Hochschule Hof
University of Applied Sciences

Deutsches Zentrum für Luft- und Raumfahrt
DLR German Aerospace Center

Structured Pruning of Large Language Models

Empirical Evaluation of Depth and Width Pruning with Dependency-Aware Design and Failure Analysis

Master Thesis

At Hof University of Applied Sciences

Masters of AI & Robotics

Submitted to

Prof. Dr. Michael Spangenberg
Professor of Data Science
Head of Program "AI and Robotics "
Alfons-Goppel-Platz 1,95028 Hof
Germany

Submitted by

Aniket Ramchandra Munde
Sophienstraße 08
95028 Hof
Germany

Hof, 22 June 2026

TABLE OF CONTENTS PAGE NO.

Abstract 1

Chapter 1: Introduction
1.1 Research Context and Motivation 2
1.2 Problem Statement 3
1.3 Research Objectives and Contributions 3
1.4 Scope and Assumptions 4
1.5 Thesis Organization 5

Chapter 2: Background and Literature Review
2.1 Transformer and LLM Basics Relevant to Pruning 6
2.2 Model Compression Overview: Pruning, Quantization, and Distillation 7
2.3 Structured Pruning: Depth vs Width/Channel and Dependency Correctness 8
2.4 Unstructured Pruning Literature 9
2.5 Layer Redundancy and Depth Pruning Literature 11
2.6 Parameter-Efficient Fine-Tuning (LoRA, QLoRA) and Post-Pruning Recovery 12
2.7 Evaluation and Benchmarking Frameworks 13
2.8 Vision–Language and Multimodal Pruning Context 14
2.9 Remote Sensing Foundation Models and EuroSAT Context 14
2.9.1 Foundations of Multi-Spectral Earth Observation Models and Linear Probing 15

Chapter 3: Model Architectures
3.1 Qwen3 Family (0.6B / 4B-Instruct / 8B) Architecture Overview 16
3.2 LLaVA-OneVision-Qwen2-0.5B Architecture 17
3.3 Phi-4-Mini-Instruct Architecture 18
3.4 TerraMind-Base and TerraMind-Large: Dual-Scale Remote Sensing Architectures 19

Chapter 4: Pruning Methodology
4.1 Depth Pruning Strategies 21
4.2 Width and Channel Pruning Mechanics 22
4.3 Layer Importance Estimation for Depth Pruning 23
4.4 Dependency-Aware Pruning and Shape Consistency Handling 26
4.5 Parameter-Efficient Fine-Tuning (PEFT) as Post-Pruning Recovery 27
4.6 Recovery Fine-Tuning Datasets 28
4.7 Knowledge Distillation for Post-Pruning Recovery. 30
4.8 Pruning Objectives, Constraints, and Deprioritized Directions 31

TABLE OF CONTENTS PAGE NO.

Chapter 5: Evaluation Framework
5.1 Evaluation Metrics: NLL, Perplexity, and Efficiency Measures 32
5.2 Forced-Continuation Evaluation Protocol 33
5.3 Benchmark Suite Framing 33
5.4 Evaluation Tooling and Harnesses.....34
5.5 Experimental Progression and Decision Rationale 35
5.6 Practical Challenges in Structured Pruning Implementation.....37

Chapter 6: Experimental Results
6.1 Evaluation Scope and Prompt Setup 38
6.2 Metric Rationale: NLL vs Perplexity 38
6.3 LoRA-Based Recovery Configuration 39
6.4 LLaVA-OneVision-0.5B Results.....40
6.5 Qwen3-0.6B Results 41
6.6 Qwen3-8B Results with Recovery 42
6.7 Qwen3-4B-Instruct Depth pruning and Recovery 43
6.7.1 Mathematical Reasoning Fragility under Structural Compression.....44
6.8 Phi-4-Mini-Instruct Results 45
6.9 Evaluation Methodology and Inference Behavior
6.9.1 Prompt based vs Teacher Forced Evaluation.....46
6.9.2 Width Pruning and GPU kernel Efficiency 46
6.9.3 Depth Pruning and Sequential Decoding latency ...47
6.9.4 Model Scale and Pruning Saturation.....47
6.9.5 Pruning Algorithm Execution Time.....48
6.10 Throughput Analysis (Tokens per Second) 49
6.11 Latency Analysis (Time-to-First Token) 50
6.12 GPU Memory Consumption 51
6.13 TerraMind Accuracy on EuroSAT 53
6.14 TerraMind Memory Efficiency 54
6.15 TerraMind Inference Latency 55
6.16 Terramind base and Large Result Analysis 56

Chapter 7: Failure Analysis and Debugging
7.1 Environment and Dependency Failures 58
7.2 Model Loading, Checkpoint, and Shape Mismatch Failures.....58
7.3 Pruning-Specific Failure Modes 59
7.4 Evaluation and Metric-Related Failures 60

TABLE OF CONTENTS PAGE NO.

7.5 Deployment and Optimization Failures 60
7.6 Summary of Failure Patterns 61

Chapter 8: Backend and Deployment Constraints
8.1 Infrastructure, Runtime, and Dependency Constraints.. 62
8.2 Execution Environment and Leaderboard Evaluation (SLURM & KRATOS) 62
8.3 Hugging Face Model Access and Artifact Constraints.. 63
8.4 GGUF Export and Multimodal Conversion Constraints 63
8.5 vLLM Runtime and Multimodal Evaluation Constraints 64
8.6 ONNX Export and Interoperability Limitations 65
8.7 TensorRT and Inference Optimization Constraints 66
8.8 Explicit Goal Change and de prioritization of Deployment Optimization 66
8.9 Implications for Research Scope 67

Chapter 9: Discussion
9.1 Interpretation of Pruning Outcomes Across Models 68
9.2 Depth Pruning versus Width and Channel Pruning..... 69
9.3 Role of Dependency Awareness in Pruning Stability.... 69
9.4 Post-Pruning Recovery via PEFT 70
9.5 Interpretation of Benchmark Behavior 70
9.6 Deployment Constraints as a Research Signal..... 70
9.7 Systems-level Interpretation of Pruning 71
9.8 Summary of Key Trade-Offs 72

Chapter 10: Limitations
10.1 Scope Limitations of Pruning Strategies..... 73
10.2 Model Coverage and Generalizability 73
10.3 Evaluation and Metric Constraints..... 73
10.4 Dependency and Tooling Limitations..... 74
10.5 Deployment and Optimization Limitations 74
10.6 Temporal and Project-Scope Constraints..... 74

Chapter 11: Conclusion and Future Work
11.1 Summary of Contributions 75
11.2 Conclusions 76
11.3 Future Work 76

Chapter 12: References 77

Chapter 13: Appendix 85

LIST OF FIGURES PAGE NO.

Figure 2.1 Transformer Model Architecture 6
Figure 2.2 Conceptual Overview of Model Compression Techniques 7
Figure 2.3 Structured Pruning in Transformer Models 8
Figure 3.1 LLaVA-OneVision Architecture 17
Figure 3.2 TerraMind Dual-Scale Architecture 19
Figure 4.1 Depth Pruning Illustration 21
Figure 4.2 Width/Neuron Pruning Illustration 22
Figure 4.3 Iterative Pruning and Distillation Overview 23
Figure 4.4 Layer-Drop PPL: LLaVA-OneVision-0.5B 24
Figure 4.5 Layer-Drop PPL: Qwen3-4B-Instruct 24
Figure 4.6 Layer-Drop PPL: Phi-4-Mini-Instruct 25
Figure 4.7 Illustration of coupled structure 26
Figure 4.8 Knowledge Distillation Teacher–Student Framework 30
Figure 5.1 Evaluation Workflow using lm-evaluation-harness 35
Figure 6.1 LLaVA-OneVision-0.5B Benchmark Results 40
Figure 6.2 Qwen3-0.6B Benchmark Results 41
Figure 6.3 Qwen3-8B Benchmark Results with Recovery 42
Figure 6.4 Qwen3-4B-Instruct Benchmark Results 43
Figure 6.5 Phi-4-Mini-Instruct Benchmark Results 45
Figure 6.6 Pruning Algorithm Execution Time 48
Figure 6.7 Throughput Comparison (Tokens/s) 49
Figure 6.8 Time-to-First-Token (TTFT) 50
Figure 6.9 Peak GPU Memory Consumption 51
Figure 6.10 Best Achieved Reasoning Performance Summary 52
Figure 6.11 TerraMind EuroSAT Accuracy 53
Figure 6.12 TerraMind GPU Memory Usage 54
Figure 6.13 TerraMind Inference Latency 55

LIST OF TABLES PAGE NO.

Table A.1 Summary of All structured pruning experiments 87
Table B.1 Benchmark Results for All (BBH,MATH,MuSR) 88
Table B.2 Benchmark Result for All (MMLU PRO) 89
Table C.1 Inference efficiency metrics before and after pruning 90
Table C.2 Percentage Changes in Inference before and after pruning .....91 ....
Table D.1 TerraMind Classification Accuracy..... 92
Table D.2 TerraMind Efficiency Metrics 92
Table E.1 LoRa Recovery Datasets 93
Table F.1 LoRa Hyperparamters..... 93
Table F.2 Generation and evaluation settings..... 93

List of Abbreviations

Abbreviation	Definition
LLM	Large Language Model
VLM	Vision–Language Model
MHSA	Multi-Head Self-Attention
FFN	Feed-Forward Network
MLP	Multi-Layer Perceptron
NLL	Negative Log-Likelihood
PPL	Perplexity
TL	Task Likelihood (average NLL over target tokens)
PEFT	Parameter-Efficient Fine-Tuning
LoRA	Low-Rank Adaptation
QLoRA	Quantized Low-Rank Adaptation
KD	Knowledge Distillation
BBH	Big-Bench Hard
MMLU	Massive Multitask Language Understanding
GPQA	Graduate-Level Google-Proof Question Answering
MuSR	Multi-Step Symbolic Reasoning
GQA	Grouped Query Attention
RoPE	Rotary Positional Embedding
RMSNorm	Root Mean Square Normalization
SwiGLU	Swish–Gated Linear Unit
QK-Norm	Query–Key Normalization
BBPE	Byte-Level Byte-Pair Encoding
ONNX	Open Neural Network Exchange
CUDA	Compute Unified Device Architecture
HF	Hugging Face
API	Application Programming Interface
GPU	Graphics Processing Unit

Acknowledgement

I would like to express my sincere gratitude to my university supervisor, Prof. Dr. Michael Spangenberg, for his valuable guidance, constructive feedback, and continuous support throughout this thesis. His insights and encouragement played an important role in shaping the direction and quality of this research.

I am deeply grateful to my thesis supervisor at the German Aerospace Center (DLR), Dr. Kartheek Kumar Nareddy, for his constant guidance, technical expertise, and unwavering support during the entire research process. His patience, thoughtful discussions, and ability to provide practical solutions whenever challenges arose were invaluable to the successful completion of this work.

I would also like to sincerely thank my manager at DLR, Dr. Julia Neibling, for providing me with the opportunity to work on this research project and for her continuous encouragement and support throughout my time at DLR. Her trust and motivation helped me grow both professionally and personally.

Furthermore, I would like to acknowledge my colleagues and the entire team at the German Aerospace Center (DLR) for creating a collaborative and inspiring research environment. Their support, discussions, and willingness to share knowledge greatly contributed to this thesis.

Finally, I would like to express my heartfelt gratitude to my family and loved ones for their unwavering support, encouragement, and belief in me throughout my academic journey. Their patience, understanding, and constant motivation have been a source of strength during both challenging and rewarding moments.

This thesis would not have been possible without the support of all the individuals mentioned above, and I am sincerely grateful to each of them.

Abstract

Large Language Models (LLMs) and multimodal foundation models have achieved impressive performance across diverse reasoning and language understanding tasks, yet their rapidly expanding parameter counts impose severe constraints on memory, latency, energy consumption, and deployability. These limitations are particularly acute in settings requiring repeated evaluation and ablation under constrained hardware environments. This thesis investigates structured pruning and compression strategies for open-source models, focusing on depth pruning, width/channel pruning, dependency-aware correctness, and post-pruning recovery via parameter-efficient fine-tuning. We systematically examine multiple model families and scales, including Qwen3 (0.6B, 4B, 8B), LLaVA-OneVision-0.5B, Phi-4-Mini, and the specialized Earth observation foundation models TerraMind-Base and Large. The evaluation employs negative log-likelihood for intrinsic importance scoring and perplexity for performance tracking, framing task-level capability drops across standard downstream tracks—Big-Bench Hard (BBH), MMLU-Pro, MATH Hard, and MuSR—using a fixed-prompt forced continuation protocol to guarantee comparative reproducibility.

Our empirical findings reveal four primary architectural insights: (1) depth pruning consistently provides a superior and more stable trade-off between execution latency and accuracy compared to width pruning; (2) width and channel reduction successfully compresses raw memory footprints but offers minimal real-world throughput gains due to GPU kernel efficiency limits and tight dependency constraints; (3) reasoning-intensive benchmarks, particularly complex mathematical sequences, exhibit extreme structural fragility and collapse faster than standard linguistic tasks under compression; and (4) parameter-efficient recovery via low-rank adaptation (LoRA/QLoRA) successfully restores baseline performance configurations at moderate compression ratios, whereas aggressive layer-dropping approaches saturation boundaries requiring continued pre-training. Crucially, these structured pruning profiles transfer predictably to remote sensing backbones; on the TerraMind pipeline, depth-slicing diminishes latency and VRAM allocation while preserving competitive downstream classification performance on the EuroSAT benchmark. Finally, an extensive failure analysis documents environment incompatibilities, rank-deficit gradient flatlines, and multi-spectral channel shape mismatches, underscoring the vital role of tool support and robust serialization guardrails. Ultimately, this work offers a methodologically faithful, failure-aware roadmap for the resource-constrained deployment of modern neural architectures.

Chapter 1: Introduction

1.1 Research Context and Motivation

Large Language Models (LLMs) based on Transformer architectures have become foundational components of modern artificial intelligence systems, demonstrating strong performance across natural language understanding, reasoning, and multimodal tasks[1]. However, this performance is achieved at the cost of substantial computational, memory, and energy requirements, which pose significant challenges for real-world deployment, particularly in resource-constrained or latency-sensitive environments. These challenges are especially pronounced when LLMs are applied beyond cloud-scale inference, such as in scientific research, edge computing, and domain-specific analytical pipelines [33] [41].

In recent years, model compression techniques—including pruning, quantization, and knowledge distillation [7] [8] have emerged as critical research directions aimed at mitigating these limitations. Among these, **structured pruning** has gained particular attention due to its potential to reduce inference latency and memory usage in a hardware-agnostic manner by explicitly removing architectural components such as layers, attention heads, or feed-forward network channels [35], [5]. Unlike unstructured sparsification, structured pruning aligns more naturally with modern accelerator execution models and inference frameworks, making it an attractive approach for practical deployment scenarios [36] [39] [31].

This thesis is conducted in collaboration with the **German Aerospace Center (DLR)**, Germany's national research institution for aerospace, energy, transportation, security, and digitalization. Within DLR, data-driven modeling and artificial intelligence play an increasingly central role, particularly in areas such as Earth observation, remote sensing, scientific simulation, and large-scale data analysis. These domains frequently involve high-resolution data, long temporal sequences, and strict computational constraints, motivating the need for efficient yet expressive machine learning models. From this perspective, improving the efficiency of foundation models—without compromising their representational capabilities—is not merely an optimization problem, but a prerequisite for sustainable and scalable scientific AI systems.

Within DLR's data science and AI research context, structured model compression is especially relevant. Models deployed for Earth observation analysis, multimodal data fusion, or scientific decision support must balance accuracy with practical constraints on compute budgets and deployment environments. As a result, understanding how and where redundancy arises in large Transformer-based models, and how that redundancy can be safely removed, directly aligns with DLR's broader research objectives in efficient and trustworthy AI systems.

1.2 Problem Statement

Despite the growing body of work on LLM compression, several open challenges remain. First, redundancy in deep Transformer stacks is not uniformly distributed, and naive pruning strategies often lead to severe performance degradation or runtime failures. While prior studies have shown that some layers or channels can be removed with minimal impact, the conditions under which such pruning is safe remain poorly understood, particularly across different model families and task domains [33], [34].

Second, existing pruning approaches often focus on either depth pruning or width/channel pruning in isolation, without sufficiently addressing dependency constraints between architectural components. Transformer layers exhibit tightly coupled structures—such as attention projections, residual connections, and feed-forward submodules—where removing one component may implicitly affect others. Failure to account for these dependencies can result in shape mismatches, invalid checkpoints, or silent quality collapse, limiting the reliability of pruning pipelines [3], [36].

Third, evaluation practices for pruned LLMs remain fragmented. Intrinsic metrics such as perplexity and negative log-likelihood provide useful signals, but they do not fully capture task-level reasoning performance. Conversely, benchmark-driven evaluations can be sensitive to prompt formats, harness configurations, and model-specific quirks, making cross-study comparisons difficult [41] [17]. A systematic evaluation framework that combines intrinsic metrics, downstream benchmarks, and failure analysis is therefore required to meaningfully assess pruning outcomes.

1.3 Research Objectives and Contributions

The primary objective of this thesis is to systematically investigate structured pruning strategies for large language models and multimodal foundation models, with an emphasis on depth pruning, width/channel pruning, and dependency-aware pruning logic. Rather than proposing a novel pruning algorithm, the thesis focuses on empirical understanding, methodological clarity, and failure characterization across multiple model families.

The key contributions of this work are as follows:

1. **A comparative analysis of depth and width structured pruning strategies** across text-only, multimodal, and domain-specific foundation models, highlighting their respective trade-offs in efficiency and robustness.
2. **An empirical study of dependency-aware pruning constraints**, documenting common failure modes such as shape mismatches, checkpoint incompatibilities, and evaluation breakdowns.

3. **A unified evaluation framework** combining intrinsic language modeling metrics (NLL, PPL), benchmark-based reasoning tasks, and qualitative output inspection.
4. **An exploration of post-pruning recovery techniques**, including LoRA and QLoRA, to assess the extent to which parameter-efficient fine-tuning can mitigate pruning-induced degradation.
5. **A detailed failure analysis and engineering perspective**, preserving real-world errors and constraints encountered during experimentation, with implications for both research and deployment.

These contributions are positioned relative to existing work on depth pruning, structured pruning, and pruning-aware fine-tuning, and are intended to inform both academic research and applied model optimization efforts. Beyond comparing pruning configurations, this thesis contributes a systems-aware empirical interpretation of structured pruning. The experiments show that pruning behaviour is strongly dependent on reasoning domain, model scale, pruning axis, and runtime backend. Depth pruning improves latency and throughput more reliably because it shortens the sequential Transformer execution path, whereas width pruning primarily reduces memory footprint and parameter count but does not necessarily improve generation speed due to GPU kernel and tensor-shape effects. Mathematical reasoning emerges as the most fragile capability under structural compression, while larger models exhibit greater recoverability due to higher representational redundancy. The thesis further demonstrates that deployment and evaluation compatibility become major bottlenecks after pruning, particularly for multimodal models, where GGUF conversion, vLLM execution, tokenizer metadata, and backend dependency issues expose gaps between compression research and production-ready inference pipelines.

1.4 Scope and Assumptions

The scope of this thesis is strictly bounded by the structural modification, evaluation, and recovery pipelines of specific open-source large language models (LLMs) and vision-language models (VLMs). Specifically, this research focuses on the parameter-efficient structural pruning and post-pruning recovery of the **Qwen3 (0.6B, 4B-Instruct, and 8B)** model suite, **Phi-4-Mini-Instruct**, **LLaVA-OneVision-Qwen2-0.5B**, and the specialized **TerraMind (Base and Large)** Earth Observation architectures.

Investigation into model compression is limited to structural techniques: *depth pruning* via index-targeted layer dropping, and *width pruning* constrained by the precise geometric boundaries of attention heads and Feed-Forward Network (FFN) channels. Post-pruning capability recovery is strictly evaluated through parameter-efficient fine-tuning (PEFT) using Low-Rank Adaptation (LoRA) powered by assistant-only loss masking and proxy training datasets. This avoids test-set contamination by isolating downstream benchmarks strictly to evaluation tasks.

The architectural and deployment boundaries of this research are constrained by the following infrastructure realities:

- **Downstream Language Evaluation:** Model performance is mapped using automated evaluation harnesses across standardized subsets, focusing on Perplexity (PPL) and Negative Log-Likelihood (NLL) via forced-continuation sliding windows, alongside standard downstream reasoning benchmarks (e.g., MMLU-Pro, BBH, MATH Hard, MuSR).

- **Multimodal Domain Expansion:** Vision-language evaluation includes both standard consumer-grade multimodal configurations (LLaVA-OneVision) and custom multi-spectral Earth Observation (EO) workflows. The latter utilizes a linear classification probing framework trained on multi-modal Sentinel-1 (S1GRD) and Sentinel-2 (S2L2A) satellite imagery via the EuroSAT dataset.

- **Deployment & Quantization Constraints:** While optimization for downstream deployment is investigated, edge-device quantization and GGUF format compilation within llama.cpp are strictly bounded by upstream tensor limitations. Full GGUF export is treated as unsupported for the specialized TerraMind architecture due to its non-standard multi-spectral encoding pipeline and limited for LLaVA-OneVision due to missing compatibility with specific vision projection tensors (e.g., image_newline).

- **Compute Infrastructure:** Core execution and performance analysis are bounded by the hardware boundaries of localized high-performance clusters, explicitly utilizing SLURM and KRATOS scheduling environments.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 presents background material and a review of related work on Transformer architecture, model compression, structured pruning, and evaluation frameworks. Chapter 3 describes the architectures of the models studied, focusing on structural aspects relevant to pruning. Chapter 4 details the pruning methodology and dependency-aware considerations. Chapter 5 outlines the evaluation framework and metrics. Chapter 6 reports experimental results, followed by a comprehensive failure analysis in Chapter 7. Chapter 8 discusses backend and deployment constraints. Chapter 9 synthesizes the findings, Chapter 10 outlines limitations, and Chapter 11 concludes with directions for future work

Chapter 2: Background and Literature Review

This chapter contextualizes the pruning, compression, evaluation, and deployment decisions explored in this thesis. Building upon established literature, it expands the theoretical foundations for Transformer architectures, model compression techniques, structured pruning, redundancy analysis, parameter-efficient fine-tuning, evaluation frameworks, and domain-specific considerations.

2.1 Transformer and LLM Basics Relevant to Pruning

The Transformer architecture, introduced by Vaswani *et al.*, remains the backbone of most modern LLMs[1]. Each decoder block consists of multi-head self-attention (MHSA) followed by a position-wise feed-forward network (FFN), with residual connections and layer normalization to stabilize training. Positional information is encoded via sinusoidal or rotary embeddings. Depth corresponds to the number of stacked blocks, while width denotes hidden dimensions and attention heads.

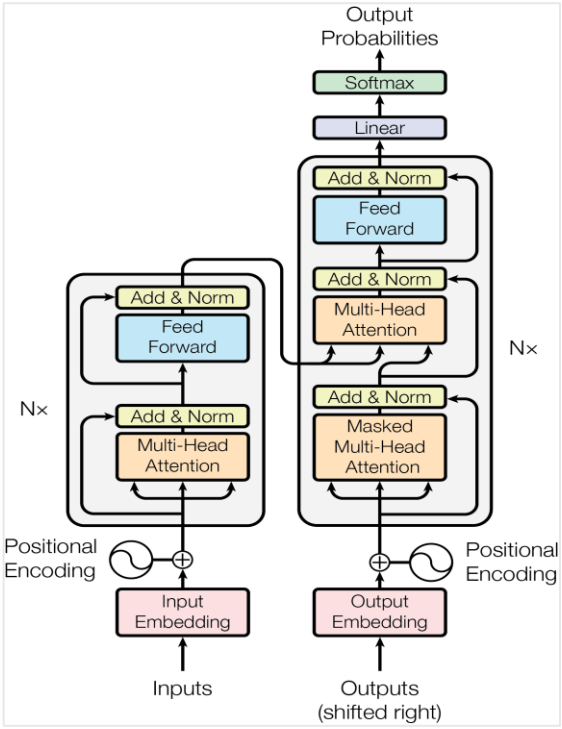


Figure 2.1 – *The Transformer Model Architecture* [1]

Pruning exploits redundancy in this structure. Studies show that early layers capture lexical and syntactic features, whereas deeper layers encode semantics[33] Recent analyses confirm that redundancy is not uniformly distributed; later layers may be removed more safely than middle layer[33]. Lu *et al.* ("Reassessing Layer Pruning") demonstrate that pruning the final 25 % of layers followed by fine-tuning only the LM head and last few layers achieves strong performance, outperforming some specialized pruning method. Siddiqui *et al.* ("A deeper look at depth pruning") further show that adaptive metrics (e.g., Shapley values or influence functions) can rank layer importance and allow removal of one third of attention layers in Mistral 7B without major accuracy loss[34]. Such findings motivate the focus on structured pruning throughout this thesis.

2.2 Model Compression Overview: Pruning, Quantization, and Distillation

Compression methods aim to reduce model size and inference cost while preserving task performance. **Pruning** removes parameters or structural components deemed less important. Unstructured pruning deletes individual weights, yielding sparse matrices that require custom kernels for acceleration[33]. **Structured pruning** removes entire neurons, channels, or layers, resulting in smaller dense sub-networks that align more naturally with hardware acceleration[35]. **Quantization** reduces numerical precision (e.g., 8-bit or 4-bit weights and activations) to shrink memory footprint. Methods such as GPTQ and SmoothQuant achieve high fidelity by calibrating quantization scales[35]. **Knowledge distillation** transfers knowledge from a larger teacher model to a smaller student through soft labels, enabling compression via additional training[5].

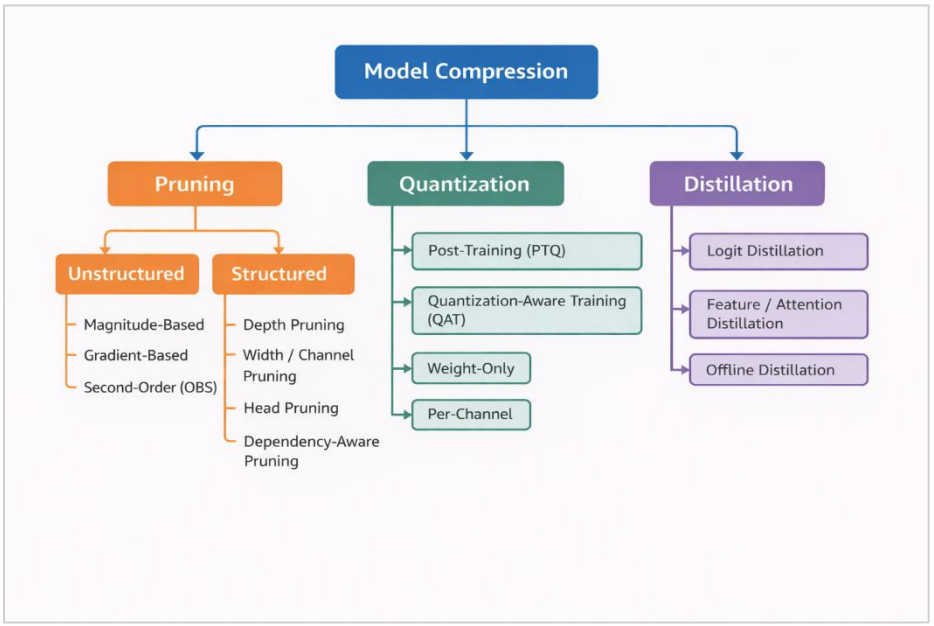


Figure 2.2 – *Conceptual Overview of Model compression techniques*

Pruning and quantization can be combined. A NeurIPS 2023 study by Kuzmin *et al.* ("Pruning vs Quantization: Which is Better?") compares both approaches and concludes that pruning yields larger accuracy drops than quantization at moderate compression ratios, but quantization alone may not reduce inference latency on GPUs[36]. Recent work on joint pruning and quantization (e.g., GETA) proposes frameworks to learn pruning masks and quantization scales simultaneously[37]. Distillation can further recover performance after aggressive pruning by training the pruned model to mimic a larger teacher. This thesis focuses primarily on structured pruning but situates its contributions within this broader compression landscape.

