Abstract
Deep Learning Models adoption was critical to Walmart Search relevancy
Walmart Search has embarked on the journey of adopting Deep Learning in the Search ecosystem for improving Search relevance in various parts. As our pilot use case, we wanted to serve the computationally intensive Bert Base model at runtime with an objective to achieve low latency and high throughput.

We had JVM hosted web applications loading and serving multiple models. The experimental models were being loaded onto the same applications. These models are large in size and computation is expensive.

- Refreshing model with the latest version or adding new experimental model would need application deployment
- Increased memory pressure on single application
- Slow startup time due to loading multiple ML models during startup
- Concurrency was not beneficial due to limited CPU (Metrics on concurrent model prediction vs sequential)

We had JVM hosted web applications loading and serving multiple models. The experimental models were being loaded onto the same applications. These models are large in size and computation is expensive.

We were facing the following limitations with this approach:
- Refreshing model with the latest version or adding new experimental model would need application deployment
- Increased memory pressure on single application
- Slow startup time due to loading multiple ML models during startup
- Concurrency was not beneficial due to limited CPU (Metrics on concurrent model prediction vs sequential)

Model Optimizations
This refers to techniques for performing computations and storing tensors at a lower bit than floating point. With some tradeoff in precision, this helps in reducing the memory footprint of the model and speeding up compute times.

Hardware support for INT8 computations is typically 2 to 4 times faster compared to FP32 compute.

PyTorch provides us 3 methods of quantization for CPU based hardware:
1. Dynamic - this happens at runtime. The weights are quantized on the fly.
2. Training Aware - this happens at train time, so the training learns to reduce quantization error and leads to the greatest accuracy.
3. Static - this happens post training. The quantized values are passed between operations, reducing inference times.

Dynamic Quantization
It allowed us decreasing inferencing latency significantly, increasing throughput while trading off little accuracy. FP16 came out promising

<table>
<thead>
<tr>
<th>Quantization</th>
<th>Model F1</th>
<th>Inference latency (PS0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F32</td>
<td>88%</td>
<td>18ms</td>
</tr>
<tr>
<td>FP16</td>
<td>86.7%</td>
<td>8ms</td>
</tr>
<tr>
<td>INT8</td>
<td>82%</td>
<td>6ms</td>
</tr>
</tbody>
</table>

Model Distillation
This process transfers learned knowledge from a trained larger teacher model to a smaller student model. The Bert Base model is used to train a distilBert model as a student network. As a result, distilBert has 40% less parameters than Bert Base model and around 1.5 times faster and lighter to serve.

<table>
<thead>
<tr>
<th>Model</th>
<th>teacher F1</th>
<th>student F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>86.7</td>
<td>81%</td>
</tr>
</tbody>
</table>

Walmart Search Model Serving Framework
Walmart’s generic base handler was developed to support most of the ML models. Additional customization and capabilities are provided on top of the vanilla TorchServe, to support ONNX Runtime and Torchscript formats.

Chaos Testing on Model Servings
Our model onboarding, either new model or new version of existing model, goes through a series of chaos testing by inducing several fault injections in the Model Serving platform. A few key fault injections such as network packet loss, n/w packet delays, Memory/CPU/Disk hogging

Chaos testing allowed us
- Control losses on revenue by finding critical issues ahead of time.
- Reduction in system or application failure.
- Better user experience with less disruption and high service availability.
- Learn about the system and gain confidence.

Acknowledgements
Shout out to Meta AI/PyTorch team for their collaboration and support.