Technical Documentation

---

1. Installation Guide

✅ Hardware Requirements

- GPU with CUDA support (NVIDIA V100 or higher recommended)
- At least 8 GB of GPU memory
- 16 GB system RAM

🧰 Software Dependencies

- Python 3.8+
- PyTorch ≥ 1.10
- NumPy
- torchvision (for prebuilt models and datasets)
- tqdm (optional, for progress bars)

Install via pip:

```
pip install torch torchvision numpy tqdm
```

⚙️ Setup Instructions

1. Clone the repository or place all Python files (`adaptive_dpc.py`, `quantized_layers.py`, `train.py`) in your working directory.
2. Prepare datasets (CIFAR/ImageNet/PTB) using torchvision or manual download.
3. Run training with:

```
python train.py --arch resnet18 --dataset cifar10 --epochs 100
```

---

2. API Reference

🧠 `AdaptiveDPC(model, initial_bits=8, min_bits=4, delta=0.5)`

- **Purpose**: Automatically assign and update bit-widths for each layer based on capacity and training dynamics.
- **Methods**:
  - `quantize_weights()`: Applies quantization to model weights layer-wise.
  - `update_precision(gradient_stats)`: Adjusts precision based on gradient norm and variance.
  - `collect_gradient_stats()`: (Optional utility) Collects norms and variances of weight gradients.
  - `bit_widths`: A dictionary containing current bit-widths per layer.

⚙️ Quantized Layers

`QuantizedConv2d(...)`

- Drop-in replacement for `nn.Conv2d` with support for dynamic bit-widths.
- Methods:
  - `forward(input)`: Performs convolution with quantized weights.
  - `set_bit_width(bits)`: Sets active bit-width for the layer.

`QuantizedLinear(...)`

- Replacement for `nn.Linear`.
- Same API as `QuantizedConv2d`.

🔁 Training Utilities

`train_with_adaptive_dpc(...)`

- Manages training loop, dynamic precision adaptation, and model checkpointing.

`validate(...)`

- Performs validation with quantized weights.

`print_bit_width_distribution(...)`

- Logs current bit-width distribution across all layers.

---

3. Usage Examples

⚡ Basic Quantization

```
model = make_quantized_resnet('resnet18')
adpc = AdaptiveDPC(model)
adpc.quantize_weights()
output = model(input)
```

🎯 Custom Precision Schedules

Set fixed precision manually:

```
for layer in model.modules():
    if isinstance(layer, (QuantizedConv2d, QuantizedLinear)):
        layer.set_bit_width(6)
```

🧪 Training with Adaptive DPC

```
model = make_quantized_resnet('resnet18')
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
criterion = nn.CrossEntropyLoss()
train_with_adaptive_dpc(model, train_loader, val_loader, criterion,
optimizer, scheduler)
```

---

4. Performance Tuning

🔧 Choosing Initial Parameters

- `initial_bits = 8`: Good balance between quality and speed.
- `min_bits = 4`: Lower bound for efficiency.
- `delta = 0.5 to 1.0`: Controls bit-width spread. Higher = more aggressive downscaling.

🐞 Debugging Tips

- Ensure `adpc.quantize_weights()` is called **before forward pass**.
- Use `print_bit_width_distribution()` to monitor adaptation.
- Disable quantization (set `bits=32`) for debugging model logic.

🚀 Optimization Guidelines

- Use **cosine similarity threshold** (≈ 0.95) for tuning lower bit-bound.
- Monitor t-SNE plots or accuracy to detect over-aggressive quantization.
- Combine with **temporal precision scheduling (CPT)** for even greater efficiency.