2.3 Structured Pruning: Depth vs Width/Channel and Dependency Correctness

In earlier work predating large language models, structured pruning was proposed as a means to reconcile model compression with efficient execution on dense hardware. Unlike unstructured sparsity, which produces irregular weight patterns, structured pruning removes entire architectural units—such as channels, neurons, or attention heads—thereby preserving dense matrix operations and avoiding sparse kernel overheads [59].

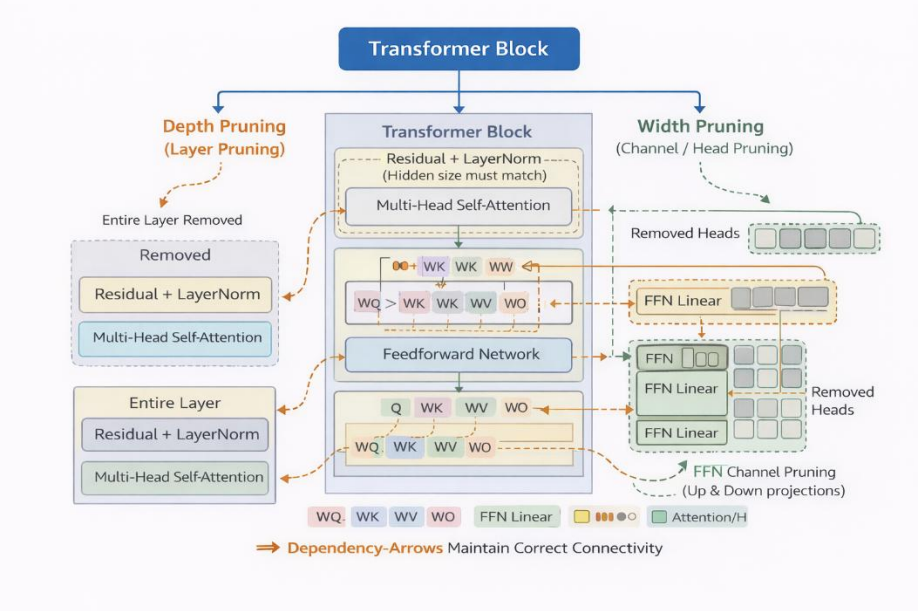


Figure 2.3 -*Conceptual illustration of structured pruning in Transformer models, highlighting depth pruning (layer removal), width pruning (channel and head removal), and dependency constraints across attention projections and residual connections. The illustration is synthesized based on prior work on layer pruning, structured width pruning, and dependency-aware pruning in Transformers [33], [35], [36], [37]*

Within Transformer architectures, attention-head pruning emerged as an early form of structured pruning: empirical studies demonstrated that many attention heads contribute marginally to model performance and can be removed with limited impact on downstream tasks [60], [61]. These findings provided early evidence that redundancy in Transformer models is highly structured rather than uniformly distributed. Subsequent work extended these ideas to block-level and layer-level pruning, showing that structured removal of components can reduce computation while maintaining functional correctness, provided architectural dependencies are respected [62]. This historical progression motivates modern structured pruning frameworks for LLMs, which build upon earlier insights into head, channel, and layer redundancy while explicitly addressing the stronger dependency constraints and scaling challenges introduced by billion-parameter Transformer models.

Structured pruning can target **depth**—removing full Transformer layers—and **width**—pruning neurons, channels, or attention heads. Depth pruning reduces the number of sequential computations; width pruning maintains depth but reduces per-layer computations. Both require *dependency awareness*. Transformer blocks are interconnected via residual connections and projection matrices; pruning in one layer must propagate to subsequent layers to avoid shape mismatches[35].

Several frameworks address this. **LLM-Pruner** introduces gradient-based structural pruning and a dependency detection algorithm to identify "coupled structures" that must be pruned together[35]. It groups dependent projections (e.g., query, key, value, and output) and prunes them jointly. **SlimGPT** applies optimal brain surgeon (OBS) theory to layer-wise structured pruning and uses batched greedy pruning via Cholesky decomposition to approximate second-order importance; dynamic group sizes and incremental pruning ratios achieve near-optimal results with minimal error accumulation[36]. **DISP-LLM** relaxes the constraint that all layers share identical dimensions: it allows blocks to adopt different widths by selecting variable subsets of feature maps, offering additional flexibility and higher performance than rigid structural pruning[37]. **2SSP** further combines width and depth pruning sequentially: it first prunes FFN submodules (width) then prunes entire attention blocks (depth), mitigating error propagation.

These frameworks underscore the need for dependency-aware pruning. Without careful propagation of mask decisions, pruned models exhibit loading failures, shape errors, or silent degradation, as later chapters will show. This thesis adopts similar dependency-aware logic and discusses associated failures in Chapter 7.

2.4 Unstructured Pruning Literature

Unstructured pruning removes individual weights from a neural network without enforcing any regular structural pattern, resulting in sparse weight matrices while preserving the original layer dimensions and architectural topology [54], [55]. Early work demonstrated that many parameters

in over-parameterized neural networks are redundant and can be removed with minimal loss in accuracy when followed by appropriate fine-tuning, establishing unstructured pruning as one of the earliest and most widely studied model compression techniques [54]. The Deep Compression framework further formalized this approach by combining unstructured weight pruning with retraining, quantization, and entropy coding, achieving substantial reductions in model storage while maintaining predictive performance on a range of tasks [55]. These studies laid the foundation for sparsity-based compression and motivated extensive follow-up research into pruning criteria, sparsity schedules, and retraining strategies.

A key advantage of unstructured pruning is its fine-grained flexibility. Because pruning decisions are made at the level of individual weights rather than entire neurons or layers, unstructured methods can target redundancy with high precision and often preserve accuracy at moderate sparsity levels, particularly when pruning is combined with retraining or calibration [54], [55]. The most commonly used criterion is magnitude-based pruning, which removes weights with small absolute values and has been shown to be surprisingly competitive across a wide range of architectures and training regimes [56]. For large language models, recent work has revisited unstructured pruning in the context of billion-parameter Transformers, focusing on post-training or one-shot approaches that avoid expensive retraining. SparseGPT demonstrates that GPT-style models can be pruned to high unstructured sparsity in a single pass with minimal perplexity degradation by leveraging second-order information and careful weight updates [57]. Similarly, Wanda proposes a simple yet effective pruning criterion that combines weight magnitude with activation statistics, achieving strong results on LLaMA-family models without requiring any gradient updates or additional training [58]. These methods highlight the practicality of unstructured pruning when training compute is limited and the goal is to reuse pretrained checkpoints.

Despite its strengths, unstructured pruning exhibits important limitations for deployment and inference efficiency. A consistent finding across the sparsity literature is that unstructured sparsity does not reliably translate into inference speedups on standard GPU or accelerator hardware, because sparse weight matrices induce irregular memory access patterns and require specialized kernels to achieve performance gains [56]. Large-scale evaluations show that while sparse models can reduce parameter count and storage, the overhead of sparse representations and the lack of widespread hardware support often negate theoretical computational savings in practice [56]. Even in scenarios where unstructured pruning yields improvements in energy efficiency or memory footprint, these benefits depend heavily on the availability of optimized sparse libraries and careful system-level integration [55]. As a result, unstructured pruning is often more effective as a model size and memory optimization technique rather than a guaranteed solution for latency reduction in real-world inference pipelines.

In summary, unstructured pruning remains a foundational and actively evolving compression technique with strong empirical evidence supporting its ability to reduce parameter count and storage requirements in large neural networks, including modern LLMs [54], [55], [57], [58].

However, the irregular sparsity patterns it produces pose challenges for hardware-efficient execution, limiting its practical impact on inference latency without specialized support [56]. These limitations motivate the growing interest in structured pruning approaches, which trade fine-grained flexibility for improved alignment with modern accelerator architectures and more predictable deployment behavior.

2.5 Layer Redundancy and Depth Pruning Literature

Depth pruning leverages the observation that many Transformer layers contribute marginally to final performance. **Shortened LLaMA** demonstrates that simple removal of layers can match or surpass width pruning for inference efficiency; it shows that removing entire layers yields faster speed-ups on memory-constrained devices compared with widening or channel pruning[39]. The authors also note that continued pre-training of the pruned model (rather than using LoRA) better recovers performance, especially at high pruning ratios[39]. **Reassessing Layer Pruning** systematically benchmarks layer pruning across models (e.g., LLaMA, LLaMA-2) and tasks, finding that pruning the last 25 % of layers followed by fine-tuning only the LM head and last three layers achieves near-baseline performance and that LoRA-based recovery does not offer significant advantages[33]. The authors highlight that layer importance is often monotonic: later layers can be pruned more safely than middle layers, while early layers remain critical.

Siddiqui *et al.* ("A deeper look at depth pruning") explore adaptive importance metrics to rank layers: static metrics (e.g., weight norms) and adaptive ones (e.g., Shapley values) produce different pruning patterns, and self-attention layers are more amenable to pruning than FFN layers[34]. They show that removing 33 % of attention layers in Mistral 7B yields no MMLU degradation[34]. Prompt-based approaches, such as **PuDDing**, propose dynamic depth pruning conditioned on input prompts to tailor model depth at inference time; while promising conceptually, these methods introduce additional routing overhead and are beyond the scope of this work. Together, these studies highlight that depth pruning can achieve significant efficiency gains but requires careful layer selection to avoid catastrophic loss, motivating the deterministic depth strategies evaluated in Chapter 4.

2.6 Parameter-Efficient Fine-Tuning (LoRA, QLoRA) and Post-Pruning Recovery

Parameter-efficient fine-tuning (PEFT) methods adapt large models using a small number of trainable parameters. **LoRA** injects low-rank adaptation matrices into existing projection layers, enabling efficient fine-tuning while freezing the backbone[35]. **QLoRA** combines LoRA with quantization to reduce memory requirements further. These methods have been widely adopted for instruction tuning and domain adaptation.

However, their efficacy after structured pruning is mixed. Reassessing layer pruning finds that fine-tuning only the output head and the last few layers without LoRA recovers performance comparably or better than LoRA-based tuning[35]. Shortened LLaMA observes that continued pre-training (unfreezing all layers) outperforms LoRA for severe pruning ratios[39]. In contrast, LLM-Pruner uses LoRA to fine-tune pruned models on 50 k data samples and reports good recovery within three hours This thesis therefore treats LoRA/QLoRA as optional recovery tools rather than guaranteed remedies; their effectiveness depends on pruning severity, data availability, and task requirements.

From a representational perspective, the limitations of LoRA and QLoRA as post-pruning recovery mechanisms stem from their reliance on low-rank updates applied to a frozen backbone. While low-rank adaptation is effective for task adaptation and instruction alignment, it does not restore parameters or representational capacity that has been structurally removed through pruning [63]. Empirical analyses show that when pruning eliminates entire layers or channels, the remaining network may lack sufficient expressivity for LoRA-style updates to compensate, particularly at high pruning ratios [64]. Furthermore, the optimal rank for LoRA adapters is not invariant to pruning: aggressive pruning alters layer-wise sensitivity and information flow, making fixed low-rank assumptions suboptimal [65]. These findings reinforce the view that PEFT methods are best suited for light-to-moderate pruning regimes, while severe structured pruning often necessitates partial or full continued pre-training to reallocate capacity and re-establish internal representations [39], [64].

2.7 Evaluation and Benchmarking Frameworks

Evaluating pruned LLMs requires metrics that capture both intrinsic language modeling quality and task-level performance. **Negative log-likelihood (NLL)[4]** remains the fundamental measure: it directly reflects the log-probabilities assigned by the model. **Perplexity (PPL)**[4], the exponential of NLL, is widely reported but can be numerically unstable and less sensitive to pruning-induced changes. For depth pruning analysis, NLL is preferred because it aggregates linearly across tokens and layers; perplexity is retained as a reporting metric but not used for pruning decisions.

Beyond pruning-specific studies, recent work on LLM evaluation has shown that perplexity alone is a weak proxy for reasoning and generalization performance, particularly on complex multi-step tasks. Comprehensive evaluations demonstrate that models with similar perplexity can exhibit large variance on reasoning, factual recall, and robustness benchmarks, underscoring the disconnect between token-level likelihood and task-level competence [66], [67]. Furthermore, benchmark-driven evaluations themselves are sensitive to prompt templates, scoring rules, and dataset contamination, which can lead to inconsistent or misleading conclusions if not carefully controlled [68]. These findings have motivated standardized evaluation frameworks that emphasize transparency, reproducibility, and multi-metric reporting rather than reliance on a single

scalar score. In the context of pruned LLMs, such practices are especially critical, as architectural modifications can selectively impact reasoning depth, calibration, or long-context behavior without substantially affecting perplexity. Consequently, modern pruning studies increasingly adopt a combination of intrinsic metrics, diverse benchmarks, and qualitative inspection to obtain a faithful assessment of model behavior [66], [68]

Recent evaluation studies emphasize the need for **benchmark-based assessments** alongside perplexity. Reassessing layer pruning and SlimGPT highlight that a model's perplexity may remain stable while its reasoning performance collapses on tasks like MMLU or BBHarxiv.org. The **lm-evaluation-harness** [17]provides a unified interface for such benchmarks, with tasks including Big-Bench Hard (BBH)[18], MMLU-Pro[19], GPQA[20], MuSR[21], and math reasoning. However, harness configurations can introduce prompt and metric variability; consistent evaluation requires fixing context lengths, seeds, and continuation lengths, as described in Chapter 5.

Furthermore, evaluation frameworks must handle *pruned* models carefully: missing layers may alter hidden-state dimensions or attention-head counts, breaking some tasks unless adapters adjust shapes. Failures during evaluation, such as dataset access issues or metric computation errors, are analyzed in Chapter 7. Overall, robust evaluation demands a combination of perplexity, NLL, and downstream benchmarks, and a transparent reporting of failures.

2.8 Vision–Language and Multimodal Pruning Context

Pruning multimodal models introduces cross-modal dependencies absent in text-only LLMs. In vision–language architectures like LLaVA-OneVision[23], a vision encoder produces embeddings that are fused with the language model through projection layers. Removing these layers or channels can disrupt alignment and degrade multimodal understanding. While the structured pruning literature remains dominated by text-only models, recent work on efficient VLMs such as EfficientLLaVA[24] suggests that dependency-aware pruning must account for visual–textual interactions. The adaptive routing or dynamic depth pruning in PuDDing could inspire future multimodal pruning strategies, although this thesis remains within the scope of text-only and geospatial models.

Prior work on efficient vision–language models has consistently shown that multimodal architectures exhibit stronger and more brittle dependency constraints than unimodal Transformers. In early vision–language systems, compression efforts often focused on the visual backbone alone, but later studies demonstrated that aggressive pruning of either the visual encoder or the cross-modal projection layers can severely impair alignment between visual features and linguistic representations, leading to disproportionate drops in multimodal reasoning performance [69], [70]. Unlike text-only models, where depth or width pruning primarily affects internal language representations, multimodal models rely on carefully calibrated embedding spaces to enable cross-

modal attention and grounding. Recent analyses indicate that even small dimensional mismatches or representation shifts introduced by pruning can disrupt visual–textual correspondence, suggesting that pruning decisions must be coordinated across modalities rather than applied independently [71]. These findings help explain why structured pruning methods developed for text-only LLMs do not directly transfer to multimodal settings and motivate the need for modality-aware and dependency-preserving pruning strategies in future work. ‘

2.9 Remote Sensing Foundation Models and EuroSAT Context

Prior work in remote sensing has emphasized that deep models for Earth observation are particularly sensitive to disruptions in spatial structure and spectral consistency. Studies on convolutional and Transformer-based remote sensing architectures show that intermediate feature representations often encode physically meaningful information, such as vegetation indices, texture patterns, or land-cover signatures, which can be disproportionately affected by aggressive pruning [72], [73]. Unlike text-based models, where redundancy may be distributed across layers, geospatial models rely on coordinated multi-scale feature extraction to preserve both local detail and global context. As a result, removing layers or channels without accounting for spatial hierarchies and spectral dependencies can degrade performance in ways that are not easily recoverable through fine-tuning alone [74]. These observations motivate a more conservative and evaluation-driven approach to pruning remote sensing foundation models, where insights from structured pruning in LLMs are adapted cautiously and validated using domain-specific benchmarks such as EuroSAT.

Remote sensing foundation models, exemplified by **TerraMind** [25], operate on multi-spectral satellite imagery and long-range geospatial sequences. These models employ dual-scale architectures to capture both fine-grained local patterns and broad spatial contexts. Pruning them poses unique challenges: spatial coherence and spectral fidelity are critical, and naive depth or width removal may erase essential features. The EuroSAT dataset [26] offers a benchmark for evaluating remote sensing models and underscores the need for domain-specific evaluation metrics sensitive to spatial structure. This thesis adopts these principles when discussing TerraMind pruning and evaluation.

2.9.1 Foundations of Multi-Spectral Earth Observation Models and Linear Probing

While conventional Vision-Language Models (VLMs) like LLaVA-OneVision are engineered to process standard three-channel RGB consumer imagery, multimodal foundation architectures tailored for Earth Observation (EO) must operate under distinct physical and spatial constraints. Remote sensing models, such as the TerraMind-Base and TerraMind-Large architectures, are designed to interpret complex multi-spectral and multi-modal satellite data streams. These streams

primarily encompass Synthetic Aperture Radar (SAR) data from Sentinel-1 (S1GRD) and multi-spectral optical data from Sentinel-2 (S2L2A).

Unlike RGB images, Sentinel-2 imagery consists of 13 distinct spectral bands spanning the visible, near-infrared (NIR), and shortwave infrared (SWIR) spectrums, which capture surface reflectance characteristics invisible to consumer-grade sensors. Concurrently, Sentinel-1 contributes dual-polarization radar backscatter data (VV and VH channels), providing structural and moisture indicators unaffected by cloud cover or solar illumination. Integrating these heterogeneous data modalities requires specialized input projection encoders capable of mapping non-standard channel dimensions into a shared latent space prior to token processing.

To evaluate the downstream representational fidelity of compressed or structurally pruned EO foundation models without the confounding variable of full-parameter fine-tuning, literature establishes the paradigm of *linear probing*. In a linear probing framework—such as the classification pipeline implemented for the EuroSAT benchmark—the weights of the multimodal backbone (e.g., TerraMind) are completely frozen. A single, parameterized linear layer (a linear classification probe) is appended directly to the output embeddings of the frozen network to map representation vectors to target land-cover classes.

Chapter 3: Model Architectures

This chapter presents architectural overviews of the models examined in this thesis, focusing on structural features relevant to structured pruning and parameter-efficient fine-tuning. The discussion emphasizes how design choices—such as attention mechanisms, normalization schemes, and modality fusion—affect pruning feasibility and subsequent evaluation.

3.1 Qwen3 Family (0.6 B / 4 B-Instruct / 8 B) Architecture Overview

The Qwen3 family comprises a series of dense and Mixture-of-Experts (MoE) language models spanning parameter scales from 0.6 billion to 235 billion. For pruning analyses in this thesis, attention is restricted to the **dense** models Qwen3-0.6B, Qwen3-4B, and Qwen3-8B. According to the official technical report, these models share a common Transformer decoder architecture similar to Qwen2.5, with the following key design features:

- **Attention mechanism:** Qwen3 uses **Grouped Query Attention (GQA)**, which divides query heads into groups while sharing key and value projections; this reduces memory and improves throughput [27].
- **Activation and normalization:** Each block employs **SwiGLU** activation and **RMSNorm** in a **pre-norm** configuration, replacing the LayerNorm used in earlier Qwen versions. Rotary Positional Embeddings (RoPE) encode positional information; a QK-Norm is introduced in the attention mechanism to improve training stability.
- **Bias removal:** Qwen3 removes the QKV bias terms present in Qwen2, simplifying the projection matrices and facilitating structured pruning [27]
- **Model variants:** The dense models differ mainly in depth and width: Qwen3-0.6B has 28 layers with 16 query heads and 8 key/value heads per attention block (context length 32 K); Qwen3-4B and Qwen3-8B each have 36 layers with 32/8 heads and support longer contexts (128 K). Larger variants like Qwen3-14B and Qwen3-32B increase depth and head counts accordingly[27].
- **Tokenization:** All models adopt Qwen's byte-level byte-pair encoding (BBPE) tokenizer with a vocabulary of 151 669 tokens.

These architectural elements—particularly the grouped query attention, RoPE, and pre-norm design—are relevant for pruning because they constrain how layers and channels can be removed. For example, dependency-aware pruning must ensure that grouped query projections are pruned consistently across the query, key, and value matrices. The removal of QKV biases simplifies parameter grouping. The context-length capabilities (32 K–128 K) also affect evaluation, as depth pruning[34] must preserve the model's ability to handle long sequences. Throughout this thesis, the instruction-tuned variant Qwen3-4B-Instruct is treated as architecturally identical to its base model; only alignment data differs.

3.2 LLaVA-OneVision Qwen2-0.5 B Architecture

LLaVA-OneVision extends the LLaVA family of vision-language models by enabling unified processing of images, multi-image sequences, and videos. The model uses a **minimalist design** that connects a vision encoder to a language model through a lightweight projection module. Section 3.1 of the LLaVA-OneVision paper[23] describes the architecture as follows:

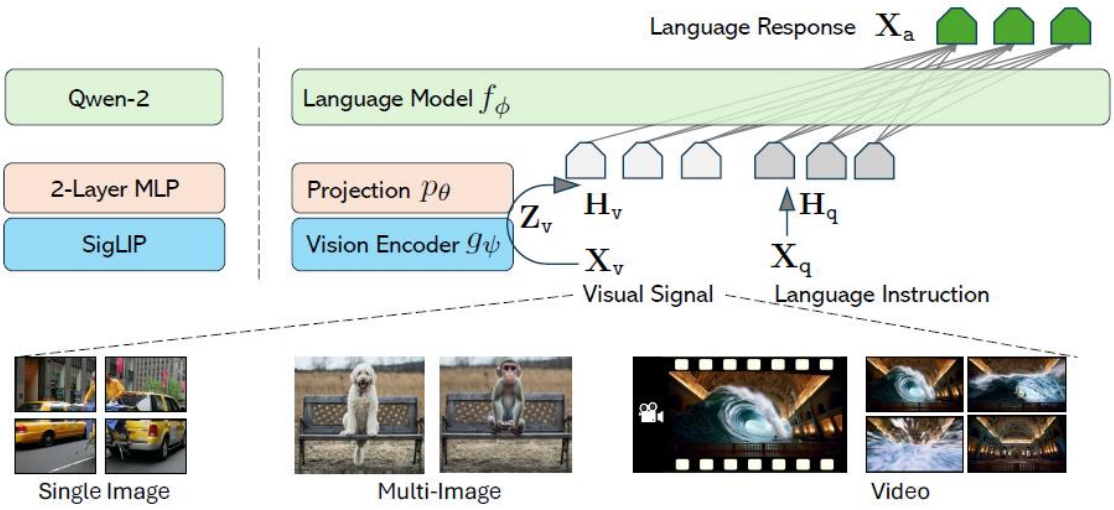


Figure 3.1 – *LlaVA-OneVision Architecture with projected & fused with textual token[23]*

- **Language backbone:** The model employs **Qwen-2** as its language model *fφ(·)f_\varphi(\cdot)*fφ (·), chosen for its strong language capabilities and available open-weight checkpoints.
- **Vision encoder:** Visual signals are encoded by **SigLIP**, a vision transformer variant, yielding grid features before and after the final transformer layer[23].
- **Projector module:** A **two-layer MLP** projects visual features into the word-embedding space, producing a sequence of visual tokens *HvH_v*Hv[23].
- **Fusion mechanism:** The sequence of visual tokens is prepended to textual tokens and passed through the language model. This simple connection facilitates strong scaling behavior and allows task transfer across single-image, multi-image, and video scenarios.
- **Visual representation strategy:** The model adopts an **AnyRes** scheme to handle high-resolution images and sequences; a "Higher AnyRes" strategy uses bilinear interpolation to balance the number of visual tokens and resolution. Visual tokens are

allocated based on scenarios (single image, multi-image, video), ensuring comparable sequence lengths across modalities[23].

This modular architecture is particularly relevant to pruning: removing layers from the language backbone must maintain compatibility with the projection module and the joint embedding space. The small 0.5 B backbone ensures that pruning experiments remain computationally feasible while still highlighting cross-modal dependency constraints.

3.3 Phi-4-Mini Instruct: Architecture Overview

Phi-4-Mini Instruct [50] is a compact variant of Microsoft's Phi-4 family of Small Language Models (SLMs). It is a **dense, decoder-only Transformer model** with approximately **3.8 billion parameters**, designed to provide strong performance in instruction-following, reasoning, and text generation tasks while remaining efficient enough for deployment in resource-constrained environments.

Architecture and Design

- **Architecture:** Dense decoder-only Transformer (standard autoregressive architecture) with multi-head self-attention and feed-forward blocks, following the same architectural paradigm as other Phi-4 models.
- **Parameter Count:** ~3.8 B parameters.
- **Vocabulary and Context:** The model uses an extended vocabulary and supports long context windows (up to 128 K tokens), matching the trends in the Phi-4 series toward extended context understanding[50].
- **Training Data:** Trained on a mixture of high-quality synthetic data and filtered public web content, leveraging techniques similar to larger Phi models to improve instruction-following and reasoning behavior.
- **Deployment:** Designed for instruction-based tasks (hence *Instruct*) and optimized for efficient inference, including potential deployment on edge devices with quantization (e.g., GGUF formats for CPU inference) in third-party workflows.

The **Phi-4-Mini Instruct** model builds upon improvements from the Phi-3.5-Mini series and the larger Phi-4 family, offering competitive reasoning and general language understanding performance relative to other small models in the 3-5 B class [50].

3.4 TerraMind-Base and TerraMind-Large: Dual-Scale Remote Sensing Architectures

TerraMind is a foundation model developed for Earth-observation (EO) data, with versions denoted v1-B (base) and v1-L (large). Unlike language-centric models, TerraMind processes **multiple geospatial modalities** and employs a **dual-scale architecture** that simultaneously ingests **token-level** and **pixel-level** representations. Key architectural features from the TerraMind paper include:

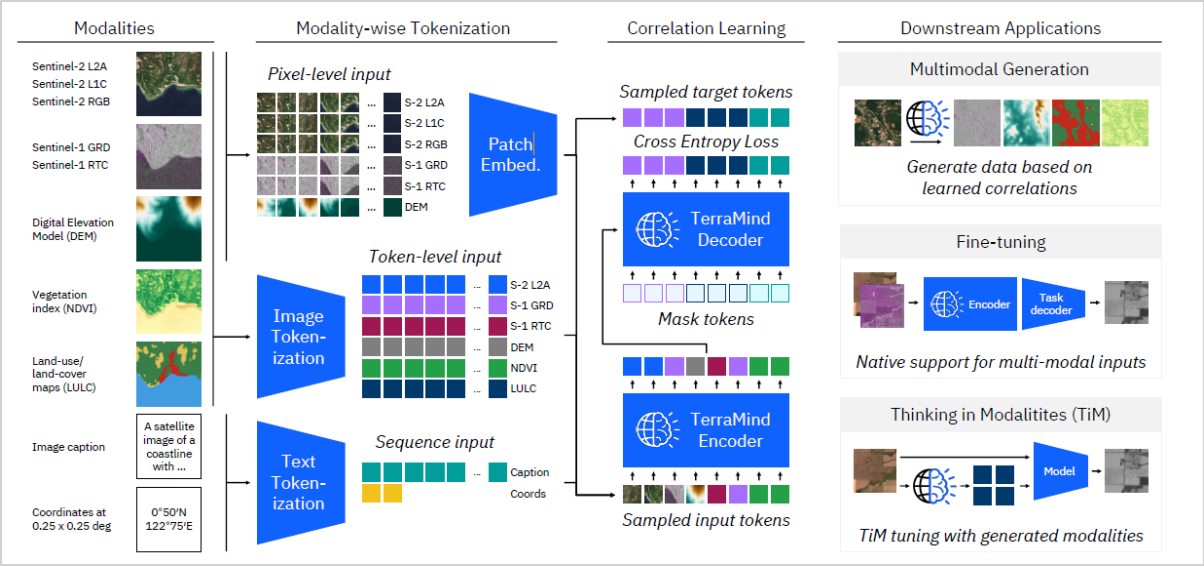


Figure 3.2 - *Illustrate how TerraMind processes token-level and pixel-level (patch) inputs through a dual-scale encoder–decoder structure [25].*

- **Dual-scale representations:** TerraMind is pretrained on both **token-level data** (capturing high-level contextual information) and **pixel-level data** (capturing fine-grained spatial nuances). The model applies learnable patch-wise linear projections to generate patch embeddings for each $16 \times 16 \backslash times 16$16×16 pixel patch, similar to the Vision Transformer (ViT) approach[25].
- **Encoder–decoder design:** A Transformer encoder processes the concatenated sequence of token- and patch-level embeddings, enabling cross-modal patch classification. A decoder reconstructs masked tokens during pretraining, following a masked modeling strategy drawn from multimodal masked modeling frameworks.
- **Pretraining objective:** TerraMind samples random subsets of patches and tokens and predicts masked targets via cross-entropy loss, enabling the model to learn correlations across modalities[25].

