Efficient Performance Recovery of Pruned LLMs for End Devices

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

Computer Science

with specialization in Data Science

by

Paul Vardhan Bethapudi

[Roll No: CS2420]

under the supervision of

Prof. Ujjwal Bhattacharya

Professor

Computer Vision and Pattern Recognition Unit

Indian Statistical Institute

Kolkata 700108, India

May 2026

Certificate

This is to certify that the dissertation entitled **"Efficient Performance Recovery of Pruned LLMs for End Devices"** submitted by **Paul Vardhan (CS2420)** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science with specialization in Data Science**, is a bonafide record of work carried out by him under my supervision and guidance.

The dissertation has fulfilled the requirements as per the regulations of the institute and, in my opinion, has reached the standard required for submission.

Prof. Ujjwal Bhattacharya
Professor
Computer Vision and Pattern Recognition Unit
Indian Statistical Institute
Kolkata 700108, India

To my family, teachers, and everyone
who supported me throughout this academic journey.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, **Prof. Ujjwal Bhattacharya**, for his guidance and tremendous support throughout this dissertation. His feedback greatly helped me streamline the pruning, recovery, and evaluation pipeline.

I am also thankful to the Indian Statistical Institute, Kolkata, for providing the academic exposure and computational support required for this work. The experiments on Llama-3.2-3B Instruct model, Wanda pruning, QLoRA, DoRA, WikiText-2, and GSM8K helped me understand the practical challenges involving efficient LLM deployment.

I thank my teachers, classmates, and friends for their encouragement and useful discussions during different stages of the project. Those suggestions helped me improve both in technical aspects and presentation of this dissertation.

I am deeply grateful to my family for their constant support, patience, and encouragement. Their belief in me gave me strength during the difficult phases of experimentation, debugging, and writing.

Finally, I thank everyone who directly or indirectly supported me in completing this dissertation.

Paul Vardhan Bethapudi
Indian Statistical Institute
Kolkata 700108, India

Abstract

Large Language Models have shown strong performance across reasoning, language understanding, and generation tasks, but their computational and memory requirements make deployment on low resource devices difficult. This dissertation studies efficient performance recovery of pruned LLMs, focusing on whether a pruned model can regain useful task performance through parameter-efficient fine-tuning while preserving the benefits of compression.

The work uses Llama-3.2-3B-Instruct as the dense reference model and applies activation-aware Wanda pruning at multiple sparsity levels(20%, 30% and 50%). The pruned models are evaluated on language modeling and reasoning tasks using WikiText-2 perplexity and GSM8K (Grade School Math 8000 problems) accuracy. Adaptation is performed using efficient parameter adaptation methods like QLoRA and DoRA. Additionally, the thesis evaluates the balance between sparsity, precision, perplexity, speed, memory usage, and device compatibility.

Based on the results obtained from the experiments conducted, we can safely say that the adapter recovery technique improves the performance of sparse models. DoRA performs better in GSM8K recovery compared to QLoRA recovery at sparsity of 20%, 30%, and 50%. Additionally, the WikiText-2 perplexity for QLoRA is slightly lower than DoRA at 30% and 50% sparsity. The largest recovery gain occurs at 50% sparsity, where GSM8K accuracy improves from 0.1800 to 0.4508 using DoRA recovery. However, the best final recovered GSM8K accuracy is obtained at 20% sparsity with DoRA.

Keywords: Large Language Models(LLM), Model Pruning, Wanda, LoRA QLoRA, DoRA, Parameter-Efficient Fine-Tuning(PEFT), Model Compression, End-Device Deployment, GSM8K, WikiText-2.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

Abbreviation	Full Form
LLM	Large Language Model
PEFT	Parameter-Efficient Fine-Tuning
QLoRA	Quantized Low-Rank Adaptation
DoRA	Weight-Decomposed Low-Rank Adaptation
LoRA	Low-Rank Adaptation
Wanda	Pruning by Weights and Activations
PPL	Perplexity
VRAM	Video Random Access Memory
GSM8K	Grade School Math 8K Dataset
HF	Hugging Face
GPU	Graphics Processing Unit
CPU	Central Processing Unit
FLOPs	Floating Point Operations
SFT	Supervised Fine-Tuning

Chapter 1

Introduction

1.1 Background

Large Language Models have been crucial to recent advances in natural language processing. They are capable of generating meaningful text, following instructions, summarizing lengthy documents, dealing with reasoning challenges, and adapting to a wide variety of downstream tasks. Nevertheless, this comes with a downside. Large Language Models even with a relatively small number of billions of parameters require a lot of memory, computation power, and time inference when put into practice anywhere except large servers.

An apparent trade-off arises. More significant models usually have better language understanding and reasoning capabilities. Yet, their size makes their use on end-users' devices or moderate hardware a challenge. From the perspective of real applications, however, accuracy alone is insufficient. Models must function within memory bounds, provide timely responses, and produce tokens efficiently under these conditions. This factor becomes especially critical when inference is to be carried out near the end-users instead of in the cloud exclusively.

One way of dealing with this problem could be through model compression. Several techniques exist to this end. Pruning stands out in that it removes or deactivates unnecessary weights. In theory, pruning is supposed to result in model sparsity and thus reduce memory footprint and lower the computational burden. Reality, though, might differ. Weights removal from the pretrained large language model would distort its representations that were learned during the pre-training and instruction tuning process. As a result, quality of language modeling goes down or reasoning suffers, or both.

The present work focuses on what can be retrieved by pruning. Sparsity is considered here merely as means to an end and not an end in itself. Besides that, an attempt is made to investigate how much of the lost performance can be recovered through lightweight adaptation. The base model being used is Llama-3.2-3B-Instruct. Wanda pruning is applied at several sparsity levels. The checkpoints produced this way are then rescued by parameter-efficient fine-tuning methods (QLoRA and DoRA). Performance evaluation was done using WikiText-2

perplexity, GSM8K accuracy, as well as memory and throughput figures.

1.2 Motivation

The rationale behind the current research is inspired by observations made throughout our initial efforts to implement compression of language models. It does not make sense to compress the model if the resulting variant lacks the necessary capabilities to perform the intended task. Sparser model loses its usefulness fast when pruning interferes with reasoning. On the other hand, if the process of recovery after pruning or the final model architecture itself becomes too bulky, the entire effort becomes impractical because of the need for excessive memory space. Therefore, the core question is not about the ability to prune a model, but rather whether the model can be pruned and recovered successfully with a preserved advantage in efficiency.

It becomes even more crucial to consider this problem in the context of end-device use cases. Cloud environments with large GPUs and enough memory space hide various inefficiencies. When working with smaller GPUs or real resource-constrained devices, such inefficiencies are readily apparent. Memory capacity defines whether or not a model can fit into memory, latency defines usability, and throughput influences efficiency of token generation. Privacy and offline usage become an issue closer to the user. Hence, deploying language models should be approached both as a modeling problem and as a systems problem, similarly to model training.

Pruning can become particularly appealing due to the idea of removing unnecessary parameters. At the same time, mere pruning may prove to be insufficient. A model that was pruned with low or medium sparsity will retain general language characteristics but will have poor performance in specific tasks. Such effects would be critical in case of reasoning, where even slight shifts of internal representation could result in incorrect answer. Another problem is that the effect of pruning on a language model varies depending on the task, which can cause a shift in perplexity score in one direction and decrease accuracy in mathematical reasoning in another direction.

Therefore, some way to recover the model after pruning needs to be considered. Full-fledged fine-tuning becomes prohibitively costly in terms of computation and time required. Instead, parameter-efficient fine-tuning can provide a more reasonable solution as a means to update only a part of model parameters. As a part of the current dissertation, the process of model recovery with QLoRA and DoRA adapters after applying Wanda pruning was tested.

The other reason for this choice is methodological. The reliance on one measure alone produces a partial account of both pruning and retraining. The GSM8K accuracy measure can be useful for measuring reasoning ability but does not capture the complete quality of language model. Similarly, WikiText-2 perplexity can be helpful for measuring next-token prediction skills but is unable to measure multi-step reasoning ability. Additionally, none of these measures can be used in isolation from measures of memory and throughput efficiency.

1.3 Problem Statement

This problem involves the optimal performance recovery of pruned Large Language Models (LLMs) in constrained computational setups. Pruning is used to decrease parameter efficiency in pretrained models, followed by adaptation in order to recover the lost performance due to the pruning process. The main idea is to increase the effectiveness of the task-specific recovered model and keep the benefits that come along with compression.

Formally, consider M to be the pre-trained model initially and consider s to be the desired sparsity. Pruning algorithm P results in the pruned model M_s as

$$M_s = P(M, s),$$

where M_s contains sparse parameters as per the described pruning policy. Since pruning has an adverse effect on the ability of the model to perform natural language processing and reasoning tasks, the recovering function R is then performed using the training data set D_{train}:

$$M_s^{\text{rec}} = R(M_s, D_{\text{train}}).$$

The objective is to achieve a model M_s^{rec} that performs increasingly similarly to the original model M while retaining advantages in memory usage, inferability, or compression. Ideally, the relationship can be stated as follows:

$$\text{Performance}(M_s^{\text{rec}}) \approx \text{Performance}(M),$$

subject to reduced deployment cost compared with the uncompressed model.

Reaching such an approximation poses certain difficulties. First, different sparsity degrees have different impacts on the model, some are enough to maintain its structure for recovering, while others take away too much capacity. Second, different recovery approaches may behave in diverse ways depending on pruning severity and specific evaluations. Thus, both pruning and recovery approaches are considered together and not separately. The study follows a consistent chain including initial evaluation, pruning, subsequent evaluation, recovery, and final evaluation.

1.4 Research Objectives

In particular, the first objective is to set up a thorough experimental pipeline for the effective pruning and recovery of a Llama-scale model. The starting point in the pipeline is the base Llama-3.2-3B-Instruct model, which must be benchmarked under the same protocol before compressing it in any way. It is crucial since all the other experiments will be conducted in

comparison with that baseline.

The second objective is related to experimenting with different degrees of sparsity for Wanda pruning. In our experiment, we treat various sparsity levels as separate compression protocols rather than as multiple iterations of one and the same experiment. That choice enables us to observe the response of our model to moderate, severe, and aggressive pruning.

The third objective is about comparing the effect of recovery performed via QLoRA and DoRA. In addition to being generally less costly to implement than fine-tuning, the two techniques offer quite different approaches to the introduction of trainable parameters. Thus, studying both of them allows for finding out whether or not the specific approach affects how well the adaptation procedure succeeds.

Additionally, we compare the models in terms of their performance in both language modeling and reasoning tasks. For the purpose, WikiText-2 perplexity measures the general capability of a language model, while GSM8K accuracy shows its skills in mathematical reasoning. As they are not equivalent, using both of them helps us have a more consistent picture of the model's pruning impact and recovery performance.

The next objective of the work is to conduct the tests of the models under identical conditions. This is significant as even minor differences in prompt construction, sample size, generation length, and answer extraction may influence the result in case of GSM8K. With a protocol fixed, the comparisons become more valid.

Finally, we measure and compare some practical metrics of efficiency of the models such as peak VRAM consumption, throughput, latency, and the ease of deployment. Taking into account the motivation of deployment to end devices with constrained hardware capacity, the necessity of these measurements becomes obvious.

1.5 Thesis Contributions

The first major contribution of this work is the formulation of a unified empirical framework for investigating LLM pruning and recovery under realistic assumptions. It should be noted that pruning is never treated as an isolated step, but the full experiment cycle – starting with a dense baseline experiment and going all the way through Wanda pruning, raw pruning evaluation, parameter-efficient fine-tuning recovery, and finally comparing with a dense baseline – is used. It allows seeing what kind of degradation results from pruning and what kind of improvements one may reasonably expect after recovery.

The second significant contribution is represented by the comparison of different levels of pruning severity performed by means of Wanda pruning for Llama-3.2-3B-Instruct. The benefit of Wanda pruning lies in its use of both weight magnitude and activation statistics, without requiring full retraining before pruning. The usage of Wanda pruning in different settings with varying degrees of pruning severity makes it possible to analyze the effects that

pruning has on language modeling and reasoning abilities.

A further point is the direct comparison of two recovery methods, namely QLoRA and DoRA, that are performed on Wanda-pruned checkpoints with equal sparsity. Although both of these techniques belong to the PEFT category, their implementation in the process of adaptation is different from one another. QLoRA is based on low-rank adaptation using the base model, loaded in a memory-saving way as a 4-bit representation of the model. On the contrary, DoRA modifies the approach to low-rank adaptation via breaking down weight representation into magnitude and direction. This dissertation investigates the reaction of both approaches to the effect of pruning on language modeling and mathematical reasoning tasks.

The third significant contribution is represented by the usage of both WikiText-2 and GSM8K datasets for evaluating language modeling and reasoning ability respectively. Using this combination of datasets allows disentangling language model degeneration and reasoning degeneration since they do not always coincide.

The fourth contribution is made by performing deployment-oriented analysis, where memory consumption, throughput, and latency are measured alongside the other metrics. This is an important part of the research because the end goal here is not just reaching a more compact model but creating a model that performs better under restricted inference.

Overall, this work makes a major contribution in the form of a structured study of the problem of compressing large language models in terms of the trade-off between compression and performance. One should not think that this paper introduces a new method of pruning; the value of this work lies elsewhere.

1.6 Scope and Assumptions

Pruning and recovery is only performed empirically under restrictive hardware settings in this paper. The base language model considered here is Llama-3.2-3B-Instruct, and the pruning scheme adopted in this research is the Wanda pruning. The methods used to perform recovery include QLoRA and DoRA parameter-efficient fine-tuning. The measures of evaluation used include perplexity on WikiText-2 dataset, accuracy on GSM8K dataset, as well as throughput, latency, and memory consumption.

The chosen benchmarks are believed to provide insights into certain aspects of quality language modeling and mathematics. WikiText-2 acts as a benchmark that measures the ability of language modeling. GSM8K is considered since arithmetic and multi-step reasoning tasks are often vulnerable to pruning operations. Nonetheless, the two datasets above cannot represent all of the abilities of LLMs. Evaluation that involves commonsense reasoning, programming instructions, coding, and question answering is outside the focus of the current work.

“End-device” in this case refers to a practical yet restrictive notion that refers to the

constrained GPU, CPU, or edge-like environment in which the amount of available resources is relatively less than that available in cloud computing servers. In other words, end-devices in this work refer to the environment used for experimental purposes and not production deployments in mobile devices or embedded devices where unstructured sparsity does not guarantee performance speedup unless exploited efficiently by the inference engine and hardware.

It is also assumed that adapter-based model recovery is a feasible and realistic approach to fine-tuning under resource-constrained settings. The recovery process would become too costly when full fine-tuning of all the parameters is applied in this experimental setting. Thus, in order to make the experiment more practical and relevant in the real world, recovery is carried out through lightweight fine-tuning.

1.7 Thesis Organization

Chapter 2 provides an overview of the necessary technical background. This section describes Large Language Models, autoregressive language modeling, transformers, transformer decoder structure, model compression, pruning, Wanda pruning, parameter-efficient fine-tuning, and evaluation metrics. Additionally, it restates the deployment requirements driving the current research.

Chapter 3 provides an overview of related work on efficient LLM inference, pruning techniques, post-pruning deterioration of model performance, recovering a compressed model, quantization techniques, low-rank model adaptation, and evaluation tailored towards deployment. Chapter 3 is intended to set out the context of the work carried out herein, with specific emphasis on the necessity for a baseline–pruning–recovery experiment under a unified framework.

Chapter 4 describes the methodology behind the work. It discusses model selection, dataset selection, baseline evaluation, Wanda pruning, post-pruning evaluation, QLoRA and DoRA recovery, handling adapters, and analysis on the end device level. Chapter 4 describes the approach taken in carrying out this research.

Chapter 5 contains a description of the algorithms, implementation specifics, and computational complexity of the project. Mainly, it deals with the description of the experimental procedure, evaluation algorithms used, and environments used to reproduce the experiments.

Chapter 6 contains the experimental results and the discussion thereof. It examines the performance of the dense baseline model, the raw pruned model, and recovered models against the metrics selected for comparison. Other factors considered include recovery ratio, efficiency trade-off, suitability for deployment, and behavioral differences in GSM8K predictions.

Chapter 7 summarizes the key findings from the research carried out in chapters 4 to 6, highlights the limitations, and discusses future plans.

Chapter 2

Preliminaries

2.1 Large Language Models

Large Language Models are large-scale neural language models trained to predict and generate text. In this dissertation, the focus is not only on their language ability, but also on their computational cost. When pruning and recovery are applied to an LLM, parameters, activations, memory usage, and inference speed all become important. This is the reason the model is evaluated not only by accuracy and perplexity, but also by throughput and memory-related behaviour.

Most recent LLMs used for text generation are based on the Transformer decoder architecture. They generate tokens autoregressively, meaning that each new token is predicted from the tokens already produced. Even when the model is moderate in size, as in the case of Llama-3.2-3B-Instruct, inference still requires repeated forward passes through many layers. This makes deployment difficult on constrained hardware.

In this work, Llama-3.2-3B-Instruct is treated as the dense reference model. The aim is not to train a new LLM from scratch, but to study what happens when a pretrained instruction model is pruned and then recovered using parameter-efficient methods. This makes the problem closer to a practical deployment setting, where a pretrained model is available but full retraining is not feasible.

2.1.1 Autoregressive Language Modeling

Autoregressive language modeling factorizes the probability of a token sequence into a product of conditional probabilities. For a sequence $x_1, x_2, \ldots, x_T$, this can be written as

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{<t}),$$

where $x_{<t}$ denotes all tokens before position t.

