Model Reduction

Pruning | Quantization | Distillation

Group 1

Why Reduce Models?

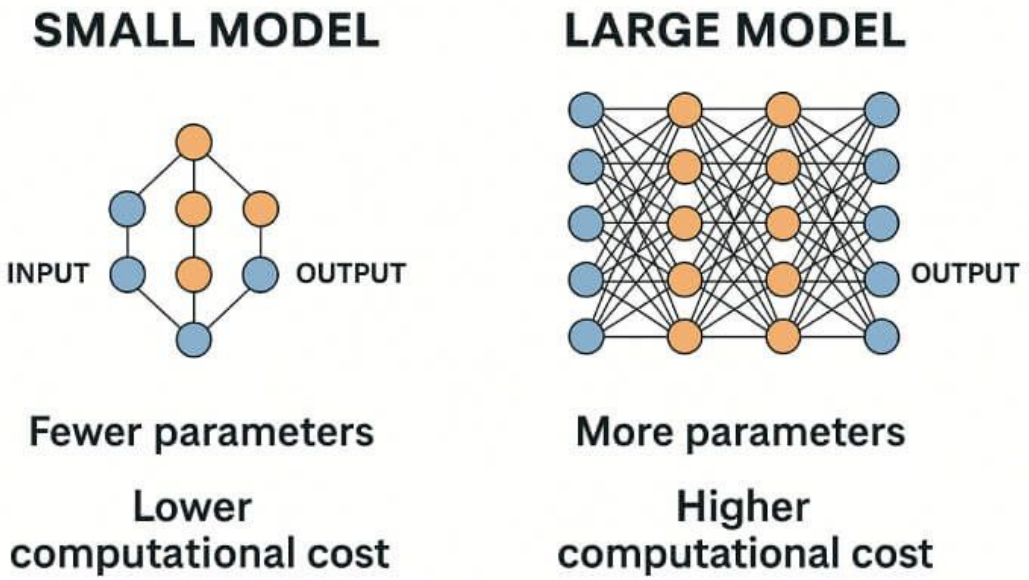
- Memory constraints
- Faster inference
- Lower energy usage
- Real-world deployment (mobile, IoT)

Model Reduction

Pruning | Quantization | Distillation

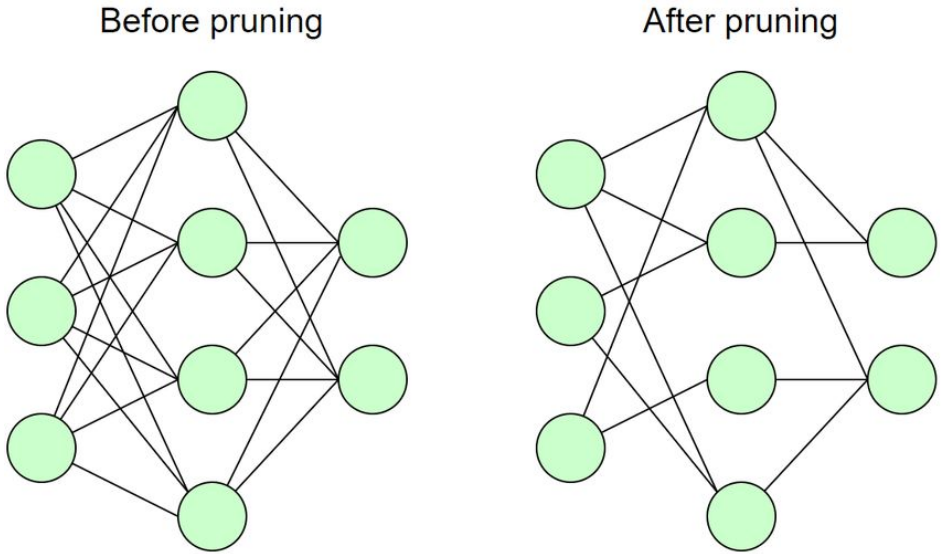
Why Pruning?

- Memory
- Speed
- Energy



What is Pruning?

- Remove unimportant weights



Lottery Ticket Hypothesis

Finding Sparse, Trainable Neural Networks

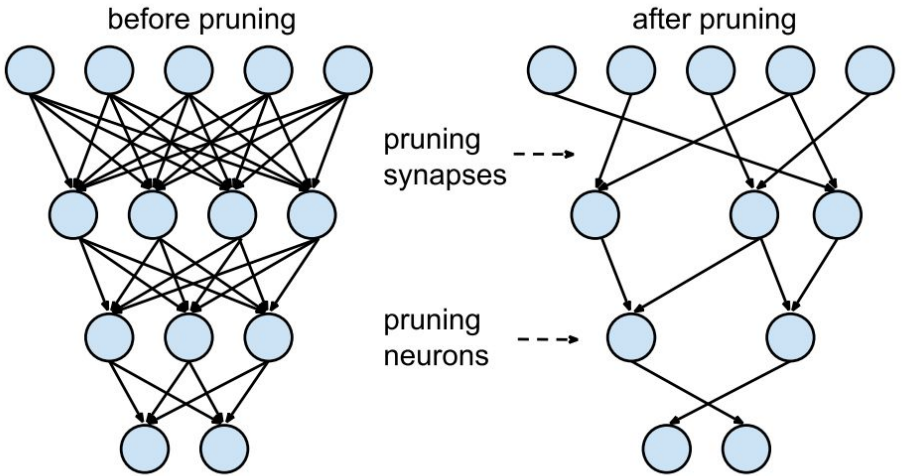
Jonathan Frankle, Michael Carbin

MIT CSAIL

ICLR 2019

Lottery Ticket Hypothesis

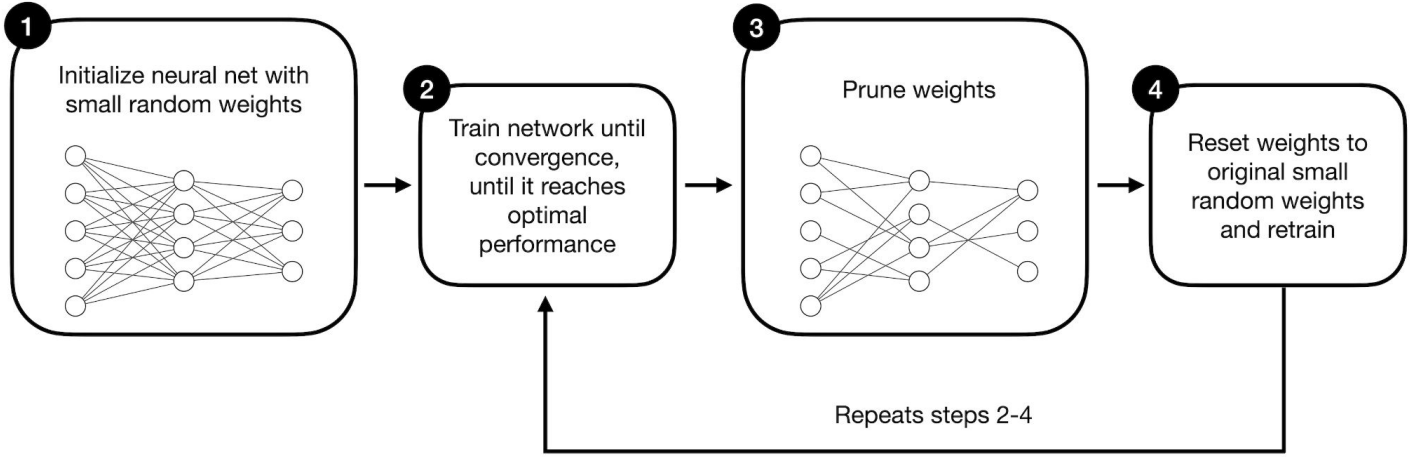
- Small subnetworks exist inside large models
- Can match full model accuracy



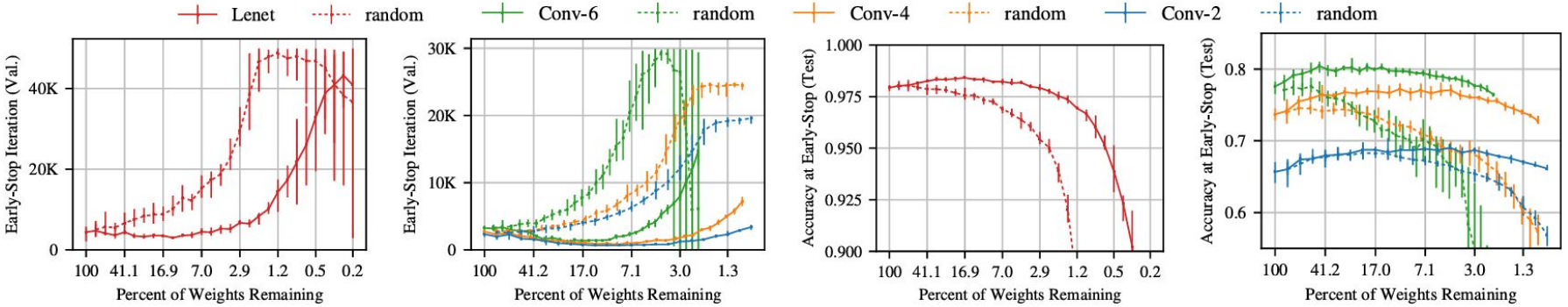
The Lottery Ticket Hypothesis. A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

[1] “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”

Lottery Ticket Hypothesis



Lottery Ticket Hypothesis



Key Results

- High sparsity possible
- Accuracy maintained
- Better than random subnetworks

[1] “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”

