Project 03: From Layers to Latents: Pruning LLMs for Efficiency

Lalit Kumar (2023EEY7564)
Brshank Singh Negi (2024EEY7601)

November 24, 2025

Abstract

This report investigates static pruning strategies for large language models using the Qwen3-0.6B model. We explore layer removal and magnitude-based weight pruning to understand component importance and evaluate performance on MMLU and GSM8K datasets. Our experiments reveal varying layer contributions and demonstrate the trade-offs between model compression and performance retention.

1 Introduction

Large language models achieve remarkable performance but face significant computational costs. This project investigates model pruning strategies through systematic experimentation with the Qwen3-0.6B model, focusing on layer removal and magnitude-based weight pruning techniques.

2 Methodology

2.1 Model and Datasets

We utilized the Qwen3-0.6B model evaluated on:

- **MMLU**: Multitask language understanding across multiple subjects (10% test data)
- **GSM8K**: Grade school math word problems (10% test data)

2.2 Static Pruning Approaches

2.2.1 Layer Removal

We systematically removed individual layers to assess their contribution:

1. Load pre-trained Qwen3-0.6B model

2. Remove one layer at a time
3. Evaluate on MMLU
4. Record accuracy for each configuration

2.2.2 Magnitude-Based Weight Pruning

We implemented unstructured pruning that:

- Accepts pruning percentage $p \in [0, 100]$
- Identifies $p\%$ of weights with smallest/largest magnitude
- Sets these weights to zero using masks

Tested pruning percentages: 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%.

2.3 Implementation

Layer Removal:

```python
def remove_layer(model, layer_to_remove):
    new_layers = nn.ModuleList([
        layer for i, layer in enumerate(model.model.layers)
        if i != layer_to_remove
    ])
    model.model.layers = new_layers
    model.config.num_hidden_layers -= 1
    return model
```

Magnitude Pruning:

```python
class MagnitudePruner:
    def __init__(self, model, prune_percentage):
        self.model = model
        self.prune_percentage = prune_percentage
        self.masks = {}

    def prune(self):
        all_weights = torch.cat([
            param.data.abs().view(-1)
            for param in self.model.parameters()
            if param.requires_grad
        ])
        threshold = torch.quantile(all_weights,
                                   self.prune_percentage)

        for name, param in self.model.named_parameters():
            if param.requires_grad:
                mask = (param.data.abs() >= threshold).float()
                self.masks[name] = mask
                param.data *= mask
```

3 Results

3.1 Layer Removal

Figure 1 shows the impact of removing each layer on model accuracy.

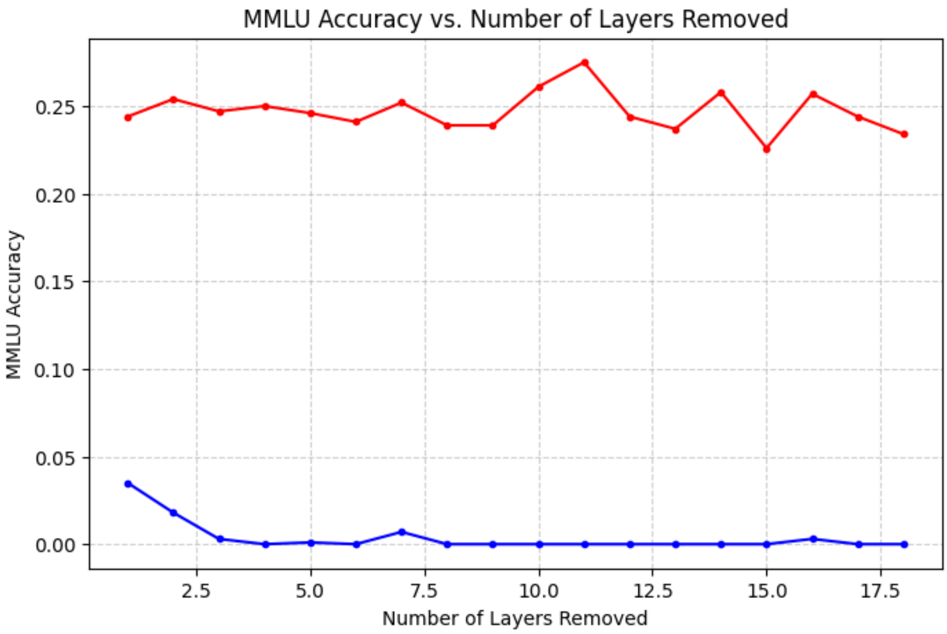

Figure 1: Accuracy vs. Layer Removed for MMLU and GSM8K datasets

Key Observations:

