# Natural Language Processing Course

## Week 11: Efficiency and Deployment

Joerg R. Osterrieder
www.joergosterrieder.com

## Week 11

# Efficiency & Deployment

From Cloud Giants to Pocket-Sized Models

# Why Your Phone Can't Run GPT-4 (Yet)

**The shocking reality of modern models:**

- GPT-3: 175B parameters = 350GB in FP16[1]
- Your phone: 6GB RAM
- Inference cost: $0.06 per 1K tokens
- Latency: 100ms+ per token
- Energy: 0.1 kWh per conversation

One ChatGPT query = 10x the energy of a Google search

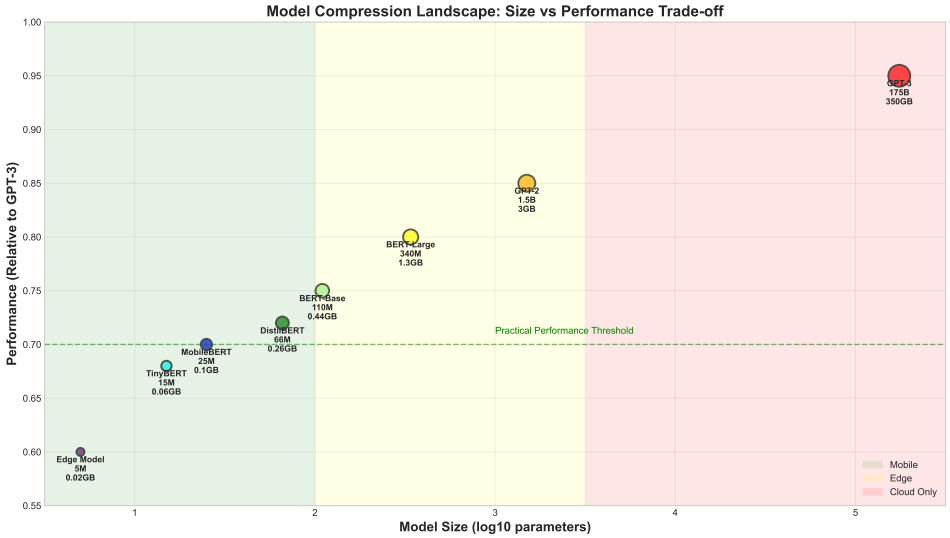
**The challenge:**

- Users want instant responses
- Privacy requires on-device processing
- Costs are unsustainable at scale
- Edge devices have limited resources

*This week: How to shrink giants into pocket-sized assistants*

[1] Brown et al. (2020); Strubell et al. (2019) on model carbon footprint

# From Supercomputers to Smartphones: The Deployment Journey




**The compression toolkit:**

# Efficient Models in Production (2024)

**Success Stories:**

- iPhone: On-device Siri (5B $\rightarrow$ 200M)[2]
- Google: Pixel voice typing
- Microsoft: Excel formula suggestions
- WhatsApp: Real-time translation
- Tesla: In-car voice commands

**Business Impact:**

- 100x cost reduction
- 10ms latency (from 1s)
- Works offline
- Privacy preserved
- Scales to billions

**Efficiency Techniques:**

- INT8 quantization: Standard
- INT4/Binary: Emerging
- Distillation: 95% performance
- Structured pruning: Hardware-friendly
- Flash attention: Memory efficient

**Deployment Targets:**

- Mobile phones: 2-4GB limit
- Browsers: WebAssembly
- IoT devices: MCUs
- Edge servers: Local processing
- Specialized chips: NPUs/TPUs

2024: Every device runs neural networks - efficiency made it possible

[2] Apple ML Research Blog (2023); Google I/O presentations

# Week 11: What You'll Master

**By the end of this week, you will:**

- **Understand** why models are so large
- **Implement** quantization techniques
- **Master** knowledge distillation
- **Apply** pruning strategies
- **Deploy** models to edge devices

**Core Insight:** 90% of weights do 10% of the work

# Why Are Models So Large? The Overparameterization Mystery

**The paradox:**
Models have way more parameters than necessary!

**Evidence:**

- Lottery Ticket Hypothesis: Small subnetworks work just as well[3]
- Magnitude pruning: Remove 90% weights, maintain accuracy
- Low-rank decomposition: Matrices are redundant
- Quantization: 32 bits $\rightarrow$ 4 bits still works

**Why overparameterize?**

- Easier optimization landscape
- Better generalization (surprisingly!)
- Redundancy provides robustness
- Training dynamics require it

Large models are like rough marble blocks - we can carve out efficient versions

[3] Frankle & Carbin (2019) "The Lottery Ticket Hypothesis"