During inference, the model receives a prompt and generates tokens one after another. This sequential nature is important for deployment because every generated token requires computation through the model. For reasoning datasets such as GSM8K, this also means that an error in an early generated step can affect the final numerical answer. Therefore, pruning must be evaluated carefully, since a pruned model may still produce fluent text while losing part of its reasoning ability.

2.1.2 Transformer Decoder Architecture

A Transformer decoder model is built from repeated blocks containing self-attention, feed-forward layers, residual connections, and normalization layers. In a causal decoder, the attention mask allows each token to attend only to previous tokens and not to future tokens. This structure makes decoder-only models suitable for autoregressive generation.

The self-attention and feed-forward projections contain a large portion of the model parameters. These linear layers are also important targets for pruning, because pruning them can reduce the number of active weights in the model. However, pruning individual weights is not trivial. A weight that appears small in magnitude may still be important if the corresponding input activation is frequently used. This motivates activation-aware pruning methods such as Wanda.

2.2 Model Compression

Model compression refers to methods that reduce the memory, storage, or computational cost of a neural network. In the context of LLMs, this is important because even a 3B-scale model can be difficult to run repeatedly under limited GPU memory and practical inference constraints.

Common compression methods include pruning, quantization, distillation, and low-rank adaptation. Pruning removes or zeros out selected weights. Quantization reduces numerical precision, for example from floating point precision to 8-bit or 4-bit representations. Distillation trains a smaller student model to imitate a larger teacher model. Low-rank adaptation reduces the number of trainable parameters during fine-tuning.

This dissertation focuses mainly on pruning followed by adapter-based recovery. Pruning is used to create sparse checkpoints, while QLoRA and DoRA are used later to recover part of the lost performance. This order is important because pruning alone may reduce model quality, especially on reasoning tasks. Therefore, compression is studied here as a complete pruning–recovery pipeline rather than as a single isolated step.

A practical point is also important. Sparse weights do not automatically give faster inference. If the inference backend still uses dense matrix multiplication, zero weights may not lead to proportional speedup. Because of this, the experiments report actual throughput and

memory behaviour instead of assuming that sparsity directly gives runtime improvement.

2.3 Neural Network Pruning

Neural network pruning removes or deactivates selected parts of a model. These parts may be individual weights, neurons, channels, attention heads, or larger blocks. The basic assumption is that not all parameters contribute equally to the final output. If less important parameters can be removed without destroying the model's useful behaviour, the model can become more compact.

For large language models, pruning is more delicate than in smaller networks. The internal representations are distributed across layers, attention modules, and feed-forward blocks. Removing weights may affect language modeling, reasoning, or instruction-following behaviour. Because of this, a pruned model must be evaluated directly before any recovery step is applied.

In this dissertation, pruning is performed after pretraining, using the existing Llama-3.2-3B-Instruct checkpoint. This is a post-training pruning setting. The model is not trained from scratch, and the purpose is to examine how much performance is lost after pruning and how much can be recovered later using parameter-efficient fine-tuning.

2.3.1 Unstructured Pruning

Unstructured pruning sets individual weights to zero while keeping the original matrix shape unchanged. If W denotes a weight matrix, pruning selects some entries of W and masks them. The matrix dimensions remain the same, but the number of non-zero weights is reduced.

The advantage of unstructured pruning is flexibility. It can reach different sparsity levels without removing full neurons, heads, or layers. In this project, Wanda pruning is used in this unstructured manner. The pruned checkpoints are produced at 20%, 30%, and 50% sparsity.

The limitation is hardware-related. A sparse matrix is not necessarily faster during inference if the software and hardware do not exploit sparsity. For this reason, this dissertation does not assume direct speedup from unstructured pruning. Instead, throughput is measured empirically during evaluation.

2.3.2 Structured Pruning

Structured pruning removes larger units such as attention heads, channels, neurons, or full blocks. Since it changes the structure of the computation, it can be more useful for hardware speedup than unstructured pruning. For example, removing entire channels can reduce the size of dense matrix operations.

However, structured pruning is more aggressive. Removing an entire head or block may

remove useful behaviour that is not easy to restore. In this dissertation, structured pruning is not used. It is discussed only to clarify the difference between hardware-friendly pruning and the Wanda-based unstructured pruning used in the experiments.

2.3.3 Magnitude-Based Pruning

Magnitude-based pruning is one of the simplest pruning methods. It assumes that weights with smaller absolute values are less important. For a weight W_{ij}, the importance is measured as

$$I_{ij} = |W_{ij}|.$$

Weights with the smallest magnitudes are pruned until the target sparsity is reached.

This method is simple and fast, but it ignores activations. A small weight connected to a frequently active input feature may still have a strong effect on the output, while a large weight connected to an inactive feature may matter less. This limitation is important in LLMs, where activation patterns differ across layers, tokens, and inputs. Hence, this work uses activation-aware pruning instead of magnitude-only pruning.

2.4 Activation-Aware Pruning

Activation-aware pruning evaluates a weight not only by its magnitude but also by the activation of the input channel connected to it. In a linear layer, the output depends on both the weight and the input activation. Therefore, a pruning score should consider both factors.

Wanda follows this principle. It first runs a small calibration set through the model and collects activation statistics for the target linear layers. These samples are not used for supervised training. They are only used to estimate which input channels are active during normal model computation.

For each weight W_{ij}, Wanda computes the importance score as

$$S_{ij} = |W_{ij}| \cdot \|X_j\|_2,$$

where W_{ij} is the weight and X_j denotes the collected activation values corresponding to input channel j. Weights with lower scores are treated as less important and are pruned according to the selected sparsity level.

This score is useful for the present work because it is simple enough to apply under limited computational resources, but still more informed than magnitude-only pruning. The quality of the pruning still depends on the calibration data, because the collected activations should reasonably represent the inputs expected during evaluation.

2.5 Wanda Pruning

Wanda pruning is the main pruning method used in this dissertation. It was selected because it considers both weight magnitude and activation statistics while deciding which weights should be removed. This makes it more suitable than simple magnitude pruning for LLMs, where a small weight may still be important if its input channel is frequently activated.

For each weight W_{ij}, Wanda computes the importance score as

$$S_{ij} = |W_{ij}| \cdot \|X_j\|_2,$$

where W_{ij} is the individual weight and X_j represents the activation statistics collected for the corresponding input channel. Weights with lower scores are treated as less important and are pruned until the required sparsity level is reached.

A useful feature of Wanda is that it does not require full retraining before pruning. A small calibration set is passed through the model to collect activation statistics, and these statistics are then used along with weight magnitude to compute pruning scores. In this work, Wanda was applied to Llama-3.2-3B-Instruct at three sparsity levels: 20%, 30%, and 50%. These levels allow the study to compare moderate, stronger, and aggressive pruning.

The following table summarizes the pruning strategies considered during the design of this work.

Table 2.1: Comparison of pruning strategies

Method	**Importance Signal**	**Main Advantage**	**Limitation**
Magnitude pruning	Weight magnitude	Simple and fast	Ignores activations
Wanda	Weight and activation	No heavy retraining before pruning	Requires calibration data
Structured pruning	Blocks, heads, or layers	More hardware-friendly	May reduce model capacity sharply

Wanda therefore provides a practical middle ground for this project. It is more informed than magnitude-only pruning but still lightweight enough to apply under limited computational resources. However, its effectiveness depends on the calibration data being reasonably representative of the model inputs.

2.6 Parameter-Efficient Fine-Tuning

Full fine-tuning updates all model parameters, which is expensive for large language models because gradients, optimizer states, and activations must be stored during training. This was not practical for the hardware setting of this dissertation. Parameter-Efficient Fine-Tuning (PEFT)

offers a more suitable alternative by freezing most of the base model and training only a small number of additional parameters.

In this work, PEFT is used after Wanda pruning. The adapter is not trained on the original dense model, but on the pruned checkpoint. Therefore, the recovery stage has to adapt to a model whose weights have already been partially removed. This makes PEFT recovery an important part of the pruning–recovery pipeline.

2.6.1 LoRA

LoRA freezes the original weight matrix W and learns a low-rank update. The modified weight can be written as

$$W' = W + \Delta W = W + BA,$$

where A and B are trainable low-rank matrices. Since only these matrices are trained, the number of trainable parameters is much smaller than in full fine-tuning. LoRA forms the basic idea behind the adapter-based recovery methods used in this dissertation.

2.6.2 QLoRA

QLoRA combines low-rank adaptation with memory-efficient quantized model loading. In this method, the base model is loaded in 4-bit precision, while the LoRA adapter parameters remain trainable. This reduces memory usage during recovery and makes it feasible to train adapters for multiple pruned checkpoints.

In the present workflow, the Wanda-pruned model acts as the fixed base, and QLoRA provides the trainable adapter used for recovery. The recovered model is then evaluated against both the dense baseline and the raw pruned checkpoint.

2.6.3 DoRA

DoRA modifies the standard LoRA approach by separating weight adaptation into magnitude and direction components. This is useful to study after pruning because Wanda changes the effective structure of the weight matrix by setting selected weights to zero. A recovery method that can adjust the direction and magnitude of weight updates may behave differently from a standard low-rank update.

DoRA is included in this dissertation to compare whether magnitude–direction based recovery improves the performance of Wanda-pruned models. Both QLoRA and DoRA are tested under the same sparsity levels and evaluation settings.

2.7 Evaluation Metrics

The evaluation in this dissertation uses both performance and efficiency metrics. This is necessary because a recovered model may improve on one metric while still being weak on another. For example, a model may show improved perplexity but still fail on mathematical reasoning, or it may improve accuracy while becoming impractical for constrained hardware.

The main evaluation metrics used in this work are WikiText-2 perplexity, GSM8K accuracy, throughput, latency, and peak VRAM usage. WikiText-2 is used to measure language modeling quality, while GSM8K is used to measure mathematical reasoning ability.

2.7.1 Perplexity

Perplexity measures how well the model predicts a sequence of tokens. Lower perplexity indicates better language modeling quality. For a sequence of N tokens, perplexity is computed as

$$\text{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p(x_i)\right),$$

where $p(x_i)$ is the probability assigned by the model to the correct token x_i.

In this work, perplexity is evaluated on WikiText-2. It helps measure whether pruning damages the general next-token prediction ability of the model and whether PEFT recovery improves it.

2.7.2 Task Accuracy

Task accuracy measures the proportion of correct outputs in a benchmark. In this dissertation, GSM8K accuracy is used to evaluate mathematical reasoning. The model generates a solution for each word problem, the final numerical answer is extracted, and the extracted answer is compared with the gold answer.

Accuracy is computed as

$$\text{Accuracy} = \frac{\text{Number of Correct Answers}}{\text{Total Number of Questions}}.$$

This metric is strict because only the final extracted numerical answer is considered. A response with fluent reasoning but an incorrect final number is counted as incorrect. For this reason, GSM8K accuracy is useful for detecting reasoning degradation after pruning.

2.7.3 Efficiency Metrics

Efficiency metrics are included because the dissertation focuses on recovery under constrained hardware conditions. Accuracy and perplexity alone do not show whether a model is practical to run.

Peak VRAM measures the maximum GPU memory required during evaluation or recovery. Throughput measures the number of generated tokens per second. Latency measures the time required for response generation. These metrics are interpreted along with accuracy and perplexity, since a useful recovered model should not only perform better but also remain feasible for deployment-oriented settings.

2.8 End-Device Deployment Constraints

In this dissertation, end-device deployment refers to running or evaluating models under constrained memory and compute conditions, not necessarily on a mobile phone or embedded board. The term is used to represent practical settings where hardware resources are limited compared to large cloud servers.

The main constraints considered are memory, latency, throughput, storage, and accuracy. These factors are important because compression is useful only when the resulting model remains practically usable. In particular, unstructured pruning does not guarantee runtime speedup unless the inference backend or hardware can exploit sparse computation.

Table 2.2: Deployment constraints considered in this dissertation

Constraint	**Meaning in this work**
Memory	Peak VRAM or RAM needed during inference or fine-tuning
Latency	Time required to generate output tokens
Throughput	Tokens generated per second
Storage	Disk size of model checkpoints and adapters
Accuracy	Retained task performance after compression and recovery

A realistic compressed model should balance these constraints rather than optimizing only one of them. High sparsity is not useful if accuracy collapses, and recovery is not useful if it removes the efficiency advantage. Therefore, this work evaluates pruning and recovery together using both model-quality and deployment-oriented metrics.

Chapter 3

Literature Review

3.1 Efficient LLM Inference

Efficient inference is one of the central issues in deploying Large Language Models outside large server environments. A model with billions of parameters requires memory not only for storing weights, but also for handling activations, key-value cache, and repeated forward passes during autoregressive generation. Since each token is generated sequentially, inference cost becomes important even when the model already fits in memory.

Several directions have been explored to make LLM inference more practical. Quantization reduces the numerical precision of weights, for example from FP16 or BF16 to 8-bit or 4-bit formats. This can reduce memory usage and make model loading easier on smaller GPUs. QLoRA follows this general direction by keeping the base model in a low-precision form while training low-rank adapters.

Pruning is another important approach. It removes selected weights or structures from the model so that the model becomes sparse or smaller. However, pruning LLMs is more difficult than pruning smaller networks because the internal representations are highly distributed across layers. Removing weights without considering their role in actual computation can damage language modeling and reasoning behaviour.

Parameter-efficient adaptation is also relevant to efficient use of LLMs. LoRA freezes the base model and trains low-rank updates, while DoRA extends this idea by separating weight representation into magnitude and direction. These methods do not compress the base model by themselves, but they reduce the cost of task adaptation and recovery.

In this dissertation, efficient inference is treated as a combined problem. Model size, sparsity, adapter cost, memory footprint, throughput, and task quality must be considered together. A model that saves memory but loses too much reasoning ability is not useful. Similarly, a sparse model is not automatically efficient unless the inference backend can benefit from sparsity.

3.2 Pruning of Large Language Models

Pruning is a long-standing compression technique in neural networks, but its use in LLMs is more delicate. The main idea is to remove weights or components that contribute less to the model output. The challenge is deciding which parts are truly less important in a model with billions of parameters and complex layer interactions.

Magnitude pruning is the simplest form of unstructured pruning. It removes weights with smaller absolute values, assuming that small weights are less important. This idea is easy to implement, but it ignores the fact that a small weight connected to a frequently active input channel may still have a strong effect on the layer output. This limitation becomes important in Transformer models, where activation behaviour changes across layers, tokens, and inputs.

SparseGPT addresses pruning as a layer-wise reconstruction problem. It uses second-order information to decide which weights can be removed while keeping the layer output close to the original output. This makes it more informed than simple magnitude pruning, but it also increases computational cost. SparseGPT showed that GPT-style models can be pruned in a one-shot manner using layer-wise reconstruction and second-order information [3].

Wanda takes a simpler activation-aware route. Unlike SparseGPT, Wanda does not require Hessian inversion, weight reconstruction, or retraining after pruning. For each weight, the importance score is computed as

$$S_{ij} = |W_{ij}| \cdot \|X_j\|_2,$$

where W_{ij} is an individual weight and X_j represents the activation statistics of the corresponding input feature.

This scoring rule is directly relevant to this dissertation because the pruning stage in our pipeline uses Wanda. The method ranks weights using both their magnitude and activation statistics, while remaining lightweight enough to produce multiple pruned checkpoints. Wanda is therefore suitable for studying 20%, 30%, and 50% sparsity under the same experimental framework [6].

Structured pruning removes larger units such as attention heads, neurons, channels, or complete blocks. It can be more hardware-friendly because it changes dense matrix dimensions, but it may also remove useful capacity more aggressively. LLM-Pruner follows this line by detecting structurally dependent components and pruning them before applying LoRA-based recovery [5]. In this dissertation, Wanda-based unstructured pruning is used because the focus is on controlled sparsity, pruning degradation, and PEFT-based recovery rather than direct hardware speedup.

3.3 Post-Pruning Performance Degradation

Pruning changes the computation of a model that was already trained. If pruning removed only redundant weights, the effect would be small. In practice, some useful weights may also be removed, especially when sparsity becomes high. Because of this, pruned LLMs can show degraded performance.

This degradation can appear in different forms. The model may lose language modeling quality, which is usually reflected through increased perplexity. It may also lose reasoning ability, which becomes visible on tasks such as GSM8K. A pruned model can still produce fluent-looking text but fail in arithmetic, ignore a condition in the question, or produce an inconsistent final answer.

For this reason, pruning must be evaluated before any recovery step. Without measuring the raw pruned checkpoint, it is not possible to know how much damage was caused by pruning or how much was repaired by recovery. This dissertation therefore follows a three-stage comparison: dense baseline, raw pruned model, and recovered pruned model.

The degree of degradation also depends on the sparsity level. Mild pruning may preserve much of the model behaviour, while aggressive pruning may remove too much useful capacity. This is why the experiments use multiple sparsity levels instead of a single pruning configuration.

Recent work also indicates that pruning quality should not be judged only from the pruning mask. The recovery step can strongly influence the final quality of the compressed model, which supports the need to study pruning and recovery together [7].

3.4 Recovery of Compressed Models

Recovery is required because raw pruning can reduce both language modeling quality and reasoning accuracy. Full fine-tuning is a direct way to recover performance, but it is expensive for LLMs because model weights, gradients, optimizer states, and activations must all be handled during training. This is not suitable for the constrained hardware setting considered in this dissertation.

Parameter-efficient fine-tuning offers a more practical recovery method by training only a small number of adapter parameters while keeping the base model mostly frozen. In post-pruning recovery, the adapter is trained on a pruned model rather than on the original dense model, so it must compensate for both task adaptation and part of the pruning damage; recent work also supports the view that pruned models benefit from recovery methods that account for information lost during pruning [2].