DTMM

Deploying TinyML Models on Extremely Weak IoT Devices with Pruning

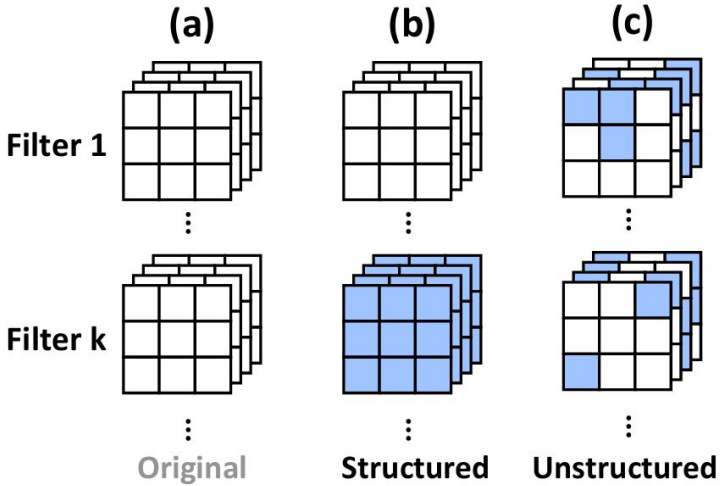
Lixiang Han, Zhen Xiao, Zhenjiang Li

Department of Computer Science, City University of Hong Kong, China

IEEE INFOCOM 2024

Limitation of Pruning

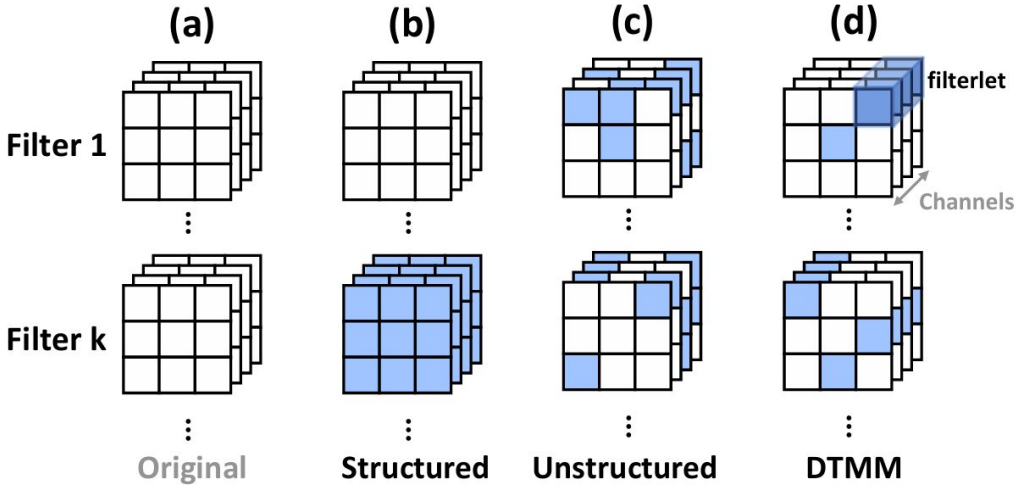
- Structured → efficient, less accurate
- Unstructured → accurate, inefficient



[2] “DTMM: Deploying TinyML Models on Extremely Weak IoT Devices with Pruning”

Limitation of Pruning

- Structured → efficient, less accurate
- Unstructured → accurate, inefficient



[2] “DTMM: Deploying TinyML Models on Extremely Weak IoT Devices with Pruning”

Deploying TinyML Models

Real-World Constraints

- Limited memory
- Limited compute
- Need fast inference

Deploying TinyML Models

DTMM Approach

- Introduces filterlets between structured and unstructured pruning
- Balances efficiency and accuracy

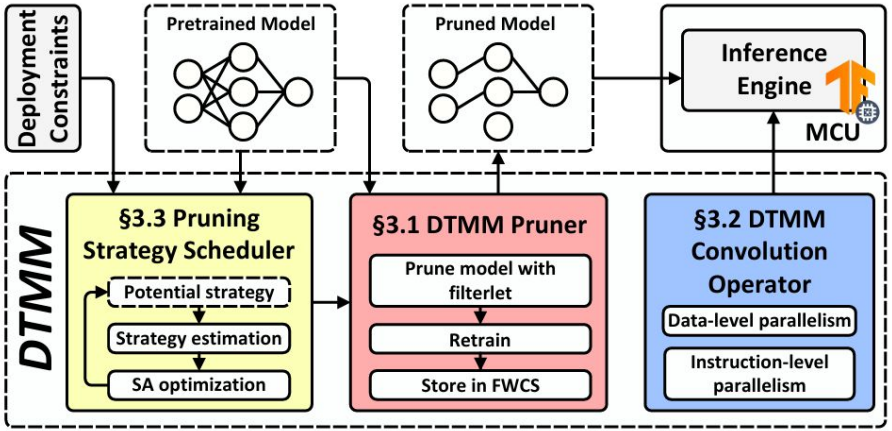


Fig. 2. Overview of the DTMM design.

Deploying TinyML Models

- Smaller model size
- Lower latency
- Fits memory constraints
- Maintains accuracy

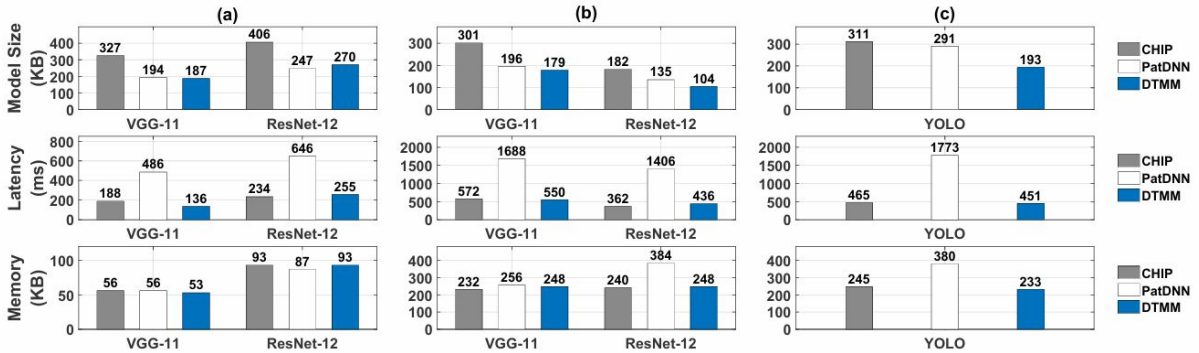


Fig. 11. Overall performance of (a) VGG-11 and ResNet-12 models on CIFAR-10, (b) VGG-11 and ResNet-12 models on VWW, and (c) YOLO model on FDDB. The latency is measured when the processor operates at 120 MHz.

Takeaways

Pruning Takeaways

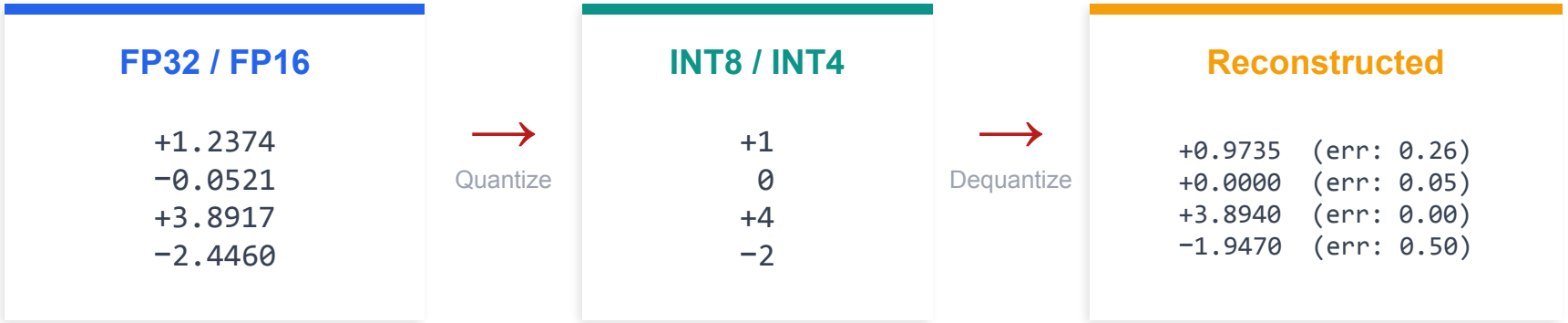
- Removes unnecessary weights
- Lottery Ticket: small subnetworks work
- Tradeoff: accuracy vs efficiency
- DTMM improves real-world deployment

Model Reduction

Pruning | **Quantization** | Distillation

What is Quantization?

Map high-precision values (FP32/FP16) to lower-precision integers (INT8/INT4)



Current Quantization Landscape

When?	What?	How?
QAT	**W8A8**	**Per-tensor**
Quantization-Aware Training — retrain with fake quantization. Best accuracy, high cost.	Both weights and activations quantized to INT8. Good for compute-bound (batched) workloads.	One scale for the entire tensor. Simplest but lowest accuracy.
PTQ	**W4A16**	**Group-wise**
Post-Training Quantization — no retraining. Fast, practical for LLMs (AWQ, GPTQ, RTN).	Weights only to INT4, activations stay FP16. Best for memory-bound (edge, low-batch) workloads.	One scale per group of channels (e.g., 128). Best accuracy–compression trade-off.

Why Quantize LLMs?

