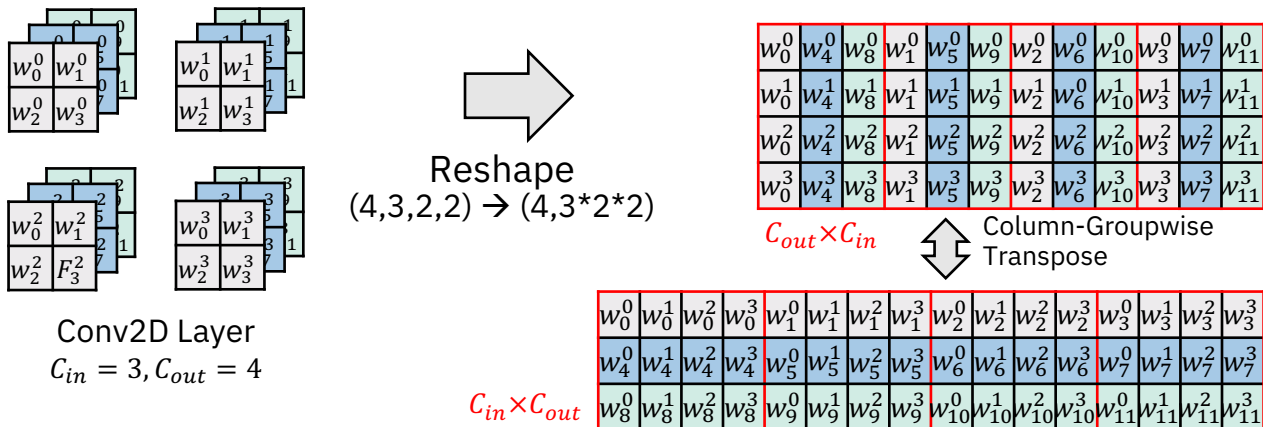
- Instead, we propose a layer-wise co-permutation algorithm that permutes adjacent layers together
 - E.g, shuffling neurons {1} → co-permute rows of W_{BC}^T and rows of the preceding weight matrix, W_{AB} , or $\text{concat}(W_{AB}, W_{BC}^T)$
- Therefore, only one P_i and P_o layers are required



Multi-Layered NN

Extending Permutation to Conv Layers

- For convolutional layers, the permutation algorithm is more nuanced as a Conv2D layer consists of 4D filter weights (dimensions: inC, outC, H, W)
- Similar to permuting **neurons** in the FC case, we permute **channels** in the Conv2D case
- This translates to a similar co-permutation algorithm as before, but with column-groupwise transposition
 - Precisely, the Conv layer is reshaped to reduce it to a 2D tensor and each group of $C_{out} \times C_{in}$ (or, $C_{in} \times C_{out}$, for alternate iterations) sub-matrix is transposed within the larger tensor
 - Reshape and transposition are done prior to model deployment, and do not add any overhead during inference



Proposed Pruning Schemes

- P2** : Train → Prune → Retrain → Pack
- P2T** : Train → Prune^{pack} → Retrain → Pack
- P3** : Train → Prune → Permute → Retrain → Pack
- P3E** : Train → Prune → Permute → Expand → Retrain → Pack
- P4** : Train → Prune → Permute → Prune^{pack} → Retrain → Pack
- P4E** : Train → Prune → Permute → Prune^{pack} → Expand → Retrain → Pack

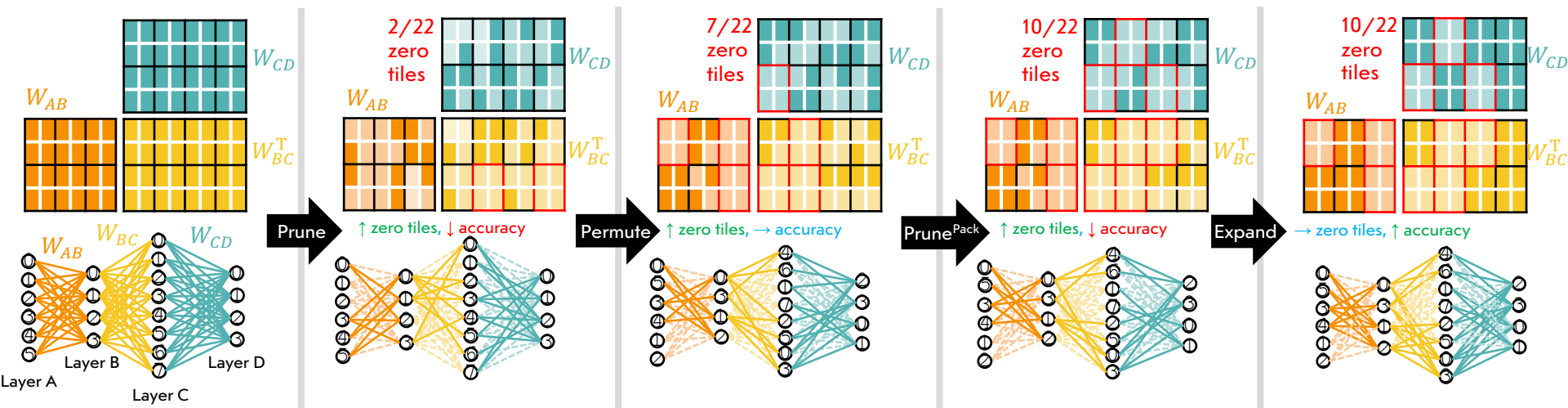
Overview of our prune, permute, expand, and pack methods