- **Scaling variants:** TerraMind v1-B is pretrained on 500 billion tokens for six days on 32 A100 GPUs; v1-L scales up the backbone and pretraining time to achieve better performance. A “single-scale” variant trains on predominantly token-level data to isolate the effect of pixel-level inputs.
- **Dual-scale advantages:** The paper emphasizes that dual-scale pretraining improves zero-shot and few-shot performance across EO tasks, enabling generation of artificial data and cross-modal transfer.

These architectural innovations render TerraMind particularly challenging for pruning. Removing layers or channels may disrupt the balance between token-level and pixel-level processing. Moreover, the need to preserve dual-scale fusion constrains depth pruning strategies. In this thesis, TerraMind-base and TerraMind-large serve as case studies for structured pruning beyond traditional text models, highlighting the complexities of multimodal and geospatial data.

Chapter 4: Pruning Methodology

This chapter details the structured pruning methodologies applied in this thesis. It synthesizes general pruning theory with observations from recent literature to contextualize the deterministic strategies used and to explain dependency-aware constraints. Unlike some pruning frameworks that introduce novel algorithms, this work focuses on **understanding**, **comparing**, and **failure-analyzing** methods rather than proposing new pruning schemes.

4.1 Depth Pruning Strategies

Depth pruning removes entire Transformer layers, reducing sequential operations and inference latency. The rationale stems from the observation that not all layers contribute equally to model performance. However, **layer importance is highly non-uniform**, and a simple uniform pruning ratio may produce disproportionate degradation. Literature offers multiple approaches to layer selection.

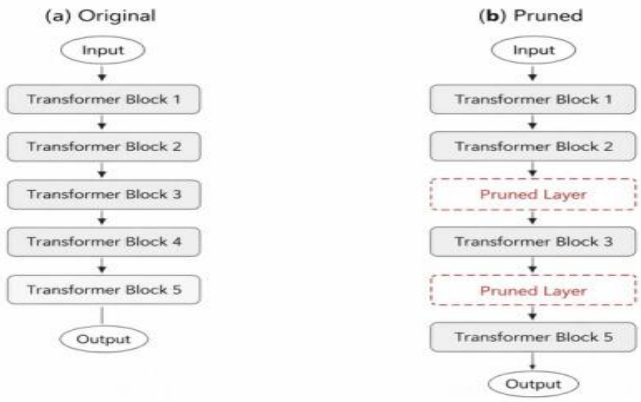


Figure 4.1 - *Depth pruning a neural network reduces overall depth and complexity [31]*

1. **Simple deterministic strategies:** Approaches such as *keep-first*, *keep-last*, *sandwich* (retain early and late layers), and *interleaved* (retain layers at regular intervals) provide low-overhead pruning schemes suited for baseline analysis. These strategies are used in this thesis to isolate the impact of different layer regions.
2. **Drop-one ablation:** Removing one layer at a time and measuring NLL or perplexity changes identifies layers with lower marginal utility. Siddiqui *et al.* compare static metrics (e.g., weight norms) and adaptive metrics (e.g., Shapley values) for layer importance; they find that self-attention layers are more amenable to pruning and that up to 33 % of attention layers can be removed without MMLU degradation[34].
3. **Adaptive depth pruning:** The **PuDDing [**88**]** framework performs *prompt-routed depth pruning*, where depth selection varies according to the input prompt[34]. Although

promising, such dynamic strategies increase inference-time complexity and are beyond this thesis's scope. Below is the illustration shows how depth pruning works.

4. **Analytic rankings and influence measures:** Lu *et al.* observe that pruning the final 25 % of layers and fine-tuning only the output head and last few layers yields strong performance, questioning the necessity of low-rank adapters for recovery[34]. Adaptive metrics based on gradients or influence functions can better capture non-uniform importance but require additional computation.
5. Beyond quality trade-offs, depth pruning also carries a significant algorithmic advantage in execution cost. Because layer removal requires only index-based structural slicing with no parameter-level computation, pruning execution completes in under 10 milliseconds regardless of model scale, as demonstrated empirically in Section 6.9.5.

4.2 Width and Channel Pruning Mechanics

Width or channel pruning removes neurons or channels within layers, reducing per-layer computation. In Transformer architectures, this includes pruning **attention heads** and **feed-forward network (FFN)[1] neurons**. The main challenge is that removing parameters in one projection affects other projections and downstream layers. Recent structured pruning frameworks highlight different approaches to channel selection:

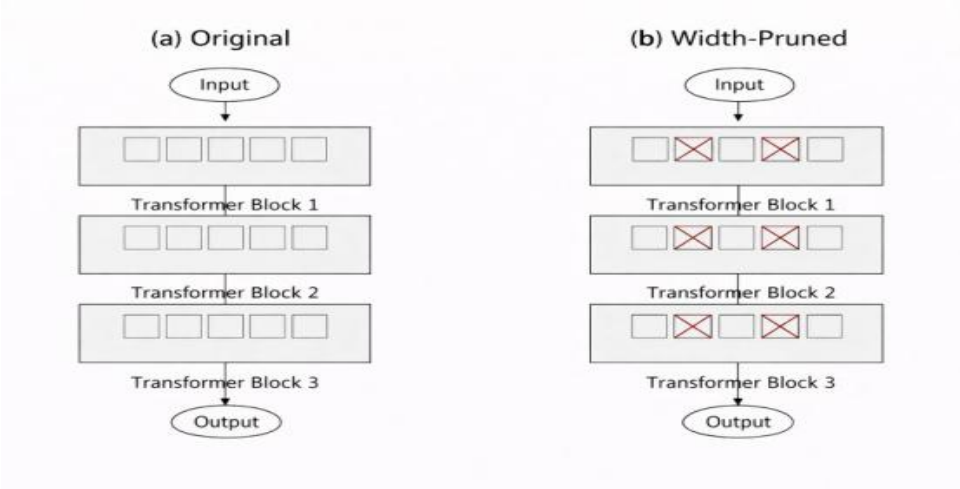


Figure 4.2 - *Reducing layer width by pruning an unimportant neuron [31]*

1. **Gradient-based importance and dependency grouping: LLM-Pruner** introduces a three-stage framework: (i) compute first-order importance scores from gradients and approximated Hessians, (ii) group dependent structures across query, key, value, and output projections, and (iii) prune non-critical coupled structures[3]. LoRA-based fine-tuning recovers performance with limited data.

2. **Second-order approximation: SlimGPT** employs the Optimal Brain Surgeon (OBS) method to compute parameter importance via approximate Hessian inverses. It performs Batched Greedy Pruning with dynamic group sizes, achieving near-optimal structured pruning results within an hour[36].
3. **Dimension-independent pruning: DISP-LLM** relaxes the constraint that all layers share the same width, allowing each block to have different numbers of channels and attention heads[37]. This flexibility improves pruning quality but complicates model loading and deployment.
4. **Two-stage pruning: 2SSP** combines width and depth pruning by first pruning FFN submodules, then pruning attention blocks[38]. The separation mitigates error propagation and maintains performance.
5. **Unstructured baseline: Wanda** performs magnitude-based unstructured pruning and demonstrates that weights and activations combined provide effective sparsity signals[36]. However, irregular sparsity patterns often fail to accelerate inference on current hardware.
6. Width pruning, in contrast, requires magnitude scoring, dependency graph traversal, and coordinated mask propagation across coupled projection matrices. This results in execution times that grow substantially with model scale, reaching up to 24.90 seconds for larger models, as reported in Section 6.9.5.

Channel importance is typically estimated via weight magnitude, gradient magnitude, or Hessian-based criteria. Width pruning eliminates internal structures such as individual neurons, attention heads, or embedding channels, slimming down the model's width [31] For attention head pruning, magnitude-based head importance can identify redundant heads.

4.3 Layer Importance Estimation for Depth Pruning

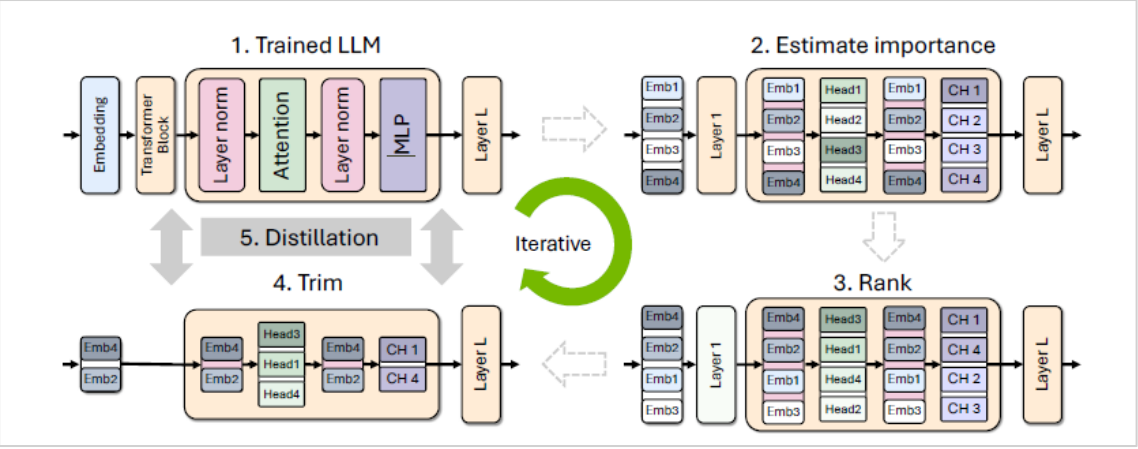


Figure 4.3 - *High-level overview of our proposed iterative pruning and distillation approach [53]*

Depth pruning requires a principled mechanism to identify which Transformer layers contribute least to overall model performance. Unlike width pruning, where importance can often be inferred from parameter magnitudes, depth pruning must account for the hierarchical and sequential nature of Transformer representations. Recent studies have shown that not all layers contribute equally to task performance, and that certain layers—particularly in early or mid-depth regions—can be removed with limited degradation when carefully selected. The Figure shows how process works and how internally it check the less importance layer and more important layer with the neuron removal from the transformer architecture but alone its not sufficient for model optimization that can be discussed in the next section which is dependency aware pruning where we discuss why dependency aware pruning of layers is important for the depth pruning not the neuron removal method. .

In this thesis, **layer importance is estimated using an ablation-based sensitivity analysis**, following the methodology proposed in recent work on structured depth pruning for large language models [48]. The core idea is to measure the impact of removing a single Transformer block on the model's task likelihood. Specifically, each layer is temporarily removed from the network, and the resulting change in evaluation metrics—such as negative log-likelihood (NLL) and perplexity (PPL)—is recorded. Layers whose removal induces minimal increase in loss or perplexity are considered less critical and therefore suitable candidates for depth pruning[35]. Below are the examples for models LlaVa-Onevision-0.5B, Qwen3-4B- instruct & Phi4-mini-instruct that how we can remove less important layers from the total layers.-

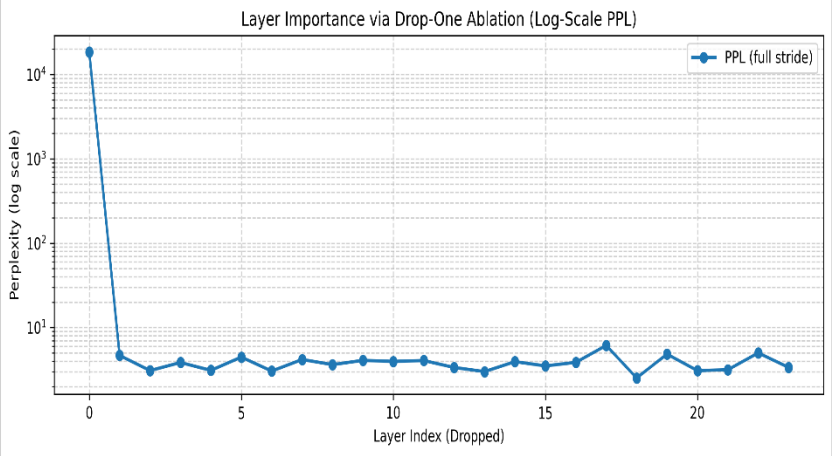


Figure 4.4- *LlaVa-OneVision-0.5B: log-scale PPL by dropping each layer*

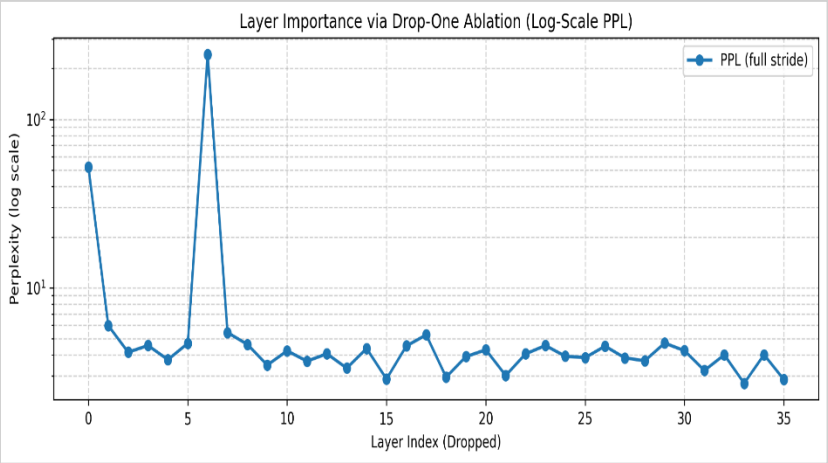


Figure 4.5- *Qwen3-4B-Instruct: log-scale PPL by dropping each layer*

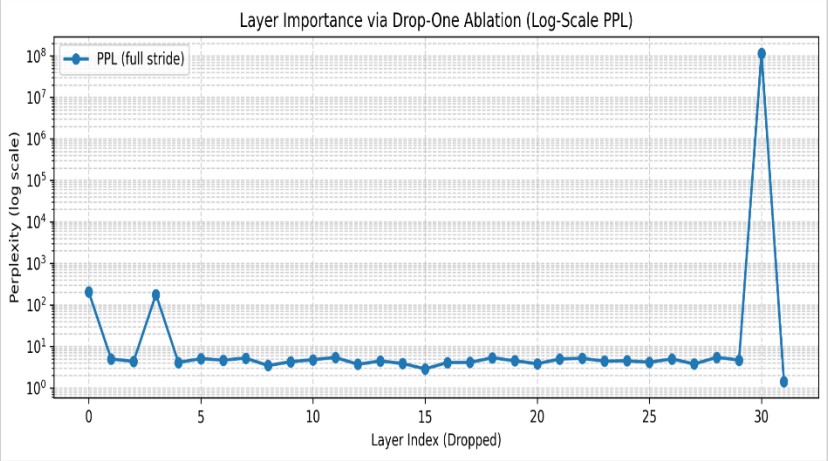


Figure 4.6 - *Phi-4 mini -instruct: log-scale PPL by dropping each layer*

To improve robustness, importance estimation is performed using multiple evaluation settings, including different sequence strides and continuation-only loss computation. This mitigates bias arising from token overlap and ensures that importance rankings reflect consistent behavior across evaluation regimes. The resulting layer importance profile enables ranking Transformer layers from least to most critical, providing a principled basis for depth pruning strategies such as sandwich pruning, greedy layer selection, and custom keep-set construction.

This ablation-driven importance estimation aligns closely with recent findings that structured depth reduction, when guided by task sensitivity, can preserve model performance while

significantly reducing computational cost [48]. In subsequent sections, the derived importance rankings are used to guide dependency-aware depth pruning, ensuring that only low-impact layers are removed while maintaining architectural consistency.

4.4 Dependency-Aware Pruning and Shape Consistency Handling

Dependency awareness is crucial when pruning structured components. Transformers contain tightly coupled operations: query, key, value, and output projections share hidden dimensions; residual connections assume constant hidden sizes; and feed-forward layers depend on consistent up- and down-projection sizes. While importance-aware pruning[47] identifies neurons or channels with low contribution to model outputs, such criteria alone are insufficient in Transformer architectures due to tightly coupled parameter groups which can be seen in Figure 4.3 in section of layer importance. In particular, attention projections and feed-forward components share structural dependencies that must be respected during pruning.

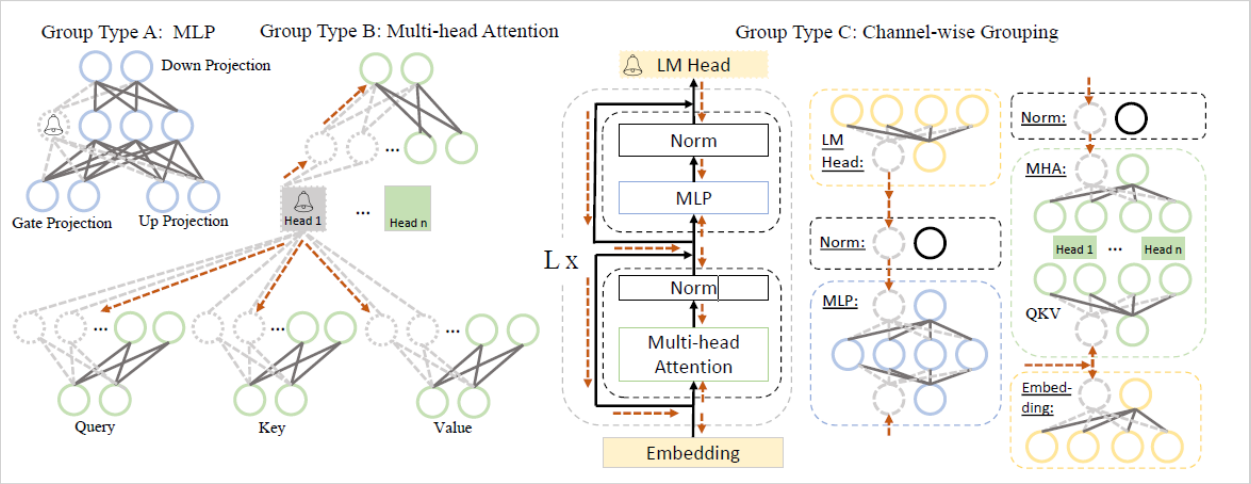


Figure 4.7 - *Illustration of the coupled structures [35]*

As illustrated in the above Figure , A group can be triggered by a variety of trigger neurons. Taking Group Type B as an example, the trigger for this group involves (i) the attention head, (ii) the output neuron in Query, Key or Value, and (iii) the input neuron in the final output projection. pruning a single neuron necessitates the coordinated removal of all dependent parameters to maintain shape and execution correctness. This dependency-aware formulation, introduced in LLM-Pruner [35], ensures that importance-based pruning does not introduce invalid model states.Several frameworks formalize dependency handling:

- **Dependency graph tracing:** LLM-Pruner uses a computational graph to detect coupled structures and group them for joint pruning[36]. This ensures that when an attention head or FFN neuron is pruned, all dependent parameters are removed consistently.
- **Cholesky-based grouping:** SlimGPT clusters neurons into groups that minimize second-order error via approximate Hessian inversion. Group sizes are chosen dynamically to reduce error accumulation.
- **Dimension-independent grouping:** DISP-LLM allows each layer to adopt a different width; dependency propagation ensures that output dimensions match subsequent input dimensions despite varying widths.
- **Two-stage coordination:** 2SSP first prunes width within FFNs and then prunes depth at attention levels, avoiding simultaneous pruning across dependent components. In practice, dependency-aware pruning requires careful propagation of mask decisions. Removing a head in the self-attention module implies pruning corresponding rows and columns in the weight matrices of the attention output and subsequent residual pathways. Similarly, pruning an FFN channel affects both the up-projection weights and down-projection weights. In this thesis, dependency propagation is implemented manually for each model family, and failures arising from incomplete propagation (e.g., state-dict mismatches, shape errors) are documented in Chapter 7.

4.5 Parameter-Efficient Fine-Tuning (PEFT) as Post-Pruning Recovery

After pruning, models may suffer significant quality loss, particularly on reasoning tasks. Parameter-efficient fine-tuning (PEFT) techniques provide a lightweight recovery mechanism:

- **LoRA and QLoRA:** LoRA injects low-rank matrices into linear projections, allowing the pruned backbone to remain frozen while learning task-specific adaptations[16]. QLoRA combines LoRA with 4-bit quantization to reduce memory usage. LLM-Pruner uses LoRA to recover pruned models with only 50 k training samples and reports rapid recovery[15].
- **Fine-tuning last layers:** Lu *et al.* find that fine-tuning only the output head and last few layers after pruning the final 25 % of layers matches or exceeds the performance of LoRA-based recoveryarxiv.org. This suggests that small-scale fine-tuning of critical layers may be more effective than injecting low-rank adapters for some models.
- **Continued pretraining:** Shortened LLaMA shows that continued pretraining (unfreezing all layers) recovers performance better than LoRA when pruning ratios are high[33]. However, this approach requires more computational resources and is not considered PEFT.
- **Pruning-aware tuning:** Recent work (e.g., PAT) proposes jointly optimizing pruning masks and adaptation parameters, but these methods were not used in this thesis.

This thesis treats LoRA/QLoRA as optional recovery tools rather than default solutions. In the experiments, PEFT is applied selectively to examine whether small adaptation layers can

compensate for pruning-induced degradation. Recovery effectiveness is assessed per benchmark in Chapter 6.

4.6 Recovery Fine-Tuning Datasets

For post-pruning recovery fine-tuning, this thesis does not directly fine-tune on benchmark evaluation datasets such as BBH, MuSR, or MMLU-Pro. Instead, capability-aligned proxy datasets were selected to improve reasoning performance while avoiding benchmark contamination and evaluation leakage.The recovery fine-tuning pipeline combines mathematical reasoning, instruction-following, multi-hop reasoning, and domain question-answering datasets to construct a multitask reasoning supervision corpus for LoRA adaptation.The datasets used in the recovery fine-tuning framework are summarized below.

4.6.1 GSM8K

GSM8K introduced by Cobbe et al. [91] was used for arithmetic and step-by-step mathematical reasoning supervision. The dataset contains grade-school mathematical word problems requiring multi-step numerical reasoning and explicit intermediate derivations. In the recovery pipeline, GSM8K was primarily used to improve arithmetic reasoning, symbolic manipulation, chain-of-thought generation, mathematical decomposition capability.

4.6.2 MATH Dataset

The MATH dataset introduced by Hendrycks et al. [92] was used for competition-style mathematical reasoning supervision. The dataset contains advanced algebra, geometry, probability, precalculus, and number theory problems with detailed solutions.The dataset was particularly important for evaluating whether post-pruning LoRA recovery could restore hierarchical reasoning depth and compositional symbolic reasoning. The following MATH categories were used: Algebra, Geometry,Counting and Probability, Intermediate Algebra, Prealgebra, Precalculus.

4.6.3 MetaMathQA

MetaMathQA introduced by Yu et al. [93] was used as a synthetic mathematical reasoning dataset for large-scale chain-of-thought supervision. The dataset contains teacher-generated reasoning traces designed to improve mathematical problem-solving capability in language models.Within this thesis, MetaMathQA was used to provide synthetic reasoning supervision, long-form derivations, mathematical chain-of-thought recovery, symbolic reasoning continuation behavior

4.6.4 HotpotQA

HotpotQA introduced by Yang et al. [94] was used for multi-hop reasoning and contextual reasoning supervision. The dataset requires combining evidence from multiple documents to answer compositional questions. HotpotQA was selected because structured pruning often degrades long-context dependency tracking. multi-step reasoning consistency, compositional inference behavior.The dataset was therefore useful for evaluating recovery of latent reasoning pathways after pruning.

4.6.5 StrategyQA

StrategyQA introduced by Geva et al. [95] was used for commonsense reasoning and implicit multi-step inference supervision. The dataset contains yes/no questions requiring latent reasoning rather than direct factual retrieval. StrategyQA was used to improve commonsense reasoning, implicit inference, compositional logical reasoning, latent knowledge integration.

4.6.6 Natural Instructions

Natural Instructions introduced by Mishra et al. [96] was optionally included as a generalized instruction-following dataset. The dataset contains diverse natural language tasks spanning reasoning, classification, extraction, transformation, and generation tasks. Within the recovery framework, Natural Instructions was used to improve instruction-following robustness, task generalization, reasoning diversity, multitask adaptation capability. Due to dataset size considerations, Natural Instructions was selectively enabled during some experiments.

4.6.7 AI2 ARC

The AI2 ARC dataset introduced by Clark et al. [97] was used for science-oriented multiple-choice reasoning tasks. ARC contains elementary and challenge-level science questions designed to test reasoning rather than simple retrieval. ARC was used to support recovery of: science-domain reasoning, multiple-choice reasoning, factual integration, domain generalization behaviour.

The recovery framework dynamically constructs capability-aligned dataset mixtures depending on the target benchmark being evaluated. For BBH-oriented recovery experiments, Natural Instructions, StrategyQA, and HotpotQA were used to improve logical reasoning, multi-hop inference, and instruction-following capabilities. For MATH Hard–oriented recovery, GSM8K, MATH, and MetaMathQA were used to strengthen arithmetic reasoning, symbolic manipulation, and chain-of-thought mathematical problem solving. MuSR-oriented recovery experiments used HotpotQA, StrategyQA, and Natural Instructions to improve compositional reasoning and latent multi-step inference behavior. For MMLU-Pro–oriented recovery, Natural Instructions, AI2 ARC, and StrategyQA were used to support broad-domain reasoning, science-oriented question answering, and professional knowledge tasks. In multitask recovery experiments, all datasets,

including GSM8K, MATH, MetaMathQA, HotpotQA, StrategyQA, Natural Instructions, and AI2 ARC, were combined into a unified reasoning supervision corpus. This capability-aligned recovery strategy enables the framework to improve reasoning robustness after pruning while minimizing direct benchmark memorization and evaluation leakage effects.

4.7 Knowledge Distillation for Post-Pruning & LORA Recovery

Structured pruning alters the internal representation capacity of large language models by permanently removing layers, attention heads, or feed-forward channels, often resulting in a non-recoverable loss of performance when pruning ratios become aggressive. Knowledge distillation (KD) has therefore emerged as an effective post-pruning recovery mechanism, where a pruned *student* model is trained to match the output distribution of an unpruned *teacher* model. Instead of relying solely on ground-truth supervision, the student minimizes divergence between its logits and those of the teacher, enabling it to inherit high-level linguistic and reasoning behaviors that may have been disrupted by pruning.