- Memory Reduction
  - FP16 → INT4 cuts model size by 4×
  - A 70B model goes from 140 GB to 35 GB, fits on a single GPU.
- Inference Speedup
  - Less memory traffic → higher effective throughput
  - Edge LLMs are memory-bound, so smaller weights = faster generation.
- Cost Savings
  - Fewer GPUs per model, lower cloud serving costs
  - Enables deployment on consumer hardware (laptops, phones, IoT).

AWQ

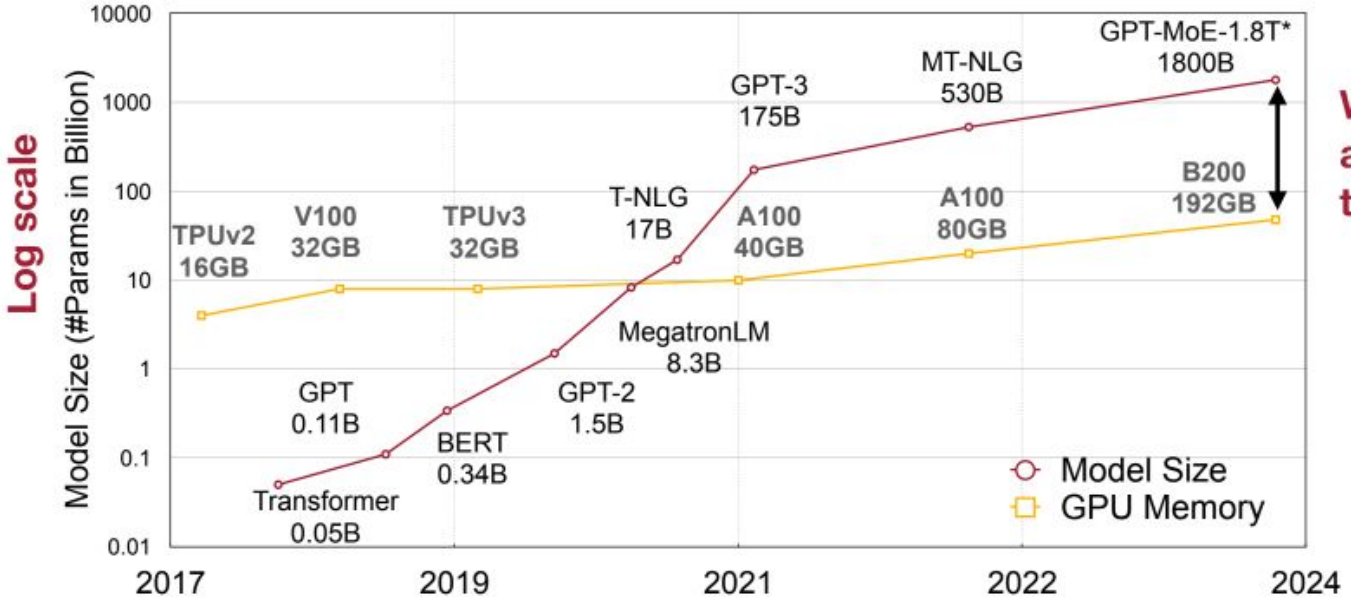
Activation-aware Weight Quantization for LLM Compression and Acceleration

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, et al

MIT | NVIDIA

MLSys 2024 Best Paper

The Challenge: LLMs Are Massive

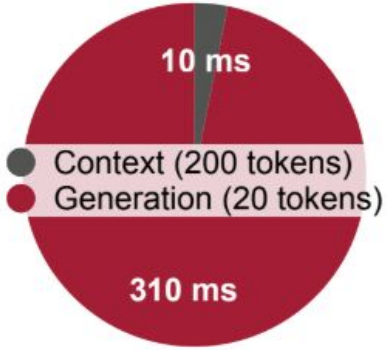


We need efficient algorithms and systems to bridge the gap.

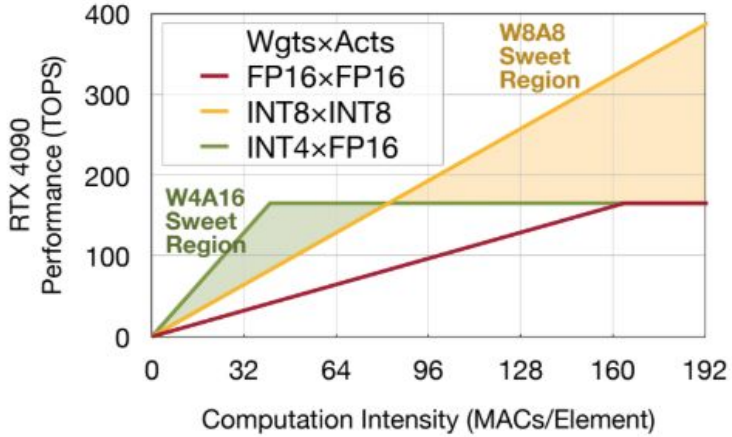
Figure(s) from slides at https://hanlab.mit.edu/projects/awq

[3] "AWQ: Activation-aware weight quantization for LLM compression and acceleration"

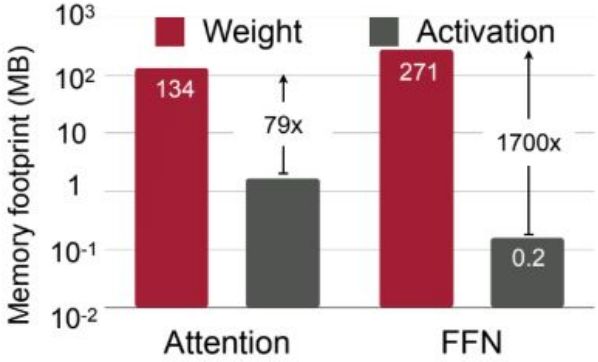
Low-bit weight quantization brings speedup



(a) Generation stage is slower



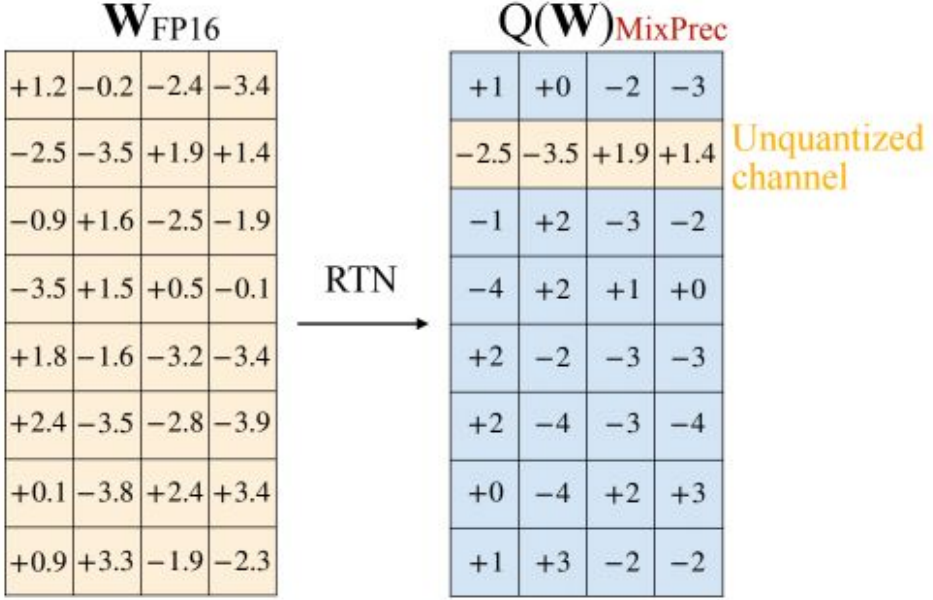
(b) Generation stage is bounded by memory bandwidth

