Model Compression for Efficient AI Computing
From TinyML to LargeLM

Song Han
MIT
songhan.mit.edu
tinyml.mit.edu
@SongHan/MIT
Model Compression
Bridges the Gap between the Supply and Demand of Computation

Specialized hardware is important but not enough.
Model compression bridges the gap.
Model Compression
Bridges the Gap between the Supply and Demand of Computation

AI Application
(demand of computation)

New Primitive → Hardware-aware NAS → Fine-Grained Pruning → CoarseGrained / Structured Pruning → Post-Training Quantization → Quantization-Aware Training

AI Hardware
(supply of computation)

Distillation → Augmentation

Bridges the Gap between the Supply and Demand of Computation
Model Compression

Applications

Large Language Model
Generative AI
Advanced Driver Assistance System
TinyML

Techniques

Hardware-aware NAS
Pruning & Sparsity
Quantization
Distillation
New Primitive
Same Principle, Diverse Applications

Applications

- **Large Language Model**: A large language model is a type of artificial intelligence that can process and generate human-like language, based on vast amounts of data it has been trained on.
- **Generative AI**:...
- **Advanced Driver Assistance System**:...
- **TinyML**:...

Techniques

- **Hardware-aware NAS**
- **Pruning & Sparsity**
- **Quantization**
- **Distillation**
- **New Primitive**
Efficient Large Language Models
Reducing LLM Serving Cost and Accelerating Inference

ChatGPT is at capacity right now

Get notified when we're back

The LLM serving costs are extremely high

Write a rap about the status of ChatGPT.

Yeah, yo, what's up ChatGPT fam
A lot of people trying to jam
But don't worry, we got your back
Just check back soon, we'll get on track

ChatGPT's the place to be
For all your AI chat needs
We're working hard to keep up the pace
So hold tight, we'll be back in this space

So don't worry, don't be mad
We're working hard, it's not so bad
Just give us time, we'll be back
ChatGPT, the future of chat, that's a fact.
Quantization cut the model size by half, but…

Existing Quantization Method is Slow or Inaccurate

- W8A8 quantization has been an industrial standard for CNNs, but not LLM. Why?
- Systematic outliers emerge in activations when we scale up LLMs beyond 6.7B. Traditional CNN quantization methods will destroy the accuracy.

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale (Dettmers et al., 2022)
SmoothQuant

Smoothing activation to reduce quantization error

- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models (Xiao et al., 2022)
SmoothQuant

Smoothing activation to reduce quantization error

- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models (Xiao et al., 2022)
SmoothQuant

Smoothing activation to reduce quantization error

- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels
- Migrate the quantization difficulty from activation to weights, so both are easy to quantize

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models (Xiao et al., 2022)
SmoothQuant
SmoothQuant is Accurate and Efficient

- SmoothQuant well maintains the accuracy without finetuning.
- SmoothQuant can both accelerate inference and halve the memory footprint.

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models (Xiao et al., 2022)
SmoothQuant
SmoothQuant is Accurate and Efficient

<table>
<thead>
<tr>
<th>Method</th>
<th>OPT-175B</th>
<th>BLOOM-176B</th>
<th>GLM-130B</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP16</td>
<td>71.6%</td>
<td>68.2%</td>
<td>73.8%</td>
</tr>
<tr>
<td>W8A8</td>
<td>32.3%</td>
<td>64.2%</td>
<td>26.9%</td>
</tr>
<tr>
<td>ZeroQuant</td>
<td>31.7%</td>
<td>67.4%</td>
<td>26.7%</td>
</tr>
<tr>
<td>LLM.int8()</td>
<td>71.4%</td>
<td>68.0%</td>
<td>73.8%</td>
</tr>
<tr>
<td>Outlier Suppression</td>
<td>31.7%</td>
<td>54.1%</td>
<td>63.5%</td>
</tr>
<tr>
<td>SmoothQuant</td>
<td><strong>71.2%</strong></td>
<td><strong>68.3%</strong></td>
<td><strong>73.7%</strong></td>
</tr>
</tbody>
</table>

- SmoothQuant well maintains the accuracy without finetuning.
- SmoothQuant can both accelerate inference and halve the memory footprint.