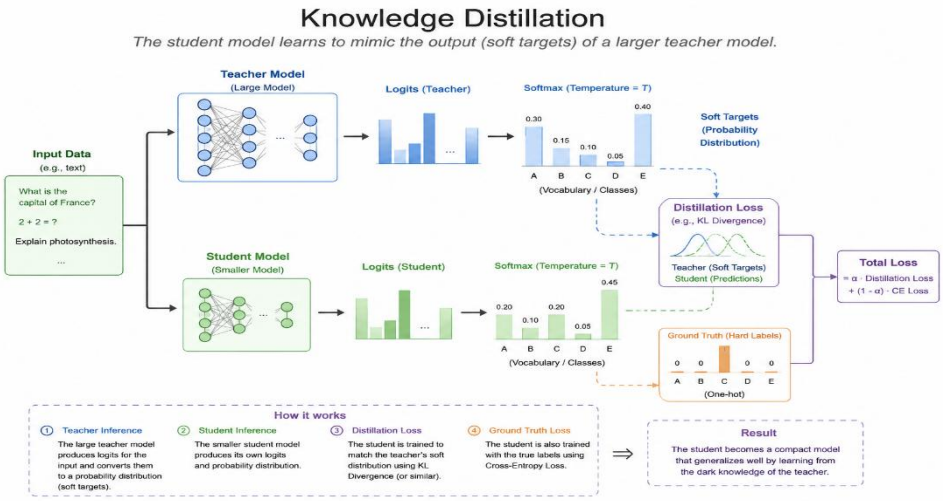


Figure 4.8 - *Knowledge distillation trained student and teacher model outputs [31].*

Recent work demonstrates that KD is particularly effective when applied **after structured pruning**, as it compensates for the representational collapse introduced by depth and width reduction. Muralidharan *et al.* show that combining structured pruning with KD yields compact language models that retain competitive zero-shot and downstream task performance, significantly outperforming pruning-only baselines at equivalent parameter counts [47]. Similarly, LLM-Pruner incorporates KD and fine-tuning stages to stabilize pruned models, emphasizing that dependency-aware pruning alone is insufficient to preserve model quality without post-training recovery [35].

Comparative studies on depth-pruned LLaMA models further indicate that KD becomes increasingly important as pruning severity increases, often surpassing parameter-efficient fine-tuning methods such as LoRA when large portions of the network are removed [3]. These findings

are reinforced by broader analyses showing that while LoRA adapts a limited number of parameters, KD provides **global distributional guidance**, making it more robust to structural damage introduced by pruning [33].

In this thesis, knowledge distillation is therefore positioned as a **post-pruning recovery strategy**, applied after depth or width pruning to mitigate degradation in perplexity and reasoning benchmarks. The methodology follows established practices in literature and does not introduce novel distillation objectives; rather, KD is used to restore functional alignment between pruned student models and their unpruned teachers.

4.8 Pruning Objectives, Constraints, and Deprioritized Directions

The overarching goal of pruning in this thesis is to **reduce model size and inference cost** while **preserving functional correctness and downstream performance**. Practical constraints include:

- **Hardware and runtime constraints:** The experiments were conducted on GPU systems with limited memory, making aggressive width pruning riskier due to memory fragmentation and potential overhead from irregular shapes. Depth pruning provides immediate latency gains but can destabilize performance.
- **Software ecosystem limitations:** Current frameworks (PyTorch, Hugging Face Transformers, vLLM) are optimized for dense, unmodified architectures. Pruned models may require custom loading logic or manual configuration adjustments, as observed in Chapter 7.
- **Evaluation limitations:** Benchmarks such as BBH, MMLU-Pro, and GPQA are sensitive to prompt formatting and sequence lengths. Pruning can alter hidden-state dynamics and degrade performance even when perplexity remains stable. The evaluation framework (Chapter 5) therefore combines NLL, perplexity, and reasoning tasks to capture degradation comprehensively.
- **Goal reprioritization:** As recorded in the experimental log, initial plans to export pruned models to ONNX and optimize with TensorRT were deprioritized after a project scope change. The thesis thus focuses on methodological understanding and evaluation rather than deployment optimization.

Finally, structured pruning must balance **efficiency and robustness**. Depth pruning offers straightforward reductions in compute but is semantically risky; width pruning requires careful dependency propagation but affords finer-grained control. PEFT can mitigate some losses, but its effectiveness varies across models. These trade-offs are revisited in the discussion and limitation chapters.

Chapter 5: Evaluation Framework

Robust evaluation is crucial to assessing the impact of structured pruning on model quality and efficiency. This chapter describes the metrics, protocols, benchmarks, and tooling used to evaluate pruned models, while situating these choices within broader evaluation literature. No experimental results are interpreted here; rather, the purpose is to clarify how results are measured and why these metrics are appropriate. During supervised fine-tuning on chat-formatted sequences, labels were initialized from the input token IDs. Padding positions were explicitly excluded from loss computation by assigning a value of -100 wherever the attention mask was zero. This ensured that loss, negative log-likelihood, and perplexity calculations were computed only over valid (non-padding) tokens.

5.1 Evaluation Metrics: NLL, Perplexity, and Efficiency Measures

Negative log-likelihood (NLL) and **perplexity (PPL)** are the primary intrinsic metrics for language modeling. For a token sequence *x1:Tx_{1:T}*x1:T , NLL is defined as the sum of the negative log-probabilities assigned by the model to each token. NLL is additive across tokens and layers, making it suitable for layer-wise ablation and pruning analysis[4]. **Perplexity** is the exponential of average NLL and reflects the effective branching factor of the model's predictive distribution. While widely reported, perplexity can be numerically unstable and less sensitive to small changes, so it is used mainly for static baseline reporting[4].

In addition to intrinsic metrics, **efficiency-oriented measures** are tracked whenever feasible:

- **Throughput** (tokens per second) measures inference speed.
- **Latency** captures end-to-end response time.
- **Memory usage** monitors GPU memory consumption.

These efficiency metrics provide insight into practical performance improvements from pruning.

For downstream tasks, the evaluation harness supports a suite of task-specific metrics beyond perplexity. NVIDIA's LM Evaluation Harness documentation lists metrics such as F1 score, Matthew's correlation coefficient, BLEU and CHRF scores for translation quality, TER (Translation Edit Rate), prompt-level strict accuracy for instruction following, and pass@1 for code-generation tasks. These metrics enable the evaluation of classification, QA, translation, instruction-following, and code-generation tasks under a unified framework. The documentation explicitly notes that lower perplexity, lower translation error rates, and higher F1/BLEU scores indicate better performance, while clarifying that not all metrics apply to every task. For our experiments, we select metrics relevant to each benchmark (see Sections 5.3 and 5.4). To evaluate structural information loss immediately following a layer-drop or width-reduction alteration, the

evaluation engine implemented in this framework rejects unconstrained open-domain token scraping, shifting instead to a localized task-continuation likelihood profile. This profiling layer measures structural stability using average Negative Log-Likelihood (avg NLL) and specialized continuation perplexity computed over precise token windows, specifically enforcing a full-stride context length of 128 tokens alongside an overlapping half-stride context of 64 tokens to expose local sequence coherence anomalies. To ensure that verification metrics isolate generative degradation without being artificially skewed by the static syntax of instruction templates, an assistant-only loss masking protocol is explicitly executed.

This mechanism clones the target sequence tensors and forces an optimization mask by resetting all indices corresponding to the prompt sequence length to a cross-entropy null value of -100, thereby disabling gradient calculations on user instructions and forcing the underlying loss function to evaluate performance solely on the model's generated response tokens. Beyond immediate prompt-level validation, macro-scale capability tracking is isolated from the core pipeline by utilizing python sub-processes to trigger the open-source lm-evaluation-harness library. This decoupled evaluation architecture programmatically runs benchmark configurations across four distinct assessment tracks—MMLU-Pro, Big-Bench Hard (BBH), MATH Hard, and MuSR—while maintaining high-throughput parallel execution by dynamically routing the inference workloads through either standard Hugging Face (hf) causal loops or an accelerated vllm engine utilizing paged attention primitives on local high-performance cluster nodes.

5.2 Forced-Continuation Evaluation Protocol

To ensure comparability across pruning configurations, we adopt a **forced continuation evaluation protocol**. Models are conditioned on a fixed context (e.g., a prompt or document) and required to predict a continuation of a specified length without sampling. Log-likelihood is accumulated over the continuation segment. This approach avoids the confounding effects of free-form generation and sampling variability, isolating the impact of structural pruning on token-level predictive quality. Where sequence length and stride parameters are provided in the experiment logs, they are preserved exactly. For example, the forced continuation evaluation of WikiText-2 uses a sliding window with sequence length *1024* and stride *1024* (full) or *512* (half), ensuring that the model processes the entire test split.

This protocol parallels standard practices in language modeling evaluation and is essential for comparing pruned and unpruned models. It also aligns with the theoretical preference for NLL over perplexity as a pruning-sensitive metric.

5.3 Benchmark Suite Framing

Beyond intrinsic metrics, evaluation includes **benchmark tasks** that probe reasoning, knowledge, and multimodal understanding. Each benchmark offers a different perspective on model capability:

- **Big-Bench Hard (BBH):** Derived from the BIG-bench benchmark, BBH focuses on challenging reasoning tasks designed to expose failures in otherwise strong models. Tasks include logical deduction, analogical reasoning, and algorithmic puzzles.
- **Massive Multitask Language Understanding (MMLU-Pro):** An extended version of MMLU that tests domain knowledge and reasoning across professional and academic subjects. Hendrycks *et al.* propose this benchmark to evaluate models on 57 tasks ranging from mathematics to law and history. Their study notes that even large GPT-3 models attain near-random accuracy on many tasks, underscoring the difficulty of comprehensive evaluation.
- **MuSR:** A multi-step symbolic reasoning benchmark requiring compositional reasoning and arithmetic precision.
- **Math Hard:** A subset of mathematically challenging tasks used to stress-test arithmetic and problem-solving capabilities.
- **EuroSAT / TerraMind evaluations:** For remote-sensing models, evaluation includes land-cover classification metrics (e.g., accuracy, F1 score) on EuroSAT. These metrics measure spatial feature fidelity and domain-specific performance.

These benchmarks complement intrinsic metrics by testing generalization and reasoning. In this thesis, benchmarks are selected according to the models' domains (text-only, multimodal, remote sensing) and availability in the experimental logs. For each benchmark, we report the metric recommended by the dataset authors (e.g., accuracy, exact match) to ensure comparability.

5.4 Evaluation Tooling and Harnesses

All benchmark evaluations in this thesis leverages standardized tooling. The **lm-evaluation-harness** provides a modular interface to load tasks, format prompts, compute metrics, and aggregate results across seeds and prompts. It supports over 60 benchmarks, including MMLU, GPQA, and GSM8K (math reasoning). The harness defines default prompt templates and evaluation scripts, reducing variance across models and runs. NVIDIA's NeMo microservices replicate similar functionality and document supported metrics and tasks, offering an alternative environment for evaluation.

The evaluation workflow shown in Figure 5.1 represents the standardized pipeline used throughout this thesis. Each model checkpoint, whether baseline, pruned, or recovery-fine-tuned, is passed through a consistent evaluation interface implemented via the lm-evaluation-harness. The harness handles prompt construction, benchmark loading, and metric aggregation, ensuring that differences in performance arise from the model modifications rather than evaluation inconsistencies. This pipeline enables systematic comparison across the three experimental stages used in this work: baseline evaluation of pretrained models, evaluation of structurally pruned

models, and evaluation of pruned models after parameter-efficient recovery techniques such as LoRA.

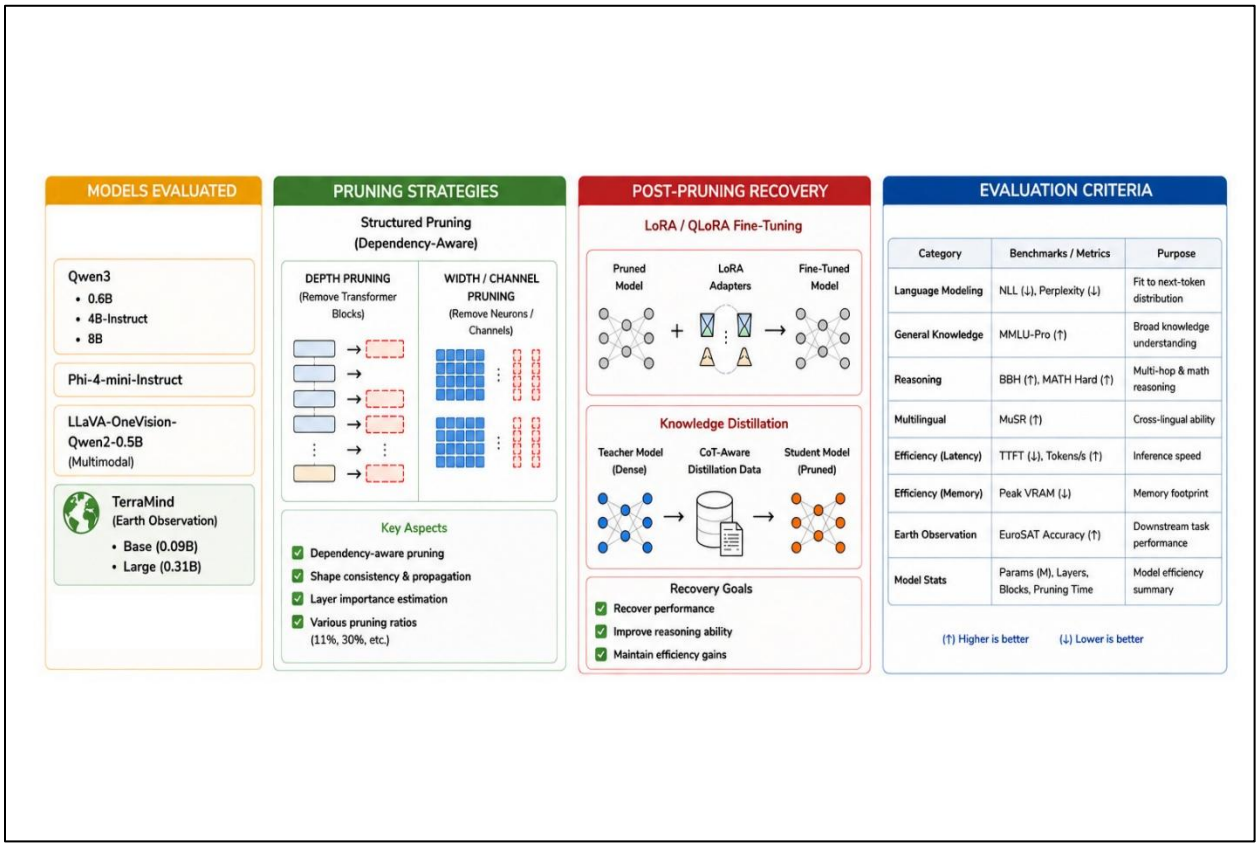


Figure 5.1 – *Conceptual Evaluation workflow using lm-evaluation-harness [17] [41].*

We also reference the Holistic Evaluation of Language Models (HELM) framework, which advocates a multi-metric approach spanning accuracy, calibration, robustness, fairness, bias, toxicity, and efficiency. Although our experiments do not encompass all HELM metrics, its emphasis on comprehensive evaluation underscores the need to look beyond single metrics like perplexity. HELM performs large-scale evaluations across 42 scenarios and demonstrates that prior evaluations covered only 17.9 % of these scenarios. This motivates our inclusion of diverse benchmarks and metrics.

5.5 Experimental Progression and Decision Rationale

This thesis followed an iterative and evidence-driven experimental progression rather than a fixed, exhaustive evaluation plan across all models and pruning strategies. Early experiments were intentionally broad, with subsequent stages refined based on observed behavior, stability, and benchmarking outcomes. This section documents the rationale behind these decisions to ensure transparency and reproducibility.

The initial phase focused on assessing whether structured pruning is suitable for **multimodal architectures**. Experiments were conducted on **LLaVA-OneVision Qwen2-0.5B**, evaluating depth-pruned variants using standard benchmark suites. These results indicated that pruning in vision–language models introduces additional cross-modal sensitivity and instability, with benchmarking outcomes showing inconsistent or degraded performance relative to text-only models. Given the added architectural complexity of multimodal fusion layers and the limited scope of this thesis, further pruning experiments on LLaVA-OneVision were deprioritized. The model was retained primarily for contextual analysis rather than extensive pruning exploration.

In parallel, **Qwen3-0.6B** was used as a **trial-scale model** to probe the basic behavior of structured pruning under controlled conditions. Both prompt-based forced-continuation evaluation and WikiText-2 perplexity evaluation were applied to depth-pruned variants. These experiments revealed that, at this scale, pruning effects were largely saturated: prompt-based and dataset-based evaluations showed minimal differentiation across pruning configurations. As a result, Qwen3-0.6B served as an exploratory baseline to validate tooling, pruning scripts, and evaluation protocols, but was not treated as a primary subject for extensive benchmarking or recovery analysis.

Based on these findings, the experimental focus shifted to **Qwen3-8B (base)** as a representative large-scale dense language model. For this model, both **depth pruning and width/channel pruning** were systematically explored. Post-pruning recovery techniques, including **LoRA fine-tuning and knowledge distillation**, were applied to depth- and width-pruned checkpoints under prompt-based evaluation. While intrinsic metrics and prompt-level behavior showed partial recovery, downstream reasoning benchmarks demonstrated limited gains, particularly for width-pruned variants. These results suggested that, for the base model, aggressive width pruning introduces representational degradation that is not easily mitigated by parameter-efficient recovery alone.

To assess whether instruction tuning alters pruning robustness, the study then transitioned to **Qwen3-4B-Instruct**. Based on the observations from the 8B base model, experiments were restricted to **depth pruning only**, as width pruning had already demonstrated unfavorable trade-offs. Depth-pruned instruct models exhibited more stable benchmarking behavior and clearer performance–efficiency trade-offs, justifying the narrowed experimental scope.

Finally, the same depth-pruning-only strategy was applied to **Phi-4-Mini-Instruct**, enabling cross-model comparison among instruction-tuned architectures. This progression ensured that later experiments were informed by earlier results, avoided redundant evaluations, and concentrated effort on pruning configurations that demonstrated meaningful and interpretable behaviour.

The growing execution overhead of width pruning at larger scales, documented in Section 6.9.5, further reinforced the decision to restrict width pruning experiments in later stages of the study, as iterative ablation under such costs would have been impractical within the available compute budget.

Overall, this staged experimental design reflects a deliberate narrowing from broad feasibility assessment to focused analysis. Decisions to exclude certain pruning strategies or models were driven by empirical evidence rather than convenience, and all reported results correspond to configurations that were evaluated meaningfully within the constraints of stability, interpretability, and available benchmarking support.

5.6 Practical Challenges in Structured Pruning Implementation

The implementation of structured pruning for large language and multimodal models introduced several practical engineering challenges beyond the theoretical pruning algorithms themselves. Many of these issues arose from the strong architectural coupling present in modern Transformer implementations and from inconsistencies between internal model configurations and serialized checkpoint structures.

One major challenge involved maintaining configuration consistency after depth pruning. Removing Transformer blocks required synchronized updates across multiple nested configuration objects, including \texttt{num_hidden_layers} fields stored within the global model configuration, language-model configuration, and modality-specific text configurations. Failure to propagate these updates consistently frequently caused checkpoint loading failures and validation mismatches during \texttt{save_pretrained} and \texttt{from_pretrained} operations.

Additional issues emerged from layer-index dependencies embedded inside attention modules. Several architectures internally store persistent \texttt{layer_idx} values that are assumed to remain contiguous across the Transformer stack. After pruning, these indices required explicit relabeling to maintain correct autoregressive attention behaviour. Without relabeling, generation failures and inconsistent KV-cache behaviour were frequently observed.

Qwen-based architectures also required synchronization of auxiliary configuration structures such as \texttt{layer_types}. When the number of Transformer blocks was reduced without simultaneously truncating these metadata lists, strict validation mechanisms produced runtime configuration errors during model loading and evaluation.

Width pruning introduced additional dependency-management complexity because feed-forward and projection layers must preserve compatible tensor dimensions across gate, up, and down projections. Improper synchronization of channel removal across these coupled layers frequently resulted in tensor shape mismatches and invalid matrix multiplications during inference.

These observations demonstrate that practical structured pruning requires significantly more than removing parameters alone. Successful deployment-oriented pruning pipelines must preserve architectural consistency across configuration metadata, serialization formats, attention indexing systems, and tensor dependency structures. The experiments therefore highlight the importance of dependency-aware pruning not only as a theoretical concept but also as a critical engineering requirement for stable Transformer deployment.

Chapter 6: Experimental Results

This chapter reports the experimental outcomes of structured pruning across the evaluated models. In accordance with good experimental practice, this chapter reports and briefly contextualizes results analysis and discussion are deferred to later chapters.

6.1 Evaluation Scope and Prompt Setup

Unless explicitly stated as **WikiText-2 (WT2)** evaluation, all results reported in this chapter are derived from a **fixed prompt-based forced continuation setup**, as defined in Chapter 5. The prompt configuration is held constant across all baseline and pruned models to ensure comparability.

Fixed 0-shot probe prompt:

"Describe the city of Stuttgart in a few sentences."

Evaluation parameters:

- Continuation length: **300 tokens**
- n-shot setting: **0-shot**
- Random seed: **123**
  (applied consistently across Python, NumPy, and PyTorch on both CPU and GPU)

For **WikiText-2 (WT2)** evaluations, the complete test split is used:

- Dataset: wikitext/wikitext-2-raw-v1
- Split: test

Sliding-window evaluation is applied with the following parameters:

- Sequence length (seq_len): **1024**
- Full stride (stride_full): **1024**
- Half stride (stride_half): **512**

These configurations follow standard language-model evaluation practices and are chosen to balance coverage and computational efficiency.

6.2 Metric Rationale: NLL vs Perplexity

The experimental logs explicitly motivate the use of **Negative Log-Likelihood (NLL)** as the primary metric for pruning analysis, rather than perplexity. The rationale is summarized as follows:

- **NLL is additive and numerically stable**, making it suitable for layer-wise and token-level aggregation.
- **Perplexity (PPL = exp(NLL)) is exponentially sensitive**, which limits its reliability for importance estimation and pruning control.
- NLL supports layer-importance scoring, enabling consistent pruning decisions across depth and width dimensions. Task Likelihood (TL) corresponds directly to NLL computed over answer tokens in forced-continuation settings. Perplexity is retained only for static baseline reporting, not for pruning decisions or optimization loops. Accordingly, all pruning decisions and importance analyses in this thesis are based on NLL, while perplexity is reported selectively for completeness and comparison with prior work.

6.3 LoRA-based recovery configuration

For all pruned language models evaluated in this work, post-pruning recovery was performed using Low-Rank Adaptation (LoRA) applied directly to the pruned base checkpoints. Pruned model weights were loaded from their corresponding depth-pruned directories and used as the initialization point for LoRA fine-tuning, without restoring any removed parameters. LoRA adapters were configured with the following hyperparameters:

- Rank (**r**): **16**
- Scaling factor (**lora_alpha**): **32**
- Dropout (**lora_dropout**): **0.05**
- Bias parameters: **disabled** (bias = none)

Adaptation was applied to both attention and feed-forward components, specifically the projection layers:

- **Attention projections**: q_proj, k_proj, v_proj, o_proj
- **Feed-forward projections**: gate_proj, up_proj, down_proj

Fine-tuning was conducted with a maximum sequence length of 2048, a batch size of 1, and gradient accumulation over 16 steps. Optimization employed the AdamW optimizer (adamw_torch) with a cosine learning-rate schedule, a weight decay of 0.01, and a **warmup ratio of 0.03`. The base learning rate was set to 2×10^{-4}.Training was executed in BF16 precision on GPU. To avoid incompatibilities with gradient checkpointing and unnecessary memory overhead, key–value caching (use_cache) was explicitly disabled during training.

6.4 LLaVA-OneVision-0.5B: Results

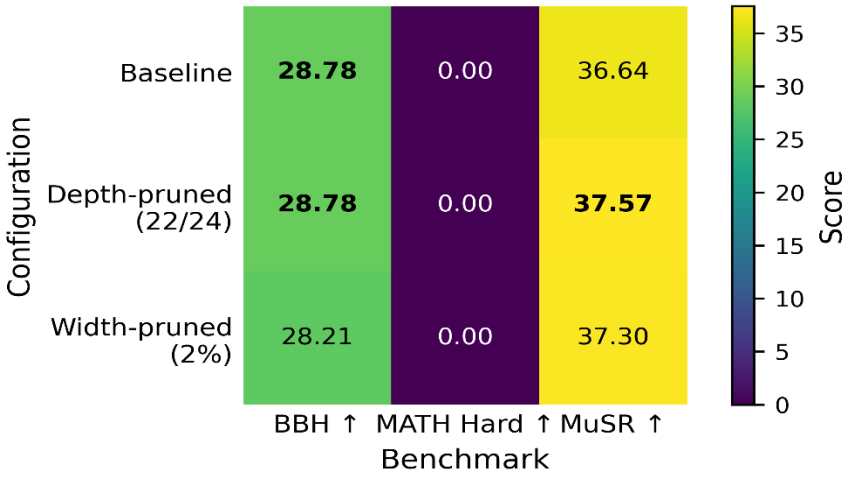


Figure 6.1 - *Benchmark performance of LLaVA-OneVision-0.5B under structured pruning. The heatmap reports BBH, MATH Hard, and MuSR scores for the baseline model, a depth-pruned variant (22/24 layers), and a width-pruned variant (2%). Scores are shown using prompt-based evaluation, with higher values indicating better performance.*

Figure 6.1 presents the impact of structured pruning on the multimodal LLaVA-OneVision-0.5B model across three reasoning benchmarks. The comparison includes the dense baseline, a depth-pruned configuration where two transformer blocks are removed, and a width-pruned configuration with 2% channel reduction.

Across all configurations, BBH performance remains largely unchanged, suggesting that shallow reasoning tasks are relatively insensitive to modest structural changes in this small multimodal model. In contrast, MATH Hard scores remain at zero for all variants, indicating that neither pruning strategy improves mathematical reasoning at this scale and that the model's limitations dominate over pruning effects.

For MuSR, both depth and width pruning result in marginal score increases relative to the baseline. However, these improvements are small and inconsistent, and do not establish a clear advantage for either pruning strategy. The absence of strong performance differentiation suggests that, at this model scale, pruning effects are largely saturated and overshadowed by multimodal fusion complexity.These observations motivated the decision to deprioritize extensive pruning exploration on LLaVA-OneVision models. The results indicate that multimodal architectures exhibit higher sensitivity and limited interpretability under pruning, leading the study to focus subsequent analysis on text-only language models where pruning effects are more stable and analytically meaningful.

6.5 Qwen3-0.6B: Results

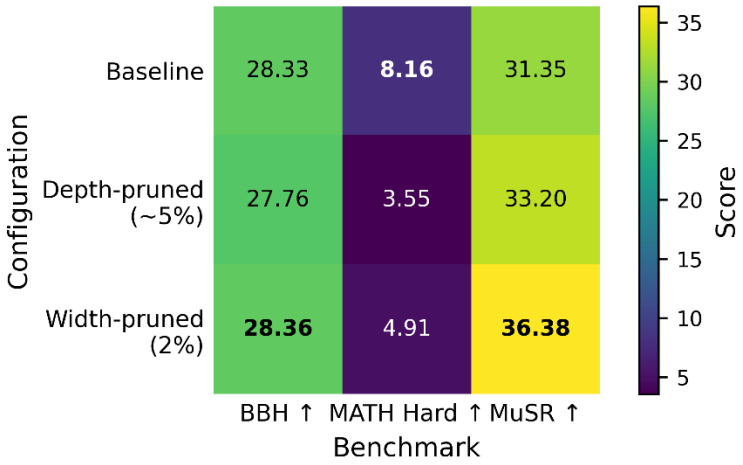


Figure 6.2 - *Benchmark performance of Qwen3-0.6B under structured pruning. The heatmap reports BBH, MATH Hard, and MuSR scores for the dense baseline, a shallow depth-pruned configuration (~5% layer removal), and a width-pruned configuration (2% channel pruning). All results are obtained using prompt-based evaluation.*

Figure 6.2 shows the effect of light structured pruning on Qwen3-0.6B across three reasoning benchmarks. This model serves as a trial-scale language model used to probe pruning behavior under controlled and relatively low-risk settings.Depth pruning at approximately 5% results in a modest reduction in BBH and MATH Hard performance, while slightly improving MuSR scores. In contrast, width pruning at 2% preserves BBH performance, yields higher MATH Hard scores than depth pruning, and produces the strongest MuSR performance among the evaluated configurations. However, the absolute differences across configurations remain small.

The limited separation between baseline, depth-pruned, and width-pruned variants indicates that pruning effects at this scale are largely saturated. Both pruning strategies introduce minor perturbations but do not lead to systematic improvements or degradations across benchmarks. As a result, Qwen3-0.6B primarily functions as a validation model for pruning tooling, evaluation pipelines, and experimental protocols rather than as a target for extensive recovery or optimization analysis.

These findings motivated the transition to larger-scale models, where structured pruning is expected to produce clearer performance–efficiency trade-offs and more interpretable failure modes.

