Performance characterization and energy efficiency of GPU-accelerated AI systems: from deep learning optimization to distributed LLM training

by

Muhammad Ali Shafique

M.S., University of Engineering and Technology Lahore, 2017

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2026

Abstract

The increasing computational demands of artificial intelligence (AI) make efficient GPU execution important for training phase, inference stage, and infrastructure design. Reduced numerical precision, multi-GPU training strategies, and newer accelerators can improve performance, but their benefits depend on workload characteristics and type. Fast execution and high GPU utilization do not necessarily result in low energy consumption. This dissertation studies these trade-offs with three studies at the model, training, and infrastructure levels.

At the model level, TensorFlow automatic mixed precision and TensorFlow–TensorRT FP32, FP16, and INT8 execution are evaluated using three vision architectures on CIFAR-10 and a Cats vs Dogs subset. The results show that the accuracy, memory, training time, and inference performance depend on the model architecture, precision, and batch size.

At the training-system level, full-parameter fine-tuning of 3B and 8B Llama models is evaluated on eight NVIDIA H100 GPU using multi-GPU techniques such as DP, DDP, FSDP, and ZeRO. For the 3B workload, ZeRO-2 achieves approximately 6,844 tokens/kJ compared to 4,537 tokens / kJ for DDP, improving energy efficiency by 51%. More aggressive sharding does not consistently improve efficiency, and the selection of strategies depends on the size and memory requirements of the model.

At the infrastructure level, eight-GPU H100 and B200 systems are compared across five language models and three vision-language models. B200 reduces the duration of language model training by almost 11–15% while consuming more GPU energy. Vision-language results reveal workload dependent platform behavior. Facility scenarios project the electricity consumption and cost implications of these differences.

Together, these findings support workload aware selection of reduced precision, multi-GPU training strategies, and the newer GPU platforms under the accuracy, memory, work completion time, and energy profiles in training and inference stages.

Performance characterization and energy efficiency of GPU-accelerated AI systems: from deep learning optimization to distributed LLM training

by

Muhammad Ali Shafique

M.S., University of Engineering and Technology Lahore, 2017

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2026

Approved by:

Co-Major Professor
Arslan Munir

Approved by:

Co-Major Professor
Don Gruenbacher

Copyright

© Muhammad Ali Shafique 2026.

Abstract

The increasing computational demands of artificial intelligence (AI) make efficient GPU execution important for training phase, inference stage, and infrastructure design. Reduced numerical precision, multi-GPU training strategies, and newer accelerators can improve performance, but their benefits depend on workload characteristics and type. Fast execution and high GPU utilization do not necessarily result in low energy consumption. This dissertation studies these trade-offs with three studies at the model, training, and infrastructure levels.

At the model level, TensorFlow automatic mixed precision and TensorFlow–TensorRT FP32, FP16, and INT8 execution are evaluated using three vision architectures on CIFAR-10 and a Cats vs Dogs subset. The results show that the accuracy, memory, training time, and inference performance depend on the model architecture, precision, and batch size.

At the training-system level, full-parameter fine-tuning of 3B and 8B Llama models is evaluated on eight NVIDIA H100 GPU using multi-GPU techniques such as DP, DDP, FSDP, and ZeRO. For the 3B workload, ZeRO-2 achieves approximately 6,844 tokens/kJ compared to 4,537 tokens / kJ for DDP, improving energy efficiency by 51%. More aggressive sharding does not consistently improve efficiency, and the selection of strategies depends on the size and memory requirements of the model.

At the infrastructure level, eight-GPU H100 and B200 systems are compared across five language models and three vision-language models. B200 reduces the duration of language model training by almost 11–15% while consuming more GPU energy. Vision-language results reveal workload dependent platform behavior. Facility scenarios project the electricity consumption and cost implications of these differences.

Together, these findings support workload aware selection of reduced precision, multi-GPU training strategies, and the newer GPU platforms under the accuracy, memory, work completion time, and energy profiles in training and inference stages.