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models (Xiao et al., 2022)
SmoothQuant
Advancing new efficient open model LLaMA

- LLaMA (and its successors like Alpaca) are popular open-source LLMs, which introduced SwishGLU, making activation quantization even harder
- SmoothQuant can losslessly quantize LLaMA families, further lowering the hardware barrier

<table>
<thead>
<tr>
<th>PIQA↑</th>
<th>LLaMA 7B</th>
<th>LLaMA 13B</th>
<th>LLaMA 30B</th>
<th>LLaMA 65B</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP16</td>
<td>78.24%</td>
<td>79.05%</td>
<td>80.96%</td>
<td>81.72%</td>
</tr>
<tr>
<td>SmoothQuant</td>
<td>78.24%</td>
<td>78.84%</td>
<td>80.74%</td>
<td>81.50%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wikitext↓</th>
<th>LLaMA 7B</th>
<th>LLaMA 13B</th>
<th>LLaMA 30B</th>
<th>LLaMA 65B</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP16</td>
<td>11.51</td>
<td>10.05</td>
<td>7.53</td>
<td>6.17</td>
</tr>
<tr>
<td>SmoothQuant</td>
<td>11.69</td>
<td>10.31</td>
<td>7.71</td>
<td>6.68</td>
</tr>
</tbody>
</table>

W8A8 per token
Same Principle, Diverse Applications

Applications

- Large Language Model
- Generative AI
- Advanced Driver Assistance System
- TinyML

Techniques

- Hardware-aware NAS
- Pruning & Sparsity
- Quantization
- Distillation
- New Primitive
### Background: The Era of AIoT on Microcontrollers

**Smart Retail** | **Personalized Healthcare** | **Smart Home** | **Precision Agriculture**
--- | --- | --- | ---

- **Problem:** restricted memory size

<table>
<thead>
<tr>
<th></th>
<th>Cloud AI</th>
<th>Mobile AI</th>
<th>Tiny AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory (Activation)</td>
<td>32GB</td>
<td>4GB</td>
<td>320kB</td>
</tr>
<tr>
<td>Storage (Weights)</td>
<td>~TB/PB</td>
<td>256GB</td>
<td>1MB</td>
</tr>
</tbody>
</table>
MCUNet
Deploy AI on MCUs that has only 256KB SRAM

The camera is OpenMV Cam.

Face/mask detection

Person detection
Inference Is Good. Can We Learn on Edge?

AI systems need to continually adapt to new data collected from the sensors. Not only inference, but also training.

- On-device learning: **better privacy, lower cost, customization, life-long learning**
- Training is more **expensive** than inference, hard to fit edge hardware (limited memory)

The camera is OpenMV Cam.
Sparse Training
Only update important layers and sub-tensors to save memory

- Sensitivity analysis
  - Later layers are more important
  - The first point-wise conv in each block contributes more

- Detailed Update Scheme for MobileNetV2
  - The activation cost is high for the early layers;
  - The weight cost is high for the later layers;
  - The overall memory cost is low for the middle layers.
    - Bias-only update
    - Update weights for the middle layers

On-Device Training Under 256KB Memory [Lin et al., NeurIPS 2022]
Low-Precision Training with Quantization Aware Scaling (QAS)

- Optimizing an INT8 quantized graph leads to memory and computing savings
  - All weights and activations are in INT8
  - Different from quantization-aware training (QAT), where operations are performed in FP16

- ... But at the cost of worse convergence
- We found the issue lies in gradient scale mismatch

Solution: quantization-aware scaling (QAS)
\[
W = s_W \cdot (W/s_W) \approx s_W \cdot W_Q, \quad G_{W_Q} \approx s_W \cdot G_W
\]

weight and gradient ratios are off by \( s_W^{-2} \)

\[
\|W_Q\|/\|G_{W_Q}\| \approx \|W/s_W\|/\|s_W \cdot G_W\| = s_W^{-2} \cdot \|W\|/\|G\|
\]

Thus, we need to re-scale the gradients \( G'_{W_Q} = G_{W_Q} \cdot s_W^{-2} \)

QAS aligns the W/G ratio

On-Device Training Under 256KB Memory [Lin et al., NeurIPS 2022]
Tiny Training Engine

Translate the theoretical savings into measured savings. 10x faster and smaller!