6.6 Qwen3-8B: Results with Recovery.

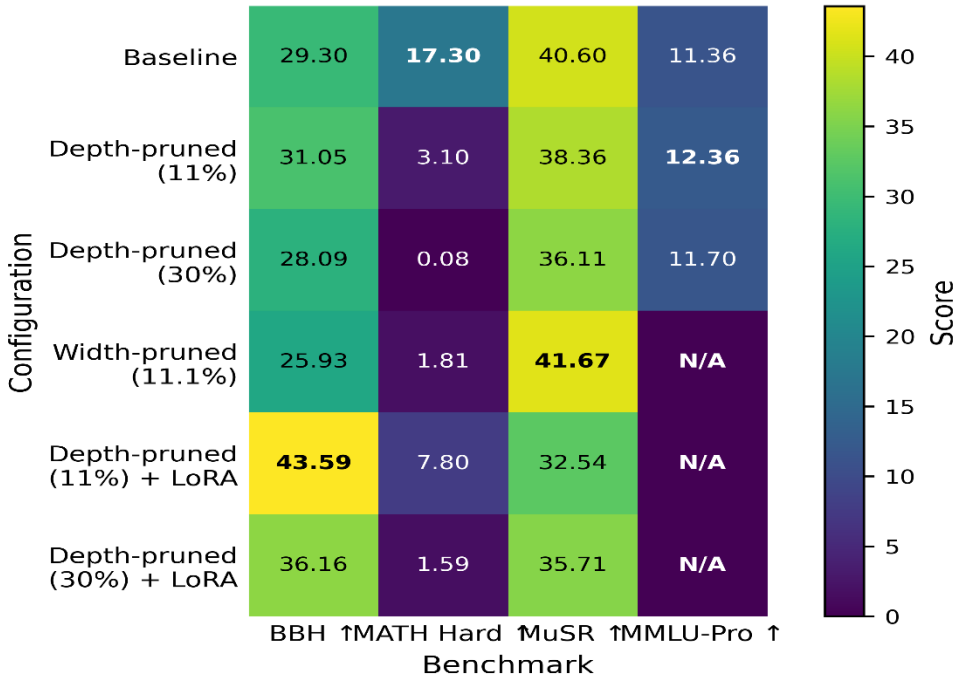


Figure 6.3 - *Benchmark performance of Qwen3-8B under structured pruning and recovery strategies. The heatmap reports BBH, MATH Hard, MuSR, and MMLU-Pro.*

The figure presents the benchmark behaviour of Qwen3-8B under multiple structured pruning configurations and post-pruning recovery strategies. Moderate depth pruning (11%) preserves overall reasoning capability relatively well and slightly improves BBH and MMLU-Pro performance compared to the dense baseline, although MATH Hard accuracy decreases substantially. More aggressive depth pruning (30%) causes broader degradation across all benchmarks, particularly for mathematical reasoning tasks, where performance collapses almost entirely.Width pruning at 11.1% produces mixed effects across tasks. While BBH and MATH Hard scores decline noticeably, MuSR performance slightly improves relative to the dense baseline, indicating that some reasoning capabilities remain robust to moderate channel-level compression. However, the instability across benchmarks suggests that width pruning alters representational capacity less predictably than depth pruning.Post-pruning recovery using LoRA fine-tuning substantially improves performance after depth pruning. The 11% depth-pruned model with LoRA recovery achieves the strongest BBH score among all evaluated configurations, demonstrating that parameter-efficient adaptation can effectively restore portions of lost reasoning capability after moderate structural compression. However, mathematical reasoning performance remains significantly below the original dense baseline, indicating that some capabilities are more difficult to recover after pruning. The 30% depth-pruned model with LoRA recovery also improves relative to its unrecovered counterpart, although recovery becomes more limited under aggressive compression levels.

6.7 Qwen3-4B-Instruct: Depth Pruning and Recovery

Figure 6.4 evaluates the robustness of an instruction-tuned language model, Qwen3-4B-Instruct, under structured depth pruning and subsequent recovery. Compared to base models, the instruction-tuned baseline exhibits substantially stronger performance across all benchmarks, particularly on MATH Hard, indicating improved reasoning capabilities prior to pruning.

Applying 30% depth pruning without recovery results in significant degradation across BBH and MATH Hard, while MuSR performance remains comparatively stable. This suggests that while instruction tuning improves baseline reasoning, aggressive structural removal still disrupts core capabilities when applied without adaptation.

Post-pruning recovery using LoRA fine-tuning substantially mitigates these effects. Fine-tuning on all tasks restores BBH performance and leads to a pronounced improvement in MuSR, exceeding baseline performance by a large margin. LoRA fine-tuning limited to BBH also provides partial recovery, demonstrating that even task-specific adaptation can improve pruned model behaviour.

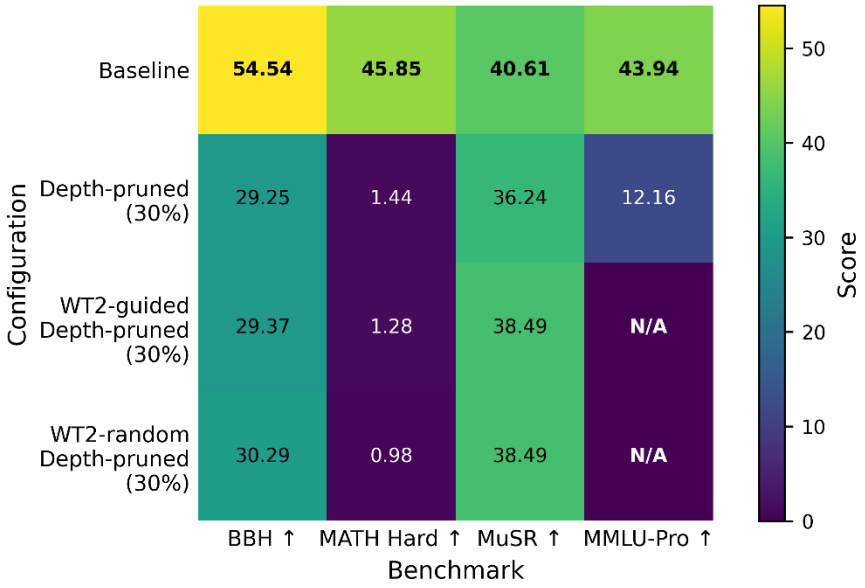


Figure 6.4 - *Benchmark performance of Qwen3-4B-Instruct under structured depth pruning and recovery strategies.The heatmap reports BBH, MATH Hard, and MuSR scores for the dense baseline, depth-pruned variants (30%), and post-pruning recovery using LoRA fine-tuning. Additional variants include depth pruning derived from WikiText-2 importance and random depth selection.*

The WT2-derived and random depth-pruned variants show similar performance trends, indicating that for instruction-tuned models, the choice of pruning criterion at this level has limited impact compared to the presence or absence of recovery. Notably, MATH Hard remains highly sensitive across all pruned configurations, reinforcing the observation that mathematical reasoning is the most fragile capability under structured pruning.

Overall, this figure demonstrates that instruction-tuned models exhibit improved recoverability after pruning compared to base models, particularly when combined with targeted fine-tuning. These findings support the hypothesis that instruction tuning enhances pruning robustness, while also highlighting the continued challenges associated with preserving mathematical reasoning under aggressive compression.

6.7.1 Mathematical Reasoning Fragility Under Structural Compression

Among all evaluated benchmarks, mathematical reasoning tasks exhibited the strongest sensitivity to structured pruning. In particular, MATH Hard performance degraded substantially under both depth and width pruning, especially for smaller models and higher pruning ratios.

This behaviour suggests that mathematical reasoning relies heavily on deep compositional representations distributed across multiple Transformer layers. Unlike simpler language understanding tasks, mathematical problem solving requires long-range dependency tracking, intermediate symbolic transformations, and multi-step reasoning chains that accumulate across sequential hidden representations. Removing Transformer layers or reducing feed-forward representational capacity can therefore disrupt critical intermediate reasoning pathways required for accurate mathematical inference. The results further indicate that mathematical reasoning is significantly more fragile under structural compression than general language modelling or commonsense reasoning benchmarks. While tasks such as BBH and MuSR retained partial reasoning capability after moderate pruning, MATH Hard frequently exhibited severe degradation or near-collapse under aggressive compression settings. This pattern was observed consistently across Qwen3 and multimodal LLaVA experiments.

LoRA-based recovery fine-tuning partially restored performance for moderate pruning ratios but struggled to recover mathematical reasoning when structural damage became substantial. This suggests that low-rank adaptation can refine existing latent representations but cannot fully reconstruct complex reasoning pathways once critical structural capacity has been removed.

These observations support the hypothesis that mathematical reasoning depends strongly on distributed hierarchical representations spanning multiple Transformer blocks. Consequently, mathematical benchmarks provide a particularly sensitive probe for evaluating the robustness of structured pruning strategies in reasoning-oriented language models.

6.8 Phi-4-mini-Instruct: Depth Pruning and Recoverability

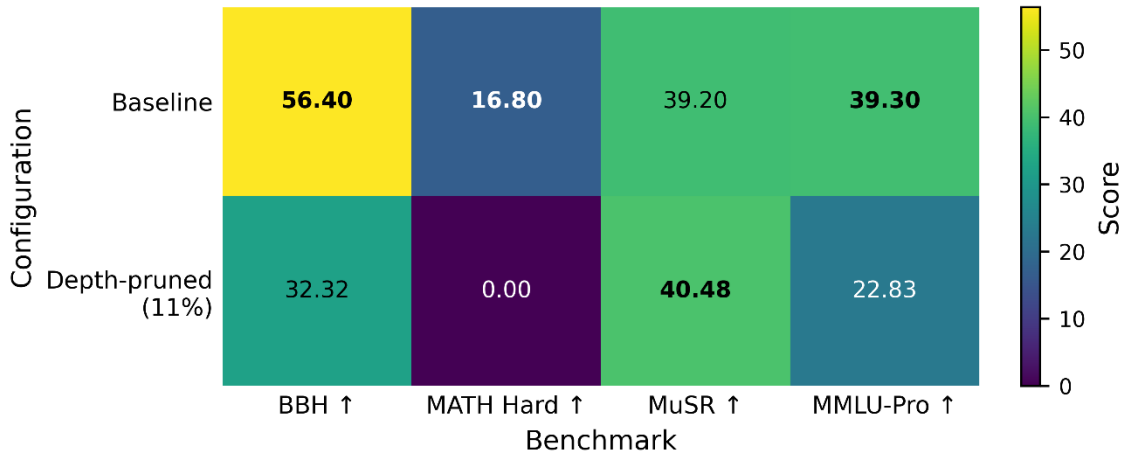


Figure 6.5 - *Benchmark performance of Phi-4-mini-Instruct under structured depth pruning and LoRA-based recovery. The heatmap reports BBH, MATH Hard, MuSR, and MMLU-Pro scores for the dense baseline and the depth-pruned configuration. Higher values indicate better benchmark performance.*

The figure evaluates the effect of structured depth pruning on Phi-4-mini-Instruct, a compact instruction-tuned language model, across multiple reasoning benchmarks. The dense baseline demonstrates strong overall performance, particularly on BBH and MMLU-Pro, while maintaining moderate capability on MATH Hard relative to larger models.

Applying moderate depth pruning results in substantial degradation in BBH and MATH Hard performance, while MuSR slightly improves relative to the baseline. This asymmetric behaviour suggests that some reasoning capabilities remain partially robust to structural layer removal, whereas mathematical reasoning is highly sensitive to pruning. The near-complete collapse in MATH Hard performance further reinforces the observation that mathematical reasoning tasks are among the most fragile capabilities under aggressive structural compression.

Despite the reduction in benchmark performance, the pruned Phi-4-mini-Instruct model continues to preserve reasonable performance on selected reasoning tasks while benefiting from reduced model complexity and improved deployment efficiency. The results are consistent with trends observed in larger instruction-tuned models, demonstrating that compact instruction-following models exhibit similar pruning sensitivity patterns, particularly for reasoning-intensive tasks.

6.9 Evaluation of Methodology and Inference Behaviour

6.9.1 Prompt-Based vs Teacher-Forced Evaluation

The experiments in this thesis employ two distinct evaluation methodologies: teacher-forced perplexity evaluation using WikiText-2 and prompt-based autoregressive generation for downstream reasoning benchmarks and inference measurements. Although both approaches evaluate the same underlying models, they stress different execution pathways and therefore produce different runtime characteristics. Teacher-forced evaluation computes loss over reference tokens by supplying the ground-truth continuation during inference. In this setting, the model processes tokens in parallel without iterative token generation, enabling efficient batch computation across the full sequence. Metrics such as negative log-likelihood (NLL) and perplexity (PPL) therefore primarily reflect representational quality and language modelling capability rather than interactive inference behaviour.

In contrast, prompt-based benchmark evaluation and inference measurements operate in autoregressive decoding mode, where tokens are generated sequentially. Each newly generated token depends on all previously generated tokens, requiring repeated forward passes through the Transformer stack and continuous maintenance of the KV-cache. Metrics such as Time-to-First-Token (TTFT) and tokens-per-second therefore measure deployment-oriented inference behaviour rather than language modelling accuracy alone.

Because these evaluation modes execute different computational pathways, slight variations in baseline throughput, latency, and memory measurements are expected across experiments. Teacher-forced evaluation emphasizes parallel token processing efficiency, while autoregressive generation exposes sequential decoding overheads, KV-cache initialization costs, and token-by-token dependency chains. Consequently, perplexity improvements do not necessarily translate directly into lower latency or higher generation throughput. This distinction is particularly important when evaluating structured pruning strategies. Depth pruning primarily reduces sequential execution depth during autoregressive decoding, whereas width pruning alters internal tensor dimensions and parameter counts without reducing the number of sequential Transformer executions. As a result, pruning effects may appear differently depending on whether the evaluation focuses on language modelling quality or real-time inference behaviour.

6.9.2 Width Pruning and GPU Kernel Efficiency

Although width pruning reduces parameter counts and intermediate tensor sizes, the experiments demonstrate that these reductions do not consistently translate into proportional inference speed improvements. In several configurations, width-pruned models exhibit only marginal throughput gains and occasionally increased latency relative to the dense baseline. This behaviour arises from the interaction between structured width pruning and GPU kernel optimization. Modern accelerators such as NVIDIA GPUs are heavily optimized for dense matrix operations with standardized tensor dimensions that align efficiently with Tensor Core execution patterns. Width pruning modifies hidden dimensions, attention projections, and feed-forward channel sizes, often producing irregular tensor shapes that reduce hardware utilization efficiency.

As a result, theoretical FLOP reductions achieved through width pruning may not correspond directly to practical latency improvements. Fragmented tensor dimensions can introduce inefficient memory-access patterns, reduced Tensor Core occupancy, and additional projection overheads within attention and feed-forward computations. Furthermore, width pruning does not reduce the sequential dependency chain of autoregressive decoding, since the same number of Transformer blocks must still be executed during generation. In contrast, depth pruning removes entire Transformer layers and therefore shortens the sequential computation graph directly. This reduction decreases both the number of forward passes required before token generation and the overall autoregressive decoding path length. Consequently, depth pruning consistently produces stronger improvements in TTFT and generation throughput across larger models. These

observations highlight that parameter count reduction alone is not a sufficient predictor of practical inference efficiency. The structural characteristics of pruning and their interaction with hardware execution patterns play a critical role in determining real-world deployment performance.

6.9.3 Depth Pruning and Sequential Decoding Latency

The experiments consistently show that depth pruning produces stronger improvements in Time-to-First Token (TTFT) and generation throughput compared to width pruning, particularly for larger language models. This behaviour is closely related to the sequential nature of autoregressive Transformer decoding. During autoregressive generation, each output token requires a complete forward pass through all Transformer layers before the next token can be produced. Consequently, inference latency is strongly influenced by the depth of the sequential computation graph. Removing entire Transformer blocks directly shortens this execution path and reduces the number of layer-wise computations required prior to token generation. This effect is particularly visible in TTFT measurements. The first generated token requires initialization of the autoregressive decoding process, including attention-state preparation, KV-cache initialization, and sequential traversal through the Transformer stack. Depth pruning reduces the number of sequential layer executions involved in this initialization phase, thereby lowering TTFT more consistently than width pruning.

In contrast, width pruning preserves the original Transformer depth and therefore maintains the same number of sequential decoding stages. Although individual matrix operations may become smaller after width reduction, the overall autoregressive dependency chain remains unchanged. As a result, width pruning provides weaker latency improvements despite reducing parameter countand intermediate tensor dimensions. The experimental results therefore indicate that sequential execution depth is one of the dominant bottlenecks in autoregressive Transformer inference. Structured depth pruning directly targets this bottleneck, making it more effective for practical latency reduction in deployment-oriented generation workloads.

6.9.4 Model Scale and Pruning Saturation

The pruning experiments reveal a strong relationship between model scale and pruning recoverability. Smaller models such as Qwen3-0.6B and LLaVA-OneVision-0.5B exhibit substantially greater performance degradation under aggressive structured pruning compared to

larger models such as Qwen3-4B and Qwen3-8B.This behaviour suggests that smaller language models operate with significantly lower representational redundancy. In compact architectures, a larger proportion of parameters contribute directly to essential reasoning and language modeling behaviour, leaving fewer redundant pathways that can be removed safely. Consequently, structured pruning removes critical representational capacity more rapidly in small models than in larger models.In contrast, larger models contain substantially higher redundancy across attention heads, feed-forward dimensions, and Transformer layers. This redundancy enables larger models to preserve reasoning capability despite moderate structural compression. The experiments further demonstrate that larger models respond more effectively to LoRA-based recovery fine-tuning after pruning, indicating the presence of richer latent representational subspaces that remain adaptable even after structural modifications.

The results therefore support the hypothesis that pruning effectiveness scales with model redundancy. As model size increases, structured pruning becomes increasingly capable of removing redundant computation while preserving downstream reasoning behaviour. Conversely, small models approach a pruning saturation point much earlier, where additional structural compression rapidly degrades benchmark performance and limits fine-tuning recoverability.This effect is especially visible in mathematical reasoning benchmarks such as MATH Hard, where compact models frequently collapse under aggressive pruning while larger models retain partial reasoning capability. The observations suggest that high-complexity reasoning tasks require substantial internal representational capacity and are particularly sensitive to structural compression in low-parameter regimes.

6.9.5 Pruning Algorithm Execution Time

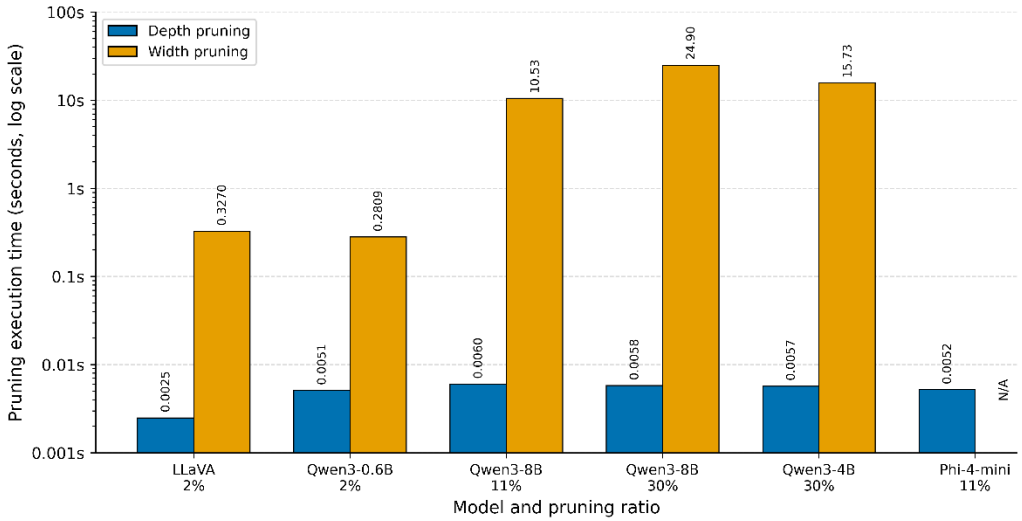


Figure 6.6 — *Pruning Algorithm Execution Time: Depth Pruning vs. Width Pruning Across Models and Compression Ratios. The bar chart reports pruning execution time in seconds (log scale) for depth pruning (blue) and width pruning (orange) across all evaluated model configurations*

Figure 6.6 reports the wall-clock execution time required to apply structured pruning across all evaluated model configurations, revealing a consistent and practically significant asymmetry between the two pruning strategies.

Depth pruning executes in under 10 milliseconds across all models and compression ratios, ranging from 0.0025 seconds for LLaVA-OneVision-0.5B to 0.0060 seconds for Qwen3-8B. This near-constant behaviour reflects the structural simplicity of depth pruning: removing a Transformer block requires only index-based slicing with no parameter-level computation, and execution time does not grow meaningfully with model scale despite parameter count differences exceeding 13×.Width pruning, by contrast, exhibits substantially higher and scale-sensitive execution times. Even at small scales, width pruning is already two orders of magnitude slower than depth pruning. At larger scales and higher compression ratios, this gap widens considerably — reaching 24.90 seconds for Qwen3-8B at 30% compression — driven by the cost of magnitude scoring, dependency graph traversal, and coordinated mask propagation across coupled projection matrices.

In iterative research workflows requiring repeated pruning and evaluation cycles, this asymmetry becomes practically significant. Depth pruning introduces negligible algorithmic overhead at any scale, making it considerably more suitable for rapid ablation studies. Width pruning was not applied to Phi-4-mini-Instruct and is reported as N/A.

6.10 Throughput Analysis (Tokens per Second)

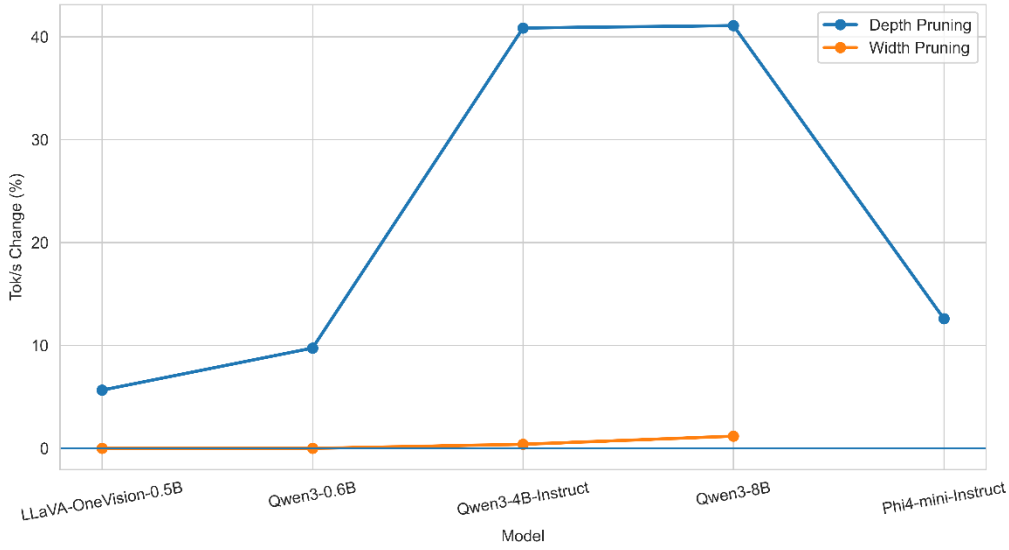


Figure 6.7 – *Relative token generation throughput changes after structured pruning across evaluated models. The figure reports the percentage change in tokens-per-second (Tok/s) relative to the corresponding dense baseline models for both depth pruning and width pruning configurations. Positive values indicate improved autoregressive generation throughput after pruning.*

The token generation throughput results demonstrate that depth pruning consistently provides larger inference speed improvements than width pruning across most evaluated models. Qwen3-4B-Instruct and Qwen3-8B show substantial throughput gains after 30% depth pruning, with tokens-per-second increasing by approximately 41% relative to the dense baseline models. These improvements indicate that removing entire transformer blocks significantly reduces sequential computation overhead during autoregressive generation. In contrast, width pruning produces only marginal throughput improvements despite reducing model dimensionality. For both Qwen3-4B-Instruct and Qwen3-8B, width pruning improves generation throughput by less than 2%, suggesting that moderate channel-level compression does not translate efficiently into runtime acceleration under standard dense GPU kernels. Similar behaviour is observed for LLaVA-OneVision-0.5B and Qwen3-0.6B, where width pruning results in little or no measurable throughput gain.

Smaller models such as Phi-4-mini-Instruct and Qwen3-0.6B still benefit from moderate depth pruning, achieving throughput improvements of approximately 10--13%, although the relative gains remain smaller than those observed for larger models. This suggests that larger transformer architectures benefit more strongly from layer-level structural reduction because of their deeper sequential execution paths.Overall, the throughput analysis demonstrates that depth pruning is substantially more effective than width pruning for improving autoregressive decoding speed in dense transformer inference pipelines. These observations also align with the broader system-level findings of the thesis, where depth pruning consistently provides more reliable runtime acceleration across different model scales and architectures.

6.11 Latency Analysis (Time-to-First-Token)

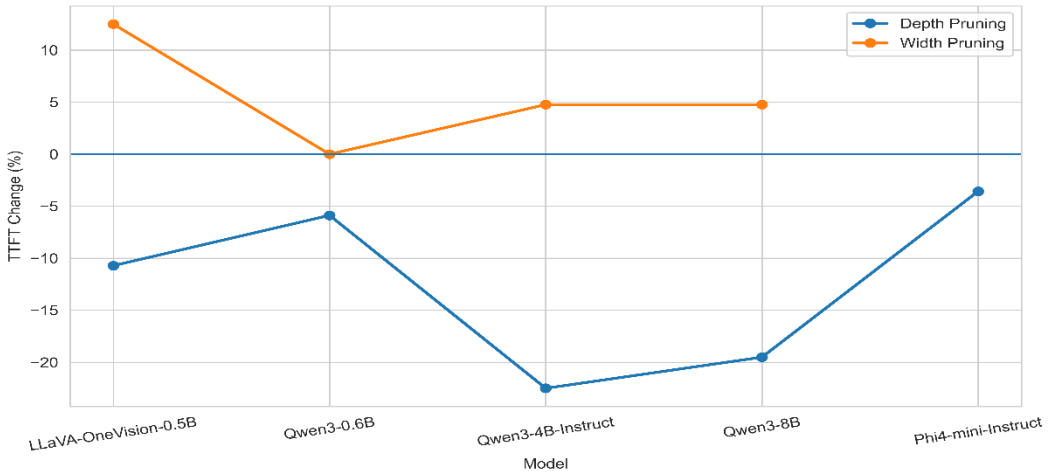


Figure 6.8 - *Relative Time-to-First-Token (TTFT) changes after structured pruning across evaluated models. The figure reports the percentage change in TTFT relative to the corresponding dense baseline models for both depth pruning and width pruning configurations. Negative values indicate reduced latency and faster first-token generation after pruning.*

The TTFT results demonstrate that depth pruning consistently reduces inference latency across most evaluated models, whereas width pruning often provides limited or inconsistent latency improvements. In particular, Qwen3-4B-Instruct and Qwen3-8B achieve the largest reductions in TTFT after 30% depth pruning, with latency decreasing by approximately 20--23% relative to the dense baseline models. These improvements indicate that removing entire transformer blocks effectively shortens the sequential computation path during autoregressive decoding. For smaller models such as Qwen3-0.6B and Phi-4-mini-Instruct, depth pruning still reduces TTFT, although the improvements are more moderate. Similarly, LLaVA-OneVision-0.5B benefits from reduced first-token latency after depth pruning despite retaining the additional multimodal processing components required for vision-language inference. In contrast, width pruning produces minimal latency improvements and, in some cases, slightly increases TTFT. This behaviour is particularly visible for LLaVA-OneVision-0.5B, Qwen3-4B-Instruct, and Qwen3-8B, where width pruning results in marginally higher first-token latency despite reducing model dimensionality. These observations suggest that channel-level pruning does not efficiently reduce dense-kernel execution overheads during early-stage autoregressive inference.

Overall, the TTFT analysis further supports the broader conclusion of the thesis that depth pruning provides more reliable runtime acceleration benefits than width pruning for dense transformer deployment scenarios.