Table of Contents

List of Figures xiv

List of Tables xvi

List of Nomenclature xvii

Acknowledgements xxi

1 Introduction 1

1.1 Motivation 2

1.1.1 Performance and Memory Constraints in Deep Learning 2

1.1.2 Energy-Aware Multi-GPU Training 3

1.1.3 Workload-Aware GPU Infrastructure Selection 4

1.2 Research Problem 4

1.3 Research Objectives 5

1.4 Dissertation Contributions 7

1.5 Dissertation Organization 8

2 Background and Related Work 11

2.1 GPU Based Deep Learning Systems 11

2.1.1 Training and Inference 12

2.1.2 Workload Geometry and Resource Bottlenecks 12

2.2 Numerical Precision and Quantization 13

2.2.1 Floating-Point and Integer Formats 13

2.2.2 Affine and Symmetric Quantization 13

2.2.3 Calibration and Post-Training Quantization . . . 14
2.2.4 Quantization-Aware and Mixed-Precision Training . . . 15
2.3 Inference Optimization with TensorRT and TF-TRT . . . 15
2.4 Multi-GPU Training Strategies . . . 16
2.4.1 Data Parallelism and Distributed Data Parallelism . . . 16
2.4.2 Fully Sharded Data Parallelism . . . 16
2.4.3 DeepSpeed ZeRO . . . 17
2.4.4 Parallelism and Sharding Layouts . . . 17
2.4.5 Memory and Communication Trade-offs . . . 17
2.5 Performance and Energy-Efficiency Metrics . . . 21
2.5.1 Accuracy, Latency, Throughput, and Memory . . . 21
2.5.2 Power and Cumulative Energy . . . 22
2.5.3 Tokens per Kilojoule and Computational Efficiency . . . 22
2.6 GPU Infrastructure and Workload-Aware Evaluation . . . 23
2.6.1 Thermal Conditions and Power Management . . . 23
2.6.2 Facility Energy and Operating Cost . . . 24
2.7 Related Work and Research Gaps . . . 24
2.7.1 Reduced-Precision Model Execution . . . 24
2.7.2 Compression, Sparsity, and Framework-Level Acceleration . . . 24
2.7.3 Distributed Training and State Sharding . . . 25
2.7.4 Workload Energy and Infrastructure Efficiency . . . 25
2.7.5 Distributed Infrastructure and Heterogeneous Execution . . . 26
2.7.6 Inference Scheduling, Memory Hierarchy, and Power Control . . . 27
2.7.7 Position of the Dissertation . . . 27
2.8 Chapter Summary . . . 28

3 Model-Level Characterization through Quantization . . . 29
3.1 Introduction . . . 29

3.2 Study Design and Optimization Workflow . . . 30
3.3 Methodology . . . 30
3.3.1 Datasets . . . 30
3.3.2 Models . . . 31
3.3.3 Hardware . . . 32
3.3.4 Software Environment and Experimental Configuration . . . 32
3.3.5 Metrics . . . 33
3.4 Experimental Results . . . 34
3.5 Analysis of Training Performance . . . 34
3.5.1 Training Time and Batch Dependence . . . 34
3.5.2 Memory Usage and Workload Feasibility . . . 40
3.6 Analysis of Inference Performance . . . 41
3.6.1 Framework Optimization and Numerical Precision . . . 41
3.6.2 Batch Size and Interpretation of Latency . . . 41
3.7 Accuracy across Training and Inference Configurations . . . 42
3.8 Implications for Model-Level Optimization . . . 42
3.9 Scope and Limitations . . . 43
3.10 Chapter Summary . . . 43

4 Training-Level Characterization through Energy-Aware Multi-GPU Execution . . . 44
4.1 Introduction . . . 44
4.2 Parallelism and Evaluation Design . . . 45
4.3 Methodology . . . 45
4.3.1 Model Architecture . . . 45
4.3.2 Dataset and Preprocessing . . . 46
4.3.3 Training Framework . . . 46
4.3.4 Hardware and Software Setup . . . 47
4.3.5 Profiling and Instrumentation . . . 48

4.3.6 Energy Computation . . . 48
4.3.7 Measurement Protocol and Scope . . . 49
4.3.8 Configuration Summary . . . 49
4.4 Results . . . 50
4.4.1 Metric Definitions . . . 50
4.4.2 3B Model Results . . . 51
4.4.3 8B Model Results . . . 53
4.4.4 Cross-Model Energy–Performance–Memory Trade-offs . . . 54
4.5 Discussion . . . 55
4.5.1 GPU Utilization Trends . . . 55
4.5.2 Power and Energy Implications . . . 56
4.5.3 Training Duration . . . 57
4.5.4 Memory Behavior . . . 57
4.5.5 A Decision Framework for Energy-Aware Strategy Selection . . . 58
4.6 Measurement Boundaries and Interpretation . . . 60
4.7 Implications for the Dissertation . . . 60
4.8 Chapter Summary . . . 61