LoRA is one common form of parameter-efficient adaptation. It represents the update

to a weight matrix using low-rank matrices:

$$W' = W + \Delta W = W + BA,$$

where A and B are trainable low-rank matrices. Since only these matrices are updated, the number of trainable parameters is much smaller than full fine-tuning. This makes LoRA-style recovery suitable for large models under limited memory.

QLoRA extends this idea by loading the base model in 4-bit precision while training LoRA adapters. This reduces memory usage during recovery and makes repeated adapter training more feasible. DoRA modifies the low-rank adaptation mechanism by separating magnitude and direction, which may behave differently when the base weights have already been changed through pruning.

Recovery should not be assumed to succeed automatically. A method may recover performance well at low sparsity but fail at high sparsity. Therefore, this dissertation evaluates QLoRA and DoRA on multiple Wanda-pruned checkpoints and compares the recovered models using both WikiText-2 perplexity and GSM8K accuracy. The recovered model must improve over the raw pruned model, remain meaningful compared with the dense baseline, and still make sense from an efficiency point of view.

3.5 Quantization and Low-Rank Adaptation

Quantization and low-rank adaptation are important for making recovery feasible under limited GPU memory. Quantization reduces the precision used to store model weights, while low-rank adaptation reduces the number of trainable parameters during fine-tuning. In this dissertation, both ideas are relevant because the recovery stage is performed on Wanda-pruned Llama-3.2-3B-Instruct checkpoints rather than through full fine-tuning.

QLoRA is important in this context because it keeps the base model in 4-bit quantized form while training only LoRA adapter weights. It uses techniques such as NF4 quantization, double quantization, and paged optimizers to reduce memory usage during fine-tuning [1]. This makes it practical for recovering multiple pruned checkpoints under memory-constrained experimental conditions.

A simple block-wise quantization scheme can be written as

$$X_{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(X_{\text{FP32}})} X_{\text{FP32}}\right),$$

where the scaling term maps floating-point values into the representable integer range. The reverse step, dequantization, approximately reconstructs the floating-point value for computation:

$$\text{dequant}(c, X_{\text{Int8}}) = \frac{X_{\text{Int8}}}{c}.$$

Quantization reduces memory and storage cost, but it can also introduce approximation error. This is especially important in language models, where outliers in weight distributions can affect the quantization scale. For this reason, QLoRA is useful not merely as a fine-tuning technique, but as a memory-efficient recovery method.

LoRA approaches efficiency from a different direction. Instead of updating the whole pretrained weight matrix, it represents the update using low-rank matrices:

$$W' = W_0 + \Delta W = W_0 + BA,$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ are trainable matrices and r is much smaller than the original matrix dimensions. The original weights remain frozen, so optimizer memory, gradient storage, and checkpoint size are reduced.

DoRA modifies this view by decomposing the pretrained weight into magnitude and direction. The decomposition can be written as

$$W = m\frac{V}{\|V\|_c} = \|W\|_c\frac{W}{\|W\|_c},$$

where m represents the magnitude component and V represents the directional component. DoRA then applies a LoRA-style update to the direction while also allowing the magnitude component to be adjusted. The adapted weight can be written as

$$W' = m\frac{V + \Delta V}{\|V + \Delta V\|_c} = m\frac{W_0 + BA}{\|W_0 + BA\|_c}.$$

This formulation is relevant after pruning because Wanda changes the effective weight structure by setting selected weights to zero. A simple additive low-rank update may recover part of the lost behaviour, while magnitude–direction adaptation may produce a different recovery pattern. Therefore, this dissertation compares QLoRA and DoRA experimentally instead of assuming one method to be superior in advance. DoRA extends LoRA by separating pretrained weights into magnitude and direction components [4].

3.6 Deployment-Oriented Evaluation

Most LLM evaluations focus on benchmark accuracy or perplexity, but deployment requires additional measures. A model is practically useful only if it can be loaded, evaluated, and used within available memory and time limits. This is especially relevant in this dissertation because the goal is not only to recover accuracy after pruning, but also to understand whether

the recovered model remains meaningful under constrained hardware conditions.

The evaluation therefore combines task quality and efficiency. GSM8K accuracy measures mathematical reasoning, while WikiText-2 perplexity measures language modeling quality. Along with these, peak VRAM, throughput, latency, and model or adapter size are considered. These measures help avoid a narrow conclusion based only on accuracy or perplexity.

The list of important deployment-oriented measurements is summarized in the following table.

Table 3.1: Deployment-oriented evaluation indicators

Indicator	**Purpose in this dissertation**	**Preferred Direction**
Peak VRAM	Measures maximum GPU memory required during evaluation or recovery.	Lower
Throughput	Measures generated tokens per second during inference.	Higher
Latency	Measures time required for generation under a fixed setting.	Lower
Perplexity	Measures language modeling quality on WikiText-2.	Lower
GSM8K Accuracy	Measures final-answer correctness on mathematical reasoning problems.	Higher
Model/Adapter Size	Measures storage burden of checkpoints and recovery adapters.	Lower

This distinction is important because unstructured sparsity does not automatically give wall-clock speedup. If the inference backend still uses dense matrix multiplication, many zero weights may remain inside dense tensors during computation. In contrast, structured or hardware-aware sparsity may offer stronger runtime benefits when supported by the hardware and software stack.

For this reason, recovery is not evaluated only by accuracy gain. A recovered model should improve over the raw pruned checkpoint, but it should also remain feasible in terms of memory, throughput, and storage. The useful region lies between performance recovery and practical deployment cost.

3.7 Research Gap

Existing LLM compression research has made strong progress in pruning, quantization, and PEFT, but these directions are often studied separately. In practice, deployment connects them. A model may be pruned successfully but still require recovery, and a PEFT method that works well on a dense model may behave differently on a pruned checkpoint.

The first gap addressed here is the need for a unified post-pruning recovery pipeline. Many works study pruning quality or adapter-based tuning independently, while this dissertation follows a fixed sequence: dense baseline evaluation, Wanda pruning at multiple sparsity

levels, raw pruned evaluation, QLoRA/DoRA recovery, and final deployment-oriented evaluation.

The second gap concerns evaluation behaviour. Perplexity and reasoning accuracy do not always move together. A recovered model may improve WikiText-2 perplexity but show limited GSM8K recovery, or the reverse may happen. Hence, both WikiText-2 and GSM8K are used so that language modeling quality and mathematical reasoning are not treated as identical.

The third gap is deployment interpretation. Compression literature often reports sparsity or parameter reduction, but actual usefulness depends on memory, throughput, latency, and task quality. In particular, unstructured sparsity may not produce proportional speedup unless the inference backend can exploit it.

Finally, different PEFT recovery methods must be compared after pruning. QLoRA is attractive because it reduces memory burden during recovery, while DoRA is relevant because of its magnitude–direction decomposition. This dissertation connects activation-aware LLM pruning [6], memory-efficient adapter recovery [1], and weight-decomposed low-rank adaptation [4], and studies them under one consistent evaluation pipeline.

3.8 Positioning of This Work

This work is positioned at the intersection of LLM pruning, parameter-efficient recovery, and constrained deployment. It does not introduce a new pruning algorithm or a new adapter architecture. Instead, it studies how established methods behave when combined in a single experimental pipeline.

The base model used in the study is Llama-3.2-3B-Instruct. Wanda pruning is selected because it provides activation-aware post-training pruning without requiring full retraining before pruning. QLoRA and DoRA are selected as recovery methods because both are parameter-efficient, but they represent adaptation differently.

The experimental pipeline can be written as

$$M_s = P_{\mathrm{Wanda}}(M, s),$$

where M is the original LLM, s is the sparsity level, and P_{Wanda} denotes Wanda pruning. Recovery is then applied as

$$M_s^{\mathrm{rec}} = R_{\mathrm{PEFT}}(M_s, D_{\mathrm{train}}),$$

where D_{train} is the recovery training data.

This structure allows the study to compare three model states: the dense baseline, the raw Wanda-pruned model, and the PEFT-recovered model.

Chapter 4

Proposed Methodology

4.1 Overview of the Proposed Framework

The methodology used for this work involves a pruning-recovery pipeline instead of the compression method. This method starts by taking a highly parameterized LLM that undergoes evaluation through an immutable framework. Then, Wanda pruning is implemented for particular levels of sparsity, followed by evaluation of the pruned checkpoint before recovering parameters to establish the degree of useful performance possible. The procedure adopted is of significant importance because recovery performance becomes relevant only if the performance of the original baseline and the pruned LLM have been quantified.

There are three main steps in the pipeline. Suppose M represents the original pretrained LLM and s represents the sparsity level desired. Wanda pruning results in a sparse model denoted as

$$M_s = P_{\text{Wanda}}(M, s),$$

where P_{Wanda} is the activation-aware pruning operation. The pruned model M_s is evaluated immediately to record the damage caused by pruning. Recovery then follows using a parameter-efficient method:

$$M_s^{\text{rec}} = R_{\text{PEFT}}(M_s, D_{\text{train}}),$$

where R_{PEFT} is either QLoRA or DoRA recovery and D_{train} is the recovery training dataset. Finally the original model M, the raw pruned model M_s, and the recovered model M_s^{rec} are compared under the same evaluation settings.

Figure 4.1 summarizes the complete experimental pipeline used in this dissertation. The baseline dense model Llama-3.2-3B-Instruct is tested initially, followed by pruning at three different sparsities: 20%, 30%, and 50% with Wanda. Then the pruned but uncorrected checkpoints are tested before any recovery operation, and finally, the QLoRA and DoRA adapters are fine-tuned and combined with the respective pruned base models.

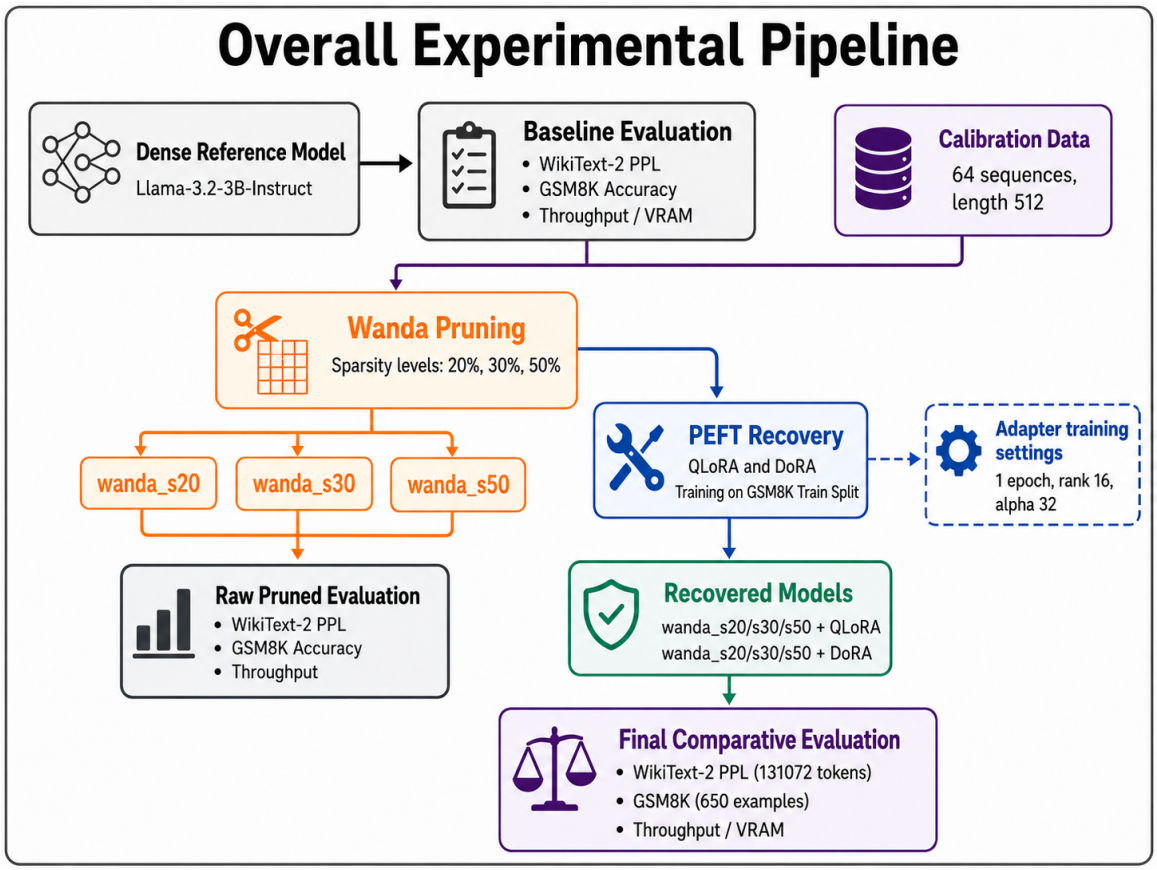

Figure 4.1: Overall experimental pipeline followed for pruning, recovery, and evaluation of Llama-3.2-3B-Instruct.

The unaltered, pruned model takes a significant place in this scheme. Without Ms measurement, it would be impossible to distinguish between deterioration caused by pruning and any potential improvement through subsequent recovery. If a restored model demonstrates high quality of performance, the interpretation of this result should take into account how much the initial deterioration was. The poor quality of the restoration process does not automatically imply the ineffectiveness of adaptation; on the other hand, it might be because the sparsity chosen is simply too large for the model.

Another aspect that shapes the design is related to implementation. Precision alone is not enough – if a pruned model is meant for limited hardware, it must be tested on parameters such as memory usage, speed, and runtime behaviour. Hence, the proposed approach combines an evaluation of the performance of the model with the analysis of its efficiency. The critical question to be asked is whether it is possible to bring the usefulness of the LLM back without undermining the reasons for pruning.

4.2 Model Selection

The experiments used the Llama-3.2-3B-Instruct base model. 3B-scale models were chosen as a good tradeoff between having enough size to learn interesting things about pruning and recovery while being feasible to fit into GPU memory and evaluate in a reasonable amount of time. Bigger models would make it impractical to perform repeated rounds of pruning and recovery at different sparsity ratios and using two different methods. Meanwhile, much sparser models will fail to demonstrate the deployment problems typical for LLMs.

At the same time, a model of this size fits well into the scope of the research focused on end-devices. While larger models may be a more likely candidate for deployment on a device with limitations, models of this size already require proper memory management without being overly large. They are too big to be run effortlessly, but they can fit a small GPU. Thus, Llama-3.2-3B-Instruct is a good middle ground for this study.

In this study, the model is viewed as a pretrained base model. The goal is not to train an LLM from scratch. The focus is on achieving certain results when you have a pretrained checkpoint available. This makes it useful for exploring real research cases where it is impossible to train an LLM from scratch, but pruning and adaptation are still viable.

To start, the dense model was evaluated before any pruning took place. This is done in order to create a basis for comparison. Once pruning was completed, the model was assessed both in relation to the raw pruned checkpoint and to the dense version of the model.

Table 4.1: Base model configuration

Component	**Description**
Dense reference model	Llama-3.2-3B-Instruct
Model family	Decoder-only transformer language model
Approximate scale	3 billion parameters
Model usage in this work	Dense reference model for Wanda pruning, PEFT recovery, and deployment-oriented evaluation
Pruning method applied	Wanda activation-aware pruning
Recovery methods applied	QLoRA and DoRA
Primary evaluation datasets	WikiText-2 and GSM8K
Main deployment indicators	Peak VRAM, throughput, latency, and model/adapter storage

4.3 Dataset Selection

The selection of the two datasets is motivated by their ability to characterize two different properties of the model. WikiText-2 is selected as the evaluation dataset for evaluating the performance of the model on language modeling using perplexity. GSM8K is used as the dataset for evaluating the mathematical reasoning capability of the model using final-answer accuracy. The reason behind choosing the two datasets is that one is useful for characterizing

the model's capability of language prediction, whereas the other is useful for the reasoning capability of the model.

This is necessary due to the fact that pruning does not necessarily affect every property of the model equally. A model can continue predicting text fluently while having a degraded reasoning capability. It may show relatively small changes in perplexity but much larger drops in the accuracy of reasoning. Having both datasets gives a balanced view regarding how much damage has been caused by pruning and whether recovery can be achieved.

Besides evaluation datasets, Wanda also needs some calibration datasets. This is because the Wanda pruning algorithm uses the activation values at the inputs to calculate the importance of each weight. The activation values are calculated by using text sequences. However, this data is not used for supervised learning in pruning.

4.3.1 WikiText-2

WikiText-2 is used for testing the performance of language models. The model operates on text sequences, and the accuracy of its predictions can be measured via perplexity, the lower the value of which, the more probable it is that the model predicts the right tokens from the test text.

For our experimental setup, WikiText-2 will primarily be used for estimating the influence of pruning on the overall prediction capability of the model. The model loses some weights, which could make its inner representation unstable and, hence, result in increased perplexity. After recovery, a decrease in perplexity would show that the model recovers its language modeling abilities to a certain extent.

Perplexity is calculated using the formula below.

$$\text{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p(x_i)\right),$$

where N is the number of evaluated tokens and $p(x_i)$ is the probability assigned to the correct token x_i. In the experiments the same evaluation script and token processing setup are kept fixed across dense, pruned, and recovered models so that the comparison remains fair.

4.3.2 GSM8K

GSM8K acts as a benchmark to test the mathematical reasoning ability of the model. This benchmark dataset consists of math word problems that require understanding of the problem, some intermediate reasoning process, and providing an end numerical answer. In the current research paper, GSM8K is especially useful since reasoning problems are known to be sensitive to pruning. Thus, although a solution generated by the model may be coherent, any arithmetic

mistake or confusion about a quantity will result in an inaccurate final solution.