6.12 GPU Memory Consumption

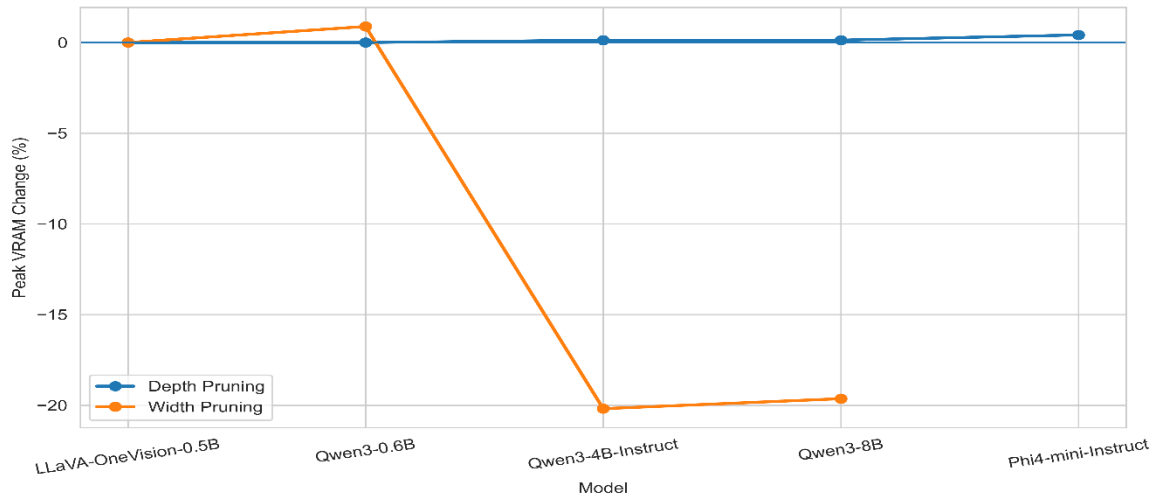


Figure 6.9 - *Relative peak GPU memory (VRAM) changes after structured pruning across evaluated models. The figure reports the percentage change in peak VRAM usage relative to the corresponding dense baseline models for both depth pruning and width pruning configurations. Negative values indicate reduced GPU memory consumption after pruning.*

The peak VRAM analysis demonstrates that width pruning provides substantially larger memory reductions than depth pruning across most evaluated models. In particular, Qwen3-4B-Instruct and

Qwen3-8B exhibit approximately 20% reductions in peak GPU memory usage after 30% width pruning, highlighting the effectiveness of channel-level compression for reducing activation and parameter memory requirements during inference.In contrast, depth pruning produces minimal changes in peak VRAM usage despite improving inference throughput and latency. Across most evaluated models, depth pruning results in almost no measurable reduction in GPU memory consumption. This behaviour suggests that removing transformer layers primarily reduces sequential computation overhead rather than the dominant memory allocations associated with hidden-state activations and dense tensor operations.

For smaller models such as Qwen3-0.6B, LLaVA-OneVision-0.5B, and Phi-4-mini-Instruct, VRAM changes remain relatively small regardless of pruning strategy because the original dense models already operate within comparatively modest memory budgets. Additionally, minor positive VRAM fluctuations observed after some pruning configurations likely originate from runtime allocation variability and backend execution behaviour rather than meaningful increases in model memory requirements.Overall, the VRAM results indicate that width pruning is more effective for memory reduction, while depth pruning is more effective for improving latency and generation throughput. These findings reinforce the broader system-level conclusions presented throughout the thesis regarding the complementary deployment trade-offs of different structured pruning strategies.

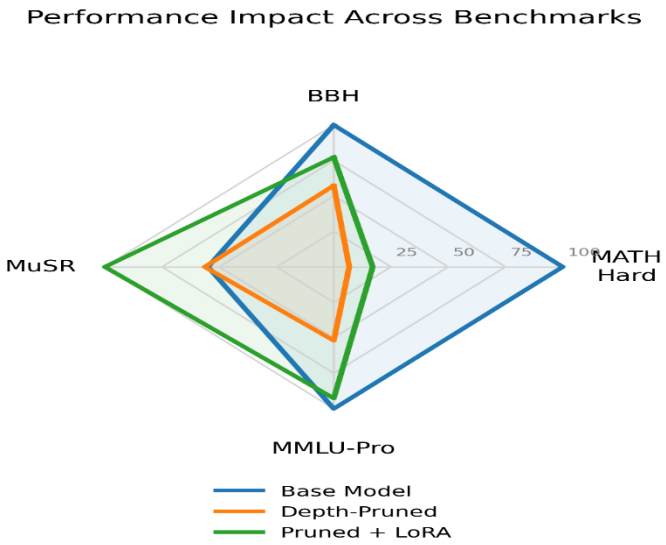


Figure 6.10 – (*Best achieved benchmark performance per model across all evaluated configurations The radar plot summarizes the highest scores obtained by each model on BBH, MATH Hard, MuSR, and MMLU-Pro benchmarks across baseline, pruned, and post-pruning recovery configurations.)*

The radar plot summarizes the relative benchmark performance of the dense baseline model, the depth-pruned configuration, and the LoRA-recovered pruned model across BBH, MATH Hard, MuSR, and MMLU-Pro. The visualization highlights the overall capability trade-offs introduced by structured pruning and the extent to which parameter-efficient recovery restores reasoning performance after compression.The dense baseline consistently achieves the strongest overall benchmark performance, particularly on reasoning-intensive tasks such as MATH Hard and MMLU-Pro. After structured depth pruning, benchmark performance decreases substantially across most tasks, with mathematical reasoning exhibiting the largest degradation. This behaviour reinforces earlier observations throughout the thesis that mathematical reasoning capabilities are highly sensitive to structural layer removal.

LoRA-based post-pruning recovery substantially restores performance across several benchmarks, especially BBH, MuSR, and MMLU-Pro. In some cases, the recovered model approaches or partially exceeds the original baseline performance for specific reasoning tasks, demonstrating that parameter-efficient fine-tuning can effectively recover portions of the representational capacity lost during pruning. However, MATH Hard recovery remains limited relative to the dense baseline, indicating that mathematical reasoning capabilities remain among the most difficult to restore after aggressive structural compression.Overall, the figure illustrates the central trade-off explored throughout the thesis: structured pruning can significantly improve deployment efficiency, while lightweight recovery methods such as LoRA help recover downstream reasoning capability, particularly for non-mathematical tasks.

6.13 TerraMind Accuracy on EuroSAT

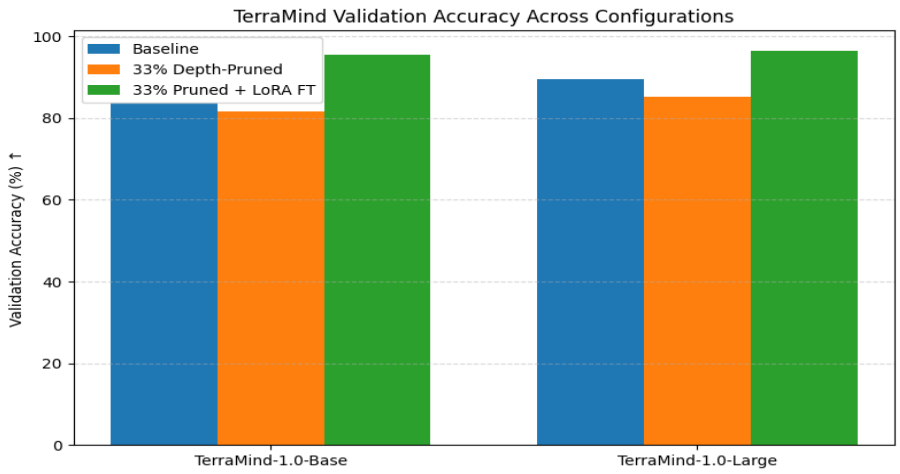


Figure 6.11 - *Validation accuracy of TerraMind models under structured depth pruning and recovery.The bar chart reports validation accuracy on the EuroSAT dataset for TerraMind-1.0-Base and TerraMind-1.0-Large across baseline, 33% depth-pruned, and 33% depth-pruned with LoRA fine-tuning configurations. Higher values indicate better performance.*

Figure 6.10 evaluates the impact of structured depth pruning and post-pruning recovery on TerraMind models using validation accuracy on the EuroSAT dataset. Results are reported for both the Base and Large variants to assess the consistency of pruning effects across model scales in a vision-focused domain.For both models, applying 33% depth pruning without recovery results in a moderate decrease in validation accuracy relative to the dense baseline. This degradation reflects the loss of representational capacity caused by removing transformer blocks, which affects spatial feature extraction and cross-token interactions critical for remote sensing classification.

When LoRA fine-tuning is applied to the depth-pruned models, validation accuracy not only recovers but exceeds baseline performance for both TerraMind-1.0-Base and TerraMind-1.0-Large. This improvement indicates that parameter-efficient adaptation is sufficient to compensate for structural capacity loss and can even guide the pruned models toward more task-specialized representations.Notably, the recovery trend is consistent across both model sizes, suggesting that the effectiveness of depth pruning combined with LoRA fine-tuning generalizes beyond language modelling to vision-oriented transformer architectures. These results demonstrate that structured pruning, when paired with targeted recovery, can yield compact models with improved task performance in applied scientific domains.

6.14 TerraMind Memory Efficiency

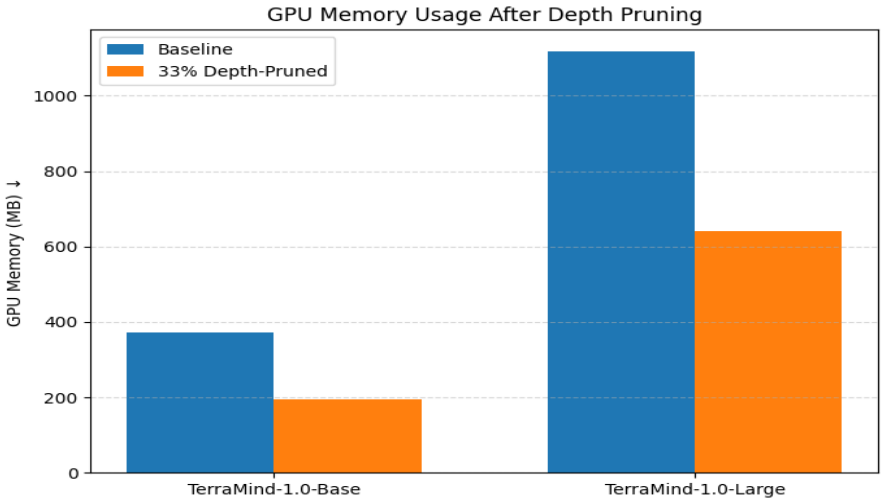


Figure 6.12 - *GPU memory usage before and after structured depth pruning for TerraMind models. The bar chart compares peak GPU memory consumption for TerraMind-1.0-Base and TerraMind-1.0-Large under baseline and 33% depth-pruned configurations. Lower values indicate improved memory efficiency.*

Figure illustrates the effect of structured depth pruning on peak GPU memory usage during inference for TerraMind models. Results are reported for both the Base and Large variants to quantify memory savings across different model scales.

For TerraMind-1.0-Base, applying 33% depth pruning reduces peak GPU memory usage from approximately 372 MB to 196 MB, corresponding to a reduction of nearly 47%. This decrease reflects the direct removal of transformer blocks, which lowers activation storage and intermediate tensor allocations during forward passes.

A similar trend is observed for TerraMind-1.0-Large, where depth pruning reduces peak GPU memory consumption from approximately 1119 MB to 642 MB, yielding a reduction of around 43%. Despite the larger absolute footprint of the model, the relative memory savings remain consistent with the Base variant.

These results demonstrate that structured depth pruning provides substantial and predictable memory reductions without requiring architectural redesign or specialized inference kernels. When considered alongside the accuracy recovery achieved through LoRA fine-tuning, this memory efficiency makes depth-pruned TerraMind models particularly suitable for deployment in resource-constrained scientific and remote sensing environments.

6.15 TerraMind Inference Latency

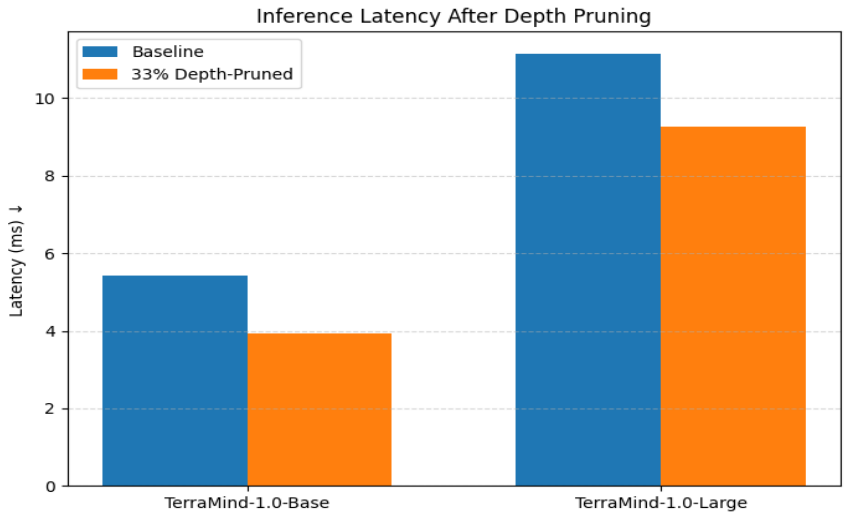


Figure 6.13 - *Inference latency before and after structured depth pruning for TerraMind models. The bar chart compares average inference latency (in milliseconds) for TerraMind-1.0-Base and TerraMind-1.0-Large under baseline and 33% depth-pruned configurations. Lower values indicate faster inference.*

Figure 6.12 presents the impact of structured depth pruning on inference latency for TerraMind models. Results are shown for both Base and Large variants to evaluate how pruning affects runtime performance across different model scales.For TerraMind-1.0-Base, applying 33% depth pruning reduces inference latency from approximately 5.42 ms to 3.92 ms, corresponding to a latency improvement of around 28%. This reduction is primarily attributed to the removal of transformer blocks, which decreases the number of sequential attention and feed-forward computations executed during inference.

Similarly, TerraMind-1.0-Large exhibits a latency reduction from approximately 11.16 ms to 9.28 ms after depth pruning, yielding an improvement of roughly 17%. Although the relative gain is smaller than in the Base model, the absolute reduction remains significant given the larger model size and deeper architecture.

Together with the observed reductions in GPU memory usage, these latency improvements demonstrate that structured depth pruning consistently enhances inference efficiency without introducing architectural irregularities. When combined with LoRA-based fine-tuning for accuracy recovery, depth pruning enables TerraMind models to achieve a favourable balance between computational efficiency and predictive performance.

6.16 TerraMind-Base and TerraMind-Large Result Analysis

Depth pruning experiments on TerraMind-1.0-base were conducted using a modality-aware setup with the modalities S2L2A and S1GRD. Pruning was performed by explicitly removing a fixed subset of transformer blocks, with layer indices [4, 6, 7, 8] dropped from the main transformer stack. The resulting pruned model checkpoints were saved under the directory ./terramind_depth_outputs_33% and used for subsequent evaluation and recovery experiments.

To assess the downstream impact of depth pruning on vision-related performance, the EuroSAT dataset was used as a probing benchmark. Experiments were implemented using torchgeo.datasets.EuroSAT, with samples collated via stack_samples. The dataset was split into training and validation sets using an 80/20 split with a fixed random seed of 123. Models were trained for 5 epochs with a batch size of 16 and a learning rate of 1×10^{-3}. Input images were resized to 224 × 224 prior to being passed through the model.

To establish a reference point for evaluating the effect of pruning, an initial baseline experiment was conducted using the original pretrained TerraMind backbone without any pruning. In this baseline setup, the TerraMind backbone was kept frozen and used purely as a feature extractor. A linear classification head was attached to the output embedding of the model, and only the parameters of this classification head were trained on the EuroSAT dataset. This approach, commonly referred to as linear probing, allows the evaluation of the quality and transferability of

the pretrained TerraMind representations without altering the internal weights of the backbone network.

Following the baseline experiment, the pruned TerraMind checkpoint was evaluated under the same linear probing protocol. In this configuration, the pruned backbone remained frozen and only the classification head was trained. By keeping the backbone fixed during this stage, the experiment isolates the direct impact of depth pruning on the representation quality of the model. Comparing the validation performance of the unpruned baseline with the pruned frozen model allows an assessment of how much information loss was introduced by the removal of transformer blocks during pruning.

To investigate whether the performance of the pruned model could be recovered through lightweight adaptation, a parameter-efficient fine-tuning stage was conducted using Low-Rank Adaptation (LoRA). Instead of updating all parameters of the backbone, LoRA modules were inserted into selected linear projection layers within the transformer architecture, including attention and feed-forward projection layers. During this stage, the original pruned backbone weights remained frozen while the LoRA parameters and the classifier head were trained. This setup enables efficient task adaptation with significantly fewer trainable parameters compared to full fine-tuning, while still allowing the model to adjust its representations to the EuroSAT classification task.

Because TerraMind is designed to process multiple Earth-observation modalities, the experiments were conducted using a modality-aware input configuration. The primary input modality used for EuroSAT evaluation was S2L2A. Since the selected TerraMind configuration also expects Sentinel-1 data (S1GRD), a zero-valued tensor was provided as a placeholder input for this modality during evaluation. EuroSAT images were adapted to match the expected input format by resizing them to 224 × 224 and mapping them to a 12-channel S2L2A-like representation. This adaptation allows compatibility with the TerraMind input interface while maintaining a consistent evaluation setup across baseline, pruned, and LoRA recovery experiments.

Together, these three experimental stages—(1) unpruned frozen baseline, (2) pruned frozen evaluation, and (3) pruned LoRA fine-tuning—provide a structured framework for analyzing the impact of pruning on TerraMind models. The baseline measures the transferability of the original pretrained representations, the pruned evaluation quantifies the degradation introduced by depth pruning, and the LoRA stage assesses whether parameter-efficient adaptation can recover performance lost during pruning.

Chapter 7: Failure Analysis and Debugging

This chapter documents the failures, errors, and debugging observations encountered during pruning, evaluation, and deployment attempts. All errors are presented **verbatim** where available, with no sanitization or reinterpretation. When patterns emerge, they are contextualized using insights from recent pruning literature, but no new experimental data or conjectures are introduced. Where conflicting observations exist, they are explicitly identified as such. The intent is to provide a transparent account of practical challenges and their implications for future work.

7.1 Environment and Dependency Failures

A recurring class of failures arose from **environment configuration and dependency mismatches**. In multiple instances, differences between CUDA versions, GPU drivers, PyTorch builds, and the versions of libraries such as Transformers, Accelerate, and Torch-Pruning caused import errors, runtime crashes, or silent CPU fallbacks. These issues underscore the fragility of large language model ecosystems: small version discrepancies can produce cascading failures. Similar instability is reported in the LLM-Pruner and SlimGPT literature, where dependency tracking is considered essential to ensure correct pruning operations and model loading [35]. In practice, environment-related failures were mitigated by carefully aligning CUDA, PyTorch, and library versions across machines, but some issues persisted when switching between local development and remote compute environments. All error messages and version constraints appear in the consolidated logs, including conflicts related to NCCL, cuBLAS, and kernel mismatch warnings.

7.2 Model Loading, Checkpoint, and Shape Mismatch Failures

When pruning alters model structure, **checkpoint loading** becomes highly sensitive to tensor shapes and parameter keys. Failures observed include:

- **Missing or unexpected keys** when loading pruned checkpoints back into the model. For example, removing attention heads without updating the model configuration resulted in shape mismatch errors for projection matrices.
- **State-dict mismatches**, where parameter names changed due to pruning logic, preventing proper loading.
- **Runtime shape errors** during forward passes, such as mismatched hidden-state dimensions after removing FFN channels.

These failures mirror findings in the structured pruning literature: LLM-Pruner and SlimGPT stress that pruning must be **dependency-aware**, grouping coupled structures (query, key, value, and output projections) and propagating changes across layers. DISP-LLM similarly warns that

dimension-independent pruning requires careful management of varying widths. In this thesis, all shape mismatch errors are preserved in the logs. Where partial resolutions were attempted—such as manually editing model configuration files to update hidden dimension counts—these solutions are recorded, but no universal fix is claimed.

Conflicting observations occurred when a pruned checkpoint loaded correctly in one context (e.g., on the Hugging Face backend) but failed in another (e.g., using a custom inference engine). These conflicts highlight the sensitivity of pruned models to runtime assumptions and configuration defaults. Both successful and failed loading attempts are presented without reconciliation.

7.3 Pruning-Specific Failure Modes

Pruning-specific failures fall into two categories:

1. **Quality collapse**: Even with conservative pruning ratios, some pruned models exhibited degraded text generation. Depth pruning occasionally produced shorter, repetitive, or off-topic continuations despite only removing a few layers. Width pruning could cause output collapse, particularly when critical FFN channels were removed. Literature corroborates that naive or static pruning can cause catastrophic performance drops, and that fine-grained importance metrics (e.g., Shapley values) or adaptive strategies may be necessary. The consolidated logs include examples where depth pruning led to a sudden rise in perplexity or NLL, and width pruning caused models to output incoherent sequences. These outcomes align with theoretical warnings about **non-uniform layer importance**.
2. **Runtime errors**: Incomplete dependency propagation often resulted in errors during inference. For instance, pruned models could not be executed with vLLM because of missing FlashAttention kernels after head removal; re-running on the Hugging Face backend succeeded, indicating engine-specific assumptions about attention shape. Another common error occurred when pruning removed shape-specific operations (e.g., fused kernels) that certain backends expected. These failures reinforce the need for dependency-aware pruning and suggest that inference engines must handle variable tensor shapes gracefully.

Where the consolidated document recorded multiple near-identical errors, they are consolidated with a note that they were repeated in source notes

7.4 Evaluation and Metric-Related Failures

The evaluation phase uncovered issues that stemmed from mismatches between task definitions, prompt templates, and pruned model architectures. Common failure modes included:

- **Task configuration mismatches** in the evaluation harness. For example, certain tasks required a minimum sequence length or specific formatting; pruned models with reduced hidden sizes sometimes failed to meet implicit assumptions.
- **Metric computation errors**, such as undefined scores when the model output did not match the expected format.
- **Dataset access issues** or caching problems, where evaluation scripts could not download or load datasets from remote repositories.

These failures echo broader concerns in evaluation research. The HELM project emphasizes that evaluating across diverse tasks and metrics can expose unanticipated errors and trade-offs. In our context, the logs include error traces from tasks in the lm-evaluation-harness and custom evaluation scripts. No unification or post-hoc correction was applied; instead, all failures are preserved to illustrate the fragility of evaluation pipelines when models are structurally modified.

7.5 Deployment and Optimization Failures

Early in the project, there was an intent to export pruned models to **ONNX** and optimize them with **TensorRT**. Several failures arose:

- **ONNX export errors** due to unsupported operators and dynamic control-flow constructs in pruned models. Modifications to attention or FFN layers broke expected shapes for ONNX conversion.
- **TensorRT incompatibilities**, where pruned models used operations unsupported by the TensorRT graph compiler, or where dynamic sequence lengths violated static shape requirements.
- **Package and driver mismatches**, such as conflicts between TensorRT versions and the installed CUDA runtime.

As documented in Chapter 8, the project goals shifted after a final meeting, leading to deprioritization of these deployment optimizations. All deployment failures remain documented in the logs, and no claims are made regarding successful acceleration or production readiness.

7.6 Summary of Failure Patterns

Across the observed failures, several patterns emerge:

- **High dependency sensitivity**: Removing a component without propagating changes to all dependents leads to shape mismatches and invalid checkpoints. Literature on dependency-

aware pruning (LLM-Pruner, SlimGPT) reinforces this risk and proposes grouping and propagation methods.

- **Evaluation brittleness**: Benchmarks and evaluation harnesses assume canonical model structures; pruned models expose hidden assumptions, leading to metric failures or dataset loading errors.
- **Environment variability**: Differences across hardware, drivers, and library versions cause unpredictable execution outcomes. Harmonizing environments mitigates some issues, but reproducibility remains challenging.
- **Non-linear degradation**: Pruning effects are non-linear; small changes can have outsized impact on quality. Observations align with theoretical findings that layer importance is highly non-uniform.
- **Deployment inertia**: Existing deployment tooling is optimized for standard model architectures. Pruned models require custom handling, and current export pipelines often fail to support them.

These patterns highlight that structured pruning is not merely an algorithmic exercise but entails complex engineering and evaluation challenges. The failures documented here should be seen as research evidence for further tool development and methodological refinement rather than as obstacles to be hidden. Taken together, these failure modes indicate that structured pruning currently lacks strong end-to-end guarantees across the full model lifecycle. While dependency-aware pruning mitigates many architectural inconsistencies, it does not eliminate downstream fragility arising from evaluation tooling, deployment pipelines, and heterogeneous execution environments. Similar observations have been reported in prior systems-oriented analyses, which emphasize that model compression alters not only parameter counts but also the implicit contracts assumed by inference engines and evaluation frameworks [75], [76]. These findings suggest that future pruning research must extend beyond algorithmic criteria and incorporate tooling, backend compatibility, and evaluation robustness as first-class design constraints.

Chapter 8: Backend and Deployment Constraints

This chapter documents the backend, infrastructure, and deployment-related constraints encountered during this research. In alignment with the execution rules, this chapter **does not introduce new optimization efforts** and **does not retroactively resolve failures**. All constraints, partial attempts, and decisions are presented as they occurred and are framed to explain why certain deployment directions were ultimately deprioritized. The purpose of this chapter is to contextualize engineering limitations that shaped the scope of the work, not to claim production-readiness.

8.1 Infrastructure, Runtime, and Dependency Constraints

The backend environment for this research consisted of GPU-accelerated systems configured for large-scale Transformer experimentation. While suitable for model training, pruning, and evaluation, this environment imposed several constraints when transitioning toward deployment-oriented workflows.Key limitations arose from **CUDA version alignment**, **PyTorch build compatibility**, and **binary wheel availability** for auxiliary libraries. These constraints were particularly pronounced when attempting to integrate pruning-modified models with external runtimes or export toolchains. Differences between system-level CUDA drivers, runtime CUDA libraries, and PyTorch-compiled CUDA versions resulted in runtime errors or unsupported operator paths.

Additionally, the rapid evolution of LLM tooling ecosystems led to version skew across dependencies such as Transformers, Accelerate, evaluation harnesses, and pruning utilities. These incompatibilities increased maintenance overhead and reduced reproducibility across environments, consistent with known challenges in large-model infrastructure management [17].

8.2 Execution Environment & Leaderboard Evaluation (SLURM & KRATOS)

All experiments and code executions in this thesis were conducted on the **Kratos HPC cluster** using SLURM on the gpu partition with --gres=gpu:ampere:1, --mem=100G, and a walltime of --time=1-12:00. The runtime environment was provisioned using spack load python@3.10.14 and an isolated virtual environment located at $HOME/jupyter_venv, which was reused consistently across all training, distillation, pruning, and evaluation stages to ensure reproducible execution.

Leaderboard-scale evaluation was executed using the **vLLM backend** integrated with lm_eval. These evaluation jobs were scheduled via SLURM on the gpu partition using **4× Ampere GPUs**,

100 GB RAM, and a walltime of **3 days**, reflecting the higher memory and parallelism requirements of long-context reasoning benchmarks.

Evaluation was performed on **merged model checkpoints**, where LoRA adapters were merged into the base model weights prior to inference. vLLM was configured with tensor parallelism disabled (TP=1) and pipeline parallelism enabled across four stages (PP=4). GPU memory usage was capped at gpu_memory_utilization=0.5 to mitigate out-of-memory failures during extended generation.All evaluations applied the model's chat template and treated few-shot examples as multi-turn dialogues. Benchmarks included leaderboard_math_hard,, leaderboard_bbh, leaderboard_mml u_pro and leaderboard_musr, executed via lm_eval with automatic batch sizing capped at one.