# Quantization: From Float32 to Int8 and Beyond

**The quantization spectrum:**

- FP32 (32 bits): Training precision
- FP16 (16 bits): Mixed precision training
- INT8 (8 bits): Standard deployment
- INT4 (4 bits): Aggressive compression
- Binary (1 bit): Research frontier

# Implementing Post-Training Quantization

```python
import torch
import torch.nn as nn

def quantize_tensor(x, num_bits=8):
    """Quantize tensor to n bits"""
    qmin = -(2**(num_bits-1))
    qmax = 2**(num_bits-1) - 1

    min_val = x.min()
    max_val = x.max()

    scale = (max_val - min_val) / (qmax - qmin)
    zero_point = qmin - min_val / scale

    q_x = torch.round(x / scale + zero_point)
    q_x = torch.clamp(q_x, qmin, qmax)

    return q_x, scale, zero_point

def dequantize_tensor(q_x, scale, zero_point):
    """Dequantize back to float"""
    return scale * (q_x - zero_point)

class QuantizedLinear(nn.Module):
    def __init__(self, weight, bias, num_bits=8):
        super().__init__()

        self.q_weight, self.w_scale, self.w_zp = quantize_tensor(
              weight, num_bits)
        self.q_weight = self.q_weight.to(torch.int8)

        if bias is not None:
            self.q_bias, self.b_scale, self.b_zp = quantize_tensor(
                  bias, num_bits)
        else:
```

**Quantization Impact:**

- Memory: 4x reduction (INT8)
- Speed: 2-4x on CPUs
- Accuracy: -1% typical
- Energy: 10x savings

**Advanced Techniques:**

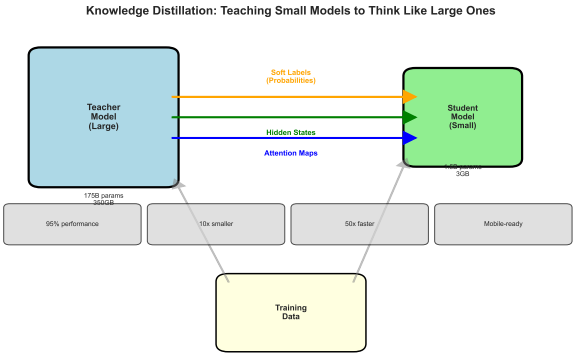
- Symmetric vs asymmetric
- Per-channel quantization
- Quantization-aware training
- Mixed bit-width

**Hardware Support:**

- INT8: All modern chips
- INT4: Newer GPUs/NPUs
- Binary: Research only

# Knowledge Distillation: Teaching Small Models to Think Big

**The teacher-student paradigm:**[4]




**Key insight: Soft labels contain more information**

- Hard label: "cat" (probability = 1.0)
- Soft labels: cat=0.9, tiger=0.05, dog=0.03, ...
- Student learns relationships between classes

# Implementing Knowledge Distillation

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class DistillationLoss(nn.Module):
    def __init__(self, temperature=3.0, alpha=0.7):
        """
        temperature: Softens probability distributions
        alpha: Weight between distillation and true label loss
        """
        super().__init__()
        self.temperature = temperature
        self.alpha = alpha
        self.ce_loss = nn.CrossEntropyLoss()

    def forward(self, student_logits, teacher_logits, labels):
        soft_targets = F.softmax(teacher_logits / self.temperature, dim=-1)
        soft_prob = F.log_softmax(student_logits / self.temperature, dim=-1)

        distillation_loss = F.kl_div(soft_prob, soft_targets,
                                     reduction='batchmean') * (self.temperature
                                           ** 2)

        student_loss = self.ce_loss(student_logits, labels)

        loss = self.alpha * distillation_loss + (1 - self.alpha) * student_loss
        return loss