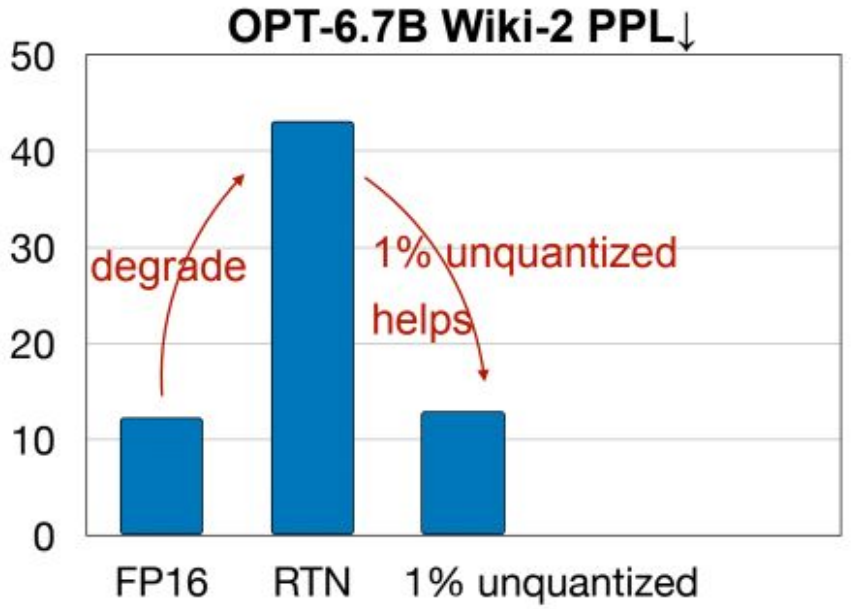


(c) Weight loading is more expensive

Figure(s) from slides at https://hanlab.mit.edu/projects/awq

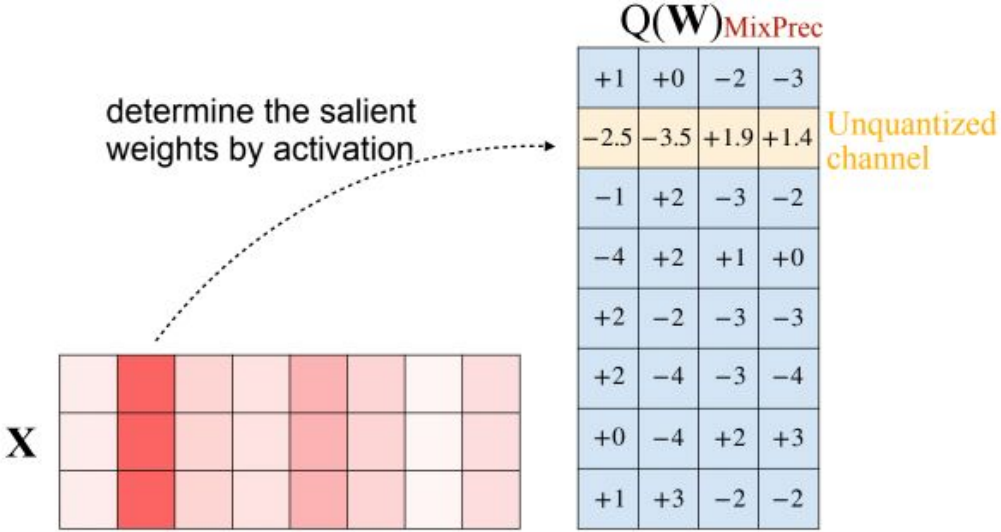
Key Insight: All weights are not equally important

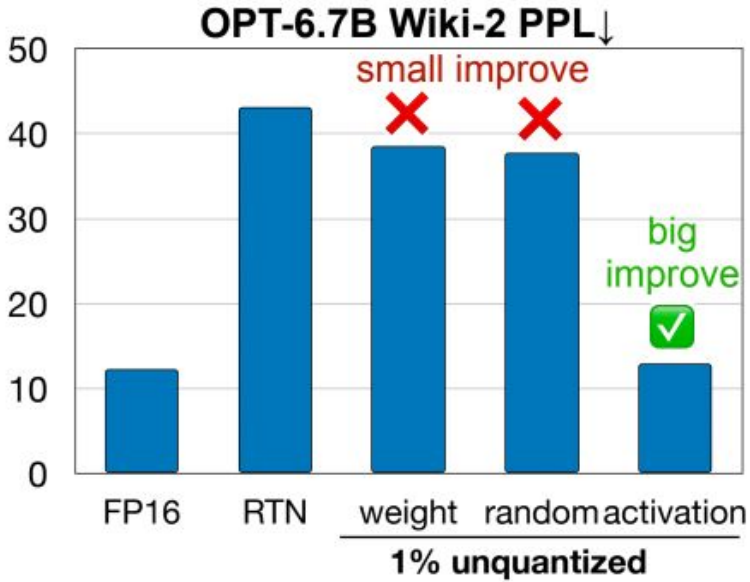




Figure(s) from slides at https://hanlab.mit.edu/projects/awq

How do we select salient channels?





Figure(s) from slides at https://hanlab.mit.edu/projects/awq

Results

PPL↓		Llama-2			LLaMA			
		7B	13B	70B	7B	13B	30B	65B
FP16	-	5.47	4.88	3.32	5.68	5.09	4.10	3.53
INT3 g128	RTN	6.66	5.52	3.98	7.01	5.88	4.88	4.24
	GPTQ	6.43	5.48	3.88	8.81	5.66	4.88	4.17
	GPTQ-R	6.42	5.41	3.86	6.53	5.64	4.74	4.21
	AWQ	**6.24**	**5.32**	**3.74**	**6.35**	**5.52**	**4.61**	**3.95**
INT4 g128	RTN	5.73	4.98	3.46	5.96	5.25	4.23	3.67
	GPTQ	5.69	4.98	3.42	6.22	5.23	4.24	3.66
	GPTQ-R	5.63	4.99	3.43	5.83	5.20	4.22	3.66
	AWQ	**5.60**	**4.97**	**3.41**	**5.78**	**5.19**	**4.21**	**3.62**

Figure(s) from slides at https://hanlab.mit.edu/projects/awq

[3] "AWQ: Activation-aware weight quantization for LLM compression and acceleration"

TinyChat: LLM Inference on Edge

On-the-fly Dequantization

Fuse INT4→FP16 conversion into GEMM kernels. No intermediate DRAM writes.

SIMD-aware Weight Packing

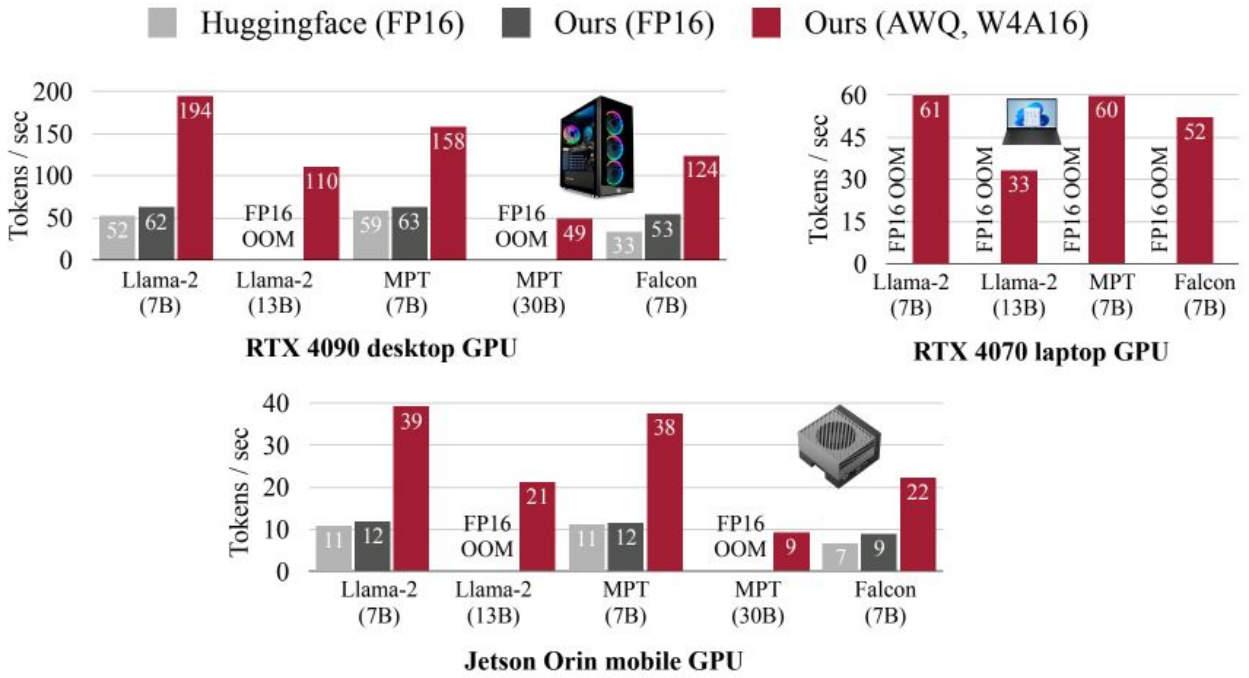
Reorder 4-bit weights to decode 32 weights with only 3 logical ops. 1.2× speedup on ARM.

Kernel Fusion

Fuse LayerNorm, QKV projections, positional embeddings, KV cache updates into single kernels.

[3] "AWQ: Activation-aware weight quantization for LLM compression and acceleration"

TinyChat significantly accelerates edge LLMs



Figure(s) from slides at https://hanlab.mit.edu/projects/awq

[3] "AWQ: Activation-aware weight quantization for LLM compression and acceleration"

Model Reduction

Pruning | Quantization | **Distillation**

Distilling the Knowledge in a Neural Network

Geoffrey Hinton, Oriol Vinyals, Jeff Dean

Google

Mismatch between Training and Deployment

Training
- large redundant datasets
- need to extract structure and patterns
- no constraint on latency or compute

Deployment
- must be fast and efficient to serve large amount of users
- latency is a huge consideration

The solution?

Train a large model to capture those complex patterns and structures, then transfer that knowledge to a smaller model.