The model's accuracy is measured by the correctness of its final solution. The model provides a solution for a specific problem, and the final answer is evaluated against the true value. Accuracy is calculated as:

$$\text{Accuracy} = \frac{\text{Number of Correct Answers}}{\text{Total Number of Questions}}.$$

A fixed sample size, seed, prompt structure, generation length, and extraction mechanism of the answers is used whenever possible. This is crucial due to the potential influence of small variations in the implementation process on the evaluation of GSM8K. For example, slight changes in the prompt format and maximum generation length could lead to different numbers of correct answers. Since the current study involves comparing three models, the priority should be given to consistency rather than personalization of the prompts.

4.3.3 Calibration Data for Pruning

Wanda pruning requires calibration data to estimate activation statistics. During calibration a small number of input sequences are passed through the model, and the input activations of linear layers are collected. These activations are then used in the Wanda importance score

$$S_{ij} = |W_{ij}| \cdot \|X_j\|_2,$$

where W_{ij} is a weight and X_j represents the activation values corresponding to the input channel j. Weights with lower scores are pruned according to the target sparsity level.

This information is significant since Wanda does not judge the importance of a weight based on its size alone, but also considers how active its corresponding input is. This relationship makes the process of pruning become more consistent with how the model computes. At the same time, calibration is based on the presumption that the selected examples are representative of the kind of text the model deals with. If this information turns out to be too specific, then the statistics about activations will prove to be misleading.

4.4 Baseline Evaluation

Baselining formed the first step in the experiment procedure. The densely connected Llama-3.2-3B-Instruct model was evaluated against the benchmark datasets without applying any form of pruning and recovery. This formed the basis of comparison throughout the experiment. Without baselining, it would not have been easy to ascertain whether pruning significantly degraded the model's performance or whether recovery improved its performance back to significant levels.

Baseline performance testing involved the use of different performance measures. WikiText-2 perplexity was used to measure language modeling capabilities, while GSM8K accuracy was used to evaluate reasoning capabilities. Practical performance measures such as generated tokens, throughputs, latencies, and peak VRAM were also collected when possible.

Table 4.2: Baseline evaluation template

Metric	**Purpose**
WikiText-2 Perplexity	Measures general language modeling quality.
GSM8K Accuracy	Measures mathematical reasoning through final-answer correctness.
Correct Answers	Counts the number of correctly solved GSM8K problems.
Total Generated Tokens	Records the total number of generated output tokens during evaluation.
Throughput	Measures generated tokens per second.
Peak VRAM	Measures maximum GPU memory usage during evaluation.
Latency	Measures response generation time under a fixed setting.

In another one, the same evaluation procedure was used throughout the entire work. Namely, the same scripts were applied to evaluate dense, pruned, and recovered models under the same conditions, whenever possible. Such a choice helps address a common problem in experiments, which is that differences in the result obtained at various stages can be attributed not to real improvement or worsening of model performance but merely to changes in the experiment setting.

Concerning GSM8K, the baseline experiment was conducted with a set amount of examples and the same random seed. The algorithm of answers' extraction was consistent among all models. Similarly, for WikiText-2, the process of text splitting and tokenization was the same. While these considerations may seem trivial at first glance, they matter within the context of a comparative study since the difference in a few answers in the GSM8K data set can affect the conclusions, especially considering the limited size of the subset.

Finally, the baseline stage also gave an idea of the original deployment cost of the model before the experiment. Specifically, the memory footprint and the speed of generation were determined, and then these metrics could be used as a basis to understand whether the process of pruning and recovering has positively affected the compatibility of the model with low-end hardware.

4.5 Wanda Pruning Methodology

After baseline evaluation, Wanda pruning was applied to the dense model. I selected Wanda because it is a post-training pruning method that uses both weight magnitude and activation information. This fit the project setting, where full retraining before pruning was not practical

under the available compute budget. Wanda instead allowed pruning through a calibration set and a relatively lightweight activation collection process.

The pruning was applied mainly to the linear layers of the Transformer model. These layers contain a large portion of the model parameters, so they were natural targets for sparsification. The first embedding layer and final output-related components were treated carefully, since aggressive pruning in such parts may directly disturb token representation or output prediction. The pruning flow was layer-wise: collect activations, compute importance scores, select low-scoring weights, and set those weights to zero according to the target sparsity.

In this study, pruning was not treated as the final solution. It was the compression stage that created sparse models for later recovery. This distinction is important. A raw pruned model may lose language modeling quality or reasoning ability. Because of that, every pruned checkpoint was evaluated before recovery, so the damage caused by pruning remained visible and measurable.

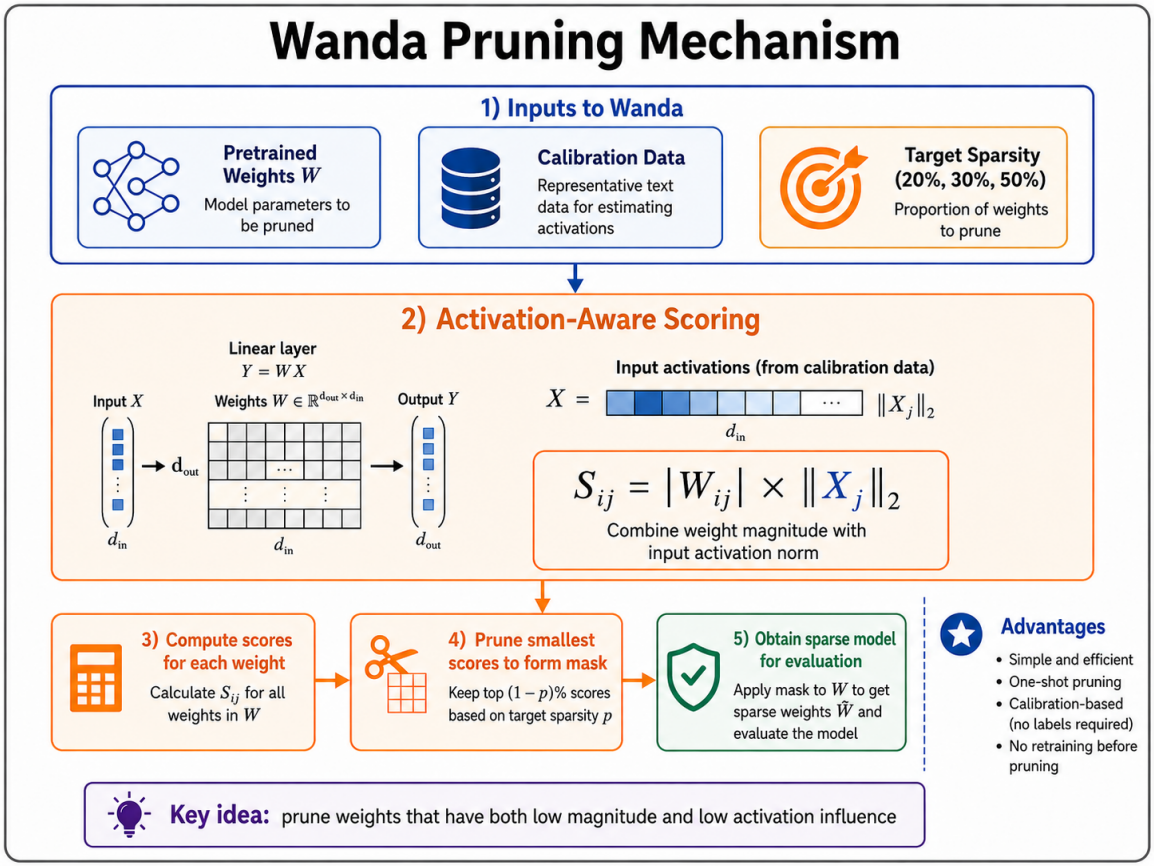

Figure 4.2: Wanda pruning mechanism used to compute activation-aware importance scores and generate sparse pruned checkpoints.

Figure 4.2 illustrates the Wanda pruning mechanism used in this work. The algorithm first uses a limited dataset for propagation within the network to obtain the activation statistics. Wanda then computes the importance of each connection based on the product of its weight magnitude and activation norm. Connection weights that are less important are pruned depend-

ing on the desired sparsity, while more important connections are preserved.

4.5.1 Activation Collection

The reason why activation statistics are necessary for Wanda is that, apart from weights themselves, the pruning algorithm needs information about how frequently the respective input channels are activated. When collecting activations, the model was fed several text sequences and the activations on input for the target linear layers were collected.

In this particular implementation, it is important to note that the size of the activation dataset was small enough so that activation collection was feasible under memory limitations of the GPU. For reproducibility purposes, the number of sequences and their length were defined in advance, and they remained constant for all variations of Wanda pruning.

It is important to note that these sequences were not used as supervised fine-tuning data. The learning process involving label information was not yet implemented. These examples were collected only to estimate the active hidden input space during regular inference passes.

4.5.2 Importance Score Computation

For the target linear layers, Wanda assigns an importance value to each weight considering both weight magnitude and input activation norms. The formula for the importance is given by

$$S_{ij} = |W_{ij}| \cdot \|X_j\|_2,$$

where W_{ij} is the weight connecting input channel j to output channel i, and X_j represents the collected activation values for the corresponding input channel. The term $\|X_j\|_2$ captures how strongly that input feature is used across the calibration samples.

The importance calculated in this manner holds great significance since only the weight magnitude may not correctly evaluate the importance. For instance, a weight which is smaller in magnitude but has very high input activation values and frequent activity might play a significant role, whereas a bigger weight having less activity might have lesser importance during actual inference.

4.5.3 Sparsity Levels

The investigation uses several sparsity levels instead of using only one. It is justified due to the fact that the degree of pruning greatly affects performance loss and the complexity of recovery. At low levels of sparsity, pruning allows retaining more behavior of the model, while at high levels of sparsity, pruning results in more compactness. However, the extent of the damage caused to the model is harder to address.

The pruning configurations used in this project are summarized in Table 4.3.

Table 4.3: Pruning configurations

Configuration	**Sparsity**	**Pruning Method**	**Recovery Applied**
wanda_s20	20%	Wanda	QLoRA / DoRA
wanda_s30	30%	Wanda	QLoRA / DoRA
wanda_s50	50%	Wanda	QLoRA / DoRA

Three scenarios provide opportunities for comparison between moderate, moderate-to-aggressive, and aggressive pruning. In case of 20%, pruning should allow keeping more capabilities of the original model. Scenario of 30% serves as the median one to understand better whether pruning damage and its recovery have been possible. Finally, scenario of 50% can be characterized as the most radical scenario that will help to investigate the possibilities of recovery.

4.6 Post-Pruning Evaluation

After each pruning of a Wanda-trained model, a pre-recovery evaluation took place before any sort of recovery attempt. This preliminary process ensures isolation of the effect caused by the pruning. Ignoring this step could lead to an inability to interpret the results, as it might be difficult to know whether the adapter managed to recover enough performance or had simply optimized the model to approach the baseline.

For the evaluation after pruning, identical datasets and process were used to those in the baseline phase. For measuring the effect of pruning on language modeling, the perplexity score was calculated on the WikiText-2 dataset. The accuracy on the GSM8K dataset was measured to determine the effect on math reasoning. Throughput and maximum VRAM usage are reported whenever possible.

The central comparison at this point is

$$M \to M_s,$$

where M is the dense model and M_s is the pruned model at sparsity level s. The difference between their results shows the performance degradation caused by pruning. Later, after recovery, the comparison becomes

$$M \to M_s \to M_s^{\text{rec}}.$$

The following three-stage approach outlines the primary experimentation process used in this paper: first, identify how well the dense model performs; second, quantify the level of impairment caused by pruning; and lastly, determine the extent to which that impairment can

be recovered using PEFT methods.

However, this pruning-induced damage may vary significantly among different evaluation criteria. For instance, while perplexity would reduce slowly, GSM8K accuracy may fluctuate more irregularly. Reasoning-oriented tasks depend on multistage generation of outputs and consistent final answers. The pruned model might still generate coherent text but be incapable of performing arithmetic, neglecting some conditions, or calculating the wrong answer. Therefore, post-pruning evaluation does not serve as an ordinary benchmark; instead, it demonstrates what abilities have been impaired by the compression process.

4.7 Recovery Using PEFT

Upon the assessment of the unrefined pruned models, recovery is carried out through parameter-efficient fine-tuning. It is worth noting that full fine-tuning is not used because it requires updates for all the parameters of the model, and it is simply too costly, especially within the confines of the current hardware resources. More importantly, this approach goes against the practical aim of the study, in which efficiency in terms of recovery is prioritized over re-training.

In this case, parameter-efficient fine-tuning approaches prove to be more applicable since they involve minimal changes to the base model by keeping it mostly frozen while training just a few new parameters. In particular, the recovery methods used here are QLoRA and DoRA. Both approaches belong to the adapter-type techniques; however, they differ in how they represent their update process. The comparison of these two will reveal whether the recovery outcome depends on the choice of adapter type, aside from pruning, that is.

For each pruned model M_s, recovery is performed as

$$M_s^{\text{rec}} = R_{\text{PEFT}}(M_s, D_r),$$

where D_r denotes the recovery training data and R_{PEFT} denotes the selected recovery method. The recovered model is then evaluated using the same protocol as the baseline and raw pruned models.

4.7.1 QLoRA Recovery

QLoRA recovery involves low-rank adaptation where the base model remains frozen and memory usage is optimized. QLoRA recovery comes in handy in this research because it allows one to conduct the recovery experiment without having to pay for the associated memory expenses that come with fine-tuning. The pruned model acts as the fixed backbone while the adaptable weights get modified.

The LoRA-style update can be written as

$$W' = W + \Delta W = W + BA,$$

where W is the frozen base weight, and A and B are low-rank trainable matrices. Since only these adapter matrices are trained, the number of trainable parameters is much smaller than the full model size.

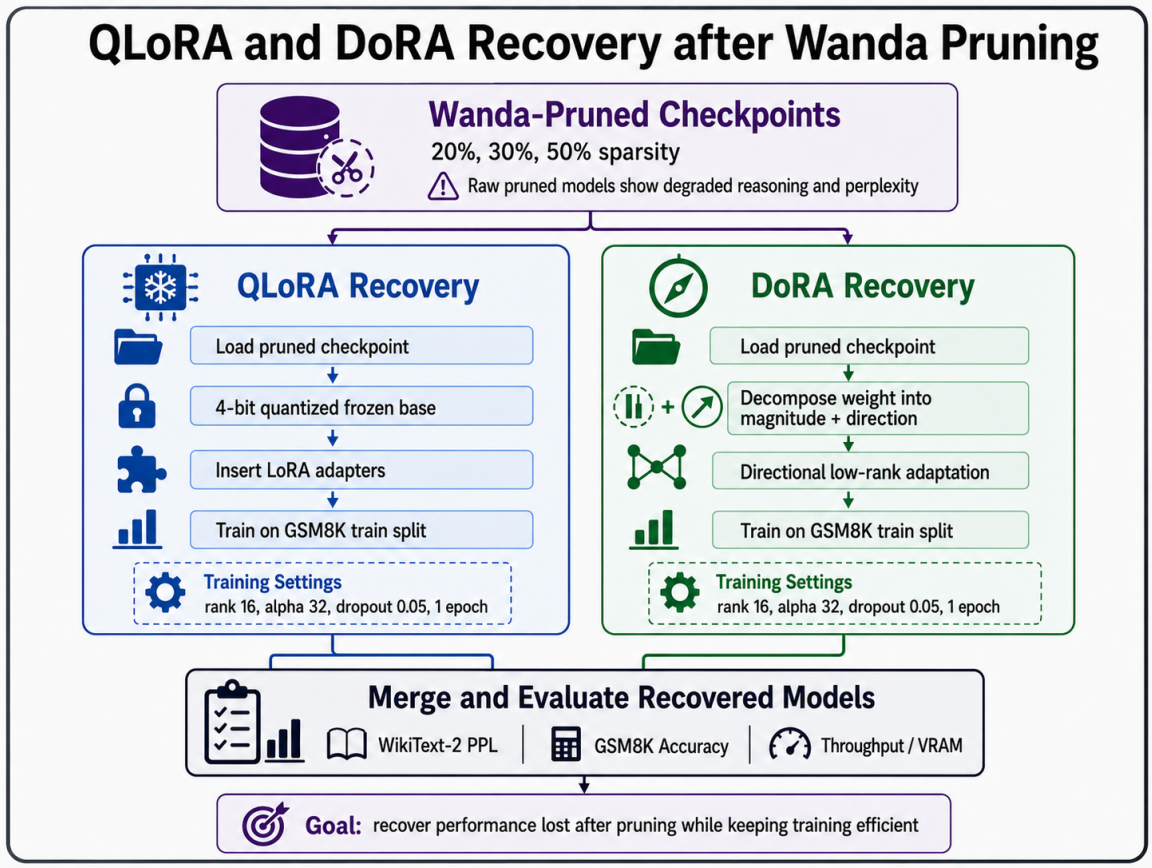

Figure 4.3: PEFT-based recovery pipeline for Wanda-pruned models using QLoRA and DoRA adapters.

Figure 4.3 shows the recovery process applied after Wanda pruning. PEFT adapters are added to the attention and projection layers in each pruned model. Memory-efficient recovery using low-rank adapters and quantization is done using QLoRA while DoRA performs magnitude–direction decomposition of the model parameters for recovery. After training on the GSM8K training subset, the trained adapters are added to the base pruned model before performing the evaluation step.

