Introduction

Building efficient and correct applications with leveled integer FHE schemes is tedious and error-prone:

- Incorrect encryption parameters will compromise either security or performance.
- Best performance requires efficient use of batching.
- For the CKKS family of schemes correctness requires careful precision selection.

CHET is a compiler and runtime that automates many parts of this process for neural network inference tasks. The compiler applies transformations based on a framework of symbolic analysis passes.

Data Layout Selection

CHET selects one of four layout policies for the runtime:

- **HW** and **CHW** use the corresponding layout throughout.
- **HW-conv** switches to HW for convolutions, CHW otherwise.
- **CHW-fc** uses CHW starting from the first fully connected layer.

The selection uses a cost analysis pass, which accounts for:

- Relative costs of operations.
- Required encryption parameters.
- Degree of parallelism in the model vs. available execution units.
- Cost of switching between layouts.

Parameter Selection

CHET supports parameter selection for both HEAAN's CKKS and SEAL's RNS-CKKS. The analysis passes simulate scaling behavior while measuring modulus consumed by rescale operations.

Rotation Key Selection

Using a network specific set of rotation keys can provide up to 2X performance improvement. This transformation uses a pass that records the necessary rotation keys.

Data Layouts for Vectorized Kernels

CHET includes kernels optimized for low-latency inference of CNNs, which operate on strided layouts of values into batched ciphertexts. We have considered two classes of layouts:

- **HW** Each channel of an image is in a separate ciphertext.
- **CHW** Each ciphertext holds multiple channels.

Consider the 2D-convolution of an image tensor of shape \((I, H, W)\) with a filter of shape \((F, FW, IC, OC)\):

\[
output_{OC,H,W} = \sum_{IC=0}^{IC} \sum_{FH=0}^{FW} \sum_{FW=0}^{FW} input_{IC,FH,FW,IC} \cdot filter_{IC,FH,FW,OC} \cdot \text{tryRescale}(output_{OC,FH,FW,IC}, cipherScale)
\]

For the HW layout the kernel is:

```python
for oc in indices(OC):
    output[oc] = zeroCipher
for ic, fh, fw in indices(IC,FH,FW):
    weight = encode(filter[fh,fw,ic,oc], scalarScale)
    rotated = leftRotate(input[ic], fh * W + fw)
    output[oc] = multiplyPlain(rotated, weight)
    tryRescale(output[oc], cipherScale)
```

The kernel for CHW is similar, but includes extra rotations and additions to handle multiple channels in a ciphertext. Compared to HW, the kernel may perform fewer multiplications. However, HW has a lower depth, because with CKKS encoding a uniform value into all slots is exact. These kinds of trade-offs make it challenging to choose the best layout manually.

Evaluation

We have evaluated CHET on a set of CNNs. To our knowledge, SqueezeNet-CIFAR is the largest network evaluated on FHE to date.

<table>
<thead>
<tr>
<th>Network</th>
<th>Layers</th>
<th>CHET best</th>
<th>Hand-written</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet-5-small</td>
<td>4</td>
<td>8s</td>
<td>14s</td>
</tr>
<tr>
<td>LeNet-5-medium</td>
<td>4</td>
<td>51s</td>
<td>140s</td>
</tr>
<tr>
<td>LeNet-5-large</td>
<td>4</td>
<td>265s</td>
<td></td>
</tr>
<tr>
<td>Industrial</td>
<td>7</td>
<td>312s</td>
<td>2413s</td>
</tr>
<tr>
<td>SqueezeNet-CIFAR</td>
<td>10</td>
<td>1342s</td>
<td></td>
</tr>
</tbody>
</table>

The following figure compares latencies for each network with different layout policies. No single policy is best for all networks.