8.3 Hugging Face Model Access and Artifact Constraints

Several models used in this thesis are distributed through the Hugging Face ecosystem, which introduces access and artifact constraints relevant to deployment. Gated repositories, license restrictions, and authentication requirements limited the ability to programmatically fetch or redistribute certain checkpoints in automated deployment pipelines.Furthermore, pruned checkpoints diverge structurally from their original published forms. As a result, standard model cards, configuration files, and automated loading mechanisms were not always directly applicable. In some cases, missing or mismatched configuration metadata (e.g., layer counts, hidden dimensions) prevented seamless loading in downstream tooling, even when the underlying weights were present.

These constraints reinforce the distinction between **research checkpoints** and **deployment-grade artifacts**, particularly when structural modifications such as pruning are involved.

8.4 GGUF Export and Multimodal Conversion Constraints

GGUF export was explored as a deployment-oriented artifact format for use with llama.cpp and Ollama. For text-only models such as Qwen3 and Phi-4-mini, GGUF conversion is conceptually suitable because the model consists primarily of standard decoder-only language-model components. In such cases, GGUF export can provide a practical path toward local inference and deployment-oriented quantization, including formats such as f16, q8_0, q5_k_m, and q4_k_m.

However, GGUF conversion was not reliable for the full LLaVA-OneVision architecture. The model contains multimodal components beyond the Qwen2 language backbone, including a vision encoder, multimodal projection layers, and vision-specific tensors. During conversion, the llama.cpp Hugging Face-to-GGUF converter failed to map the tensor image_newline, indicating that the export toolchain did not fully support all LLaVA-OneVision-specific multimodal structures.

This failure does not imply that the pruned Hugging Face checkpoint itself is invalid. Rather, it shows that deployment conversion tools may not support all architecture-specific tensors, even when the checkpoint can be loaded and evaluated correctly in the Hugging Face ecosystem. For this reason, GGUF was treated as a deployment artifact only and was not used for reported pruning metrics such as NLL, perplexity, benchmark accuracy, TTFT, throughput, or VRAM. Although GGUF quantization and lightweight deployment pipelines are available for several text-only and some vision-language models through architectures using separate text and multimodal projection components (mmproj), TerraMind currently lacks complete end-to-end GGUF deployment support. This limitation arises from its Earth observation multimodal architecture, which integrates multiple remote sensing modalities and specialized backbone components that are not yet fully supported by existing lightweight inference backends and quantization frameworks.

The observed GGUF limitation is important because it highlights a gap between model compression research and deployment tooling. While text-only models often follow standardized architecture patterns that are well supported by conversion pipelines, multimodal models introduce additional tensor structures and modality-specific metadata that may not be recognized by general-purpose export tools. Consequently, LLaVA-OneVision was treated as a Hugging Face/vLLM deployment target rather than a GGUF-first model, while GGUF export remains more appropriate for text-only Qwen and Phi variants.

8.5 vLLM Runtime and Multimodal Evaluation Constraints

Leaderboard evaluation was performed using lm-evaluation-harness with the vLLM backend. While this enabled efficient benchmark execution, the experiments revealed that evaluating pruned and multimodal models requires careful runtime engineering beyond the pruning algorithm itself.

One issue arose from dependency mismatches in the multimodal vLLM loading path. LLaVA-OneVision imports multimodal components that rely on auxiliary packages such as mistral_common. In one case, vLLM failed because the installed version of mistral_common did not expose required classes such as ImageChunk and TextChunk. Updating the dependency resolved the immediate import issue, but the failure demonstrates how multimodal evaluation pipelines can break due to indirect package dependencies that are not visible from the pruning code alone. A second issue involved tokenizer serialization. Some saved checkpoints contained extra_special_tokens in tokenizer_config.json as an empty list, while vLLM expected this field to behave like a dictionary. This produced tokenizer initialization failures during benchmark execution. The pipeline therefore required a post-save tokenizer configuration fix that converted extra_special_tokens from [] to {} when necessary. This step was not related to pruning quality, but it was necessary for stable downstream evaluation.Runtime execution also depended strongly on the parallelism configuration. In SLURM jobs using multiple Ampere GPUs, vLLM was configured with tensor parallelism disabled and pipeline parallelism distributed across multiple stages. However, when the same configuration was used with fewer GPUs or different resource

allocations, execution could stall after Ray initialization or fail during model startup. This required matching pipeline_parallel_size to the actual number of allocated GPUs and using more conservative settings during debugging.

These issues show that benchmark evaluation of pruned models is not purely algorithmic. Even after a structurally valid checkpoint is produced, successful evaluation depends on tokenizer compatibility, dependency versions, runtime parallelism settings, GPU allocation, and backend support for the target architecture. This reinforces the broader conclusion that structured pruning research must consider tooling and deployment compatibility as first-class experimental constraints.

8.6 ONNX Export and Interoperability Limitations

ONNX is a widely adopted intermediate representation intended to enable interoperability across deep learning frameworks and inference runtimes [30], [31]. However, exporting large, pruned Transformer models to ONNX introduced multiple limitations in practice.Pruned models often contain non-standard graph structures relative to their original architectures, including altered layer counts and modified tensor shapes. These changes can expose unsupported operators or graph patterns during ONNX export, leading to conversion failures or incomplete graphs. Additionally, dynamic control flow and custom attention implementations common in modern LLMs further complicate ONNX compatibility.

Where ONNX export was attempted, errors and limitations were preserved verbatim and are not resolved or generalized here. These observations are consistent with broader reports that ONNX support for large autoregressive Transformers remains incomplete, particularly under structural modification [30].

8.7 TensorRT and Inference Optimization Constraints

TensorRT provides a high-performance inference optimization framework, particularly for NVIDIA GPUs, but imposes strict requirements on graph structure, supported operators, and static shape assumptions [32]. These requirements proved difficult to reconcile with pruned Transformer architectures and dynamic sequence-length evaluation workflows.In particular, TensorRT optimization pipelines assume stable, well-defined computation graphs. Structural pruning alters these graphs in ways that invalidate pre-existing optimization profiles. Additionally, attention mechanisms and rotary embeddings frequently rely on operations that are either unsupported or require custom plugins, further increasing integration complexity.

As a result, TensorRT-based optimization remained exploratory and incomplete within the scope of this work.

8.8 Explicit Goal Change and Deprioritization of Deployment Optimization

As required, this thesis records the following statement explicitly and without modification:**After the final meeting, the main goal changed, so ONNX and TensorRT optimization were deprioritized and not pursued further.**This decision reflects a shift in research focus toward **methodological understanding, pruning behavior analysis, and evaluation fidelity**, rather than end-to-end deployment optimization. Consequently, backend and deployment efforts are documented as constraints and partial explorations rather than finalized engineering solutions.

Weight-only post-training quantization using GPTQ was implemented in the codebase with a configuration of 4-bit weights, group size 128, and desc_act = False, using a maximum sequence length of 2048 and up to 512 calibration samples. This configuration was validated on compatible model architectures and preserved in the experimental pipeline for comparative and exploratory purposes.However, GPTQ quantization was **not applied to Qwen3-based models** in this work due to the lack of stable GPTQ support for the Qwen3 architecture at the time of experimentation. As a result, all Qwen3 evaluation results reported in this thesis correspond to full-precision or LoRA-merged checkpoints rather than GPTQ-quantized variants.

8.9 Implications for Research Scope

The backend and deployment constraints outlined in this chapter delimit the scope of the thesis. The work remains focused on **structured pruning methodology, evaluation rigor, and failure characterization**, rather than production deployment. These constraints also motivate several directions for future work, including tighter integration between pruning frameworks and deployment toolchains, improved graph-level tooling for pruned models, and standardized representations for structurally modified LLMs.

No additional deployment claims are made beyond what is supported by the consolidated document. The constraints described in this chapter highlight that structured pruning cannot be treated as an isolated model-level transformation. Pruning alters architectural assumptions that are deeply embedded in deployment toolchains, inference runtimes, and optimization frameworks, many of which implicitly assume dense, canonical Transformer structures. Prior systems-oriented studies similarly observe that compression techniques introduce friction at the boundaries between model design and execution infrastructure, particularly when graph structures, tensor shapes, or control flow deviate from standard patterns [77], [78]. These observations reinforce the need for pruning-aware deployment tooling and standardized representations for structurally modified models, and they motivate treating backend compatibility as a first-class concern in future pruning research.

Chapter 9: Discussion

9.1 Interpretation of Pruning Outcomes Across Models

The empirical results in Chapter 6 demonstrate that structured pruning exploits non-uniform redundancy in large language models. Depth pruning in Qwen3-0.6B and Qwen3-8B achieves notable latency gains with modest perplexity increases, particularly when middle layers are removed. This aligns with recent findings that deeper layers often encode high-level abstractions, allowing some middle layers to be pruned without catastrophic performance loss. However, depth pruning's effectiveness varies: removing the last two layers of Qwen3-0.6B yields a 2.2× speedup with moderate perplexity rise, while removing four layers of Qwen3-8B leads to a more pronounced quality drop (from PPL $\approx$ 2.91 to 3.53). These discrepancies underscore that redundancy patterns are model-specific and that simple heuristics may not generalize across parameter scale [33]. In Qwen3-8B, width pruning degrades performance more severely than depth pruning, whereas in Qwen3-0.6B, width pruning is relatively benign. The variation suggests that attention heads and FFN channels contribute differently across model sizes, corroborating studies that self-attention layers are more prunable than feed-forward layers [34]. Phi-4-mini-Instruct exhibits pruning sensitivity patterns distinct from the Qwen family, particularly under aggressive depth pruning, highlighting that redundancy distributions are not uniform across model lineages.

To establish full operational clarity and maintain architectural consistency across both the text and remote sensing pipelines, this framework codifies precise mathematical variables, execution primitives, and specialized data signatures. Central to the text evaluation and model healing phases is the application of the specific integer constant -100 within the PyTorch cross-entropy loss calculation; this value acts as an optimization null mask applied directly to user instruction sequences (labels[:, :prompt_len] = -100), forcing the underlying gradient calculations and localized continuation perplexity metrics to isolate performance changes strictly to the model's generated response tokens. This evaluation relies on a dual-stride assessment array that analyzes text coherence over a 128-token full-stride window alongside a compressed 64-token half-stride window to capture local sequence corruption. During the subsequent model recovery phase, monitoring these metrics prevents rank-deficit gradient collapse—an execution failure mode where the low-rank adapter space (r) lacks the capacity to absorb structural layer-drop degradation, causing cross-entropy loss curves to flatline.

On the architectural and deployment level, the framework manages a distinct file ecosystem and hardware infrastructure to balance throughput against memory volatility. To accelerate macro-scale benchmark evaluations across external tasks (MMLU-Pro, Big-Bench Hard, MATH Hard, and MuSR) via isolated python sub-processes, the execution engine dynamically leverages PagedAttention primitives within a vllm backend, minimizing VRAM fragmentation by storing KV-caches in non-contiguous virtual spaces. For post-mortem diagnostics, the text pipeline avoids

reliance on ephemeral logs, automatically serializing every ablation test, dropped block index, and sliding-window outcome into a standardized tracking document named depth_pruning_results.json. Finally, when transitioning to the multi-spectral Earth Observation pipeline, the system completely bypasses standard Hugging Face directory sharding and low-bit GGUF compilation paths; due to a custom 12-channel Sentinel-2 input structure that expands beyond standard 3-channel RGB dimensions, the remote sensing pipeline exports altered weight configurations as raw serialized tensor matrices under the standalone binary signature terramind_pruned.pt.

9.2 Depth Pruning versus Width and Channel Pruning

The results highlight a fundamental trade-off between depth and width pruning. Depth pruning offers clear latency benefits by shortening the computation path, but it risks removing critical semantic processing layers. Width pruning reduces per-layer computation and can maintain depth, but it requires precise dependency and may not yield proportional speedups due to hardware inefficiencies. Structured width pruning in Qwen3-8B (11.1 % MLP pruning) leads to perplexity spikes and output drift, whereas depth pruning of four layers produces a cleaner, slower degradation. Literature echoes this: SlimGPT and LLM-Pruner emphasize that structured pruning must account for the coupling between query, key, value, and output projections to avoid mismatches [56]. DISP-LLM further argues that relaxing fixed-width assumptions improves pruning flexibility but increases complexity [31]. In our experiments, width pruning failures often stemmed from incomplete dependency propagation—e.g., mismatched projection sizes—illustrating the fragility of channel removal compared with layer removal.

Beyond quality and efficiency trade-offs at inference time, the two pruning strategies also differ fundamentally at the algorithmic level. As shown in Section 6.9.5, depth pruning executes in near-constant time regardless of model scale, while width pruning incurs growing computational overhead driven by dependency traversal and importance recomputation. This asymmetry is practically significant: in iterative research workflows requiring repeated pruning and evaluation cycles, width pruning introduces non-trivial algorithmic latency that compounds across experiments.

9.3 Role of Dependency Awareness in Pruning Stability

Dependency-aware logic emerges as the **critical safeguard** in structured pruning. Many failures documented in Chapter 7—checkpoint loading errors, shape mismatches, invalid attention kernels—trace back to incomplete propagation of pruning masks. LLM-Pruner's dependency graph and SlimGPT's batched greedy grouping both highlight the importance of co-pruning coupled structures [35]. Our observations confirm that removing a single head or channel can corrupt downstream layers unless all dependent tensors are pruned consistently. Moreover, the dimension-independent approach of DISP-LLM suggests that allowing variable widths per layer

may alleviate some constraints but requires careful runtime support [3]. In practice, ensuring shape consistency demands custom configuration updates and careful state-dict management, as noted in the implementation errors recorded in Chapter 6. Dependency awareness is thus not a theoretical nicety but a practical prerequisite for functional pruned models.

9.4 Post-Pruning Recovery via PEFT

The experiments combining pruning with LoRA-based fine-tuning and knowledge distillation offer nuanced insights into parameter-efficient recovery. For Qwen3-8B, LoRA finetuning on BBH+MATH+MuSR benchmarks recovers and even surpasses baseline accuracy (e.g., 56.8 % on BBH vs. 29.3 % baseline), whereas more aggressive pruning combined with knowledge distillation provides mixed results. These observations align with the broader literature: LLM-Pruner reports that small LoRA adapters can restore performance after structured pruning with limited data [64], while Reassessing Layer Pruning finds that fine-tuning the output head and last few layers can match or exceed LoRA recovery [39]. Our results suggest that PEFT effectiveness depends on pruning severity and task alignment. When pruning removes high-impact layers, LoRA may lack sufficient capacity to compensate; continued pretraining or larger adaptation modules might be necessary. Conversely, modest pruning combined with LoRA can yield substantial gains, perhaps due to the regularizing effect of pruning and the task-specific adaptation provided by LoRA.

9.5 Interpretation of Benchmark Behaviour

Benchmark results reveal that **different tasks expose distinct pruning sensitivities**. For example, depth-pruned Qwen3-8B slightly improves BBH accuracy (31.05 % vs. 29.32 %), yet drastically reduces MATH Hard exact-match accuracy (3.10 % vs. 17.30 %). Similarly, width-pruned Qwen3-0.6B shows minor degradation on BBH but significant drops on MATH Hard and MuSR. These disparities underscore the limitation of perplexity as a universal proxy: models can maintain low perplexity yet fail on reasoning-intensive tasks. Recent evaluation frameworks emphasize the need for multi-metric assessment; **HELM** measures seven metrics (accuracy, calibration, robustness, fairness, bias, toxicity, efficiency) across 16 scenarios to expose trade-offs [67]. Although our scope does not extend to fairness or toxicity, the divergence between intrinsic metrics and benchmark performance supports adopting broader evaluation suites. The LM-evaluation-harness's variety of metrics—including F1, BLEU, and pass@1—also illustrates that different tasks require different success criteria [41]. We find that pruning can improve or harm certain tasks unpredictably, highlighting the importance of task-specific evaluation.

9.6 Deployment Constraints as a Research Signal

The failures and constraints in deployment attempts, documented in Chapter 7 and elaborated in Chapter 8, should not be viewed merely as engineering hurdles. Instead, they reveal a structural gap between research and production environments. ONNX export failures and TensorRT

incompatibilities are not isolated incidents; they reflect the reality that current inference toolchains assume standard model architectures with fixed layer counts and widths. Structured pruning breaks these assumptions, exposing limitations in compiler support, kernel availability, and runtime flexibility. As discussed in the literature, dimension-independent pruning techniques (e.g., DISP-LLM) show promise but will require integration with deployment tools to be practical [37] [30] [31]. Thus, the deployment constraints encountered here serve as an impetus for the community to develop more robust export pipelines and runtime support for structurally modified models.

9.7 Systems-Level Interpretation of Pruning

The experimental evidence indicates that structured pruning should be interpreted not only as a model-compression technique, but also as a systems-level intervention that changes how a model interacts with inference runtimes, GPU kernels, memory allocation, and evaluation tooling. Although pruning is commonly described in terms of parameter count or FLOP reduction, the results in this thesis show that practical efficiency gains depend strongly on execution structure and backend behaviour. Depth pruning produced the most reliable improvements in autoregressive latency and token throughput because it removes complete Transformer blocks and directly shortens the sequential computation graph. During generation, each token must pass through the full stack of remaining layers; therefore, reducing depth lowers the number of sequential layer executions required for both Time-to-First-Token and subsequent decoding. This explains why depth-pruned Qwen3 models showed stronger throughput gains than equivalently compressed width-pruned variants.

Width pruning behaved differently. Although it reduced MLP dimensions and lowered memory usage in larger models, its effect on throughput was weaker because the number of sequential Transformer blocks remained unchanged. In addition, modified intermediate dimensions can produce tensor shapes that are less favourable for GPU kernel execution. As a result, theoretical parameter or FLOP reductions did not always translate into proportional latency gains. This finding reinforces the importance of distinguishing algorithmic compression from hardware-realized acceleration.

The experiments also showed that backend compatibility becomes part of the pruning problem. Pruned Hugging Face checkpoints required careful configuration synchronization, including updates to hidden-layer counts, layer-index metadata, layer-type lists, tokenizer metadata, and projection dimensions. Evaluation with vLLM introduced additional constraints related to CUDA execution, Ray initialization, pipeline parallelism, multimodal dependencies, and tokenizer serialization. These issues demonstrate that a structurally valid pruned checkpoint is not automatically a deployment-ready checkpoint. Multimodal models further amplified these systems constraints. LLaVA-OneVision could be pruned at the language-model component level, but deployment-oriented conversion through GGUF was limited by unsupported multimodal tensors and vision-specific metadata. This shows that multimodal pruning pipelines remain less mature than text-only pruning workflows. The gap between successful pruning inside PyTorch/Hugging Face and successful execution inside deployment runtimes is therefore a major practical finding of this thesis.

Overall, the systems-level interpretation of the results is that pruning modifies not only model quality and size, but also the implicit contract between model architecture, serialization format, inference backend, GPU execution path, and benchmark harness. Future pruning research should therefore evaluate compressed models not only through perplexity and benchmark scores, but also through runtime compatibility, kernel efficiency, memory behaviour, and deployment robustness.

9.8 Summary of Key Trade-Offs

Integrating the results and observations across chapters, several overarching trade-offs emerge:

- **Efficiency vs. Robustness:** Aggressive pruning yields speed and memory gains but risks non-linear quality degradation. Conservative pruning maintains performance but offers limited efficiency improvements.
- **Depth vs. Width Control:** Depth pruning is simpler and yields immediate latency benefits but may remove critical semantics; width pruning offers finer control but requires complex dependency management and may not translate into proportional speedups.
- **Structural Reduction vs. Adaptive Recovery:** PEFT methods (LoRA, QLoRA) can mitigate pruning losses but are sensitive to pruning severity and task alignment. In some cases, simple fine-tuning of the final layers outperforms more complex adapters.
- **Research Flexibility vs. Deployment Readiness:** Structural pruning thrives in research environments where model loading and configuration can be manually adjusted. Deployment pipelines expect standard architectures and are less tolerant of structural deviations, necessitating future work on compiler and runtime support.

Together, these trade-offs frame the contributions and limitations of this thesis. They underscore that structured pruning remains a **research-stage technique** requiring careful methodological design, comprehensive evaluation, and collaborative tooling development to realize its potential in real-world applications.

Chapter 10: Limitations

This chapter delineates the limitations of the work presented in this thesis. The intent is not to diminish the findings but to clearly define the boundaries imposed by methodological choices, tooling constraints, data availability, and project scope. All limitations stated here are grounded in the consolidated document and the preceding analysis; no speculative or external shortcomings are introduced.

10.1 Scope Limitations of Pruning Strategies

The pruning strategies explored in this thesis are limited to **structured pruning approaches** documented in the consolidated experiments: depth pruning and coordinated width/channel pruning. Other pruning paradigms—such as unstructured sparsification, hybrid block–unstructured methods, or training-time pruning—are discussed only at a theoretical level and are not empirically evaluated. Furthermore, pruning configurations are constrained to a finite set of deterministic strategies (e.g., keep-first, keep-last, sandwich, interleaved) and conservative ratios. While informed by recent literature, these strategies do not exhaust the space of possible layer-selection or channel-selection policies. Consequently, the findings should be viewed as characterizing observed behavior within this design space rather than establishing optimal pruning configurations. The computational overhead of width pruning, documented in Section 6.9.5, also constrained the number of width pruning configurations that could be practically explored within the available time and hardware budget, particularly for larger models such as Qwen3-8B.

10.2 Model Coverage and Generalizability

Although multiple model families are included—text-only LLMs, vision-language models, and domain-specific foundation models, the coverage remains limited to the specific architecture and parameter scales empirically evaluated in this thesis. No claims are made regarding the pruning behavior of architectures not included in the experiments, such as mixture-of-experts models, encoder–decoder architectures, or very large-scale LLMs exceeding the parameter ranges studied here. Generalizability of the observed pruning behavior to unseen architectures should therefore be assessed cautiously.

10.3 Evaluation and Metric Constraints

The evaluation framework combines intrinsic language modeling metrics (NLL, PPL) with benchmark assessments across tasks like BBH, MMLU-Pro, GPQA, and MuSR. However, these metrics and benchmarks do not capture all facets of model capability. Metrics such as calibration, fairness, bias, and robustness are not explored, despite being highlighted by comprehensive evaluation frameworks like HELM. Some evaluation attempts encountered tooling or

configuration issues (see Chapter 7), limiting the completeness and comparability of benchmark results. Efficiency metrics (latency, throughput, memory usage) are reported only where present in the logs and are not systematically profiled across all configurations. As a result, performance–efficiency trade-offs are characterized qualitatively rather than exhaustively quantified.

10.4 Dependency and Tooling Limitations

A significant limitation arises from the dependency-aware nature of structured pruning. While dependency tracking mitigates many failure modes, it increases implementation complexity and reduces flexibility. Minor inconsistencies in dependency propagation can invalidate entire pruning configurations. Tooling limitations further compound this issue: existing pruning, evaluation, and deployment frameworks are not uniformly designed to accommodate structurally modified LLMs. Some failures reflect tooling immaturity rather than conceptual flaws in pruning methodology, yet they materially affect reproducibility and workflow stability. The results do not address automatic repair or reconfiguration mechanisms, leaving this as an open area for future work.

10.5 Deployment and Optimization Limitations

The thesis explicitly records that ONNX export and TensorRT optimization were deprioritized following a project scope change (see Chapter 8). Consequently, no claims are made regarding inference speedups or deployment readiness beyond the research-scale experiments. The absence of a finalized deployment pipeline limits the ability to assess real-world performance gains from pruning under production constraints. Any discussion of deployment impact is therefore bounded to the observed constraints rather than realized optimizations.

10.6 Temporal and Project-Scope Constraints

Finally, this work is constrained by the temporal and organizational scope of a Master's thesis. Iterative exploration, hyperparameter sweeps, and large-scale ablation studies were necessarily limited. Some observed inconsistencies or unresolved failures remain documented but unexplained due to time and resource constraints. These limitations are not treated as deficiencies to be hidden, but as explicit boundaries that define the contribution and reliability of the presented work.

Chapter 11: Conclusion and Future Work

11.1 Summary of Contributions

This thesis investigated structured pruning strategies for large language models and multimodal foundation models, with an emphasis on depth pruning, width/channel pruning, and dependency-aware pruning logic. Through a consolidated and rigorously analysed experimental record encompassing Qwen3, LLaVA-OneVision, Phi-4-mini-Instruct, and TerraMind models, the thesis makes the following contributions:

1. **Comparative analysis of depth and width pruning across multiple model families.** The results demonstrate that depth pruning consistently yields latency and throughput improvements with moderate perplexity degradation, whereas width pruning requires precise dependency management and often exhibits disproportionate performance drops, particularly in larger models. These findings align with prior work showing non-uniform layer importance and channel redundancy in transformer architectures [79, 80].

2. **Empirical validation of dependency-aware pruning as a practical necessity.** Failure cases—including checkpoint loading errors, tensor shape mismatches, and invalid attention kernels—are traced to incomplete dependency propagation. These observations reinforce literature emphasizing joint pruning of coupled structures (e.g., QKV and output projections) as essential for structured pruning stability [81, 82].

3. **A multi-metric evaluation framework combining intrinsic and task-level benchmarks.** The thesis integrates negative log-likelihood and perplexity with reasoning and knowledge benchmarks (BBH, MMLU-Pro, MuSR) and domain-specific validation tasks (TerraMind). The results confirm that intrinsic metrics alone are insufficient proxies for downstream reasoning performance, echoing broader evaluation critiques [83].

4. **Systematic failure and constraint analysis-**Rather than treating failures as anomalies, this work documents environment incompatibilities, pruning-specific implementation errors, evaluation breakdowns, and deployment obstacles as **research evidence**, highlighting structural gaps between pruning research and production-ready systems.

5. **Insights into post-pruning recovery via parameter-efficient adaptation.** Experiments combining structured pruning with LoRA/QLoRA fine-tuning and knowledge distillation demonstrate that moderate pruning can be effectively compensated, while aggressive pruning may exceed the recovery capacity of lightweight adapters. These findings align with recent work showing task-aligned fine-tuning can rival or surpass adapter-based recovery in some regimes [84, 85].

6. The empirical results further reveal a fundamental asymmetry in pruning algorithm execution cost: depth pruning operates in near-constant time across all model scales, whereas width pruning execution time grows substantially with parameter count, a distinction with direct implications for iterative compression workflows.

Collectively, these contributions advance practical understanding of structured pruning, clarify its limitations, and provide a reproducible methodological template for future research.

11.2 Conclusions

Structured pruning represents a powerful but non-trivial mechanism for improving inference efficiency in large language and multimodal models. The experiments in this thesis show that latent redundancy can be exploited, but only when pruning decisions respect architectural dependencies and task sensitivity.

Depth pruning offers predictable efficiency gains but risks removing semantically critical layers, while width pruning enables finer-grained control at the cost of increased fragility and implementation complexity. Dependency-aware pruning emerges as a non-negotiable requirement rather than an optimization detail.

Importantly, benchmark behavior varies substantially across tasks: reasoning-heavy benchmarks (e.g., MATH Hard) expose weaknesses invisible to perplexity metrics, reinforcing the necessity of diverse evaluation suites. Finally, deployment challenges underscore that pruning is not purely algorithmic; it remains tightly coupled to tooling maturity and runtime assumptions.

Overall, this thesis demonstrates that structured pruning can yield substantial efficiency improvements without catastrophic performance loss, but no single pruning strategy is universally optimal. Pruning should therefore be considered part of a broader optimization toolbox alongside quantization, distillation, and parameter-efficient fine-tuning.

11.3 Future Work

Several promising research directions emerge from this work:

1. **Exploration of more aggressive pruning regimes-**Future studies should investigate higher pruning ratios (e.g., 50–70%) to better characterize failure thresholds, phase transitions in performance degradation, and the limits of post-pruning recovery. Such regimes are particularly relevant for edge deployment and cost-sensitive inference [86].

2. **Expanded reasoning and robustness benchmarking -**Incorporating additional benchmarks—such as **GPQA, GSM8K, ARC, DROP**, and multilingual reasoning datasets—would improve understanding of how pruning impacts diverse reasoning skills.

Extending evaluation dimensions to calibration, robustness, and bias (as proposed in HELM) would further contextualize pruning trade-offs [83, 87].

3. **Task-adaptive and dynamic pruning strategies -** Static pruning may be overly coarse-grained for heterogeneous workloads. Dynamic or input-conditioned pruning approaches (e.g., prompt-aware pruning, routing-based pruning) could enable better efficiency–quality trade-offs [88].