5 Infrastructure-Level Characterization through Workload-Aware GPU Evaluation . 62
5.1 Introduction . . . 62
5.2 Methodology and Experimental Setup . . . 63
5.2.1 Training Workloads . . . 63
5.2.2 Measurement and Profiling Methodology . . . 65
5.2.3 Benchmarking Objective . . . 66
5.3 Measured Performance and Energy Results . . . 70
5.3.1 LLM Performance and Energy Trade-offs . . . 70
5.3.2 VLM Performance and Workload Dependence . . . 71
5.4 Facility-Level Energy and Electricity-Cost Scenarios . . . 72

5.4.1 Scenario Assumptions . . . 73
5.4.2 Projected Energy and Cost . . . 73
5.5 Workload-Aware Infrastructure Selection . . . 74
5.6 Scope, Limitations, and Future Evaluation . . . 75
5.7 Chapter Summary . . . 75

6 Unified Analysis and Discussion . . . 76
6.1 Purpose and Scope of the Unified Analysis . . . 76
6.2 Shared Findings across Optimization Levels . . . 76
6.2.1 A Common Principle of Workload-Aware Selection . . . 76
6.2.2 Multi-Objective Efficiency . . . 77
6.3 Comparison of Experimental Scope and Evidence . . . 78
6.4 Practical Configuration-Selection Framework . . . 78
6.5 Summary . . . 79

7 Conclusions and Future Research . . . 80
7.1 Dissertation Overview . . . 80
7.2 Summary of Research Findings . . . 81
7.2.1 Model Level Optimization: Quantization and Mixed Precision . . . 81
7.2.2 Training-Level Optimization: Distributed Execution and Tokens/kJ . . . 82
7.2.3 Infrastructure-Level Optimization: GPU Platforms and Operating Conditions . . . 83
7.3 Dissertation Contributions . . . 84
7.4 Future Research Directions . . . 85
7.4.1 Joint Precision, Parallelism, and Platform Evaluation . . . 85
7.4.2 Multi-Node Scaling and Communication . . . 85
7.4.3 Power and Thermal Operating Policies . . . 85
7.5 Concluding Remarks . . . 85

Bibliography 87

Statement on the Use of Generative Artificial Intelligence 98

List of Figures

1.1 Dissertation research framework 9

2.1 DDP and FSDP layouts 18
2.2 ZeRO sharding layouts 19
2.3 Per-GPU training-state placement for each parallelism strategy 20

3.1 Training time speedup comparison with AMP for CIFAR-10 35
3.2 Memory performance comparison with AMP for CIFAR-10 36
3.3 CIFAR-10: Latency speedup comparison with TF-TRT for VGG16 37
3.4 Cats vs Dogs: Latency speedup comparison 37
3.5 CIFAR-10: Throughput improvement factor comparison 38
3.6 Cats vs Dogs: Throughput improvement factor comparison 39

4.1 Training time and average per-GPU power draw for the 3B model 53
4.2 Energy–memory–time trade-off overview 54
4.3 Octagonal GPU memory comparison across (a) DP vs. DDP memory distribution 56
4.4 Energy efficiency comparison (Tokens per kJ) across multi-GPU training techniques 58

5.1 Metrics classification based on measurement and calculation in this study . . 64
5.2 Tokens-per-kilojoule comparison for the five LLM workloads 68
5.3 Token-efficiency comparison for the three VLM workloads 69
5.4 Estimated TFLOPs/GPU comparison for LLM training on H100 and B200 . 69
5.5 training-duration and energy visualization from the platform study 70

5.6 Modeled annual facility electricity costs 74

List of Tables

3.1 Documented dataset configurations for the quantization study. 33

3.2 Training time and Speedup calculation with AMP for CIFAR-10 34

3.3 Training time and Speedup calculation with AMP for cats_and_dogs 35

3.4 Memory Reduction calculation with AMP for CIFAR-10 35

3.5 Memory Reduction calculation with AMP for cats_and_dogs 36

3.6 CIFAR-10: Latency speedup results 36

3.7 Cats vs Dogs: Latency speedup results 37

3.8 CIFAR-10: Throughput improvement factor results 38

3.9 Cats vs Dogs: Throughput improvement factor results 38

3.10 CIFAR-10: Accuracy for different models 39

3.11 Cats vs Dogs: Accuracy for different models 39

4.1 Documented configuration of the distributed-training experiments. 50

4.2 3B model: performance and telemetry measurements across training strategies 51