[5] "*Distilling the knowledge in a neural network*"

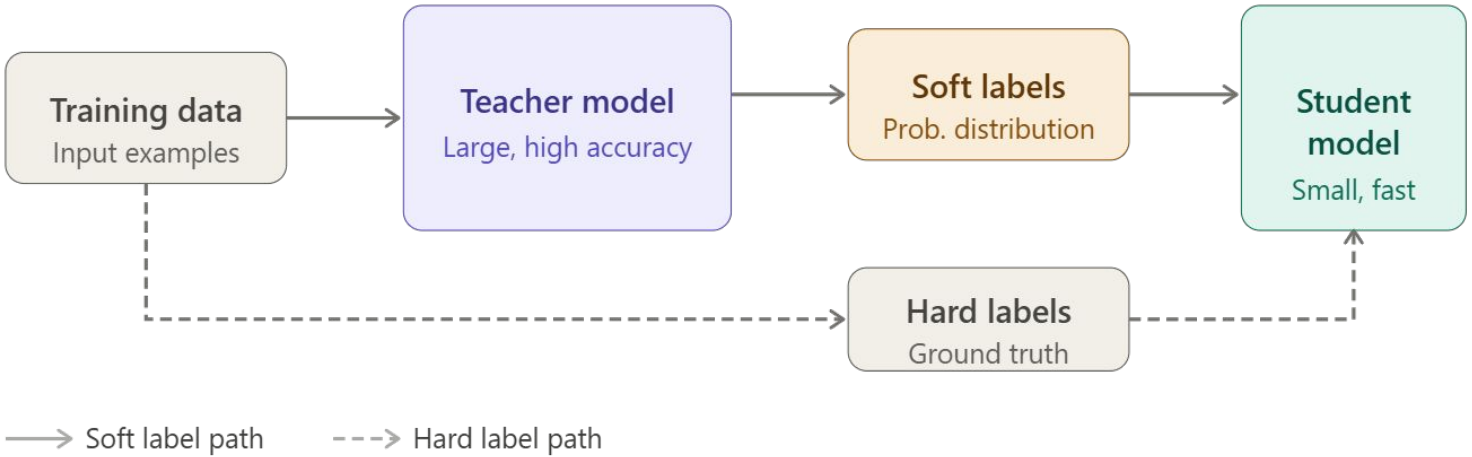
Intuition Behind Distillation

Models learn a mapping between input vectors and output vectors.

The relative probabilities of incorrect answer tells us how the teacher model generalizes.

Rather than optimizing for correct labels alone, distillation aligns the student's objective with the teacher's learned understanding of the data.

[5] "*Distilling the knowledge in a neural network*"



[5] “Distilling the knowledge in a neural network”

MNIST Classification

Baseline Distillation

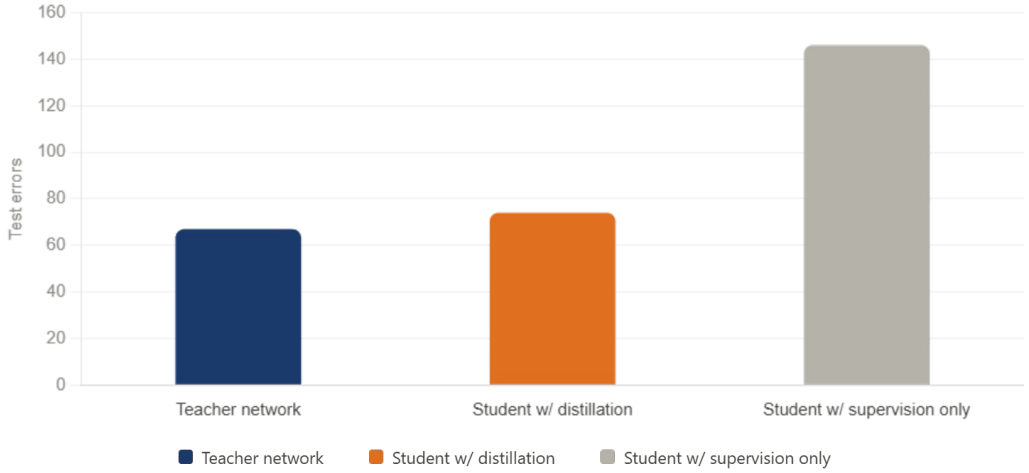
- Teacher Network: 67 test errors
- Student Network w/ Distillation: 74 test errors
- Student Network w/ Supervised Training: 146 errors

Omitting the Number 3

- 98.6% accuracy on test 3s
- this is with a scalar bias adjustment

Only Training on 7 and 8

- 47.3% → 13.2% test error with correction



Distillation on Speech Recognition

DNN

- input audio frames
- outputs probability distribution over possible speech sounds
- decoder assembles this into words
- 85M parameter model
- Trained on 2000 hours of spoken english data (700M training examples)

System	Test Frame Accuracy	WER
Baseline	58.9%	10.9%
10xEnsemble	61.1%	10.7%
Distilled Single model	60.8%	10.7%

Table 1: Frame classification accuracy and WER showing that the distilled single model performs about as well as the averaged predictions of 10 models that were used to create the soft targets.

[5] “Distilling the knowledge in a neural network”

Key Takeaways

Distillation works

- Student models can successfully learn to generalize from a teacher model's soft labels

Knowledge transfers without complete data

- A student model can learn to recognize classes it never directly trained on, because soft targets from related classes leak structural information about missing ones.

Soft targets are richer than hard labels

- The probability distributions produced by a trained model encode learned relationships between classes, carrying more signal than a one-hot label.

[5] “Distilling the knowledge in a neural network”

An Application of Knowledge Distillation: Federated Learning

FedKD: Communication Efficient Federated Learning via Knowledge Distillation

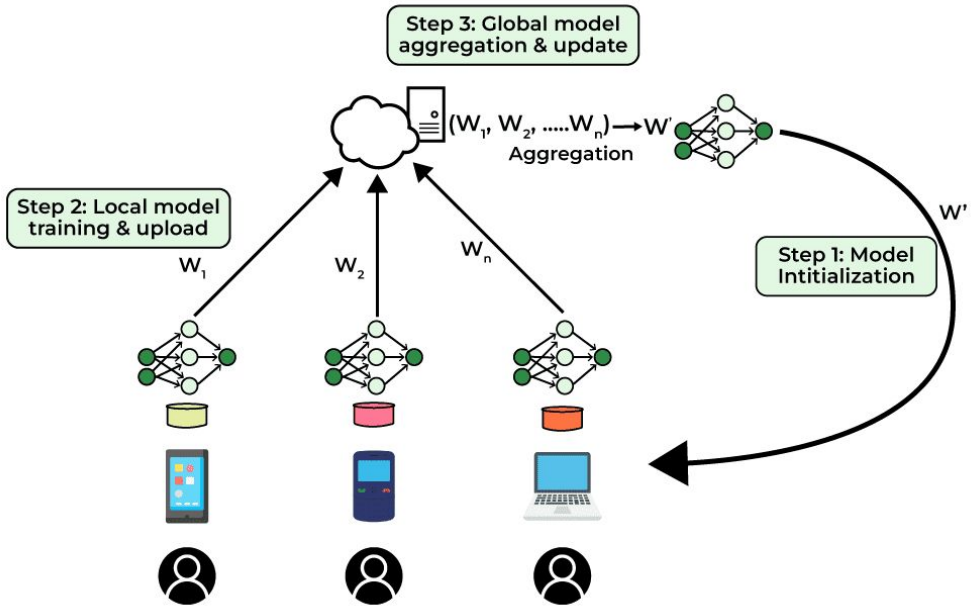
Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, Xing Xie

Microsoft | *Sony AI*

Federated Learning

Way of training models across devices without transferring data

- Global models are downloaded on device
- Model trains locally
- Only model updates are uploaded
- Central server aggregates local updates to update global model



[6] "*FedKD: Communication efficient federated learning via knowledge distillation*"

Problem Definition

Problem: In federated learning, clients must repeatedly exchange model updates with a central server. When models are large, this communication cost becomes prohibitively expensive.

Core Idea: Instead of communicating updates for the full large model, FedKD has each client maintain two models: a large teacher and a small student. Only the small student model's updates are shared across clients, drastically cutting communication costs. The teacher stays local and never leaves the client.

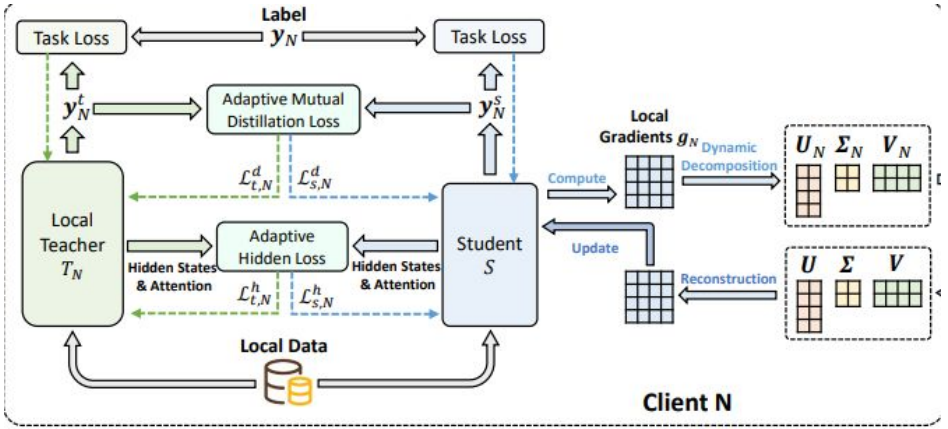
FedKD

Each client stores its own private data locally.

Every client runs two models: a large teacher (private, stays local) and a small student (shared across all clients).

A central server coordinates the clients, aggregating student model updates without ever seeing raw data.

Goal: strong model performance with minimal communication cost and full privacy preservation.



