**Energy-Efficient AI with CNN Compression**

**1. Problem & Motivation**

* Deep learning models (CNNs) like **ResNet-18** are accurate but **heavy**: large size, high computation, and high energy use.
* This makes them slow on CPUs, expensive to run at scale, and increases **carbon footprint**.
* The goal is to make them **lighter and faster** without losing too much accuracy.

**2. ✅ Our Objective**

We designed a **framework for energy-efficient neural networks** that:

1. **Trains a baseline model** (ResNet-18 on CIFAR-10).
2. Applied **compression techniques**:
   * **Pruning** (remove unnecessary filters).
   * **Quantization** (reduce weight precision, e.g., FP32 → INT8).
   * **Knowledge Distillation** (train a smaller student model using the teacher’s knowledge).
3. Compare trade-offs: **accuracy vs size vs latency vs throughput**.
4. Provides a **Streamlit UI** for interactive testing.

**3. Dataset**

* **CIFAR-10**:
  + 60,000 images (32×32 color, 10 classes: airplane, car, bird, cat, etc.).
  + Train: 50k, Test: 10k.
* It’s small enough for quick experiments but complex enough to show real trade-offs.

**4. 🏗️ Models Used**

1. **ResNet-18** → Baseline "Teacher" model.
2. **Pruned ResNet-18** → Smaller, faster version after removing redundant channels.
3. **Quantized ResNet-18** → Compressed to INT8 precision for faster inference on CPU.
4. **MobileNetV2 (Student)** → A lightweight CNN trained using **Knowledge Distillation**.

**5. Implementation Steps**

**Step 1: Baseline Training**

* Train **ResNet-18** on CIFAR-10.
* Save the best checkpoint (resnet18\_best.pt).
* This gives ~76–80% accuracy in our quick runs (with more training, >90% is possible).

**Step 2: Structured Pruning**

* Remove less important convolution channels (filters).
* Fine-tune the model so accuracy recovers.
* Result: smaller computation, slightly smaller model, and still competitive accuracy.
* Saved as pruned\_resnet18.pt.

**Step 3: Quantization**

* Convert FP32 weights → INT8 using **dynamic quantization**.
* Reduces precision and speeds up inference on CPU.
* Accuracy drops slightly, but model runs faster.
* Saved as quantized\_resnet18.pt.

**Step 4: Knowledge Distillation**

* Use **ResNet-18 (teacher)** to train a smaller model (**MobileNetV2 student**).
* Loss = combination of teacher’s soft outputs + true labels.
* Students learn faster and achieves decent accuracy with much smaller size (~9 MB vs ~43 MB).
* Saved as kd\_mobilenetv2.pt.

**Step 5: Benchmarking**

* Run all saved models through a benchmarking script.
* Collect:
  + **Accuracy (%)**
  + **File size (MB)**
  + **Latency (ms per image)**
  + **Throughput (images/sec)**
* Results saved in reports/metrics.csv.

**Step 6: Interactive Demo (Streamlit UI)**

We built a full UI to **show results live**:

1. **Choose a model checkpoint** (ResNet18, Pruned, Quantized, KD).
2. **Pick a test image** from CIFAR-10 and see:
   * The actual image.
   * Predicted label vs Ground Truth.
   * Top 5 probabilities (bar chart).
3. **Upload your own image** → see the prediction.
4. **Metrics Table & Bubble Chart**:
   * Compare accuracy, latency, size, throughput visually.
   * Shows trade-offs between models.
5. **Latency test button** → measure speed instantly on CPU.

**6. 🎓What this shows in review**

* We didn’t just train CNN.
* We **optimized it** with state-of-the-art techniques:
  + **Pruning** (energy saving).
  + **Quantization** (hardware efficiency).
  + **Knowledge Distillation** (student-teacher transfer).
* We proved **trade-offs**:
  + Larger models = higher accuracy but slower.
  + Smaller models = lighter and faster but slightly less accurate.
* We built a **live demo tool** where anyone can test the models interactively.

**Remaining Work**

**🏗️ Compression Enhancements**

* Add **Structured Pruning** (real channel removal).
* Apply **Knowledge Distillation** (ResNet-18 teacher → MobileNetV2 student).
* Support **Mixed Precision (FP16)** training for faster, memory-efficient execution.

**🌱 Energy & CO₂ Dashboard**

* Integrate **CodeCarbon** to log energy & CO₂ emissions.
* Store results in a **database** (SQLite/Postgres).
* Visualize **Energy vs Accuracy**, **Carbon savings per method**.

**🖥️ UI Enhancements**

* Upload custom images for testing.
* Add **trade-off charts** (Accuracy vs Latency vs Size).
* Include **Confusion Matrix** & **Grad-CAM** heatmaps.
* Show **Energy & CO₂ plots**.

**🌐 Deployment**

* Export optimized models to **ONNX / TorchScript**.
* Package backend with **Docker** for portability.

**📊 Advanced Visualization & Reporting**

* **Pareto curves**: Accuracy vs Energy vs Latency.
* **CO₂ vs Accuracy** trade-off plots.
* **Automatic PDF reports** with graphs + summaries.

**🤖 Gemini AI Integration**

* **Carbon Footprint Narrator** → Gemini explains CO₂ savings in real-world terms (e.g., *“This model saves energy equal to running a bulb for 10 hours”*).
* **Auto-Documentation Assistant** → Gemini generates human-readable experiment reports from logs & benchmarks.