4.3 8B model: performance and telemetry measurements across training strategies 52

5.1 Documented workload settings for the H100–B200 comparison. 65

5.2 LLM execution and energy comparison for the evaluated H100 and B200 . . 67

5.3 VLM execution and energy comparison for the evaluated H100 and B200 . . 68

5.4 Facility scenario energy and cost 72

6.1 Scope of the three research studies 78

Nomenclature

Abbreviations

Term	Definition
AI	Artificial intelligence
AMP	Automatic mixed precision
API	Application programming interface
BF16	16-bit brain floating-point format (bfloat16)
CNN	Convolutional neural network
CPU	Central processing unit
CUDA	Compute Unified Device Architecture
DDP	Distributed Data Parallel
DNN	Deep neural network
DP	Data Parallel
FLOP	Floating-point operation
FP16	16-bit floating-point format
FP32	32-bit floating-point format
FSDP	Fully Sharded Data Parallel
FSDP-1 / FSDP-2	PyTorch FSDP version 1 / version 2
GPU	Graphics processing unit
HBM / HBM3	High-bandwidth memory / third-generation HBM
INT8	8-bit integer format
I/O	Input/output
IPMI	Intelligent Platform Management Interface

Term	Definition (continued)
IT	Information technology; the IT boundary used in PUE
LLM	Large language model
LoRA	Low-Rank Adaptation
MSE	Mean squared error
NVMe	Non-Volatile Memory Express
OOM	Out of memory
PEFT	Parameter-efficient fine-tuning
PTQ	Post-training quantization
PUE	Power usage effectiveness
QAT	Quantization-aware training
QLoRA	Quantized Low-Rank Adaptation
ReLU	Rectified linear unit
SFT	Supervised fine-tuning
SM	Streaming multiprocessor
TCO	Total cost of ownership
TDP	Thermal design power
TF–TRT	TensorFlow–TensorRT integration
TRL	Transformer Reinforcement Learning library
TRT	TensorRT
VLM	Vision-language model
VRAM	Video random-access memory
W&B	Weights & Biases
ZeRO	Zero Redundancy Optimizer
ZeRO-1 / -2 / -3	DeepSpeed Zero Redundancy Optimizer stages

Symbols

Term	Definition
x	Real-valued input to a quantizer
q, x_q	Quantized integer value
$\hat{x}$	Approximate reconstructed real value after dequantization
s, S	Positive quantization scale
z, Z	Integer quantization zero-point
$q_{\min}$, $q_{\max}$	Lower and upper bounds of the integer quantization range
$f_{\min}$, $f_{\max}$	Lower and upper real-valued calibration bounds in the quantization source
a, b	Lower and upper selected real-valued bounds
G	Number of GPUs within the stated measurement boundary
g	GPU index
i	Telemetry sample index
t, t_i	Elapsed time and timestamp of sample i
t_0, t_1	Start and end timestamps of a measurement interval
T	Training-run duration
Δt, Δt_i	Sampling interval; interval associated with sample i
$P_g(t)$	Instantaneous power of GPU g at time t, in watts
$\bar{P}$	Average per-GPU power in the supplementary accounting check
$\bar{P}_{\mathrm{IT}}$	Average IT power per node
E_{total}, E_{GPU}	Integrated GPU energy in joules when power is in watts and time in seconds
E_{kJ}, $E_{\mathrm{GPU,kJ}}$	GPU energy expressed in kilojoules
E_{facility}	Energy at the facility boundary
E_{IT}	Energy at the IT-equipment boundary

Term	Definition (continued)
N_{tok}, N_{token}	Processed training-token count over the measured interval
η_{tok}, η_{token}	Token-processing efficiency, in tokens/kJ
H	Annual operating hours used in facility projections
C	Electricity cost under the specified tariff

Units and Reported Metrics

Term	Definition
B (model size)	Billion parameters when used in model-size labels
GB / TB	Gigabytes / terabytes
s / ms	Seconds / milliseconds
°C	Degrees Celsius
W / kW	Watts / kilowatts; units of power
J / kJ	Joules / kilojoules; units of energy
kWh / GWh	Kilowatt-hours / gigawatt-hours; units of energy
TFLOPs/GPU	Reported achieved computational rate per GPU, in trillions of floating-point operations per second
TFLOPs/kW	Reported computational rate normalized by power in kilowatts
tokens/s	Processed tokens per second
tokens/kJ	Processed tokens per kilojoule within the declared energy boundary
images/s	Processed images per second
$/kWh	Electricity price per kilowatt-hour