In the context of model recovery, QLoRA serves two purposes: first, as an adapter that helps adapt the model to a new task, and second, as a tool to correct degradation caused by Wanda pruning. The idea is that the adapter will be able to learn to correct the errors of the pruned model so as to perform better on the given dataset.

The setup for QLoRA model recovery includes parameters such as the adapter rank, alpha, dropout, learning rate, batch size, gradient accumulation, number of training steps, number of epochs, and maximum sequence length. All the mentioned values are fixed for each degree of sparsity of the model.

4.7.2 DoRA Recovery

DoRA recovery is included because it changes the standard low-rank adaptation mechanism. Instead of treating adaptation only as an additive low-rank update, DoRA decomposes the pre-trained weight into magnitude and direction components. The adapted weight can be written as

$$W' = m\frac{W_0 + BA}{\|W_0 + BA\|_c},$$

where m represents the magnitude component and BA gives the low-rank directional update. This is what makes DoRA unique compared to traditional LoRA-based recovery. It allows the model to control the scale and direction of the adapted weight representation.

This difference gains importance after pruning. Wanda changes the structure of the weights by removing some of them. Recovery that can control both the scale and direction of weight might behave differently from a simple additive adapter. But this does not necessarily mean DoRA will always be superior to QLoRA. The comparison takes place under equal pruning and testing conditions.

In this experiment, DoRA is run on the same set of pruned checkpoints where QLoRA was used for recovery. This will allow for a comparative study between the two recovery algorithms at levels of sparsity of 20%, 30%, and 50%.

4.7.3 Adapter Merging and Inference

After adapter recovery training, adapters can either be loaded separately with the pruned base model or merged into the model depending on the experiment setup and framework compatibility. Loading adapters separately keeps the pruned checkpoint and recovery adapter modular, which is useful for comparing QLoRA and DoRA adapters trained on the same pruned model. However, adapter merging simplifies inference because the adapter weights are incorporated into the base model.

The choice between separate adapter loading and merging depends on memory availability, precision requirements, and whether multiple adapters need to be switched during experiments. In this study, the main priority was consistent evaluation rather than deployment optimization through adapter handling alone. Therefore, the trained adapters were merged with their corresponding pruned base models before final evaluation on GSM8K and WikiText-2.

A practical point is that PEFT recovery does not make the pruned base model unnec-

essary for deployment. Although the adapter is much smaller than the base model, it still functions on top of it before merging. Hence, deployment analysis must consider the pruned model, recovery adapter, VRAM consumption, model size, and inference throughput together.

4.8 End-Device-Oriented Evaluation

Evaluation oriented toward end-device restores the connection to the actual target of our experiment. In this case, by end-device we do not mean that the produced model would actually run on some mobile phone or a real embedded machine. Instead, we use this terminology from the research perspective – to refer to inference environments that suffer from memory, computing power, and speed limitations in comparison to large cloud servers. It could be, for example, a smaller GPU, a CPU setup, or consumer machine.

The recovered pruned models are evaluated according to both task-related metrics and efficiency criteria. Among the key metrics are the following: model size after pruning, adapter size, VRAM used in inference, token throughput, latency per request, and task performance. This combination is important since no model can be considered deployable simply because it is sparse; it should also be able to work under certain hardware conditions and produce sensible outputs.

At this step, we should understand trade-offs as well as consequences. Higher sparsity allows lowering non-zero weight counts but might negatively affect GSM8K accuracy or raise perplexity. Recovering task performance increases the number of adapter parameters and might affect inference memory consumption depending on the way the adapter was trained. This means that the best model may not be the most compact but the one that achieves a reasonable trade-off.

The following metrics are collected during end-device-oriented evaluation:

Efficiency Profile = {Peak VRAM, Throughput, Latency, Model Size, Adapter Size}.

These values are interpreted together with

Performance Profile = {WikiText-2 PPL, GSM8K Accuracy}.

However, there exists one aspect that should not be overlooked. Sparsity is not proportional to the wall-clock speed-up by nature unless used along with sparse kernel computations or hardware-aware libraries for inference. Thus, I do not make any exaggerations in regards to the speed-ups achieved through the process of pruning. The analysis provided relies on real-time metrics and behavior observed during the experiment.

4.9 Complete Experimental Flow

It should be noted that there was no separation between the two phases, meaning both pruning and recovery were in one pipeline because they depended on each other sequentially. Initially, the dense model sets the standard for comparison. Subsequently, pruning based on Wanda is performed and sparse checkpoints of the selected sparsities are obtained. Then the evaluation of the raw pruned checkpoints takes place without the recovery step.

Let M denote the original dense model and let s denote a target sparsity level. Wanda pruning produces a sparse model

$$M_s = P_{\text{Wanda}}(M, s).$$

The pruned model M_s is then recovered using a PEFT method R_r, where r is either QLoRA or DoRA:

$$M_s^{\text{rec}} = R_r(M_s, D_{\text{train}}).$$

This is where we compare the three model states, namely the densely-connected base model M, the sparse pruned model M_s, and the recovered modelM_s^{rec}. There are two main hypotheses being tested here - the degree of performance loss resulting from pruning, and the recoverability of such losses.

The final stage base model in this work is Llama-3.2-3B-Instruct. Wanda activation-aware pruning was done at three different levels of sparsity: 20%, 30%, and 50%. The respective checkpoints are named as `wanda_s20`, `wanda_s30`, and `wanda_s50`. Both QLoRA and DoRA recovery were performed on each checkpoint separately, giving a total of six configurations.

It is crucial to specify what the data used in Wanda pruning means. The activation data that was used in Wanda is the calibration data, but not the validation data in the supervised learning paradigm. Namely, a relatively small batch of input sequences is passed through the model to gather activation statistics from linear layers of interest. Then this information, along with the absolute value of the weights, is used to calculate the Wanda importance score. In the pruning stage, the amount of activation collection data was equal to 64 calibration sequences with the length of 512 tokens. This data did not serve for the model training, its purpose is to provide some estimates on which weights should be pruned.

The GSM8K test was done with fixed number of samples of 650 problems and random seed of 42. Also, the maximum number of generated tokens was set to 512 per problem. The same prompt schema, the same answer extraction rules, and the same evaluation script were used in the evaluation of dense baseline model, raw pruned models, and recovered models. The decoding configuration was kept unchanged between the models. Top-p sampling parameter or other generation parameters were fine-tuned individually for each checkpoint.

Algorithm 1 Efficient Recovery Pipeline for Pruned LLMs

Input: Original pretrained model M, sparsity levels $S = \{20\%, 30\%, 50\%\}$, recovery methods $R = \{\text{QLoRA}, \text{DoRA}\}$, evaluation datasets D_{eval}, recovery training data D_{train}

Output: Evaluation results for dense, raw pruned, and recovered models

Evaluate dense baseline model M on D_{eval}
for each sparsity level $s \in S$ **do**
    Apply Wanda pruning:

$$M_s \leftarrow P_{\text{Wanda}}(M, s)$$

    Evaluate raw pruned model M_s on D_{eval}
    for each recovery method $r \in R$ **do**
        Perform PEFT recovery:

$$M_{s,r}^{\text{rec}} \leftarrow R_r(M_s, D_{\text{train}})$$

        Merge the trained adapter with the corresponding pruned base model
        Evaluate recovered model $M_{s,r}^{\text{rec}}$ on D_{eval}
    end for
end for
Compare all configurations using perplexity, GSM8K accuracy, throughput, and deployment-oriented indicators
Select the most suitable sparsity–recovery configurations under the chosen evaluation constraints

For WikiText-2 perplexity, the test data contained 131,072 tokens. The sequence length was set to 1,024 tokens with a stride of 512 tokens, ensuring that dense, pruned, and recovered models were evaluated on the same text.

During recovery, the GSM8K training split was used with 7,473 samples, and each adapter recovery task was trained for one epoch. The adapter rank was fixed at 16, LoRA alpha was set to 32, dropout was set to 0.05, and the learning rate was 2×10^{-4}. The same target modules were used for both QLoRA and DoRA: `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. Keeping the adapter positions, recovery data, pruning level, and training setup fixed made the QLoRA–DoRA comparison more controlled. For the final tests, the trained adapters were merged into the corresponding pruned base models before GSM8K and WikiText-2 evaluation, so inference used a simpler model-loading setup and avoided extra adapter computation during generation.

Chapter 5

Algorithms, Implementation, and Complexity

5.1 System Pipeline Overview

The implementation itself followed the pipeline principle, with each step built upon the preceding one. Such a structure was preserved since each subsequent step relied on the results of the previous stage. The dense model had to be evaluated first, after which the pruned checkpoints were to be created, and the recovery adapters were to be fine-tuned on the pruned checkpoints. Only then could the comparison of all the model variants be done according to the same conditions. Otherwise, the final experiment would be hard to validate.

Generally speaking, the process includes five main blocks: baseline evaluation, Wanda pruning, recovery training, evaluation, and logging/artifact management. The baseline evaluation scripts are to evaluate the original Llama-3.2-3B-Instruct model prior to the compression process. The pruning script uses the Wanda approach on the required sparsity. The recovery block involves training both QLoRA and DoRA adapters on the pruned checkpoints. Evaluation scripts are used multiple times for baseline, pruned, and recovery models to allow for comparison. Metrics, predictions, performance, and memory logs are stored with the help of logging scripts.

It may be assumed that reproducibility relies not only on the algorithms involved but also on the manner in which outputs are stored. Thus, for GSM8K, output answers will also have to be saved alongside the metrics to better understand the nature of the mistake. For the perplexity calculation, the exact configuration for evaluation will have to be preserved to interpret any differences in PPL as model-related and not script-dependent. This approach makes the experiments much easier in the case of varying sparsities and multiple recoveries.

Thus, the implementation serves as an experimental control framework. Due to the fact that multiple states of the model are compared, the evaluation process has to be carefully repeated for each state. In particular, in the case of GSM8K, even slight discrepancies in answer

extraction and length influence the evaluation result significantly.

Table 5.1: Major components of the implementation

Component	**Purpose**
Baseline evaluation scripts	Evaluate the original dense model on WikiText-2 perplexity and GSM8K accuracy.
Pruning script	Apply Wanda pruning at selected sparsity levels and save the pruned checkpoints.
Recovery scripts	Train QLoRA and DoRA adapters on the Wanda-pruned models.
Evaluation scripts	Evaluate dense, raw pruned, and recovered models under the same protocol.
Logging utilities	Save metrics, JSONL predictions, throughput, latency, and hardware usage.
Upload utilities	Save model checkpoints and adapters locally or upload them to Hugging Face Hub when required.

5.2 Baseline Evaluation Algorithm

The baseline evaluation algorithm evaluates the dense Llama-3.2-3B-Instruct model before any pruning or recovery is applied. This step establishes the original performance level of the model and provides the reference values against which all later results are compared. Since the later experiments involve several pruned and recovered variants, the baseline must be measured using the same datasets, prompt format, generation settings, and answer extraction logic. Otherwise, differences in results may come from changes in evaluation procedure rather than from pruning or recovery itself.

The algorithm first loads the tokenizer and the dense base model, then computes WikiText-2 perplexity to measure language modeling quality. After that, it evaluates GSM8K by generating answers for mathematical word problems, extracting the final numerical response, and comparing it with the gold answer. Along with perplexity and accuracy, the algorithm also records total generated tokens, throughput, latency-related behaviour, and peak VRAM consumption. These additional values are useful because the dissertation does not evaluate only accuracy, but also the practical feasibility of using the model under constrained hardware conditions.

Although baseline evaluation may look like a simple preliminary step, it plays an important role in the entire experimental design. It defines the common evaluation standard for dense, pruned, and recovered models. If the baseline results were not reliable, later claims about degradation after pruning or improvement after recovery would become weak. For that reason, the baseline evaluation is treated as the foundation of the experimental study.

Algorithm 2 Baseline Evaluation

Input: Dense model M, WikiText-2 evaluation set D_{ppl}, GSM8K evaluation set D_{gsm}

Output: Baseline perplexity, GSM8K accuracy, generated outputs, and efficiency metrics

Load tokenizer and dense model M
Tokenize D_{ppl} using the model tokenizer
Compute next-token prediction loss on D_{ppl}
Calculate perplexity from the average loss
for each question $q_i \in D_{\text{gsm}}$ **do**
    Generate model response $\hat{y}_i$ using the fixed prompt template
    Extract the final numerical answer from $\hat{y}_i$
    Compare the extracted answer with the gold answer
end for
Compute GSM8K accuracy
Record generated tokens, throughput, latency, and peak VRAM
Save metrics and prediction outputs

5.3 Wanda Pruning Algorithm

The Wanda pruning technique is implemented after baseline analysis. In this pruning technique, the magnitude of the weight is used along with the activation norm of its input channel. Therefore, this technique can be considered more appropriate than basic pruning because the value of LLM weights cannot always be interpreted correctly through their magnitude.

To compute the Wanda score, the activation for the selected target layer will be extracted using calibration data, and the Wanda score will be calculated as

$$S = |W_l| \cdot \|X_l\|_2,$$

where W_l denotes the weight matrix of layer l and X_l denotes the collected input activation statistics for that layer. The lowest-scoring weights are selected according to the required sparsity level and set to zero. The resulting sparse model is saved as a pruned checkpoint.

This process is carried out for all the selected sparsity rates. During pruning, there is no recovery process as well. The pruning stage will only yield the pruned model without recovery. Before adapting the pruned model, it will be assessed individually to separate the impact of pruning from adaptering process.

Algorithm 3 Wanda Pruning

Input: Dense model M, calibration data D_c, target sparsity level s

Output: Wanda-pruned model M_s

Load dense model M
for each target linear layer l in M **do**
    Pass calibration samples from D_c through the model
    Collect input activation statistics X_l for layer l
    Obtain the layer weight matrix W_l
    Compute Wanda importance score:

$$S_l = |W_l| \cdot \|X_l\|_2$$

    Identify the lowest-scoring weights according to sparsity level s
    Set the selected weights to zero
end for
Save the pruned checkpoint as M_s
Return M_s

5.4 Recovery Training Algorithm

After pruning, there is usually some degradation in the language modeling and reasoning capabilities of the model. This happens because pruning removes selected weights from a model that has already learned its internal representations during pretraining and instruction tuning. Even when the pruned model remains fluent, its reasoning steps may become less reliable.

Recovery aims to reduce this degradation through parameter-efficient fine-tuning. Unlike full fine-tuning, all base model parameters are frozen, and training is performed only on adapter parameters. This keeps the recovery process feasible under the limited-resource assumption used in this dissertation. At a conceptual level, the recovery algorithm is common to both QLoRA and DoRA.

In the chosen PEFT approach, trainable adapter parameters are introduced into the pruned model. The model is then fine-tuned using batches from the recovery dataset, the language modeling loss is computed, and only the adapter parameters are updated. All other model weights remain fixed during this process, which makes the recovery stage lightweight compared with full model fine-tuning.

The adapter architecture differs between QLoRA and DoRA. QLoRA follows the standard low-rank update approach with memory-efficient model loading, while DoRA decomposes weights into magnitude and direction components before adaptation. Although their internal mechanisms are different, the high-level recovery pipeline remains the same for both

methods.

Algorithm 4 PEFT-Based Recovery

Input: Pruned model M_s, recovery dataset D_r, PEFT method $A \in \{\text{QLoRA}, \text{DoRA}\}$

Output: Adapter checkpoint and recovered model $M_{s,A}^{\text{rec}}$

Load pruned model M_s
Freeze base model parameters
Insert trainable adapter parameters according to method A
for each training step **do**
    Sample a mini-batch from D_r
    Run the forward pass through the pruned model and adapter
    Compute language modeling loss
    Backpropagate only through adapter parameters
    Update adapter parameters using the selected optimizer
end for
Save the trained adapter checkpoint
Merge adapter weights with the corresponding pruned base model for final evaluation
Return recovered model $M_{s,A}^{\text{rec}}$

Recovery is an important part of our experiments as pruning by itself only answers how much the model can be compressed without significant degradation. However, what we care about is whether the lightweight finetuning procedure can restore enough capability of a pruned model to make its use practical.

5.5 GSM8K Evaluation Procedure

The GSM8K evaluation methodology is performed on the baseline, pruned, and restored models to evaluate their mathematical reasoning ability. I don't consider GSM8K as a standard text-generation evaluation benchmark. In the current research, it is employed to assess the ability of the model to understand the word problem, go through the required arithmetic reasoning, and return the accurate numeric result.

The evaluation begins with the preloading of the fixed subset of examples in GSM8K with a fixed random seed. A fixed seed is critical here since multiple model variants are evaluated. The variation of the question set leads to variance in results due to the sample selection rather than model performance. Also, the prompt templates are consistent across all models to provide reliable comparison results.

For each question, the model produces a solution. The resulting text could contain the arithmetic reasoning process, calculations, and the final result of calculation. As the correct an-

swer should be numeric, the script extracts the last numeric value from the produced solution and compares it to the ground truth. Accuracy is calculated as

$$\text{Accuracy} = \frac{\text{Number of Correct Answers}}{\text{Total Number of Evaluated Questions}}.$$

As for the settings in the experiments, the number of generated tokens in each case is fixed to the same maximum number. This is important since too low generation length might cut off the reasoning process before reaching the conclusion, while too large one may result in useless reasoning continuation.

Additionally, the decoding setting remains constant so that the variations introduced by sampling procedure will not be a part of the comparison.

It should be noted that the accuracy of GSM8K is highly dependent on the answer extraction. A model can output a solution, yet use a different form of writing it down. At the same time, another one may get the correct number, but provide a weak reasoning trail. Thus, the predictions in JSONL form will be kept for qualitative analysis, as it will help to establish whether pruning helps to improve the reasoning process or just the form of the final solution.