Federated Knowledge Distillation

Each round, every client updates both its teacher and student simultaneously using local data and knowledge from each other

Teacher → Student: the large teacher passes down rich knowledge it has learned from local data via soft labels

Student → Teacher: the student passes back global knowledge it has absorbed from all other clients

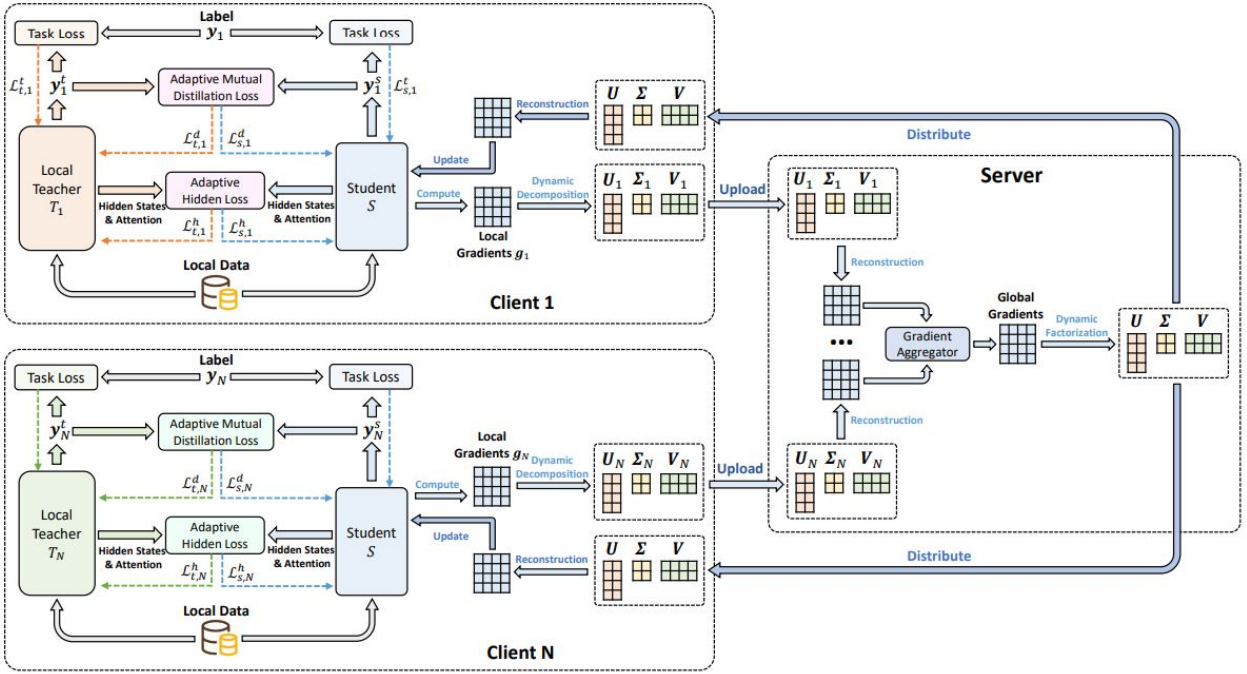


Figure 1: The framework of our *FedKD* approach.

Experiment Setup

Datasets

- MIND - Microsoft News, 1 million users, predicting whether a user will click a news article based on browsing history
- SMM4H - 16,694 tweets, detecting whether a tweet mentions an adverse drug reaction (binary classification)

Setup

- Training data split across 4 clients to simulate real federated learning conditions Teacher model: UniLM-Base (a BERT-scale language model)
- Student models: smaller versions of UniLM using only the first 4 or 2 transformer layers

MIND			
# users	1,000,000	# impressions	15,777,377
# news	161,013	# clicks	24,155,470
avg. title len.	11.52	# training samples	2,186,683
# validation samples	365,200	# test samples	2,341,619
SMM4H			
# tweets	16,694	# positives	1,355
avg. tweet len.	16.48	# negatives	15,336

Table 1: Statistics of the datasets.

[6] "*FedKD: Communication efficient federated learning via knowledge distillation*"

Experiment Results

Methods	AUC	MRR	nDCG@5	nDCG@10	Comm. Cost per Client
UniLM (Local)	68.8±0.5	33.5±0.4	36.6±0.5	42.4±0.6	-
UniLM (Cen)	**71.0**±0.1	**35.8**±0.1	**39.0**±0.1	**44.8**±0.1	-
UniLM (Fed)	70.9±0.3	35.7±0.2	38.9±0.3	44.7±0.4	2.05GB
DistilBERT_6	69.3±0.2	34.0±0.2	37.5±0.2	43.0±0.1	1.03GB
DistilBERT_4	69.0±0.2	33.7±0.1	37.0±0.1	42.6±0.2	0.69GB
BERT-PKD_6	69.6±0.2	34.4±0.3	37.7±0.3	43.4±0.2	1.03GB
BERT-PKD_4	69.2±0.2	33.8±0.2	37.1±0.3	42.9±0.3	0.69GB
TinyBERT_6	69.7±0.2	34.5±0.2	37.9±0.1	43.5±0.2	1.03GB
TinyBERT_4	69.4±0.3	33.9±0.3	37.5±0.2	43.1±0.2	0.17GB
UniLM_4	69.6±0.1	34.4±0.2	37.7±0.1	43.4±0.2	0.69GB
UniLM_2	68.9±0.2	33.6±0.2	36.8±0.2	42.5±0.1	0.35GB
FetchSGD	70.5±0.4	35.2±0.3	38.2±0.3	44.0±0.4	0.51GB
FedDropout	70.5±0.2	35.1±0.2	38.3±0.3	44.2±0.3	1.23GB
FedKD_4	**71.0**±0.1	35.6±0.1	38.9±0.1	**44.8**±0.1	0.19GB
FedKD_2	70.5±0.1	35.3±0.2	38.6±0.1	44.3±0.2	**0.11GB**

Table 2: Performance of different methods on *MIND*.

Methods	Precision	Recall	Fscore	Comm. Cost per Client
UniLM (Local)	53.2±1.3	54.6±1.4	53.9±1.1	-
UniLM (Cen)	**60.3**±0.7	61.6±0.8	**60.8**±0.4	-
UniLM (Fed)	59.1±0.6	62.3±0.6	60.6±0.4	1.37GB
DistilBERT_6	56.8±0.8	59.2±0.8	57.9±0.5	0.69GB
DistilBERT_4	56.5±0.9	58.4±1.1	57.1±0.7	0.46GB
BERT-PKD_6	56.9±0.9	60.4±0.8	58.4±0.6	0.69GB
BERT-PKD_4	56.3±1.1	59.9±0.7	58.0±0.6	0.46GB
TinyBERT_6	57.4±0.8	60.5±0.6	58.6±0.5	0.69GB
TinyBERT_4	57.0±0.7	59.9±1.2	58.3±0.7	0.12GB
UniLM_4	56.1±0.9	60.6±0.9	58.2±0.5	0.46GB
UniLM_2	53.8±0.8	59.1±1.0	56.3±0.6	0.24GB
FetchSGD	57.5±0.9	60.4±1.1	59.0±0.8	0.34GB
FedDropout	57.8±1.0	61.0±0.8	59.4±0.6	0.82GB
FedKD_4	59.4±0.6	**62.8**±0.9	60.7±0.5	0.12GB
FedKD_2	58.2±0.7	62.4±0.9	59.8±0.6	**0.07GB**

Table 3: Performance of different methods on *SMM4H*.

[6] "*FedKD: Communication efficient federated learning via knowledge distillation*"

Adaptive Mutual Distillation

Mutual distillation helps both models

- Removing mutual distillation hurts both teacher and student performance
- The teacher benefits most because the student brings in global knowledge

Adaptive Weighting

- the weighting mechanism that scales distillation by prediction confidence is what separates FedKD from a simpler baseline

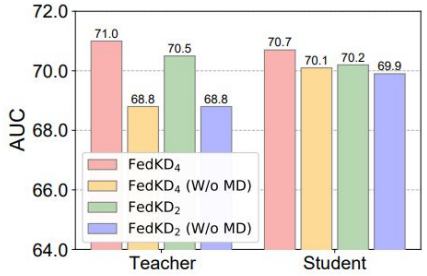


Figure 2: Influence of mutual distillation on the student and teacher models.

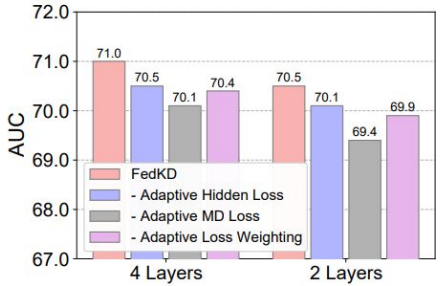


Figure 3: Effect of adaptive mutual distillation.

[6] "*FedKD: Communication efficient federated learning via knowledge distillation*"

Number of Clients

Unlike standard federated learning (FedAvg) which degrades with more clients, FedKD actually improves

Each additional client brings another teacher model with unique local knowledge, enriching the global student

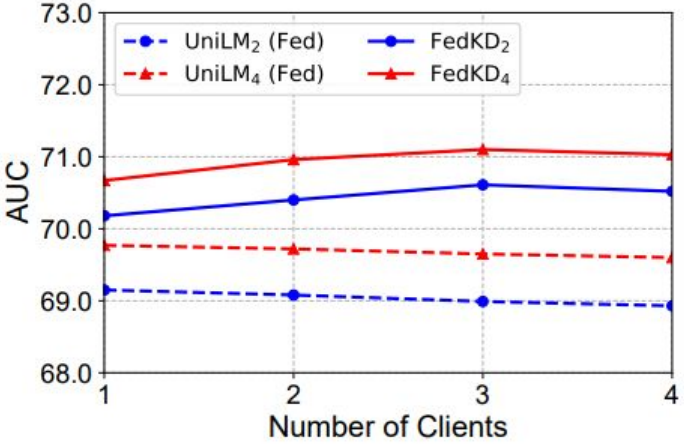


Figure 4: Influence of client number.