Acknowledgments

I am very grateful to my major advisors, Dr. Arslan Munir and Dr. Don Gruenbacher, for their support, guidance, and mentorship throughout my doctoral work at Kansas State University. Their advice helped me develop as a researcher and connect questions in deep learning with the practical and real challenges of GPU performance and energy efficiency. I appreciate the time and attention they devoted to my work. I also thank Dr. Munir for the opportunity to pursue this research in ISCAAS Lab.

I also thank my dissertation committee members, Dr. Andrew Rys and Dr. Mitchell Neilsen, for their time, thoughtful feedback, and contributions to my doctoral work. I also thank the faculty and staff of the Department of Electrical and Computer Engineering at Kansas State University for their assistance throughout my studies.

I gratefully acknowledge the Fulbright Foreign Student Program for supporting my doctoral work and making this academic journey possible. Studying in the United States has provided opportunities to learn, collaborate, and develop a broader perspective on research and its applications.

I thank my coauthors, research collaborators, and ISCAAS lab colleagues for their discussions, shared expertise, and contributions to the studies in this dissertation. I also appreciate the assistance of the technical staff and computing resource providers who supported this experimental work. Their efforts helped make the evaluation of GPU based deep learning systems possible.

Finally, I thank my family and friends for their encouragement, patience, and support throughout this journey. Their understanding during demanding periods of research and writing has meant a great value to me. I am grateful for their continued belief in me and for the strength they have given me to complete this work.

Chapter 1

Introduction

Deep learning has become an important component of modern artificial intelligence (AI) that enables the advances in computer vision, natural language processing, robotics, and the recommender systems[1–4]. The development of large language models (LLMs) and vision–language models (VLMs) has further expanded the applications supported by deep neural networks. However, increasing the capacity of the model introduces substantial computational and infrastructure demands. Large models require significant memory capacity, extended training time, and large execution time during inference. At the infrastructure level, these demands convert into increased accelerator usage, power consumption, and cumulative energy cost[5;6]. Consequently, the efficient development and deployment of AI systems require joint consideration of model accuracy, execution performance and resource efficiency.

Graphics processing units (GPUs) provide the parallel computing capability that is required by large AI workloads. However, the availability of high-performance accelerators does not ensure efficient execution. Performance obtained from a GPU accelerator depends on numerical precision, model architecture, workload configuration, memory requirements, and the software used to execute the computation. The energy consumed by the AI system also depends on the way the power varies between workload characteristics and accelerator execution. These interactions make efficiency a system-level problem that cannot be understood from the accuracy of the model, the accelerator computational capability, or the

utilization of the GPU considered independently[7–9].

This dissertation investigates performance and the resource efficiency across three complementary levels of GPU based AI systems:

1. Numerical precision during training and inference stages
2. Choice of parallelism strategy when an LLM is trained on several GPUs
3. Accelerator selection for LLM and VLM training.

The dissertation begins with automatic mixed precision (AMP) and TensorFlow–TensorRT (TF-TRT) optimization for deep learning vision models, proceeds to the energy–memory–time trade-offs of multi-GPU LLM training, and concludes with a comparison of NVIDIA H100 and B200 GPU accelerators. Together, these studies examine how software and hardware choices affect the efficiency of the work within practical memory, time, and energy constraints.

1.1 Motivation

1.1.1 Performance and Memory Constraints in Deep Learning

The computational requirements of deep learning models affect both development and deployment phases. During training, long execution times slow down experimentation process, fine-tuning, and evaluation. In addition to model parameters, training stage requires memory for activations, gradients, and optimizer states. During inference stage, latency and throughput determine whether a model meets application requirements, particularly in real-time systems such as autonomous driving and robotics[10–12]. Improving accuracy is therefore only one part of the design problem; a model must also operate within the available computing and memory resources of the hardware.

Reduced precision computation provides a optimization mechanism for addressing these constraints. Quantization reduces the numerical precision used to represent model values.