def train_student(student_model, teacher_model, dataloader,
                  epochs=10, temperature=3.0):
    """Train student to mimic teacher"""
    teacher_model.eval()
    optimizer = torch.optim.Adam(student_model.parameters(), lr=1e-3)
    criterion = DistillationLoss(temperature=temperature)

```

**Distillation Results:**

- Size: 10-100x smaller
- Speed: 5-50x faster
- Accuracy: 95-98% retained
- Examples: DistilBERT, TinyBERT

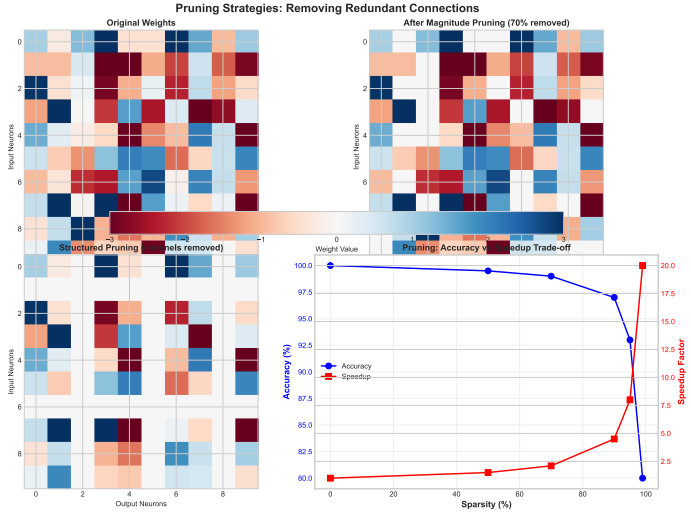
**Temperature Effect:**

- T=1: Normal softmax
- T=3-5: Typical for NLP
- T>10: Very soft labels
- Higher T = more dark knowledge

**Advanced Methods:**

- Feature distillation
- Attention transfer
- Progressive distillation

# Pruning: Finding the Essential Subnetwork




**Pruning approaches:**

- Magnitude pruning: Remove small weights

# Mobile-First Architectures: Designed for Efficiency

**MobileBERT:**[5]

- Bottleneck structure
- 4.3x smaller, 5.5x faster
- Depth-wise convolutions
- Progressive knowledge transfer

**ALBERT:**

- Parameter sharing
- Factorized embeddings
- 18x fewer parameters
- Same performance as BERT

**EdgeBERT:**

- Hardware-aware design
- Adaptive computation
- Early exit mechanisms
- Dynamic depth/width

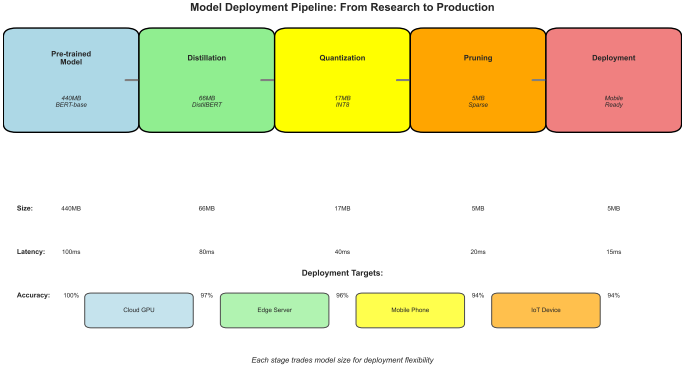
**Design Principles:**

- Prefer depth over width
- Share parameters
- Use separable operations
- Minimize memory access

Mobile architectures: Not compressed models, but efficient by design

[5] Sun et al. (2020) "MobileBERT"; Lan et al. (2020) "ALBERT"

# Deployment Optimization Pipeline




## Key Insights

- Start with pretrained model
- Apply compression techniques
- Optimize for target hardware
- Profile and iterate

# Hardware Acceleration: From GPUs to Edge TPUs

**Compute Platforms:**

- GPUs: Training & cloud inference
- TPUs: Optimized for transformers
- NPUs: Mobile inference
- DSPs: Ultra-low power
- FPGAs: Custom acceleration

**Optimization Techniques:**

- Kernel fusion
- Memory pooling
- Graph optimization
- Operator scheduling
- Cache optimization

**Framework Support:**

- TensorFlow Lite: Mobile/edge
- ONNX Runtime: Cross-platform
- Core ML: iOS devices
- TensorRT: NVIDIA optimization
- OpenVINO: Intel hardware

**Performance Gains:**

- Quantization: 2-4x speedup
- Pruning: 2-10x speedup
- Hardware opt: 5-50x
- Combined: 100x+ possible

2024: Every major chip has AI acceleration built-in

# Real-World Deployment: BERT on Mobile

```python
import torch
import torch.quantization as quant
from transformers import BertModel

def prepare_mobile_bert(model_name='bert-base-uncased'):
    """Complete pipeline for mobile deployment"""

    model = BertModel.from_pretrained(model_name)
    model.eval()

    print(f"Original size: {get_model_size(model):.1f} MB")

    distilled_model = distill_bert(model, num_layers=6)
    print(f"After distillation: {get_model_size(distilled_model):.1f} MB")

    model.qconfig = quant.get_default_qconfig('qnnpack')
    quant.prepare(model, inplace=True)
    quant.convert(model, inplace=True)
    print(f"After INT8 quantization: {get_model_size(model):.1f} MB")

    pruned_model = magnitude_prune(model, sparsity=0.9)
    print(f"After pruning: {get_model_size(pruned_model):.1f} MB")

    optimized_model = torch.jit.script(pruned_model)
    optimized_model.save("mobile_bert.pt")