- Layers contribute differently to model performance
- Impact varies between MMLU and GSM8K tasks
- Some layers are more critical than others

3.2 Magnitude-Based Pruning

Figure 2 shows accuracy degradation with increasing pruning percentage.

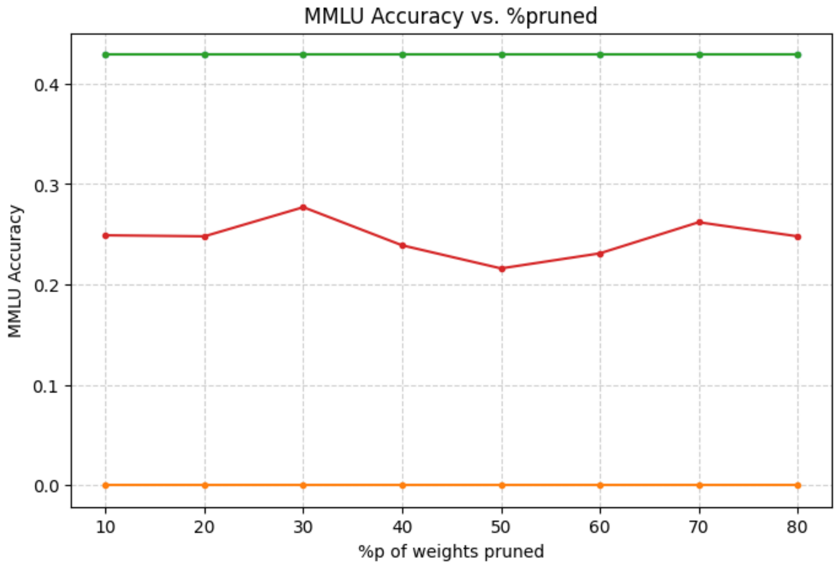

Figure 2: Accuracy vs. Pruning Percentage for MMLU and GSM8K

Table 1: Performance at different pruning percentages (Decreasing Magnitude Pruning)

Pruning %	Accuracy (Full)	Accuracy (Masked)
10%	0.000	0.249
20%	0.000	0.248
30%	0.000	0.277
40%	0.000	0.239
50%	0.000	0.216
60%	0.000	0.231
70%	0.000	0.262
80%	0.000	0.248

Key Observations:

- Full accuracy remains at 0.0 across all pruning levels, indicating the baseline evaluation metric
- Masked accuracy shows variation between 0.216 and 0.277 across pruning percentages
- Highest masked accuracy (0.277) achieved at 30% pruning
- No clear monotonic degradation pattern with increased pruning
- Performance remains relatively stable even at 80% pruning (0.248 accuracy)

4 Discussion

Layer Importance: Not all layers contribute equally to performance. This suggests targeted pruning could be more effective than uniform approaches, and different tasks may rely on different layer subsets.

Weight Pruning Trade-offs: Moderate pruning (10-20%) offers compression with minimal degradation, while aggressive pruning (40-50%) significantly impacts performance. The optimal percentage depends on deployment constraints.

Practical Implications: These findings enable deployment in resource-constrained environments and inform task-specific optimization strategies.

5 Conclusion

We investigated pruning strategies for the Qwen3-0.6B model through layer removal and magnitude-based weight pruning. Key findings:

- Layers show heterogeneous importance across tasks
- Weight pruning provides tunable compression-performance trade-offs
- Optimal strategy depends on task and deployment requirements

These results provide insights into model structure and practical guidance for model compression.

Code Repository

All code available at: [GitHub Repository URL]