P2T	Scope	Local (Lc), Global (Gl)
	Criterion	Average/Maximum/Minimum of tile (T-Avg/T-Max/T-Min)
	Target	Weight (-)
P2, P3, P3E	Scope	Local (Lc), Global (Gl)
	Criterion	L1 (L1), Rand (Rnd)
	Target	Weight (Wei), Neuron (Neu)
P4, P4E	Scope	Local (Lc), Global (Gl) [1st and 2nd prune]
	Criterion	L1 (L1), Rand (Rnd) [1st prune], threshold fraction of non-zeros in tile to prune [2nd prune]
	Target	Weight (Wei), Neuron (Neu) [1st prune]

Scope, criterion and target of pruning for each scheme

- **P4E is our best-proposed scheme**
 - Prior to expansion, we perform a second pruning-aware-packing step
 - Reduces tiles that contain “mostly” zeros
 - Tiles are marked for pruning if it has more than a certain fraction of non-zeros
- **P2T uses a tile packing-based pruning scheme**
 - Pick a tile shape and split every matrix into tiles
 - For each tile, we compute the minimum/maximum/average metric of its absolute values
 - Prune the tiles with the lowest value of the metric
- **P2T is our adaptation of a state-of-the-art scheme called Hunter [7], which we compare with**

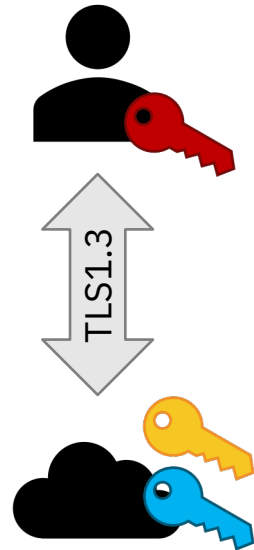
P4E Scheme Illustration






- For a 4-layer (A-D) MLP network with 2×2 tiles
- Perform $54/88 = 61\%$ weight-based pruning
 - After **pruning** we can only reduce $2/22 \approx 10\%$ tiles
 - After **permutation**, however, it is possible to reduce $7/22 = 35\%$ of the tiles
- Prune^{pack} removes tiles that have “a few” non-zeros in them, e.g., $10/22 = 45\%$ in this case
- Expansion is useful to restore the accuracy loss without affecting tile sparsity