Thus, the purpose of using GSM8K in this study is double – first, it helps to measure reasoning skills of each model numerically, and second, it helps to get examples of generation that could shed light on the effect of pruning and recovery on reasoning process.

5.6 Perplexity Evaluation Procedure

Perplexity analysis is applied to analyze the quality of language modeling. Although the GSM8K data set is designed to test math reasoning, the perplexity test is performed to evaluate the general ability of the model to predict natural language. In the current research, WikiText-2 will be utilized to perform perplexity testing.

Firstly, the evaluation text is tokenized using the tokenizer of the base model. Then, the tokenized sequence is split into blocks within the limits of context size and memory capacity. For each block, the next-token prediction loss is calculated. The losses are summarized for the evaluated tokens, and the perplexity is calculated as

$$\text{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p(x_i)\right),$$

where N is the number of evaluated tokens and $p(x_i)$ is the probability assigned by the model to the correct token x_i. Lower perplexity indicates better language modeling quality.

In the experiment, the identical text splitting, tokenizer, maximum token or char limit, and method of averaging loss are used for all model variants, as perplexity depends on the

evaluation text, splitting strategy, and tokenizer used. It was important in the experiments as we compare the performance of dense, pruned, and recovered models, and thus, the evaluation strate- gy has to be kept constant throughout.

Perplexity is beneficial because it is a more stable metric compared to exact match accuracy. The fact that a model makes a mistake while generating a GSM8K response does not automatically mean that language modeling performance is lost, although perplexity would help us determine this fact. On the other hand, perplexity is not an indicator of reasoning perform- ance, meaning that even a good one might produce incorrect results.

5.7 Hardware and Software Environment

The tests are conducted within certain constraints of the GPU environment. This is not a matter of implementation but the essence of the work since the research is based on the idea of efficient recovery and deployability.

These constraints have an impact on all aspects of the experiment, including model loading, pruning, recovery training, batch sizes, selections of precision, and the duration of evaluations. The initial exploratory experiments were run on a scaled-down Tesla P6 setup; however, the final recovery training and evaluation experiments reported in this thesis were done on a RunPod NVIDIA A40 GPU.

In case of necessary recovery or other related processes, tests can be conducted in notebook or Kagglelike environments. The implementation utilizes mainly the following tools: PyTorch, Hugging Face Transformers, PEFT, Datasets, and Accelerate. Depending on the task, the model is loaded in one of three formats: FP16, BF16, 4-bit.

The final recovery training and evaluation was performed using a paid RunPod GPU setting. Although initial tests were carried out using small GPU settings, the final results were obtained using an NVIDIA A40 GPU. The latter had enough memory to recover and test all six models, which included recovery using three sparsity values with QLoRA and three sparsity values with DoRA.

The hardware setup influences a number of design decisions. The ability to do full fine-tuning of the parameters of the models used is out of the question in the proposed setup, hence the choice of PEFT-based model recovery. Other recovery setup parameters such as batch size, gradient accumulation, sequence length, and quantized loading are also set to accommodate memory constraints so that experiments will still align with the motivation for the study.

The hardware setup is not merely a background factor in the study. It influences the research question. With unlimited computing power, full fine-tuning and model comparisons are possible tasks. In the face of constrained GPU memory, the pertinent question is whether pruning and model recovery could deliver usable results.

Table 5.2: Final hardware and software environment used for recovery and evaluation

Item	**Final Configuration**
Evaluation platform	RunPod paid GPU environment
GPU	NVIDIA A40
GPU memory	Approximately 46 GB VRAM
Base model	Llama-3.2-3B-Instruct
Pruning method	Wanda activation-aware pruning
Pruning sparsity levels	20%, 30%, and 50%
Recovery methods	QLoRA and DoRA
Framework	PyTorch
Libraries	Transformers, PEFT, Datasets, Accelerate, BitsAndBytes
Artifact backup	Hugging Face Hub

5.8 Complexity Analysis

The complexity discussion in this work is limited to the three main computational stages of the pipeline: Wanda pruning, PEFT-based recovery training, and inference. The aim is not to derive the full cost of every Transformer operation, but to explain where most of the computation and memory cost arises in the implemented pruning–recovery workflow.

5.8.1 Pruning Complexity

In Wanda pruning, the main cost comes from activation collection and importance score computation. Calibration samples are passed through the model to collect input activations for the target linear layers. This requires forward passes through the model, but no backpropagation is performed.

For a target linear layer with weight matrix W_l and collected activation statistics X_l, Wanda computes the score

$$S = |W_l| \cdot \|X_l\|_2.$$

The score is obtained by combining weight magnitude with the corresponding activation norm. Weights with the lowest scores are then selected according to the target sparsity level and set to zero. Compared with second-order pruning methods, Wanda is lighter because it avoids Hessian inversion, weight reconstruction, and retraining during pruning. Since pruning is repeated separately for 20%, 30%, and 50% sparsity, the total pruning cost increases with the number of sparsity configurations.

5.8.2 Recovery Training Complexity

Recovery training uses parameter-efficient fine-tuning, so only adapter parameters are trained while the pruned base model remains frozen. This reduces the number of trainable parameters and lowers the memory required for gradients and optimizer states compared with full fine-tuning.

For LoRA-style adaptation, the weight update is written as

$$\Delta W = BA,$$

where A and B are low-rank trainable matrices. If the original weight matrix has shape $d \times k$ and the adapter rank is r, then the number of trainable parameters for that adapted matrix is approximately

$$r(d + k),$$

which is much smaller than dk when $r \ll \min(d, k)$. This reduction is the main reason PEFT is suitable for recovering pruned models under constrained hardware. QLoRA further reduces memory pressure through quantized base-model loading, while DoRA changes the adaptation mechanism through magnitude–direction decomposition. In both cases, the expensive part of the computation still comes from forward and backward passes, but only adapter parameters are updated.

5.8.3 Inference Complexity

Inference in autoregressive LLMs remains expensive because tokens are generated sequentially. Each new token requires a forward pass through the model, and the cost depends on model size, sequence length, precision, batch size, and the efficiency of the inference backend.

Although Wanda pruning introduces unstructured sparsity, this does not automatically produce proportional speedup. If dense matrix multiplication kernels are used, zero weights may still be stored and processed inside dense tensors. Therefore, a model with 50% sparsity should not be assumed to run twice as fast unless sparse kernels or hardware support are available.

For this reason, inference efficiency in this dissertation is interpreted using measured throughput and latency rather than theoretical sparsity alone. Adapter merging can simplify the recovered model during evaluation, but the pruned base model still has to be loaded and executed. Hence, practical inference cost is judged from the observed runtime behaviour reported in the experimental results.

5.9 Reproducibility Details

Reproducibility becomes especially important in this study because of the presence of multiple models and evaluations. Minor changes in the random seed, prompts, lengths of generation, and answer extraction may impact results, particularly for GSM8K. In order to achieve this, the design of the experiment takes into account all the important parameters associated with each of the outputs produced by the study.

Important parameters associated with reproducibility include random seed value, path to the checkpoint, version of the data set, number of samples used in the evaluation, maximal length of generation, hyperparameters used in adapters, and file paths where outputs are saved. Generated responses from GSM8K are saved in JSONL format along with answers extracted and correctness labels. Summary metrics are also separately saved in JSON format to reproduce tables even if not all experiments are reproduced.

The following information is recorded for each major run:

- model name or checkpoint path;
- pruning sparsity level, where applicable;
- recovery method, such as QLoRA or DoRA;
- dataset name and split;
- evaluation sample size and random seed;
- maximum generation length for GSM8K;
- adapter hyperparameters such as rank, alpha, dropout, learning rate, batch size, and training steps;
- output paths for JSONL prediction files and metric files;
- hardware-related values such as peak VRAM and throughput.

There were many learnings involved in creating this system. Just documenting accuracy alone isn't sufficient because there's much more information conveyed through the outputs of the model than through that one number. Outputs allow for detecting incorrect computations, problems with the generation process, following the prompt, and errors with extraction. The reproducibility pipeline is also key for writing up the dissertation.

The reproducibility pipeline ensures support for writing up the final dissertation, which involves making sure the model analysis takes place under specific conditions where the outputs are saved.

Chapter 6

Experimental Results and Discussion

6.1 Experimental Design

The experiments in this chapter evaluate how much performance is lost after Wanda pruning and how much of it can be recovered using PEFT methods. The same base model, Llama-3.2-3B-Instruct, is studied in three stages: dense baseline, raw Wanda-pruned checkpoint, and recovered checkpoint.

Wanda pruning is applied at 20%, 30%, and 50% sparsity. The 20% case represents mild pruning, the 30% case represents an intermediate setting, and the 50% case represents aggressive pruning. Each pruned checkpoint is then recovered separately using QLoRA and DoRA.

The evaluated groups are the dense base model, raw Wanda-pruned models, QLoRA-recovered models, and DoRA-recovered models. All models are tested under the same protocol. WikiText-2 perplexity is used for language modeling quality, GSM8K accuracy is used for mathematical reasoning, and throughput is used as the main efficiency indicator. The raw pruned model is evaluated before recovery so that pruning damage and recovery improvement can be measured separately.

6.2 Baseline Results

The dense Llama-3.2-3B-Instruct model is evaluated first to establish the reference point for all later comparisons. The baseline results are shown in Table 6.1. In the GSM8K evaluation, 650 questions are used with random seed 42 and maximum generation length of 512 tokens. For WikiText-2, perplexity is evaluated using 131,072 tokens.

Model	**PPL**	**Correct**	**Accuracy**	**Throughput**
Llama-3.2-3B-Instruct	10.4627	462 / 650	0.7108	46.33 tok/s

Table 6.1: Baseline performance of the dense Llama-3.2-3B-Instruct model

The dense baseline gives the strongest overall result in this setup. It has the lowest WikiText-2 perplexity and the highest GSM8K accuracy among all evaluated configurations. Therefore, the recovered models must be interpreted relative to both the dense baseline and their corresponding raw pruned checkpoints.

6.3 Effect of Wanda Pruning

After establishing the dense baseline, Wanda pruning is applied at 20%, 30%, and 50% sparsity. The raw pruned checkpoints are evaluated before applying QLoRA or DoRA recovery. This step shows the direct effect of pruning without mixing it with adapter-based recovery. The results are shown in Table 6.2.

Model	**Sparsity**	**PPL**	**Correct**	**Accuracy**	**Throughput**
`wanda_s20`	20%	23.3884	328 / 650	0.5046	48.06 tok/s
`wanda_s30`	30%	25.4533	314 / 650	0.4831	47.56 tok/s
`wanda_s50`	50%	45.1633	117 / 650	0.1800	48.02 tok/s

Table 6.2: Performance of raw Wanda-pruned models before recovery

The results show clear degradation after pruning. All raw pruned models have higher perplexity and lower GSM8K accuracy than the dense baseline. The drop is moderate at 20% and 30% sparsity, but it becomes severe at 50% sparsity, where GSM8K accuracy falls to 0.1800.

This confirms that aggressive pruning affects both next-token prediction and mathematical reasoning. The 20% and 30% pruned models still retain useful capability, while the 50% model loses a much larger portion of its reasoning performance. These observations justify the need for a recovery stage using QLoRA and DoRA.

6.4 Recovery Results Using QLoRA

After evaluating the raw Wanda-pruned checkpoints, QLoRA recovery was applied at each sparsity level. The pruned checkpoint acted as the frozen base model, while the adapter parameters were trained using the GSM8K training split. The same recovery settings were used across 20%, 30%, and 50% sparsity so that the comparison remained consistent.

The QLoRA recovery results are shown in Table 6.3.

QLoRA improves all three raw pruned checkpoints. The improvement is seen in both GSM8K accuracy and WikiText-2 perplexity. The largest recovery effect appears at 50% sparsity, where the raw pruned model is most degraded. However, even after recovery, the QLoRA models remain below the dense baseline, which is expected because only adapter parameters are trained instead of the full model.

Model	Sparsity	PPL	Correct	Accuracy	Throughput
wanda_s20 + QLoRA	20%	21.7185	344 / 650	0.5292	47.49 tok/s
wanda_s30 + QLoRA	30%	23.2286	329 / 650	0.5062	46.80 tok/s
wanda_s50 + QLoRA	50%	33.8288	289 / 650	0.4446	47.77 tok/s

Table 6.3: Recovery results using QLoRA

6.5 Recovery Results Using DoRA

DoRA recovery was applied to the same Wanda-pruned checkpoints used for QLoRA recovery. The base models, sparsity levels, training data, adapter rank, and evaluation protocol were kept fixed. The main difference is that DoRA uses magnitude–direction decomposition during adaptation, unlike the standard LoRA-style update.

The DoRA recovery results are shown in Table 6.4.

Model	Sparsity	PPL	Correct	Accuracy	Throughput
wanda_s20 + DoRA	20%	21.5220	352 / 650	0.5415	47.37 tok/s
wanda_s30 + DoRA	30%	23.8493	343 / 650	0.5277	45.30 tok/s
wanda_s50 + DoRA	50%	34.4044	293 / 650	0.4508	47.64 tok/s

Table 6.4: Recovery results using DoRA

DoRA also improves all raw pruned checkpoints. Its recovered models show better GSM8K accuracy than the corresponding raw Wanda-pruned models at every sparsity level. The 20% DoRA model gives the best final recovered accuracy, while the 50% DoRA model shows the largest improvement over its raw pruned version. Perplexity also improves compared with the raw pruned checkpoints, showing that DoRA recovery helps both reasoning and language modeling quality.

6.6 Comparison of QLoRA and DoRA

Table 6.5 compares raw Wanda-pruned models with their QLoRA and DoRA recovered versions. Since both recovery methods use the same pruned checkpoints and evaluation settings, the comparison mainly reflects the difference in recovery behaviour.

Sparsity	Raw Acc.	QLoRA Acc.	DoRA Acc.	Raw PPL	QLoRA PPL	DoRA PPL
20%	0.5046	0.5292	0.5415	23.3884	21.7185	21.5220
30%	0.4831	0.5062	0.5277	25.4533	23.2286	23.8493
50%	0.1800	0.4446	0.4508	45.1633	33.8288	34.4044

Table 6.5: Comparison of raw pruned, QLoRA-recovered, and DoRA-recovered models

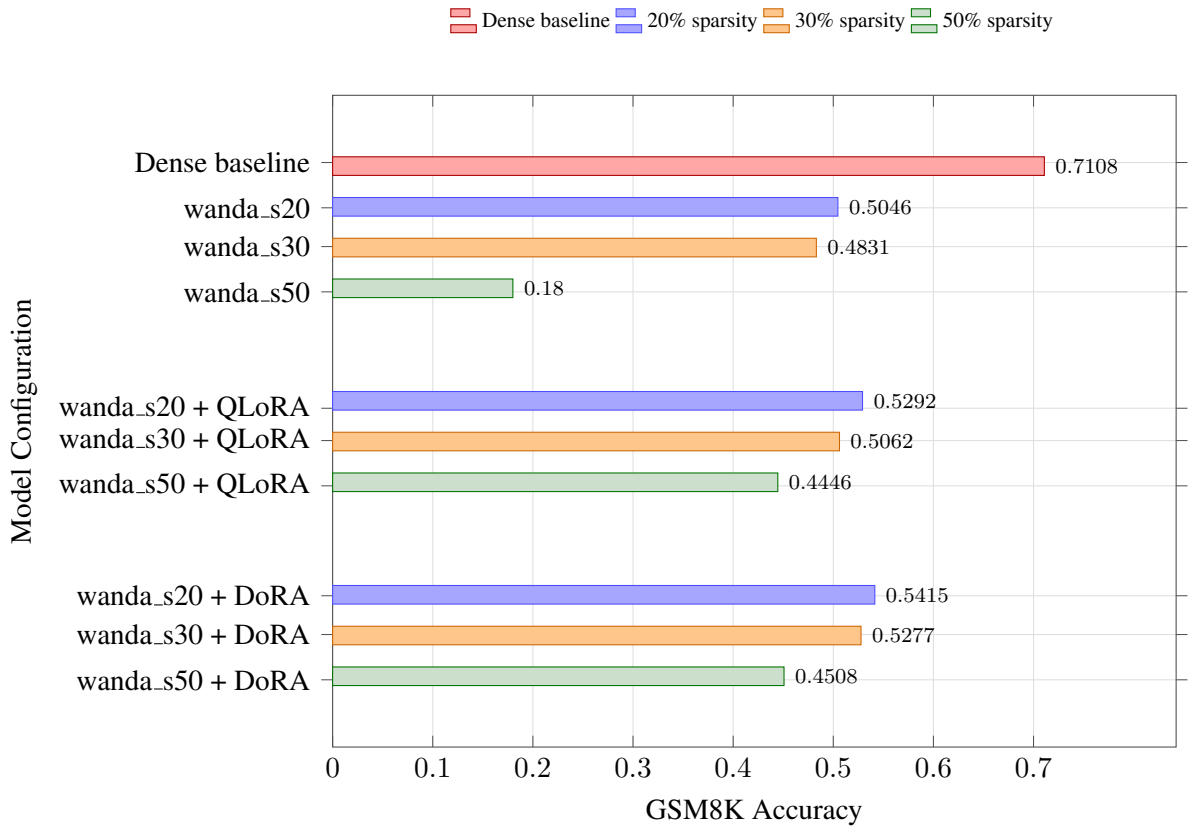

Figure 6.1: GSM8K accuracy comparison across dense baseline, raw Wanda-pruned models, and PEFT-recovered models.

Figure 6.1 shows that both QLoRA and DoRA improve GSM8K accuracy after Wanda pruning. DoRA gives the highest GSM8K accuracy at all three sparsity levels. The difference between QLoRA and DoRA is not large in every case, but the pattern is consistent across 20%, 30%, and 50% sparsity.

