RCD for Circuit Sparsity: Enabling Interpretable LLMs

Problem: Large Language Models (LLMs) are black boxes. OpenAI researchers (Elhage, Nanda, et al., 2022) introduced the concept of *Circuit Sparsity* in their interpretability work, showing that **enforcing sparsity during optimization** reveals interpretable “circuits” of non-zero weights, linking model function to specific connections.

Our Solution: The Random Coordinate Descent (RCD) optimization framework intrinsically supports this paradigm. By design, RCD performs sparse updates, making it a natural fit for cultivating interpretable circuits from the ground up.

---

1. RCD’s Core Logic for Circuit Sparsity

Abstract Logic:
Traditional optimizers update *all* parameters simultaneously, then prune. RCD, however, **selects only one coordinate (weight) `i` at each step** and updates it. This inherent sparsity of updates means most weights remain untouched, naturally fostering a sparse weight matrix (circuit).

Mathematical Abstract:
Given an objective function $f(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^N$ represents the model’s weights.
In RCD, at iteration k:

1. **Coordinate Selection:** Randomly choose an index $i \in \{1, \dots, N\}$ (or use a biased selection, see below).
2. **Gradient Computation (Partial):** Compute $\nabla_i f(\mathbf{x}^k) = \frac{\partial f}{\partial x_i}$ while holding all other x_j constant.
3. **Update:** $x_i^{k+1} = x_i^k - \eta \cdot \frac{\partial f}{\partial x_i} (\mathbf{x}^k)$
4. **Sparsity Enforcement (Zero-Clip):** If $|x_i^{k+1}| < \epsilon_{clip}$, then $x_i^{k+1} = 0$.
5. **Remaining Coordinates:** For all $j \neq i$, $x_j^{k+1} = x_j^k$.

This process directly enforces sparsity: only the selected coordinate changes, and small changes are zeroed out. The “circuit” emerges from the *non-zero* x_j values.

---

2. Workable Proof-of-Concept (PoC) Code (Pythonic)

This PoC demonstrates RCD with a simple L1 regularization to encourage sparsity on a toy problem, illustrating the core mechanics. (Adapted from standard coordinate descent methods; inspired by OpenAI's circuit sparsity framing.)

```
import numpy as np

# Toy objective: quadratic + L1 penalty
def toy_objective(x, lambda_l1=0.1):
    return np.sum((x - np.arange(len(x)))**2) + lambda_l1 * np.sum(np.abs(x))

def rcd_sparse_optimizer(func, x0, max_iter=5000, eta=0.01, tol=1e-7, clip_threshold=1e-4):
    n = len(x0)
    x = x0.copy()
    history_val = [func(x)]
    non_zero_counts = [np.sum(np.abs(x) > clip_threshold)]

    for iteration in range(max_iter):
        i = np.random.randint(n)
        x_plus_delta = x.copy()
        x_plus_delta[i] += eta * 0.1
        grad_i = (func(x_plus_delta) - func(x)) / (eta * 0.1)

        x[i] -= eta * grad_i
        if np.abs(x[i]) < clip_threshold:
            x[i] = 0.0

        current_val = func(x)
        history_val.append(current_val)
        non_zero_counts.append(np.sum(np.abs(x) > clip_threshold))

        if iteration > 0 and abs(history_val[-1] - history_val[-2]) < tol:
            break

    return x, history_val, non_zero_counts
```

---

3. Integration and Optimization Plan for Developers

Phase 1: RCD-Sparse Optimizer Core Development (1 Month)

- Implement `RCDSparseOptimizer` (PyTorch/JAX/TF).
- Coordinate selection strategies: uniform random, magnitude-biased (MB-RCD).
- Sparsity enforcement: zero-clip.
- Monitoring hooks for sparsity level and circuit mask.

Phase 2: LLM Integration & Benchmarking (2 Months)

- Target models: TinyLlama, GPT-2 small.
- Apply RCD to Linear layers (attention, feed-forward).
- Benchmark against AdamW/SGD and OpenAI's circuit sparsity method.
- Metrics: loss, perplexity, sparsity, inference speed.

Phase 3: Circuit Analysis & Interpretability (2 Months)

- Circuit identification module: extract non-zero masks, visualize circuits.
- Replicate circuit-level findings from OpenAI's interpretability work.
- Study robustness and generalization of RCD-induced circuits.

Optimization Focus Points

- **Efficient Partial Gradient:** Use Jacobian-vector products (JVP) or sparse ops.
- **Asynchronous RCD:** Distributed workers update disjoint coordinates.
- **Hardware Acceleration:** Mixed-precision, GPU/TPU sparse ops.

Conclusion

RCD offers a **principled, optimization-native approach** to achieving circuit sparsity in LLMs. By integrating RCD, developers can build models that are not only performant but also intrinsically interpretable, directly addressing a critical frontier in AI research. This builds on OpenAI's foundational work on circuit sparsity, extending it with a new optimizer paradigm.

Visual Aid (Conceptual Diagram)

```
graph TD
    A[Full LLM Weights x (Dense)] --> B{RCD Iteration}
    B --> C{1. Coordinate Selection<br> (Random/Biased i)}
    C --> D{2. Compute ∂f/∂x_i<br> (Partial Gradient)}
    D --> E{3. Update x_i Only<br> (x_i = x_i - η * grad_i)}
    E --> F{4. Zero-Clip if |x_i| < ε<br> (Enforce Sparsity)}
    F --> G[New Sparse LLM Weights x']
    G --> B
    G --> H[Extract Non-Zero Edges<br> (The "Circuit")]
    H --> I[Mechanistic Interpretability]
```

References

1. Elhage, N., Nanda, N., et al. (2022). *Toy Models of Superposition* and related work on Circuit Sparsity. OpenAI. Available at: https://transformer-circuits.pub/2022/toy_model/index.html