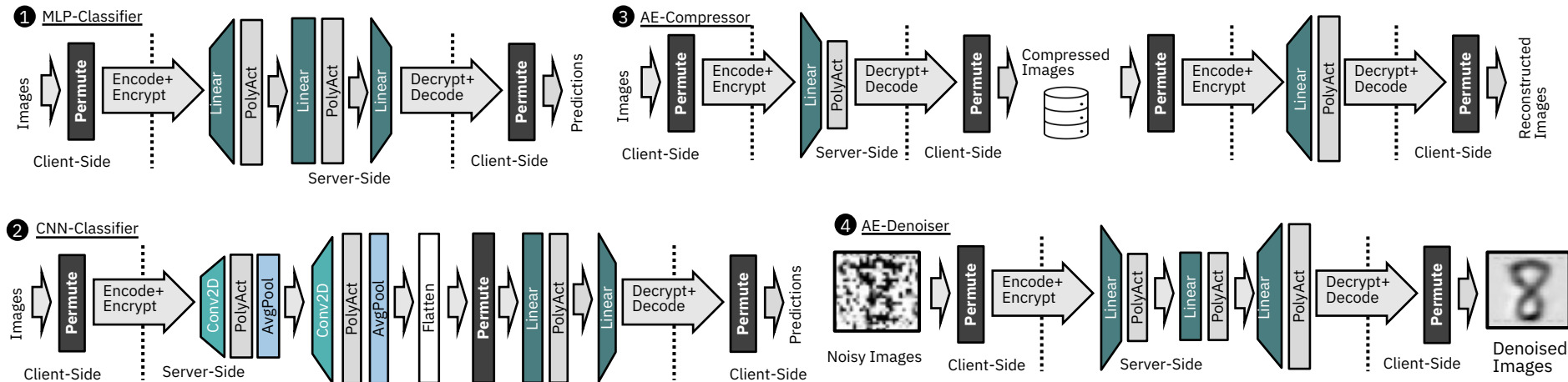
Threat Model

- We consider a simple 2-party threat model in this work
 1. A data owner with a pre-trained NN model and private data samples
 2. A cloud server that run HE inference as a service
- During computation, the cloud learns nothing about the underlying encrypted samples of the user or about the encrypted weights of the model owner, although it does learn the structure of the NN
- We assume
 - Secure protocols for inter-party communication, such as TLS 1.3
 - Computationally-bounded and semi-honest adversaries
 - That the data arrangement is modified before encryption and thus does not impact the semantic security of the HE scheme
 - 128-bit security



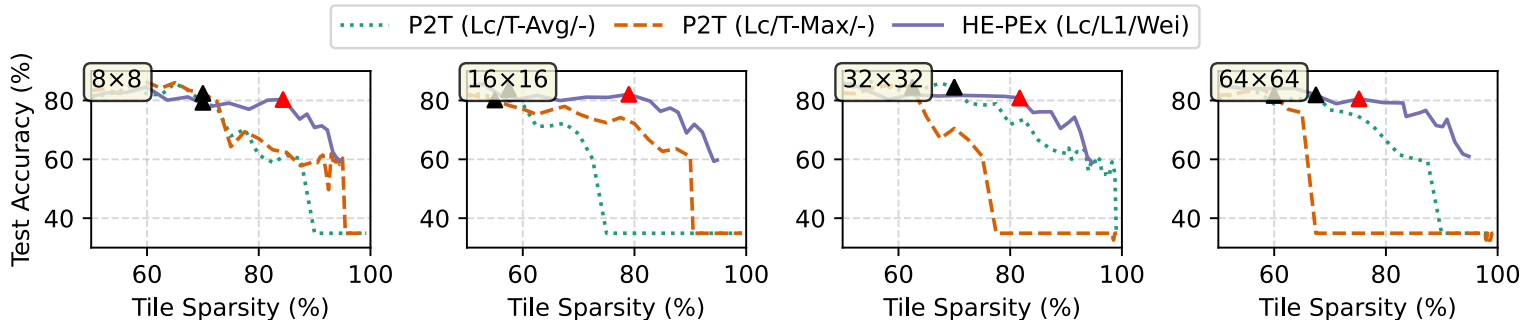
-  Secret key
-  Evaluation keys
-  Public key

Experimental Setup and HeLayers Integration



- We evaluate our methods on four applications that use MLPs and CNNs
 - In addition, we showcase our P4E technique on a 21-layer HE-friendly AlexNet with trained activation functions
- We consider the MNIST, CIFAR-10 and SVHN datasets + COVIDx CT-2A dataset for AlexNet
- We perform accuracy measurements and pruning using [PyTorch](#) on a system with V100 GPUs and Xeon Gold CPU
- Our proposal is integrated into HeLayers
 - Enhanced HeLayers to automatically identify zero vector inputs using a “zero-flag” and remove storage for zero vectors
- End-to-end memory and latency are measured using
 - [HeLayers](#) running on **8 threads**, averaged over 10 runs
 - [SEAL CKKS](#) implementation while targeting **128b** security, with N=32,768