Figure 6.2 shows that perplexity does not follow the exact same pattern as GSM8K accuracy. DoRA gives the lowest perplexity at 20% sparsity, while QLoRA gives slightly lower perplexity at 30% and 50% sparsity. This means that language modeling recovery and reasoning recovery are not identical. Therefore, DoRA is preferable when GSM8K reasoning accuracy is the main priority, while QLoRA remains competitive for perplexity recovery at higher sparsity.

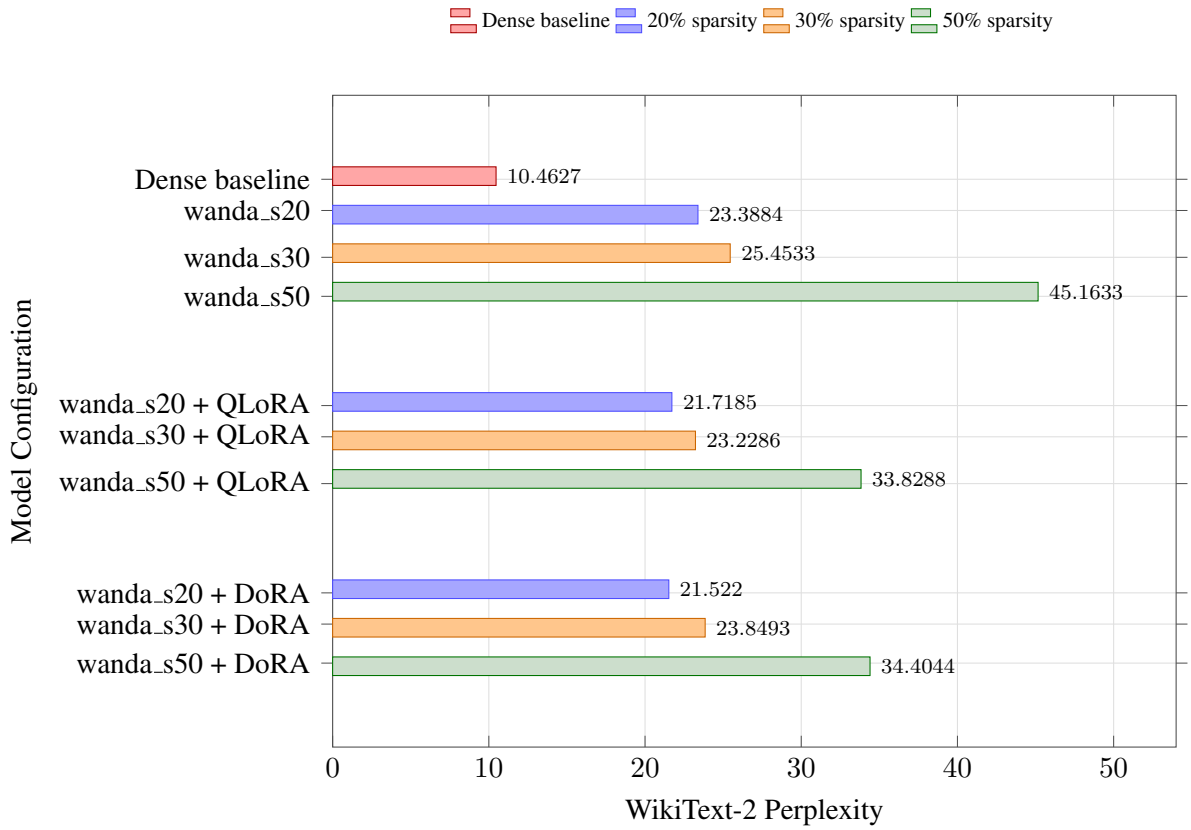

Figure 6.2: WikiText-2 perplexity comparison across dense baseline, raw Wanda-pruned models, and PEFT-recovered models. Lower perplexity indicates better language modeling quality.

6.7 Recovery Gain Analysis

Accuracy alone does not fully explain recovery behaviour, because each sparsity level starts from a different raw pruned accuracy. Therefore, GSM8K recovery gain is used to measure how much accuracy is restored after applying the recovery method.

For GSM8K, the recovery gain is calculated as

$$\text{Recovery Gain} = \text{Recovered Accuracy} - \text{Raw Pruned Accuracy}.$$

Since DoRA gives the best GSM8K accuracy at all sparsity levels, Table 6.6 reports the best recovered model and the corresponding gain for each sparsity level.

Sparsity	Raw Acc.	Best Method	Best Recovered Acc.	Absolute Gain
20%	0.5046	DoRA	0.5415	+0.0369
30%	0.4831	DoRA	0.5277	+0.0446
50%	0.1800	DoRA	0.4508	+0.2708

Table 6.6: GSM8K recovery gain over raw Wanda-pruned models

The largest absolute gain occurs at 50% sparsity, where DoRA improves GSM8K accuracy from 0.1800 to 0.4508. This shows that recovery is most visible when pruning damage is severe. However, this does not make the 50% recovered model the best final model, because its final accuracy is still lower than the 20% and 30% recovered models.

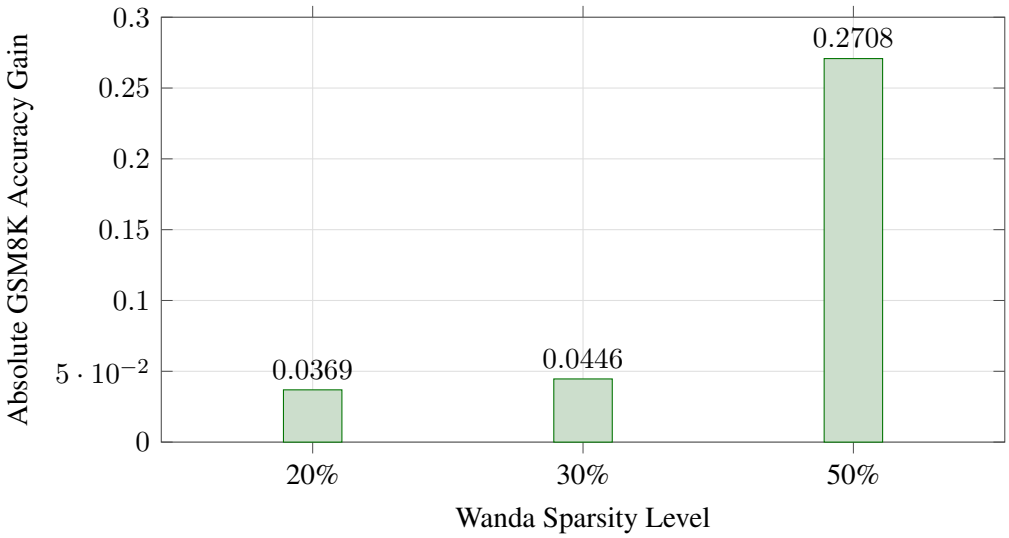

Figure 6.3: Best GSM8K recovery gain over raw Wanda-pruned models at each sparsity level. The gain is computed as recovered accuracy minus raw pruned accuracy.

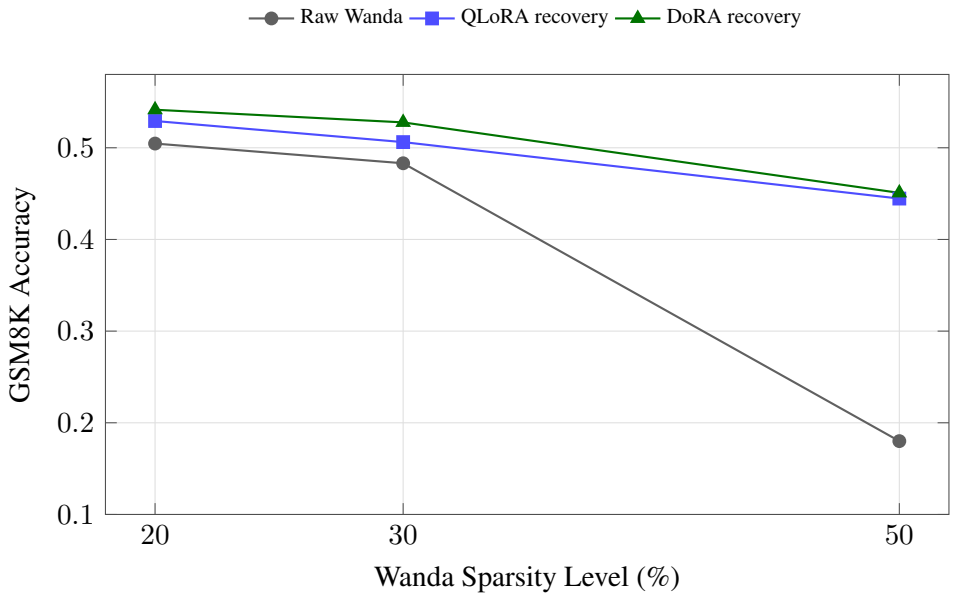

Figure 6.4: GSM8K accuracy trend across sparsity levels for raw Wanda-pruned, QLoRA-recovered, and DoRA-recovered models.

Figures 6.3 and 6.4 show two related but different conclusions. The 20% DoRA model gives the best final recovered accuracy, while the 50% DoRA model gives the largest recovery gain. Thus, final quality and recovery gain must be interpreted separately.

6.8 Efficiency Analysis

Efficiency is evaluated mainly through throughput during GSM8K generation. Since Wanda produces unstructured sparsity, speedup cannot be assumed unless the inference backend can exploit sparse computation. Table 6.7 reports the accuracy and throughput values for the dense, pruned, and recovered models.

Model	**Sparsity**	**GSM8K Accuracy**	**Throughput**
Dense baseline	0%	0.7108	46.33 tok/s
`wanda_s20`	20%	0.5046	48.06 tok/s
`wanda_s20 + QLoRA`	20%	0.5292	47.49 tok/s
`wanda_s20 + DoRA`	20%	0.5415	47.37 tok/s
`wanda_s30`	30%	0.4831	47.56 tok/s
`wanda_s30 + QLoRA`	30%	0.5062	46.80 tok/s
`wanda_s30 + DoRA`	30%	0.5277	45.30 tok/s
`wanda_s50`	50%	0.1800	48.02 tok/s
`wanda_s50 + QLoRA`	50%	0.4446	47.77 tok/s
`wanda_s50 + DoRA`	50%	0.4508	47.64 tok/s

Table 6.7: GSM8K accuracy and throughput comparison across dense, pruned, and recovered models

The throughput values remain close across all configurations. The dense baseline reaches 46.33 tok/s, while raw and recovered models stay in a similar range. This shows that Wanda pruning did not produce a meaningful wall-clock speedup in the tested setup.

This behaviour is expected because unstructured sparsity alone does not guarantee faster inference. If the backend still uses dense matrix multiplication, zero weights may remain inside dense tensors during computation. Therefore, even 50% sparsity should not be interpreted as a $2\times$ speedup without sparse-kernel or hardware support.

Figure 6.5 supports the same observation. The recovered models retain throughput close to the dense baseline, but they do not show proportional acceleration from sparsity. Therefore, the main value of recovery in this experiment is performance restoration rather than direct runtime speedup.

Among the recovered models, `wanda_s20 + DoRA` gives the best GSM8K accuracy while maintaining throughput comparable to the dense model. The `wanda_s50 + DoRA` model gives the largest recovery gain, but its final accuracy is still lower than the 20% and 30% recovered models.

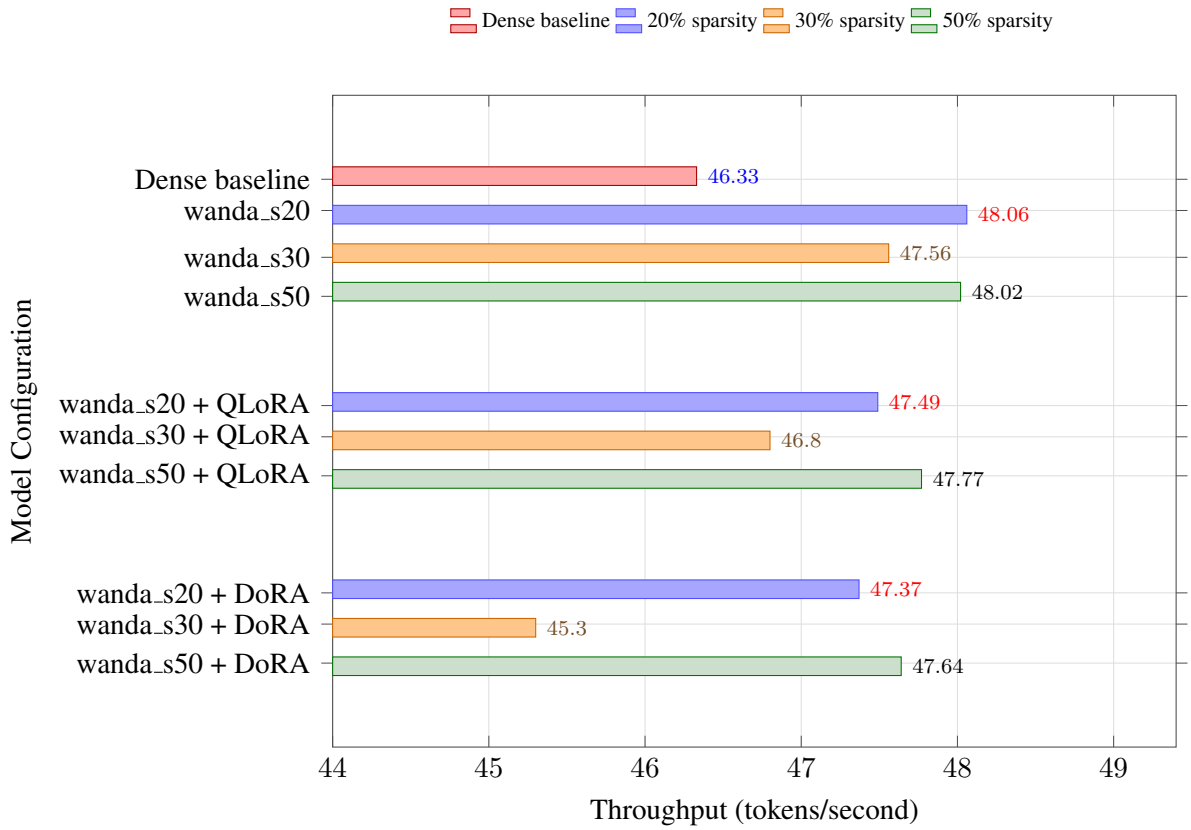

Figure 6.5: Throughput comparison across dense baseline, raw Wanda-pruned models, and PEFT-recovered models during GSM8K evaluation.

6.9 End-Device Feasibility Study

The end-device feasibility study connects the experimental results to the deployment motivation of this dissertation. Here, end-device does not mean that the model has already been deployed on a mobile phone or embedded board. It refers to the broader problem of running LLMs under constrained memory, storage, and generation-speed conditions.

Since Wanda pruning produces unstructured sparsity, these results should not be treated as evidence of hardware acceleration. A sparse checkpoint becomes useful for speed only when the software framework or hardware can exploit sparsity. Therefore, feasibility is interpreted through the balance between accuracy, perplexity, throughput, and sparsity.

Based on the results, `wanda_s20 + DoRA` is the safest recovered configuration when reasoning accuracy is the main priority. The `wanda_s30 + DoRA` configuration provides a stronger pruning level while still retaining useful GSM8K recovery. The 50% recovered models are useful for studying aggressive pruning recovery, but they are not the best final choice for reasoning accuracy.

Overall, feasibility cannot be judged from sparsity alone. A useful compressed model must balance pruning level, recovered accuracy, perplexity, throughput, and the availability of

Configuration	Suitability Interpretation
`wanda_s20 + DoRA`	Best recovered GSM8K accuracy among the recovered models. This is the safest recovered configuration when reasoning accuracy is the main priority.
`wanda_s30 + DoRA`	Stronger sparsity than the 20% model while still retaining useful recovered accuracy. This is a reasonable middle-ground configuration.
`wanda_s50 + DoRA`	Largest recovery gain over the raw pruned checkpoint, but final accuracy remains clearly below the 20% and 30% recovered models. Useful for studying aggressive pruning recovery.
`wanda_s30/s50 + QLoRA`	Competitive perplexity compared with DoRA at higher sparsity. Useful when language modeling quality is prioritized over GSM8K accuracy.

Table 6.8: End-device-oriented interpretation of recovered model configurations

sparse inference support.

6.10 Ablation Studies

The ablation analysis in this dissertation focuses on two controlled factors: Wanda sparsity level and recovery method. Other settings such as adapter rank, LoRA alpha, dropout, learning rate, number of epochs, recovery data, and evaluation protocol were kept fixed. This makes the comparison mainly reflect the effect of pruning severity and the choice between QLoRA and DoRA.

The first factor is sparsity. As sparsity increases from 20% to 50%, the raw pruned model becomes weaker, especially on GSM8K. The second factor is the recovery method. Table 6.9 summarizes the best recovery method for GSM8K accuracy and WikiText-2 perplexity at each sparsity level.

Sparsity	Best GSM8K Method	Best GSM8K Acc.	Best PPL Method	Best PPL
20%	DoRA	0.5415	DoRA	21.5220
30%	DoRA	0.5277	QLoRA	23.2286
50%	DoRA	0.4508	QLoRA	33.8288

Table 6.9: Ablation over sparsity level and recovery method

The ablation results show that DoRA is consistently better for GSM8K accuracy across all three sparsity levels. However, perplexity does not follow the same pattern. DoRA gives the best perplexity at 20% sparsity, while QLoRA gives slightly better perplexity at 30% and 50% sparsity. This confirms that recovery should be judged using more than one metric. Further ablations over adapter rank, learning rate, dropout, training epochs, and recovery data are left for future work.