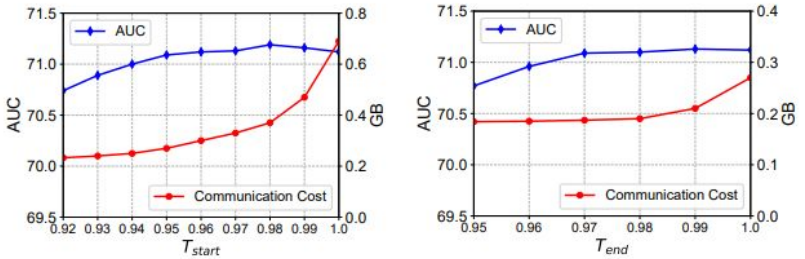
Energy Threshold

Below Tstart = 0.95

- performance drops sharply
- this is the floor below which too much useful gradient information is discarded

Tend = 0.98

- chosen as the sweet spot between accuracy and communication cost at convergence



(a) T_{start}. (b) T_{end}.

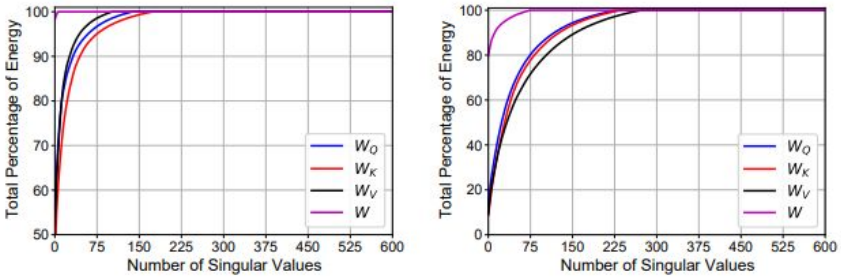
Figure 5: Influence of T_{start} and T_{end} on model performance and communication cost.

SVD Compression

Compression is aggressive early, precise late

Early in training: gradients are dominated by large coarse updates → very few singular values needed → compress heavily

Late in training: gradients become small and subtle → more singular values needed → compress less



(a) Beginning of training. (b) End of training.

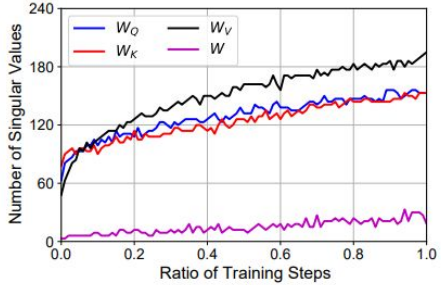


Figure 7: Evolution of the number of required singular values under $T = 0.95$.

[6] "*FedKD: Communication efficient federated learning via knowledge distillation*"

Takeaways

The problem it solves

- Standard federated learning is too expensive to communicate large models
- FedKD cuts this by up to 10× without sacrificing accuracy.

How it works

- Keep the large teacher local, share only the small student, and let them teach each other — the student brings global knowledge back to the isolated teacher.

It actually works at scale

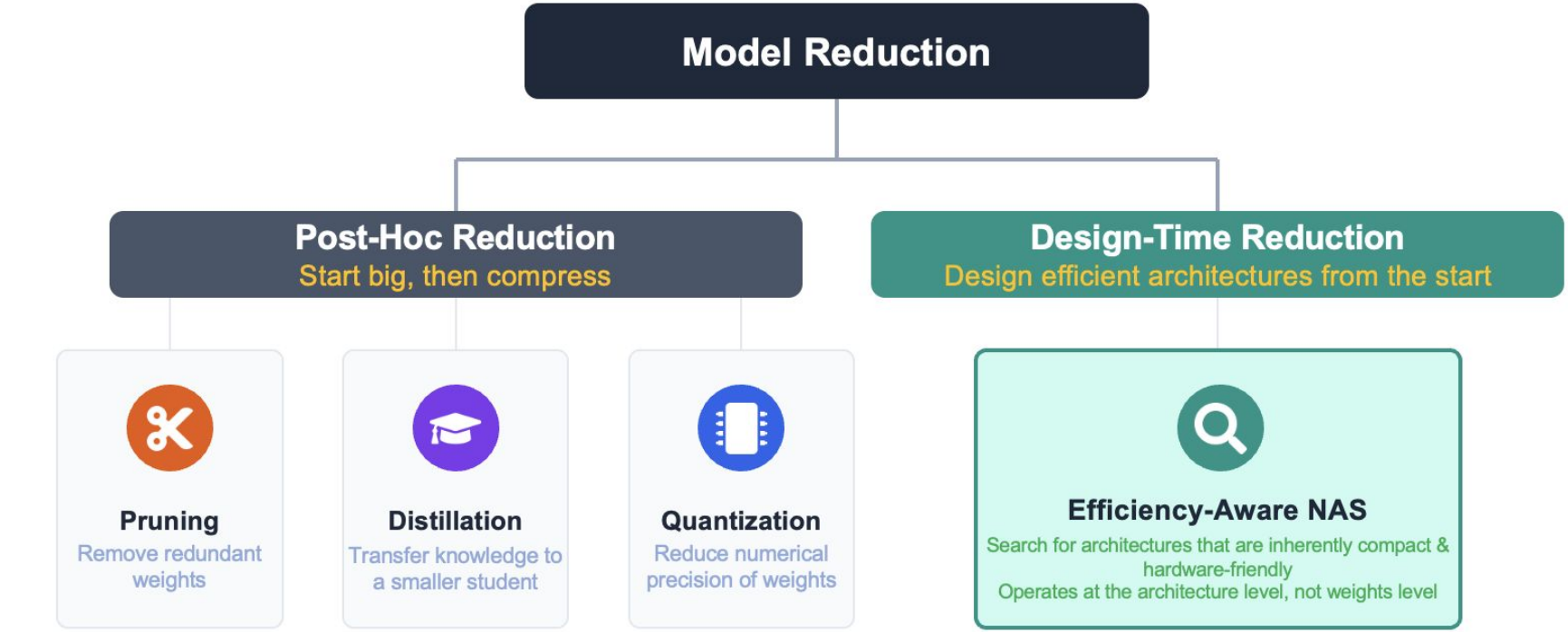
- Matches centralized training quality across two real-world privacy-sensitive tasks (eg. news recommendation and medical tweet detection).

[6] "*FedKD: Communication efficient federated learning via knowledge distillation*"

Neural Architecture Search (NAS)

An Orthogonal but Relevant Model Reduction Method

Beyond Post-Hoc Reduction



Efficiency-Aware NAS

General NAS

Optimizes for accuracy alone. No cost constraints in the search objective.

Efficiency-Aware NAS

Co-optimizes accuracy and a cost metric: latency, FLOPs, memory, or energy.

LANDMARK EXAMPLES

MnasNet (2019)

Introduced measured latency directly into the NAS reward function

EfficientNet (2019)

Compound scaling of depth, width, and resolution via NAS-found base model

Once-for-All (2020)

Train one supernet, extract specialized subnets for diverse hardware targets

Where the Approaches Intersect

Pruning

Can be seen as coarse NAS — deciding which connections to keep

NAS

Found architectures can be further pruned, quantized, or distilled

Distillation

Student architectures are often designed to be small, a NAS-adjacent choice

Quantization

Quantization-aware training reshapes the architecture's numerical precision

MCUNet

Tiny Deep Learning on IoT Devices

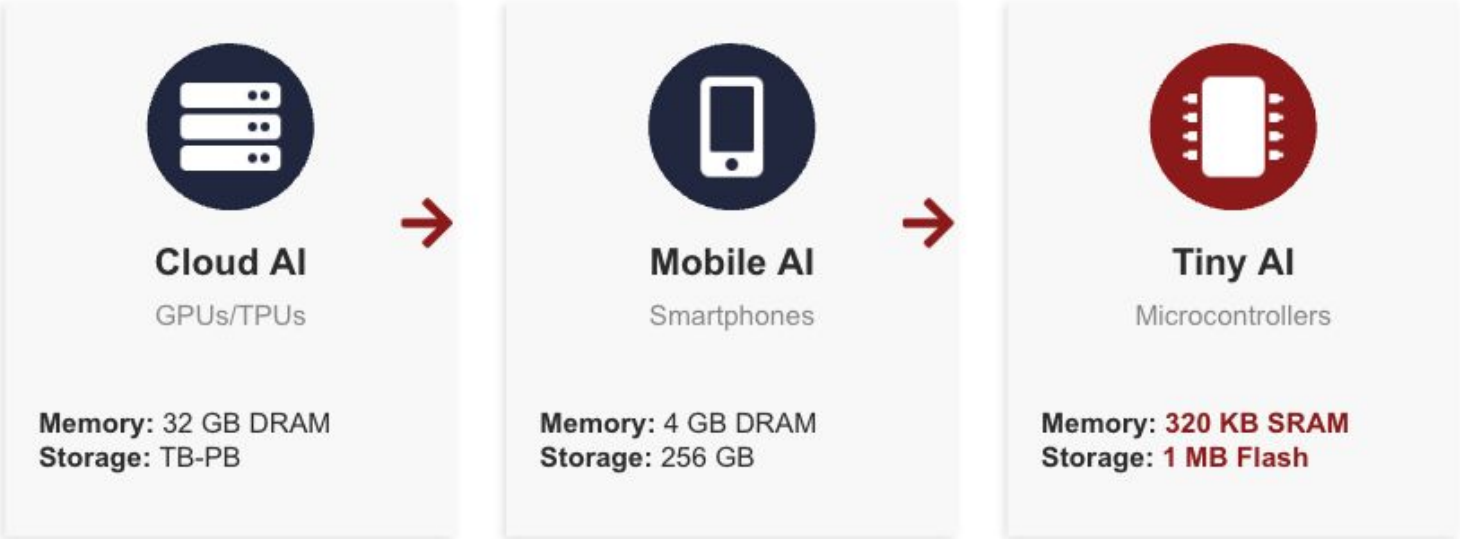
Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, Song Han

MIT | National Taiwan University | MIT-IBM Watson AI Lab

NeurIPS 2020

Why TinyML?