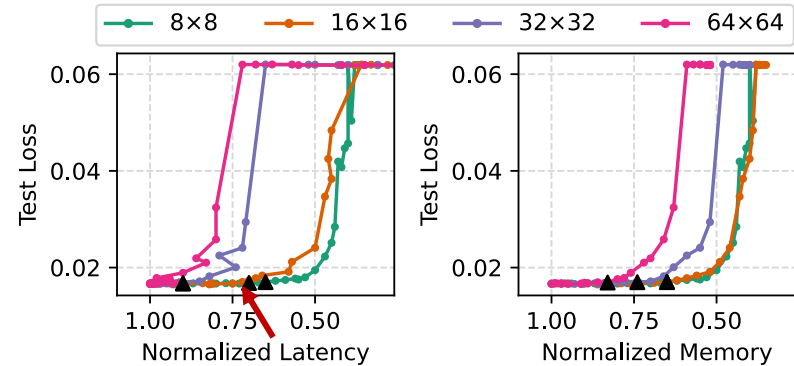
Comparison with P2T Schemes



- Inference accuracy vs. tile sparsity (% of zero tiles) comparison; t_1 and t_2 swept s.t. $t_1 = t_2$ and $t_1 \cdot t_2 \cdot t_3 = \frac{N}{2}$
 - Using our best pruning technique (P3E for Conv and P4E for FC with Lc/L1/Wei, labeled HE-PEx)
 - Against P2T variants for a 21-layer AlexNet NN on the COVIDx CT-2A dataset; local-scope variants far outperform global-scope ones
 - Markers show the NN that has the highest tile sparsity within at most 2.5% worse accuracy compared to the unpruned NN
- Through careful pruning, our best scheme achieves a tile sparsity of **75–84%** with, at worse, a 2.5% degradation in inference accuracy over the unpruned NN
- Overall, across the other four applications (omitted for brevity), we observe that HE-PEx improves the tile sparsity of each NN by **48%**, **57%**, and **104%** over P2T for tile sizes of 8x8, 16x16 and 32x32, resp.
- The efficacy of P2T schemes degrades as the tile size increases, whereas HE-PEx remains more stable
 - The larger the tile, the greater the probability that it holds a set of “important” weights along with “unimportant” ones
 - Pruning all of these weights at once in P2T leads to severe accuracy degradation, since important weights are pruned out

Impact on End-to-End Latency and Memory for Inference

- Experiments performed with HE layers+HE-PEX on NN inference using the AE-Denoiser NN, on CIFAR-10
- Our results show **memory** and **latency** reductions of **10–35%** and **17–35%**, resp., compared to the unpruned model
 - With < 2.5% degradation in NN inference accuracy
- The benefits are higher for smaller t_1, t_2 , as the tile sparsity is greater for smaller tiles
 - However, smaller tile shapes require a large t_3 , and therefore a large batch size of the inputs
 - The final tile shape depends on the optimization requirements, e.g., latency, memory, throughput
- Our method reduces **29–63%** of **HE additions** and **53–63%** of **HE multiplications** compared to the unpruned NN
- There is a potential 37% latency to reduce even with 99% tile sparsity
 - Reason: outputs of the FC layers may be denser than the weights, and these need to run through polynomial activations
 - Can mitigate using other methods, e.g., activation pruning

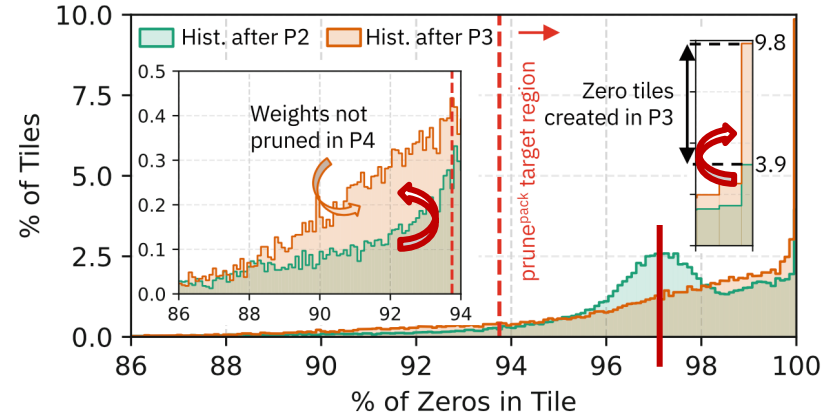


Reduction in HE operations with HE-PEX

HE Op Tile Shape	Add (before)	Add (↓) (after)	Mul (before)	Mul (↓) (after)
8×8	28,840	10,801 (63%)	25,600	9,367 (63%)
16×16	7,360	3,954 (46%)	6,400	3,002 (53%)
32×32	2,200	1,542 (30%)	1,600	937 (41%)
64×64	760	542 (29%)	400	188 (53%)

Impact of Permutation

- We plot histograms of the zero values in 32x32 tiles of the layers of AlexNet trained on COVIDx CT-2A dataset, after performing Lc/L1/Wei pruning with 97.5% pruning fraction
- As shown in green, after pruning the weights as part of the first prune step, we see a Gaussian distribution centered at 97.5% with a squashed tail
- After permutation, we observe a two-fold set of benefits
 - First, the weight tensors are rearranged so that more zeros are clustered together, thus increasing the tile sparsity from 3.9% to 9.8%
 - Second, trained non-zero weights have migrated from sparse tiles into denser tiles
 - This leads to better inference accuracy after re-training, as these weights are not pruned away



Conclusion

- Presented a framework of packing-aware pruning for HE-enabled NNs that combines four critical primitives – **pruning, permutation, expansion & packing**
 - In particular, our novel permutation approach lends improvement in tile sparsity without affecting functionality or accuracy of the NN
- Described how it operates in a non-client-aided HE-enabled inference use-case, and integrated it with a general HE packing framework called HElayers
- Adapted a state-of-the-art pruning technique called Hunter and compared our best scheme against theirs, in terms of NN loss/accuracy and the fractions of ciphertexts eliminated
- Our results with running various privacy-preserving ML applications on different datasets showed significant reductions in memory and latency for inference under HE under a small loss in accuracy

Thank You! Questions?