Device: Jetson Nano; Backend: Tiny Training Engine; Task: Speech Recognition
Model Compression for Diverse Applications

- Predictive Maintenance
- Gesture Recognition
- Video Recognition
- Health Monitoring

- Art Generation
- Question Answering
- Storytelling
- Music Composition
- Fashion Design

- Search Engine Revolution
- Augmented Reality
- Autonomous Driving
- Machine Translation
- Video Synthesis
- Sentiment Analysis
- Blind Spot Detection
- Chatbots
- Augmented Reality
- Video Synthesis
- Sentiment Analysis

Application (demand of computation)

Hardware (supply of computation)
TinyML and Efficient AI

HAN Lab Students: Yujun Lin (Arch PhD), Hanrui Wang (Arch PhD), Zhijian Liu (ML PhD)

Potential product impact for NVIDIA: future TensorRT and cuDNN libraries.

Project 1: "Efficient Training/Inference"  (EIE) [Han'16] is a first hardware accelerator for sparse DNN, it's efficient but it lacks flexibility. TACO [Kjolstad'17] is a flexible compiler for sparse linear algebra on CPU, but it lacks accelerator support. Therefore, I plan to work on an specialized accelerator for sparse linear algebra. There are two basic operations to be accelerated: union (OR) and join (AND). Software implementation need O(n) cycles. I plan to work on O(log(n)) time complexity, O(n^2) space complexity arrays. For hardware implementation (need register lookup), O(n) time complexity, O(1) area complexity arrays; or O(1) time complexity, O(n) space complexity arrays. Hence, I will use a combination of both methods to design a hardware accelerator for sparse linear algebra. This accelerator will have O(n) time complexity and O(1) area complexity.

Project 2: "Optimal Number Representation for EIE"  EIE is an efficient but it lacks flexibility. TACO is a flexible compiler for sparse linear algebra on CPU, but it lacks accelerator support. Therefore, I plan to work on an specialized accelerator for sparse linear algebra. There are two basic operations to be accelerated: union (OR) and join (AND). Software implementation need O(n) cycles. I plan to work on O(log(n)) time complexity, O(n^2) space complexity arrays. For hardware implementation (need register lookup), O(n) time complexity, O(1) area complexity arrays; or O(1) time complexity, O(n) space complexity arrays. Hence, I will use a combination of both methods to design a hardware accelerator for sparse linear algebra. This accelerator will have O(n) time complexity and O(1) area complexity.

Potential product impact for NVIDIA: future DLA architectures in Xavier, Orin, etc.

Efficient Hardware Primitives for Sparse Linear Algebra

Software implementation need O(n) cycles. I plan to work on O(log(n)) time complexity, O(n^2) space complexity arrays. For hardware implementation (need register lookup), O(n) time complexity, O(1) area complexity arrays; or O(1) time complexity, O(n) space complexity arrays. Hence, I will use a combination of both methods to design a hardware accelerator for sparse linear algebra. This accelerator will have O(n) time complexity and O(1) area complexity.

"Number representation" is a fundamental problem for efficient machine learning. For training, inference, Linear Quantization [TensorRT] or Kmeans Quantization [Han'16] are two choices for deep learning. The design space include:

- [training, inference] x [channel number] x [layer number] x [bit width] x [decimal point]
- [weight, activation, gradient] x [linear quantization, log quantization, kmeans quantization]

This is a large design space that is hard to be explored by human. It should be explored by AI. I plan to use machine learning techniques to find the best number representation for machine learning. It's a co-design of number representation together with model expressiveness. The former has easy hw implementation but poor inference, Linear Quantization [TensorRT] or Kmeans Quantization [Han'16] are two choices. The latter has ine...

granularity of sparsity that fits the accelerator. Lastly, I'll demonstrate a few machine learning models that are not only pruned to be sparse, but also with the optimal complexity arrays. After that, I'd like to implement this architecture in FPGA or ASIC, then integrate the HW primitive into TACO. Then, I want to co-design the machine learning application with such sparse primitives: machine translation, speech recognition, image classification, and Progressive GAN, which makes real-time learning applications accelerated with such sparse primitives.