Deep learning is moving from Cloud → Mobile → Microcontrollers (?)



[3] "MCUNet: Tiny Deep Learning on IoT Devices"

The Challenge: Neural Nets Don't Fit

Existing CNNs exceed MCU memory by 5–23x. Even int8 MobileNetV2 needs 5.3x more SRAM than available.

Post-hoc reduction alone isn't enough ! You need model reduction at the architecture level

[3] "MCUNet: Tiny Deep Learning on IoT Devices"

Key Insight: Reduce Weights and Activations

At ~70% ImageNet Top-1 Accuracy (INT8)

Model	Params (MB)	Peak Activation (MB)
ResNet-18	11.2	0.9
MobileNetV2-0.75	2.5 (4.6x ↓)	**1.7 (1.8x ↑!)**
MCUNet	**1.9 (6.1x ↓)**	**0.5 (3.4x ↓)**

MobileNetV2 reduces model size but increases peak activation — making it harder to fit in SRAM. MCUNet reduces both.

How CNNs map to MCU memory:

Flash (Storage) = Model Weights — static, holds entire model

SRAM (Memory) = Activations — dynamic, peak matters

[3] "MCUNet: Tiny Deep Learning on IoT Devices"

MCUNet: System-Algorithm Co-Design

Joint optimization of neural architecture + inference engine

TinyNAS

1. Automated Search Space Optimization
Scale resolution R and width W to fit MCU constraints

2. Resource-Constrained Model Specialization
One-shot NAS with weight sharing under memory budget

TinyEngine

1. Code Generation (not interpretation)
Eliminates runtime memory overhead

2. Model-Adaptive Memory Scheduling
Global topology-aware, not layer-wise

3. In-place Depthwise Convolution
Reduces activation memory from 2N → N+1

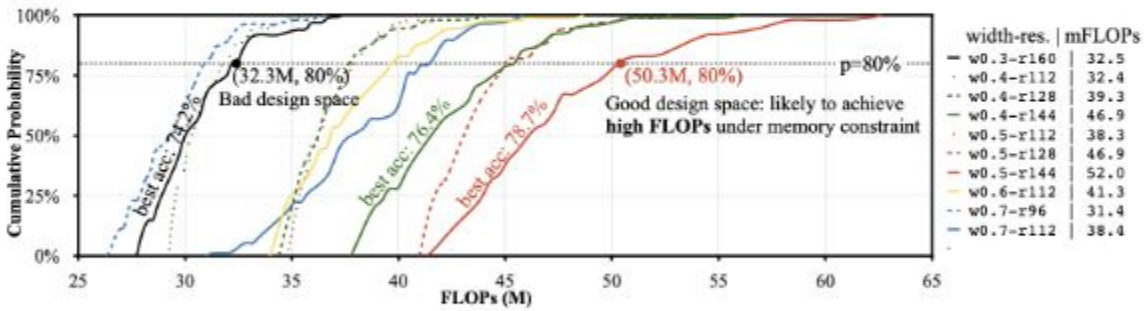
Co-design: TinyEngine expands the feasible search space → TinyNAS finds better architectures

[3] "MCUNet: Tiny Deep Learning on IoT Devices"

TinyNAS: Automated Search Space Optimization

Problem: Mobile NAS spaces (e.g., MnasNet) are too large for MCUs. How to pick the right (R, W)?

Solution: Analyze FLOPs distribution of satisfying models across candidate spaces



Core Insight

Higher mean FLOPs
→ larger model capacity
→ higher accuracy

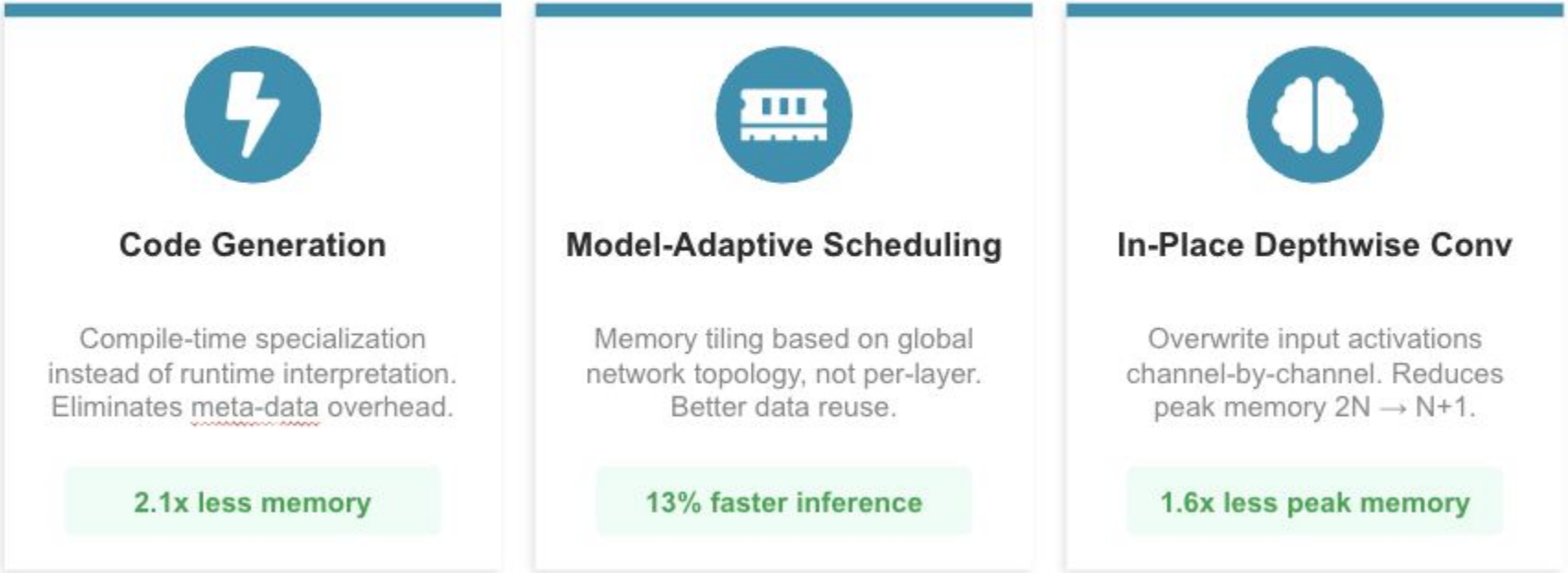
Best space: 78.7%
Bad space: 74.2%

+4.5% accuracy gap

Finding: More SRAM → higher resolution; more Flash → wider channels

[3] "MCUNet: Tiny Deep Learning on IoT Devices"

TinyEngine: Memory-Efficient Inference



[3] "MCUNet: Tiny Deep Learning on IoT Devices"

Results

70.7%

ImageNet Top-1 on STM32H743
First >70% on a commercial MCU

Visual Wake Words

3.4x faster

3.7x smaller peak SRAM

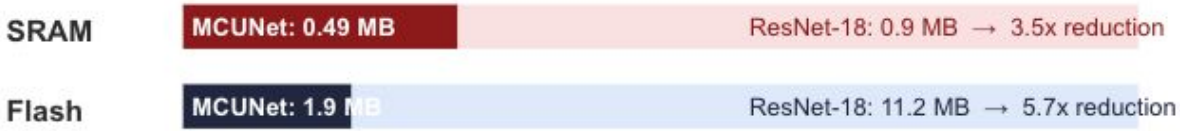
vs. MobileNetV2 at higher accuracy

Speech

2.8x faster

4.1x less SRAM
+2% accuracy

Resource Savings vs. ResNet-18 / MobileNetV2 at ~70% Accuracy:



[3] "MCUNet: Tiny Deep Learning on IoT Devices"

Takeaways

- TinyML is fundamentally different from mobile AI — memory is the bottleneck, not latency
- System-algorithm co-design (TinyNAS + TinyEngine) is key: neither alone is sufficient
- Automated search space optimization selects the right (R, W) per device without manual tuning
- First to achieve >70% ImageNet accuracy on commercial MCUs with 3.5x less SRAM

[3] "MCUNet: Tiny Deep Learning on IoT Devices"

Sources Cited

[1] Lottery Ticket Hypothesis. Frankle, J., & Carbin, M. (2018). *The lottery ticket hypothesis: Finding sparse, trainable neural networks*. arXiv preprint arXiv:1803.03635.

[2] DTMM. Han, L., Xiao, Z., & Li, Z. (2024). *DTMM: Deploying TinyML models on extremely weak IoT devices with pruning.* arXiv preprint arXiv:2401.09068.

[3] AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., & Han, S. (2023). *AWQ: Activation-aware weight quantization for LLM compression and acceleration.* arXiv preprint arXiv:2306.00978.

[4] MCUNet: Tiny Deep Learning on IoT Devices. Lin, J., Chen, W.-M., Lin, Y., Cohn, J., Gan, C., & Han, S. (2020). *MCUNet: Tiny deep learning on IoT devices.*

[5] Distillation (Hinton). Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the knowledge in a neural network.* arXiv preprint arXiv:1503.02531.

[6] FedKD. Wu, C., Wu, F., Lyu, L., Huang, Y., & Xie, X. (2021). *FedKD: Communication efficient federated learning via knowledge distillation.* arXiv preprint arXiv:2108.13323.