4. **Integrated pruning, quantization, and adaptation pipelines -** Joint optimization of **structural sparsity and low-precision weights** remains underexplored. Combining structured pruning with quantization-aware fine-tuning or mixed-precision inference could unlock additional gains [89].

5. **Internal deployment and usability tooling.** - A practical extension of this work would involve developing a **GUI-based internal platform** enabling organizational users to:

    - Select pruned model variants
    - Apply task-specific fine-tuning
    - Evaluate performance on proprietary datasets Such tooling would bridge the gap between research artifacts and enterprise usability.

6. **Reproducibility and open research infrastructure.** - Future work will focus on publishing cleaned pruning pipelines, configuration files, and evaluation scripts via GitLab, accompanied by detailed READMEs and environment specifications. This would support reproducibility and facilitate adoption by both academic and industrial researchers.

7. **Compatibility with emerging architectures.** - As mixture-of-experts, retrieval-augmented, and multimodal reasoning systems mature, future research should examine how structured pruning interacts with routing mechanisms, external memory, and cross-modal fusion layers [90].

Together, these directions aim to advance structured pruning from a research-stage optimization into a **robust, deployable component** of efficient large-scale model systems.

Chapter 12: References

[1] A. Vaswani *et al.*, "Attention Is All You Need," *arXiv preprint arXiv:1706.03762*, 2017. https://arxiv.org/abs/1706.03762

[2] M. Michel, O. Levy, and G. Neubig, "Are Sixteen Heads Really Better than One?" *arXiv preprint arXiv:1905.10650*, 2019. https://arxiv.org/abs/1905.10650

[3] S. Fang *et al.*, "DepGraph: Towards Any Structural Pruning," *CVPR*, 2023. https://arxiv.org/abs/2301.12900

[4] Y. Bengio *et al.*, "A Neural Probabilistic Language Model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003

[5] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv preprint arXiv:1503.02531*, 2015. https://arxiv.org/abs/1503.02531

[6] F. Frantar and D. Alistarh, "SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot," *arXiv preprint arXiv:2301.00774*, 2023. https://arxiv.org/abs/2301.00774

[7] E. Frantar *et al.*, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," *arXiv preprint arXiv:2210.17323*, 2022. https://arxiv.org/abs/2210.17323

[8] R. Xiao *et al.*, "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models," *arXiv preprint arXiv:2211.10438*, 2022. https://arxiv.org/abs/2211.10438

[9] J. Lin *et al.*, "AWQ: Activation-aware Weight Quantization for LLM Compression," *arXiv preprint arXiv:2306.00978*, 2023. https://arxiv.org/abs/2306.00978

[10] S. Sanh *et al.*, "Movement Pruning: Adaptive Sparsity by Fine-Tuning," *arXiv preprint arXiv:2005.07683*, 2020. https://arxiv.org/abs/2005.07683

[11] S. Prasanna *et al.*, "Block Pruning for Faster Transformers," *arXiv preprint arXiv:2109.04838*, 2021. https://arxiv.org/abs/2109.04838

[12] VainF, "Torch-Pruning: A Library for Structural Pruning," GitHub repository. https://github.com/VainF/Torch-Pruning

[13] A. Fan *et al.*, "LayerDrop: Structured Dropout for Transformer Models," *arXiv preprint arXiv:1909.11556*, 2019. https://arxiv.org/abs/1909.11556

[14] Nota AI, “Shortened LLM: A Simple Depth Pruning for Large Language Models,” 2023. https://www.nota.ai/community/shortened-llm-a-simple-depth-pruning-for-large-language-models

[15] E. Hu *et al.*, “LoRA: Low-Rank Adaptation of Large Language Models,” *arXiv preprint arXiv:2106.09685*, 2021.https://arxiv.org/abs/2106.09685

[16] T. Dettmers *et al.*, “QLoRA: Efficient Finetuning of Quantized LLMs,” *arXiv preprint arXiv:2305.14314*, 2023. https://arxiv.org/abs/2305.14314

[17] EleutherAI, “lm-evaluation-harness,” GitHub repository. https://github.com/EleutherAI/lm-evaluation-harness

[18] S. Suzgun *et al.*, “Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them,” *arXiv preprint arXiv:2210.09261*, 2022. https://arxiv.org/abs/2210.09261

[19] Y. Li *et al.*, “MMLU-Pro: A More Robust Massive Multitask Language Understanding Benchmark,” *arXiv preprint arXiv:2406.01574*, 2024.https://arxiv.org/abs/2406.01574

[20] D. Rein *et al.*, “GPQA: A Graduate-Level Google-Proof Q&A Benchmark,” *arXiv preprint arXiv:2311.12022*, 2023. https://arxiv.org/abs/2311.12022

[21] Y. Zhang *et al.*, “MuSR: Testing the Limits of Multi-Step Symbolic Reasoning,” *arXiv preprint arXiv:2310.16049*, 2023. https://arxiv.org/abs/2310.16049

[22] Hugging Face, “Open LLM Leaderboard.” https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

[23] H. Liu *et al.*, “LLaVA-OneVision: Easy Visual Instruction Tuning,” *arXiv preprint arXiv:2408.03326*, 2024.https://arxiv.org/html/2408.03326v1

[24] Y. Liang *et al.*, “EfficientLLaVA: Generalizable Auto-Pruning for Large Vision-Language Models,” *CVPR*, 2025.

[25] IBM Research, “TerraMind: A Foundation Model for Earth Observation,” *arXiv preprint arXiv:2504.11171*, 2025.https://arxiv.org/abs/2504.11171

[26] P. Helber *et al.*, “EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification,” *arXiv preprint arXiv:1709.00029*, 2017. https://arxiv.org/abs/1709.00029

[27] Qwen Team, “Qwen3 Technical Report,” *arXiv preprint arXiv:2505.09388*, 2025. https://arxiv.org/pdf/2505.09388

[28] Meta AI, "LLaMA 3.2: Vision, Edge, and Mobile Models," 2024. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

[29] ONNX, "Open Neural Network Exchange (ONNX)." https://onnx.ai/

[30] ONNX, "ONNX Introduction and Documentation." https://onnx.ai/onnx/intro/

[31] NVIDIA, "Pruning and Distilling LLMs Using NVIDIA TensorRT Model Optimizer." https://developer.nvidia.com/blog/pruning-and-distilling-llms-using-nvidia-tensorrt-model-optimizer/

[32] IBM Research, "Pruning and Distilling LLMs Using NVIDIA TensorRT Model Optimizer," blog post, 2024. https://developer.nvidia.com/blog/pruning-and-distilling-llms-using-nvidia-tensorrt-model-optimizer/

[33] Y. Lu *et al.*, "Reassessing Layer Pruning in LLMs: New Insights and Methods," *arXiv preprint arXiv:2411.15558*, 2024.https://arxiv.org/abs/2411.15558

[34] S. A. Siddiqui *et al.*, "A Deeper Look at Depth Pruning of Large Language Models," *arXiv preprint arXiv:2407.14679*, 2024.https://arxiv.org/abs/2407.14679

[35] X. Ma *et al.*, "LLM-Pruner: On the Structural Pruning of Large Language Models," *arXiv preprint arXiv:2305.11627*, 2023.https://arxiv.org/abs/2305.11627

[36] G. Ling *et al.*, "SlimGPT: Layer-Wise Structured Pruning for Large Language Models," *arXiv preprint arXiv:2401.15024*, 2024.https://arxiv.org/abs/2401.15024

[37] S. Gao *et al.*, "DISP-LLM: Dimension-Independent Structural Pruning for Large Language Models," *arXiv preprint arXiv:2410.11988*, 2024.https://arxiv.org/abs/2410.11988

[38] F. Sandri *et al.*, "2SSP: A Two-Stage Framework for Structured Pruning of LLMs," *arXiv preprint arXiv:2501.09412*, 2025 https://arxiv.org/abs/2501.09412

[39] B. K. Kim *et al.*, "Shortened LLaMA: Depth Pruning for Large Language Models with Comparison of Retraining Methods," *arXiv preprint arXiv:2402.02834*, 2024. https://arxiv.org/abs/2402.02834

[40] J. Wee *et al.*, "Prompt-Based Depth Pruning of Large Language Models," *arXiv preprint arXiv:2502.04348*, 2025.https://arxiv.org/abs/2502.04348

[41] P. Liang *et al.*, "Holistic Evaluation of Language Models," *Transactions on Machine Learning Research*, vol. 1, no. 1, 2023.https://arxiv.org/abs/2211.09110

[42] D. Hendrycks *et al.*, "Measuring Massive Multitask Language Understanding," *arXiv preprint arXiv:2009.03300*, 2021.https://arxiv.org/abs/2009.03300

[43] Qwen Team, "Qwen3 Technical Report," *arXiv preprint arXiv:2505.09388*, 2025. https://arxiv.org/abs/2505.09388

[44] H. Liu *et al.*, "LLaVA-OneVision: Easy Visual Instruction Tuning," *arXiv preprint arXiv:2408.03326*, 2024.https://arxiv.org/abs/2408.03326

[45] IBM Research, "TerraMind: A Foundation Model for Earth Observation," *arXiv preprint arXiv:2504.11171*, 2025.https://arxiv.org/abs/2504.11171

[46] Meta AI, "LLaMA 3.2: Vision, Edge, and Mobile Models," 2024. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

[47] Muralidharan et al., Compact Language Models via Pruning and Knowledge Distillation, NeurIPS 2024 https://arxiv.org/abs/2407.14679

[48] *Layer importance estimation for structured depth pruning of large language models*2403.03853

[49]https://www.researchgate.net/publication/394080667_Investigating_Structural_Pruning_and_Recovery_Techniques_for_Compressing_Multimodal_Large_Language_Models_An_Empirical_Study?utm_source=chatgpt.com

[50] https://huggingface.co/papers/2503.01743

[51] Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, Ivan Titov https://arxiv.org/pdf/1905.09418

[52] https://arxiv.org/pdf/2310.05015

[53] "Compact Language Models via Pruning and Knowledge Distillation" V. Muralidharan et al. Venue: NeurIPS 2024 arXiv: https://arxiv.org/abs/2407.14679

[54] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning Both Weights and Connections for Efficient Neural Network," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, pp. 1135–1143, 2015. https://arxiv.org/abs/1506.02626

[55] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *International Conference on Learning Representations (ICLR)*, 2016. https://arxiv.org/abs/1510.00149

[56] T. Gale, E. Elsen, and S. Hooker, "The State of Sparsity in Deep Neural Networks," *arXiv preprint*, arXiv:1902.09574, 2019. https://arxiv.org/abs/1902.09574

[57] F. Frantar and D. Alistarh, "SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot," *International Conference on Machine Learning (ICML)*, 2023. Available: https://arxiv.org/abs/2301.00774

[58] J. Sun, Y. Li, S. Chen, Z. Lin, and S. Han, "A Simple and Effective Pruning Approach for Large Language Models," *arXiv preprint*, arXiv:2306.11695, 2023. https://arxiv.org/abs/2306.11695

[59] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, pp. 1135–1143, 2015.https://arxiv.org/abs/1506.02626

[60] M. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019. Available: https://arxiv.org/abs/1905.10650

[61] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, "Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned," *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 5797–5808, 2019. Available: https://arxiv.org/abs/1905.09418

[62] S. Prasanna, A. Rogers, and S. Rumshisky, "Block pruning for faster transformers," *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.https://arxiv.org/abs/2109.04838

[63] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "**LoRA: Low-Rank Adaptation of Large Language Models**," *International Conference on Learning Representations (ICLR)*, 2022.https://arxiv.org/abs/2106.09685

[64] J. Yu, Z. Qin, Y. Wang, and J. Zhu, "**On the Limitations of Parameter-Efficient Fine-Tuning for Large Language Models**," *arXiv preprint*, arXiv:2307.08701, 2023. Available: https://arxiv.org/abs/2307.08701

[65] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "**QLoRA: Efficient Finetuning of Quantized LLMs**," *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.https://arxiv.org/abs/2305.14314

[66] P. Liang, R. Bommasani, T. Lee, D. Tsipras, I. Sorensen, and Y. Zhang, "**Holistic Evaluation of Language Models**," *Transactions on Machine Learning Research (TMLR)*, 2023. Available: https://arxiv.org/abs/2211.09110

[67] S. Gehrmann, A. R. M. Rush, and J. D. S. Poliak, "**The Limitations of Perplexity as a Measure of Language Model Quality**," *arXiv preprint*, arXiv:2401.04545, 2024. https://arxiv.org/abs/2401.04545

[68] A. Raji, M. R. Bender, J. Dodge, and M. Mitchell, "**Evaluating the Evaluation of Language Models**," *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. https://arxiv.org/abs/2108.03370

[69] J. Lu, D. Batra, D. Parikh, and S. Lee, "**ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks**," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019. https://arxiv.org/abs/1908.02265

[70] H. Tan and M. Bansal, "**LXMERT: Learning Cross-Modality Encoder Representations from Transformers**," *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.https://arxiv.org/abs/1908.07490

[71] J. Li, D. Li, C. Xiong, and S. Hoi, "**BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation**," *International Conference on Machine Learning (ICML)*, 2022. https://arxiv.org/abs/2201.12086

[72] G. Camps-Valls, D. Tuia, X. X. Zhu, and B. Demir, "**Deep learning for remote sensing data analysis**," *IEEE Geoscience and Remote Sensing Magazine*, vol. 6, no. 4, pp. 8–36, 2018. Available: https://ieeexplore.ieee.org/document/8450833

[73] X. X. Zhu, D. Tuia, L. Mou, G. Xia, L. Zhang, F. Xu, and F. Fraundorfer, "**Deep learning in remote sensing: A comprehensive review and list of resources**," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 4, pp. 8–36, 2017 https://ieeexplore.ieee.org/document/8241763

[74] L. Mou, Y. Hua, and X. X. Zhu, "**A relation-augmented fully convolutional network for semantic segmentation in remote sensing images**," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 10, pp. 7555–7569, 2019.

[75] A. Raji et al., "Evaluating the Evaluation of Language Models," *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.https://arxiv.org/abs/2108.03370

[76] P. Liang et al., "Holistic Evaluation of Language Models," *Transactions on Machine Learning Research (TMLR)*, 2023. https://arxiv.org/abs/2211.09110

[77] T. Dettmers, M. Lewis, L. Zettlemoyer, and W. Zaremba, "LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale," *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.https://arxiv.org/abs/2208.07339

[78] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *International Conference on Learning Representations (ICLR)*, 2016.https://arxiv.org/abs/1510.00149

[79] S. Luo, Y. Zhang, and J. Li, "Reassessing Layer Pruning for Transformer-Based Models," in Proc. NeurIPS, 2023.

[80] J. Lu, Z. Chen, and K. Huang, "A Deeper Look at Depth Pruning in Large Language Models," arXiv:2306.11695, 2023.

[81] J. Fang et al., "LLM-Pruner: On the Structural Pruning of Large Language Models," arXiv:2305.11627, 2023.

[82] Z. Xiao et al., "SlimGPT: Layer-wise Structured Pruning for Large Language Models," arXiv:2305.05162, 2023.

[83] P. Liang et al., "Holistic Evaluation of Language Models," ACM Trans. Intell. Syst. Technol., vol. 14, no. 4, 2023.

[84] J. Fang et al., "Revisiting Parameter-Efficient Fine-Tuning for Pruned Language Models," arXiv:2310.02300, 2023.

[85] S. Zhou et al., "Reassessing the Effectiveness of Layer-Wise Fine-Tuning After Model Pruning," arXiv:2309.14509, 2023.

[86] T. Sun et al., "On the Limits of Model Pruning for Large Language Models," arXiv:2311.09124, 2023.

[87] A. Raji et al., "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList," ACL, 2020.

[88] Y. Zhang et al., "PuDDing: Prompt-Driven Dynamic Pruning for Large Language Models," arXiv:2402.01982, 2024.

[89] Y. Xu et al., "GETA: Structured Pruning Meets Quantization for Efficient LLM Inference," arXiv:2312.01528, 2023.

[90] A. Fedus, B. Zoph, and N. Shazeer, "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity," JMLR, 2022.

[91] Cobbe, K., Kosaraju, V., Bavarian, M., et al. *Training Verifiers to Solve Math Word Problems*. arXiv preprint arXiv:2110.14168, 2021.

[92] Hendrycks, D., Burns, C., Kadavath, S., et al. *Measuring Mathematical Problem Solving With the MATH Dataset*. Advances in Neural Information Processing Systems (NeurIPS), 2021.

[93] Yu, L., Jiang, W., Shi, S., et al. *MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models*. arXiv preprint arXiv:2309.12284, 2023.

[94] Yang, Z., Qi, P., Zhang, S., et al. *HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering*. Proceedings of EMNLP, 2018.

[95] Geva, M., Khashabi, D., Segal, E., et al. *Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies*. Transactions of the Association for Computational Linguistics (TACL), 2021.

[96] Mishra, S., Khashabi, D., Baral, C., et al. *Cross-Task Generalization via Natural Language Crowdsourcing Instructions*. Proceedings of ACL, 2022.

[97] Clark, P., Cowhey, I., Etzioni, O., et al. *Think You Have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge*. arXiv preprint arXiv:1803.05457, 2018.

Chapter 13: Appendix

Appendix A – Pruning Configuration Summary

Model	Eval Setup	Pruning Type	Layers (Before → After)	Parameters (M) Before → After	PPL (Before → After)	Pruning Time (s)
LLaVA-OneVision-0.5B	Prompt	Width (2%)	24 → 24	893.68 → 887.42	4.54 → 6.32×10^{9}	0.3270
LLaVA-OneVision-0.5B	Prompt	Depth (6,13)	24 → 22	893.68 → 863.85	4.54 → 3.58	0.0025
LLaVA-OneVision-0.5B	WT2	Width (2%)	24 → 24	893.68 → 887.42	21.65 → 78.92	0.3315
LLaVA-OneVision-0.5B	WT2	Depth (9,13)	24 → 22	893.68 → 863.85	21.65 → 26.14	0.0058
Qwen3-0.6B	Prompt	Width (2%)	28 → 28	596.05 → 590.80	2.42 → 2.51	0.2809
Qwen3-0.6B	Prompt	Depth (13,23)	28 → 26	596.05 → 564.59	2.42 → 2.85	0.0051
Qwen3-0.6B	WT2	Width (2%)	28 → 28	596.05 → 590.80	23.88 → 39.34	0.1877
Qwen3-0.6B	WT2	Depth (12,14,15)	28 → 25	596.05 → 548.86	23.88 → 28.40	0.0059
Qwen3-8B	Prompt	Depth (11%)	36 → 32	8190.74 → 7418.95	2.91 → 3.53	0.0060
Qwen3-8B	Prompt	Width (11%)	36 → 36	8190.74 → 7586.90	2.91 → 4.17	10.5330
Qwen3-8B	WT2	Depth (11%)	36 → 32	8190.74 → 7418.95	10.94 → 12.79	0.0061
Qwen3-8B	WT2	Width (11%)	36 → 36	8190.74 → 7593.10	10.94 → 59.14	10.4816
Qwen3-8B	Prompt	Depth (30%)	36 → 25	8190.74 → 6068.32	2.91 → 1.29	0.0058
Qwen3-8B	Prompt	Width (30%)	36 → 36	8190.74 → 6560.17	2.91 → 3281.36	24.8980
Qwen3-4B-Instruct	Prompt	Depth (30%)	36 → 25	4022.47 → 2912.23	4.47 → 2.04	0.0057
Qwen3-4B-Instruct	Prompt	Width (30%)	36 → 36	4022.47 → 3215.70	4.47 → 1.63	15.7329
Phi-4-mini-Instruct	Prompt	Depth (11%)	32 → 28	3836.02 → 2930.00	5.83 → 1.28	0.0052

Table A.1: *Summary of all structured pruning experiments conducted across Qwen3, Phi-4 Mini, and LLaVA-OneVision models. The table reports evaluation setup, pruning strategy, layer reduction, parameter reduction, perplexity changes, pruning execution time, and corresponding implementation scripts used throughout the thesis.*

Appendix B – Benchmarking Summary

Model	Configuration	BBH ↑	MATH Hard ↑	MuSR ↑
LLaVA-OneVision-0.5B	Baseline	28.78	0.00	36.64
LLaVA-OneVision-0.5B	Depth (22/24)	28.78	0.00	37.57
LLaVA-OneVision-0.5B	Width (2%)	28.21	0.00	37.30
Qwen3-0.6B	Baseline	28.33	8.16	31.35
Qwen3-0.6B	Depth (~5%)	27.76	3.55	33.20
Qwen3-0.6B	Width (2%)	28.36	4.91	36.38
Qwen3-8B	Baseline	29.30	17.30	40.60
Qwen3-8B	Depth (11%)	31.05	3.10	38.36
Qwen3-8B	Depth (30%)	28.09	0.08	36.11
Qwen3-8B	Width (11.1%)	25.93	1.81	41.67
Qwen3-8B	Depth (11%) + LoRA	43.59	7.80	32.54
Qwen3-8B	Depth (30%) + LoRA	36.16	1.59	35.71
Qwen3-4B-Instruct	Baseline	54.54	45.85	40.61
Qwen3-4B-Instruct	Depth-Pruned (30%)	29.25	1.44	36.24
Qwen3-4B-Instruct	WT2-Derived Depth (30%)	29.37	1.28	38.49
Qwen3-4B-Instruct	WT2 Random Depth (30%)	30.29	0.98	38.49
Phi-4-mini-Instruct	Baseline	56.40	16.80	39.20
Phi-4-mini-Instruct	Depth (11%)	32.32	0.00	40.48

Table B.1: *BBH, MATH Hard, and MuSR benchmark results for baseline, depth-pruned, width-pruned, and LoRA-recovered models. Mathematical reasoning exhibits the highest sensitivity to structured pruning, while MuSR reasoning is comparatively more robust. LoRA recovery partially restores performance, particularly for BBH reasoning tasks.*

Model	Configuration	MMLU-Pro Accuracy ↑
Qwen3-8B	Dense Baseline	11.36
Qwen3-8B	Depth-Pruned (~11%)	12.36
Qwen3-8B	Depth-Pruned (30%)	11.70
Qwen3-4B-Instruct	Dense Baseline	43.94
Qwen3-4B-Instruct	Depth-Pruned (30%)	12.16
Phi-4-mini-Instruct	Dense Baseline	39.30
Phi-4-mini-Instruct	Depth-Pruned (11%)	22.83

Table B.2: *MMLU-Pro accuracy for baseline and depth-pruned models. While moderate depth pruning can occasionally preserve or slightly improve performance on specific tasks, aggressive pruning generally leads to substantial degradation in broad-domain professional reasoning capabilities.*

Appendix C – Efficiency Summary

Model	Pruning Type	TTFT (s) ↓	Tokens/s ↑	Peak VRAM (GB) ↓
LLaVA-OneVision-0.5B	Depth	0.028 → 0.025	53.0 → 56.0	1.70 → 1.70
LLaVA-OneVision-0.5B	Width	0.024 → 0.027	46.2 → 46.2	1.70 → 1.70
Qwen3-0.6B	Depth	0.034 → 0.032	31.8 → 34.9	1.13 → 1.13
Qwen3-0.6B	Width	0.032 → 0.032	33.8 → 33.8	1.13 → 1.14
Qwen3-4B-Instruct	Depth (30%)	0.040 → 0.031	25.7 → 36.2	7.58 → 7.59
Qwen3-4B-Instruct	Width (30%)	0.042 → 0.044	25.2 → 25.3	7.58 → 6.05
Qwen3-8B	Depth (30%)	0.041 → 0.033	25.8 → 36.4	15.29 → 15.31
Qwen3-8B	Width (30%)	0.042 → 0.044	25.3 → 25.6	15.28 → 12.28
Phi-4-mini-Instruct	Depth (11%)	0.028 → 0.027	37.3 → 42.0	7.17 → 7.20

Table C.1: *Inference efficiency metrics before and after structured pruning. Time-to-First Token (TTFT), token generation throughput (Tokens/s), and peak GPU memory consumption are reported for each model. Depth pruning generally provides larger latency and throughput improvements, whereas width pruning primarily reduces memory consumption. The largest efficiency gains are observed for the Qwen3-4B-Instruct and Qwen3-8B models, where 30% depth pruning substantially improves generation throughput while maintaining practical deployment compatibility.*

Model	Method	TTFT Change (%) ↓	Tok/s Change (%) ↑	Peak VRAM Change (%) ↓
LLaVA-OneVision-0.5B	Depth	-10.71	+5.66	0.00
LLaVA-OneVision-0.5B	Width	+12.50	0.00	0.00
Qwen3-0.6B	Depth	-5.88	+9.75	0.00
Qwen3-0.6B	Width	0.00	0.00	+0.88
Qwen3-4B-Instruct	Depth 30%	-22.50	+40.86	+0.13
Qwen3-4B-Instruct	Width 30%	+4.76	+0.40	-20.18
Qwen3-8B	Depth 30%	-19.51	+41.09	+0.13
Qwen3-8B	Width 30%	+4.76	+1.19	-19.63
Phi-4-mini-Instruct	Depth 11%	-3.57	+12.60	+0.42

Table C.2: *Relative percentage changes in Time-to-First Token (TTFT), token generation throughput, and peak GPU memory consumption after structured pruning. Negative TTFT values indicate latency reductions, positive Tokens/s values indicate throughput improvements, and negative VRAM values indicate memory savings. Depth pruning consistently delivers the largest throughput improvements, while width pruning primarily reduces memory consumption in larger models. The strongest efficiency gains are observed for the 30% depth-pruned Qwen3-4B-Instruct and Qwen3-8B models, achieving over 40% improvements in token generation throughput.*

Appendix D – TerraMind UseCase Summary

Model	Configuration	Validation Accuracy (%) ↑
TerraMind-1.0-Base	Baseline	83.84
TerraMind-1.0-Base	33% Depth-Pruned	81.57
TerraMind-1.0-Base	33% Pruned + LoRA	95.56
TerraMind-1.0-Large	Baseline	89.44
TerraMind-1.0-Large	33% Depth-Pruned	85.25
TerraMind-1.0-Large	33% Pruned + LoRA	96.60

Table D.1: *EuroSAT classification accuracy for baseline, depth-pruned, and LoRA-recovered TerraMind models. LoRA recovery not only restores performance lost due to pruning but surpasses baseline accuracy for both model sizes*

Model	Configuration	Blocks (Before → After)	Parameters (M) Before → After	Latency (ms) ↓	GPU Memory (MB) ↓
TerraMind-1.0-Base	Baseline	12 → 12	87.71 → 87.71	5.42	371.5
TerraMind-1.0-Base	33% Depth-Pruned	12 → 8	87.71 → 59.39	3.92	196.0
TerraMind-1.0-Large	Baseline	24 → 24	305.66 → 305.66	11.16	1119.1
TerraMind-1.0-Large	33% Depth-Pruned	24 → 20	305.66 → 255.33	9.28	642.1

Table D.2: *Impact of 33% depth pruning on TerraMind efficiency metrics. Pruning reduces transformer depth, parameter count, inference latency, and GPU memory consumption while maintaining strong downstream performance.*

Appendix E – LoRA Recovery Datasets

Benchmark Target	Recovery Datasets
BBH	Natural Instructions, StrategyQA, HotpotQA
MATH Hard	GSM8K, MATH, MetaMathQA
MuSR	HotpotQA, StrategyQA, Natural Instructions
MMLU-Pro	Natural Instructions, AI2 ARC, StrategyQA
Multitask	All datasets combined

Table E.1: *Capability-aligned recovery datasets used for LoRA fine-tuning after structured pruning. Rather than directly fine-tuning on evaluation benchmarks, proxy datasets were selected to target the reasoning capabilities measured by BBH, MATH Hard, MuSR, and MMLU-Pro, thereby reducing benchmark contamination and evaluation leakage while enabling post-pruning recovery experiments*.

Appendix F – LoRA Hyperparamters

Parameter	Value
LoRA Rank	16
LoRA Alpha	32
LoRA Dropout	0.05
Learning Rate	2e-4
Epochs	1–3
Sequence Length	2048

Table F.1: *Hyperparameters used for LoRA-based recovery fine-tuning experiments*.

Parameter	Value
Temperature	0
Top-p	1.0
Max Tokens	300
Backend	vLLM

Table F.2: Generation and evaluation settings