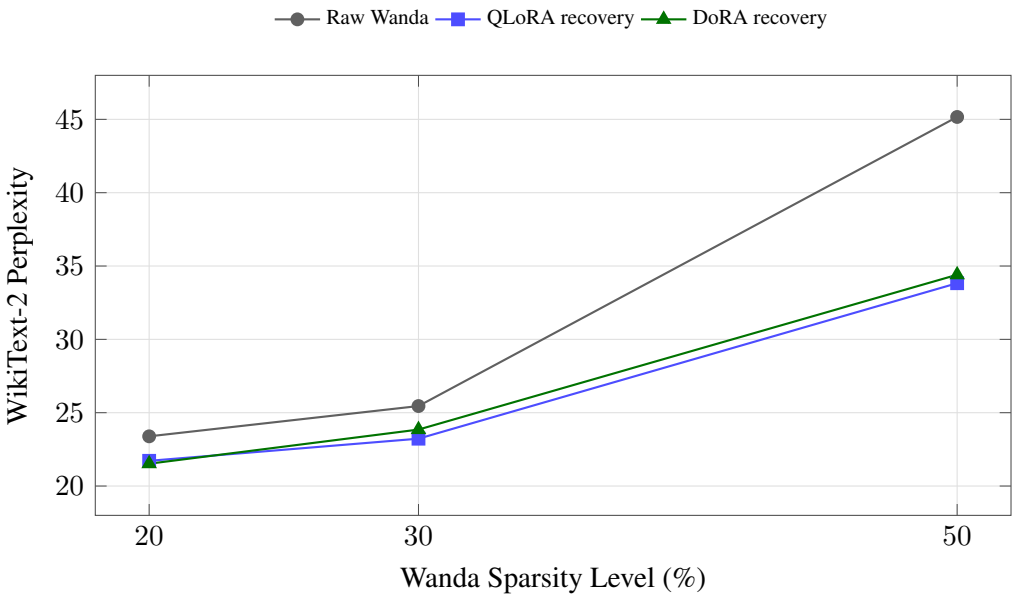

Figure 6.6: WikiText-2 perplexity trend across sparsity levels for raw Wanda-pruned, QLoRA-recovered, and DoRA-recovered models. Lower perplexity indicates better language modeling.

6.11 Qualitative Analysis of GSM8K

Case Type	Behaviour Observed	Interpretation
Correct after recovery	The raw pruned model gives an incorrect final answer, while the recovered model gives the correct final number.	Adapter recovery helps restore task-specific reasoning or answer formatting.
Still incorrect after recovery	Both raw pruned and recovered models give incorrect final answers.	The pruning damage may be too strong, or the adapter training is not enough for that reasoning pattern.
Partial reasoning error	The model follows part of the arithmetic correctly but makes an error in one intermediate step.	GSM8K accuracy is sensitive to small reasoning mistakes, especially after pruning.
Format or extraction issue	The model produces a response where the final answer is unclear or not written in a clean numerical form.	Evaluation depends not only on reasoning but also on stable final-answer formatting.
Preserved behaviour	Both the raw pruned and recovered models answer correctly.	Some questions remain unaffected by pruning, especially when the required reasoning is simpler.

Table 6.10: Common qualitative behaviours observed in GSM8K predictions

GSM8K accuracy gives a direct numerical comparison, but it does not show how the models succeed or fail. Therefore, the generated prediction files were inspected using the

model response, extracted answer, gold answer, and correctness label.

The common behaviours observed during this inspection are summarized in Table 6.10. The qualitative results show that pruning does not immediately remove fluency. Pruned models may still generate grammatically correct responses, but their reasoning can become unstable through arithmetic errors, wrong quantity interpretation, or unclear final-answer formatting.

Recovered models are more stable than raw pruned models, especially at 20% and 30% sparsity. At 50% sparsity, recovery still helps, but more reasoning errors remain. This supports the numerical results and shows why WikiText-2 perplexity and GSM8K accuracy must be interpreted separately.

6.12 Discussion

The experimental results show that Wanda pruning causes measurable degradation in both language modeling and mathematical reasoning. The dense Llama-3.2-3B-Instruct model remains the strongest configuration, with WikiText-2 perplexity of 10.4627 and GSM8K accuracy of 0.7108. After pruning, perplexity increases and GSM8K accuracy decreases at all sparsity levels.

The degradation depends strongly on sparsity. At 20% and 30% sparsity, the raw pruned models still retain some reasoning ability, but at 50% sparsity the drop is much larger, with GSM8K accuracy falling to 0.1800 and perplexity increasing to 45.1633. This shows that aggressive pruning removes useful capacity for multi-step reasoning.

Both QLoRA and DoRA recover part of the lost performance. DoRA gives better GSM8K accuracy at all sparsity levels, while QLoRA remains competitive for WikiText-2 perplexity, especially at 30% and 50% sparsity. Therefore, the preferred recovery method depends on the target metric. Recovery gain gives another useful view. The largest absolute GSM8K gain occurs at 50% sparsity, where DoRA improves accuracy from 0.1800 to 0.4508. However, the 50% recovered model is not the best final model, since the 20% and 30% recovered models still achieve higher final accuracy.

The throughput results show that Wanda pruning does not automatically create runtime speedup in the tested setup. Since the pruning is unstructured, dense inference kernels cannot fully exploit the zero weights. Therefore, this dissertation does not claim proportional acceleration from sparsity; the main value of recovery is performance restoration after pruning.

Overall, pruning and recovery should be studied together. Raw pruning causes performance loss, while QLoRA and DoRA recover part of that loss without full fine-tuning. Among the tested configurations, `wanda_s20 + DoRA` is the safest choice for GSM8K accuracy, `wanda_s30 + DoRA` gives a useful middle ground, and the 50% recovered models are mainly useful for studying aggressive pruning recovery.

Chapter 7

Conclusion and Future Work

7.1 Summary of Work

This dissertation studied performance recovery in pruned large language models under constrained hardware conditions. The work used Llama-3.2-3B-Instruct as the dense reference model and followed a fixed experimental pipeline: dense baseline evaluation, Wanda pruning, raw pruned evaluation, PEFT-based recovery, and final recovered-model evaluation. This order was important because recovery results are meaningful only when both the dense baseline and the raw pruned checkpoints are evaluated under the same protocol.

Wanda activation-aware pruning was applied at 20%, 30%, and 50% sparsity. The raw pruned checkpoints were evaluated before recovery to measure the direct effect of pruning. The results showed that pruning increased WikiText-2 perplexity and reduced GSM8K accuracy, with the strongest degradation at 50% sparsity. At this level, GSM8K accuracy dropped to 0.1800 and perplexity increased to 45.1633.

After pruning, QLoRA and DoRA recovery were applied to each pruned checkpoint. The recovered models were evaluated using WikiText-2 perplexity and GSM8K accuracy so that both language modeling quality and mathematical reasoning could be compared. Both recovery methods improved the raw pruned checkpoints. DoRA gave the best GSM8K recovery at all sparsity levels, while QLoRA remained competitive in WikiText-2 perplexity, especially at 30% and 50% sparsity.

The results also showed that Wanda pruning did not automatically provide runtime speedup in the tested setup. Throughput values for dense, pruned, and recovered models remained close to each other. This indicates that unstructured sparsity alone is not enough for practical acceleration unless the inference backend or hardware can exploit sparse computation. Therefore, the main contribution of this work is the study of pruning damage, adapter-based recovery, and the trade-off between sparsity, perplexity, reasoning accuracy, and practical feasibility.

7.2 Key Findings

The dense Llama-3.2-3B-Instruct model remained the strongest overall model in the experiments. It achieved a WikiText-2 perplexity of 10.4627 and GSM8K accuracy of 0.7108. None of the recovered models fully reached this baseline, although they improved clearly over the corresponding raw pruned checkpoints.

Wanda pruning caused increasing degradation as sparsity increased. At 20% sparsity, the raw pruned model achieved GSM8K accuracy of 0.5046 with perplexity 23.3884. At 30% sparsity, accuracy became 0.4831 with perplexity 25.4533. At 50% sparsity, the damage was much larger, with GSM8K accuracy falling to 0.1800 and perplexity rising to 45.1633. This shows that aggressive pruning affects reasoning-oriented evaluation more severely.

Both QLoRA and DoRA were effective as recovery methods. For every sparsity level, the recovered models performed better than the corresponding raw Wanda-pruned models. DoRA gave better GSM8K accuracy than QLoRA at 20%, 30%, and 50% sparsity. The best recovered GSM8K accuracy was obtained by `wanda_s20 + DoRA`, with accuracy 0.5415.

QLoRA was still competitive for language modeling recovery. Although DoRA gave the best GSM8K accuracy, QLoRA produced slightly better WikiText-2 perplexity than DoRA at 30% and 50% sparsity. This confirms that the preferred recovery method depends on the target metric. If reasoning accuracy is the priority, DoRA is stronger in this setup. If perplexity is the focus at higher sparsity, QLoRA remains useful.

The largest absolute recovery gain occurred at 50% sparsity. DoRA improved GSM8K accuracy from 0.1800 to 0.4508, which is the largest gain among the tested configurations. However, this recovered model was still weaker than the 20% and 30% recovered models in final accuracy. Hence, recovery gain and final model quality must be interpreted separately.

Finally, the throughput results showed that sparsity did not produce proportional runtime improvement. This is because Wanda creates unstructured sparsity, and dense inference kernels do not automatically benefit from zero weights. Thus, compression should not be judged only from sparsity percentage; practical speedup depends on sparse-kernel or hardware support.

7.3 Limitations

This work has several limitations. First, all experiments evaluated a single base model, Llama-3.2-3B-Instruct. While this ensured a controlled study, architectural variations in other models like Mistral, Gemma, Phi, or larger Llama variants might yield different responses to Wanda pruning and PEFT recovery.

Second, evaluation was restricted to WikiText-2 and GSM8K. Though effective for testing language modeling and mathematical reasoning, these benchmarks omit other critical ca-

pabilities of instruction-tuned models, such as commonsense reasoning, factual QA, code generation, long-context understanding, and safety compliance.

Third, the applied pruning methodology yields unstructured sparsity. While optimal for analyzing degradation mechanics, it failed to provide clear hardware-level acceleration during inference; structured or hardware-aware alternatives remain necessary for runtime speedups.

Fourth, most recovery hyperparameters—including rank, alpha, dropout, learning rate, epochs, sequence length, and target modules—were kept fixed to guarantee a fair comparison between QLoRA and DoRA. Alternative configurations might further optimize recovery performance.

Fifth, this study lacks physical validation on edge hardware. Here, ”end-device” denotes a resource-constrained experimental environment rather than a production deployment, which would require direct profiling on CPU-only systems, consumer GPUs, mobile NPUs, or sparse accelerators.

Finally, using the GSM8K training set for adaptation introduces task bias. While effective for mathematical recovery, this narrow dataset risks overfitting the adapters to specific reasoning structures, whereas a more generalized corpus might better preserve overall language capabilities.

7.4 Future Work

Future work can extend this study in several directions. First, the same pruning–recovery pipeline can be tested on other LLM families and model sizes. Applying the method to models such as Mistral, Gemma, Phi, or larger Llama variants would help determine whether the trends observed here are specific to Llama-3.2-3B-Instruct or more general.

Second, future experiments should include a wider set of benchmarks. WikiText-2 and GSM8K gave a useful view of language modeling and mathematical reasoning, but additional datasets such as MMLU, ARC, HellaSwag, HumanEval, and instruction-following benchmarks would provide a broader evaluation of factual knowledge, commonsense reasoning, coding ability, and general instruction following.

Third, structured pruning can be explored. Since the present work used Wanda-based unstructured pruning, the throughput improvement was limited. Structured pruning methods such as attention-head pruning, feed-forward channel pruning, or block-level pruning may provide better runtime gains when supported by the inference backend.

Another direction is combining pruning with quantization. Wanda pruning can be followed by quantized inference, or pruning and QLoRA-style quantized loading can be studied together in a more integrated way. Such combinations may reduce memory usage more effectively than pruning alone.

Future work can also study recovery hyperparameters more deeply. Adapter rank, learning rate, dropout, number of epochs, target modules, and recovery data size can all affect performance. A systematic ablation over these choices may improve the recovered models beyond the fixed setting used in this dissertation.

A more complete deployment study is also needed. The recovered models should be tested on CPU-only systems, consumer GPUs, laptop GPUs, edge accelerators, or inference frameworks that support sparse computation. Such experiments would give a clearer picture of latency, memory usage, energy cost, and practical usability.

Finally, future recovery experiments can use more general training data. In this dissertation, recovery was based on GSM8K training data. A mixture of instruction data, general language modeling data, and reasoning data may help recover a broader set of capabilities instead of mainly improving mathematical reasoning.

7.5 Final Remarks

This dissertation treated pruning and recovery as one connected process. The results show that pruning alone is not enough because raw pruned LLMs lose both language modeling quality and reasoning ability. At the same time, pruned checkpoints are not useless. With QLoRA and DoRA recovery, a meaningful part of the lost performance can be restored without full fine-tuning.

The dense Llama-3.2-3B-Instruct model remained the best overall model, so recovery should not be interpreted as a complete replacement for the original dense model. Instead, recovery is useful because it improves the damaged pruned checkpoints while keeping the adaptation process lightweight.

Among the recovered models, DoRA gave better GSM8K recovery than QLoRA at all tested sparsity levels. The strongest recovered reasoning result came from the 20% Wanda-pruned model recovered with DoRA. The 30% DoRA model provided a useful middle-ground, while the 50% recovered models showed that severe pruning can be partially repaired but remains risky for reasoning-intensive tasks.

The study also shows that efficiency must be interpreted carefully. Wanda pruning introduces unstructured sparsity, but this did not lead to clear throughput improvement in the tested setup. Practical acceleration would require sparse inference support, structured pruning, quantization, or hardware-aware optimization.

The main conclusion is that efficient LLM deployment requires balance. A compressed model cannot be judged only by sparsity, and a recovered model cannot be judged only by accuracy. The useful point lies in the interaction between pruning, recovery, language modeling quality, reasoning accuracy, and practical inference behavior.

Bibliography

[1] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[2] Zijian Feng, Hanzhang Zhou, Zixiao Zhu, Tianjiao Li, Jia Jim Deryl Chua, Lee Onn Mak, Gee Wah Ng, and Kezhi Mao. Restoring pruned large language models via lost component compensation. *arXiv preprint arXiv:2510.21834*, 2025.

[3] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

[4] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[5] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023.

[6] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, 2024.

[7] Moritz Wagner, Christophe Roux, Max Zimmer, and Sebastian Pokutta. A free lunch in llm compression: Revisiting retraining after pruning. *arXiv preprint arXiv:2510.14444*, 2026.

Appendix

A.1 Hyperparameters and Configuration

Table A.1: Final pruning, recovery, and evaluation configuration

Hyperparameter / Setting	**Value**
Base model	Llama-3.2-3B-Instruct
Pruning method	Wanda activation-aware pruning
Sparsity levels	20%, 30%, 50%
Wanda calibration sequences	64
Wanda calibration sequence length	512
Recovery methods	QLoRA and DoRA
Recovery dataset	GSM8K training split
Recovery training samples	7473
Recovery epochs	1
Training steps	934
Adapter rank	16
LoRA alpha	32
Dropout	0.05
Learning rate	2×10^{-4}
Maximum sequence length during recovery	512
Per-device batch size	2
Gradient accumulation steps	4
Effective batch size	8
QLoRA optimizer	paged adamw 8bit
DoRA optimizer	adamw torch
QLoRA quantization	4-bit NF4 with double quantization
Target modules	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj
GSM8K evaluation sample size	650
GSM8K random seed	42
GSM8K maximum new tokens	512
WikiText-2 evaluated tokens	131072
WikiText-2 sequence length	1024
WikiText-2 stride	512
Final evaluation platform	RunPod NVIDIA A40 GPU
Adapter handling during evaluation	Merged into pruned base model before inference

A.2 Model and Adapter Artifacts

For reproducibility, the models, their checkpoints, recovery adapters, and backup repositories used in this dissertation are available in the following Hugging Face repositories. They include Wanda pruned checkpoints, QLoRA/LoRA recovery adapters, and backup repositories. Links to their Hugging Face repositories are provided below.

Artifact Type	**Configuration**	**Hugging Face Repository Link**
Wanda-pruned model	20% sparsity	paul258/llama32-3b-wanda-s20
Wanda-pruned model	30% sparsity	paul258/llama32-3b-wanda-s30
Wanda-pruned model	50% sparsity	paul258/llama32-3b-wanda-s50
QLoRA adapter	20% sparsity pilot recovery	paul258/wanda-s20-qlora-gsm8k-pilot
QLoRA adapter	20% sparsity full recovery	paul258/wanda-s20-qlora-gsm8k-full
QLoRA adapter	20% sparsity gentle recovery	paul258/wanda-s20-qlora-gsm8k-gentle-v1
LoRA adapter	20% sparsity clean recovery without bitsandbytes	paul258/wanda-s20-lora-clean-gsm8k-v2-no-bnb
Recovery adapters	Fast recovery adapter collection	paul258/llama32-3b-wanda-recovery-adapters-fast
Backup repository	Pruning and recovery backup	paul258/llama-pruning-recovery-backup

Table A.2: Hugging Face Hub repositories containing the Wanda-pruned checkpoints, recovery adapters, and backup artifacts used in this dissertation.