    print("\nDeployment stats:")
    print(f"Final size: {get_model_size(optimized_model):.1f} MB")
    print(f"Inference time: {benchmark_model(optimized_model):.1f} ms")
    print(f"Memory usage: {get_memory_usage(optimized_model):.1f} MB")

    return optimized_model

def export_to_mobile(model, example_input):
    """Export for iOS/Android"""
```

**Compression Results:**

- BERT-base: 440MB → 13MB
- 33x size reduction
- 20x speedup on mobile
- 95% accuracy retained

**Deployment Checklist:**

- Profile target device
- Choose compression mix
- Validate accuracy
- Optimize runtime
- Monitor in production

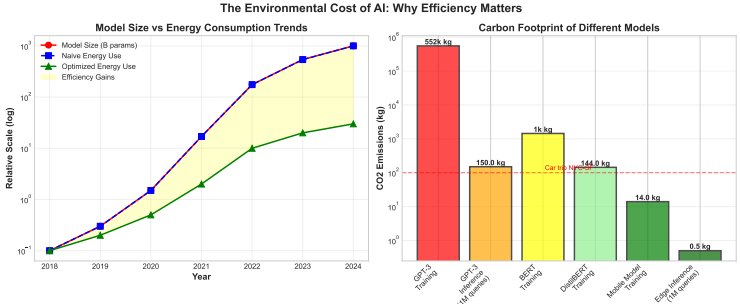
# The Green AI Movement: Energy-Efficient Models

**The environmental cost of AI:**[6]

- Training GPT-3: 1,287 MWh (123 cars for a year)
- One ChatGPT query: 0.0003 kWh
- Global AI energy: Doubling every 3.4 months

**Efficiency improvements:**

- Sparse models: 10x less energy
- Quantization: 4x less energy
- Better algorithms: 100x over 10 years
- Hardware efficiency: 1000x since 2012



# The Future of Efficient AI (2024 and Beyond)

**Emerging Techniques:**

- 1-bit models (BitNet)
- Mixture of Experts (MoE)
- Dynamic neural networks
- Neuromorphic computing
- Photonic processors

**Research Frontiers:**

- Sub-1-bit quantization
- Learned compression
- Architecture search for efficiency
- Federated model compression

**Industry Trends:**

- Every device runs AI (AIoT)
- Privacy-preserving inference
- Real-time translation everywhere
- Personalized on-device models
- Zero-latency assistants

**Hardware Evolution:**

- Analog AI chips
- In-memory computing
- Quantum advantage?
- Brain-inspired chips

The future: AI everywhere, invisible, instant, and sustainable

# Week 11 Exercise: Deploy Your Own Mobile Model

**Your Mission:** Take a large model and make it mobile-ready

**Part 1: Baseline and Profiling**

- Choose a pretrained model (BERT, GPT-2, etc.)
- Profile: size, latency, memory, energy
- Identify bottlenecks
- Set target constraints (e.g., 50MB, 100ms)

**Part 2: Apply Compression**

- Implement INT8 quantization
- Distill to smaller student
- Apply magnitude pruning
- Combine techniques

**Part 3: Deploy and Benchmark**

- Export to ONNX/TFLite
- Run on real device (phone/Raspberry Pi)
- Measure real-world performance
- Compare accuracy vs efficiency
- Create deployment package

**Bonus:** Build demo app showcasing your efficient model!

# Key Takeaways: Efficiency Enables Everything

**What we learned:**

- Models are vastly overparameterized
- 90% compression with minimal loss
- Quantization: Smaller and faster
- Distillation: Transfer capabilities
- Hardware matters immensely

**The efficiency toolkit:**

$$\text{Quantization} \rightarrow \text{Distillation} \rightarrow \text{Pruning} \rightarrow \text{Hardware optimization}$$

**Why it matters:**

- Enables edge deployment
- Reduces costs dramatically
- Improves user experience
- Environmental sustainability

**Next week: Ethics and Future Directions**
With great power comes great responsibility - what should we build?

# References and Further Reading

**Foundational Papers:**

- Hinton et al. (2015). "Distilling the Knowledge in a Neural Network"
- Han et al. (2016). "Deep Compression: Compressing DNNs"
- Frankle & Carbin (2019). "The Lottery Ticket Hypothesis"

**Quantization:**

- Jacob et al. (2018). "Quantization and Training of Neural Networks"
- Gholami et al. (2021). "A Survey of Quantization Methods"
- Dettmers et al. (2022). "LLM.int8(): 8-bit Matrix Multiplication"

**Practical Resources:**

- PyTorch Quantization Documentation
- TensorFlow Lite Guide
- ONNX Runtime Optimization
- Edge Impulse Platform