Mixed precision quantization combines reduced precision with application operations and retains higher precision where it is needed during the training stage[13–18]. However, the benefits of these techniques depend on the implementation within an execution framework. The model architecture, input size, batch size, and precision mode can affect the final model accuracy, memory usage, training time, latency, and the throughput. Therefore, System-level profiling is necessary to determine which configurations provide desirable performance for a particular AI workload.

1.1.2 Energy-Aware Multi-GPU Training

The growth of LLMs extends the efficiency problem from one model execution to coordinated multi-GPU training. Scaling laws describe relationships among model size, training data, and computational requirements[19]. As these requirements increase, research institutions and cloud providers invest in large AI computing systems[20]. The associated power and energy demands encourage training methodologies that account for infrastructure constraints with model development goals[6;21].

Distributed Data Parallelism (DDP), Fully Sharded Data Parallelism (FSDP), and DeepSpeed Zero Redundancy Optimizer (ZeRO) provide different mechanisms for multi-GPU training across GPUs. Replication supports parallel execution but it increases memory requirements, while sharding reduces the training states and introduces additional communication. These methods consequently favor different regions of the energy–memory–train-to-time design space[8]. Their energy behavior requires additional measurement because reduced memory or higher utilization do not necessarily imply lower cumulative energy consumption.

An energy-efficiency figure means little unless it states how much work was done for the energy spent. For a comparable LLM training workload, Tokens per Kilojoule (Tokens/kJ) provides the relationship by expressing the number of processed tokens per unit of measured energy. Read next to training time and memory use, Tokens/kJ shows whether a strategy that finishes sooner, or needs less memory, is also spending its energy well[22–24]. For this reason the parallelism study looks at energy, memory, and time-to-train together instead of

choosing a strategy by throughput.

1.1.3 Workload-Aware GPU Infrastructure Selection

The GPU itself is a further variable. Each new generation brings more compute, more memory, higher bandwidth, and a different power envelope, yet a workload rarely speeds up in proportion to the datasheet. Measured GPU power changes with what the workload is doing and can sit well away from the rated value[7;25]. Two accelerators should therefore be compared by running real AI workloads on both and recording speed and energy, not by reading their specifications.

The point is sharpest when LLMs and VLMs are compared. Their different combinations of attention architecture, vision encoder processing, and other operations produce different execution profiles. A newer accelerator may improve training throughput but consume more energy for the same AI workload. Whereas a compute intensive workload may convert into faster execution and reduce the total energy consumption. At the facility level, these differences interact with the characteristics of the workload, the cooling overhead and the price of electricity[26]. Turning accelerator capability into a sound infrastructure decision thus depends on testing the actual workload.

1.2 Research Problem

The central research problem addressed in this dissertation is the lack of a integrated empirical understanding of how numerical precision, multi-GPU training strategy, and GPU platform selection affect the performance and system efficiency under real AI workloads. Each choice improves one objective while introducing costs in another objective. Lower precision can raise throughput at some cost to accuracy. Sharding the training state saves memory, yet the added communication lengthens the run. On a newer GPU the computation finishes sooner, though fewer tokens may be processed per unit of energy. Because these costs fall on different metrics, no one metric is enough to pick an efficient configuration.

The overarching research question is: *How can empirical profiling of reduced precision, multi-GPU training, and newer accelerator platforms guide the choice of AI system configurations that balance model accuracy, performance, memory requirements, and energy consumption?* It is split into three narrower problems, one for each study.

1. **Performance and execution-framework behavior:** What happens to accuracy, memory usage, training time, latency, and throughput when Automatic Mixed Precision (AMP) is turned on for training and TF-TRT precision modes are used for inference, and how does the answer change from one model or configuration to another?

2. **Multi-GPU training trade-offs.** Where do replicated and sharded parallelism strategies fall in the energy–memory–time-to-train space of LLM training, and can Tokens/kJ help pick a strategy when memory, time, or energy is limited?

3. **Cross-platform efficiency.** How do H100 and B200 GPU accelerators differ in computational performance, power consumption, and efficient work per unit energy across LLM and VLM workloads, and what do these differences imply for infrastructure planning?

These problems are connected by a common methodological principle: efficiency should be evaluated through the behavior of complete AI workloads with multiple performance metrics rather than inferred solely either from an optimization technique, or a single metric, or a hardware specification. The three studies provide complementary evidence at different levels of the computing system.

1.3 Research Objectives

The overall objective of this dissertation is to develop an empirical framework for selecting efficient and energy-aware GPU based AI system configurations for deep learning and LLM/VLM workloads. The specific objectives are as follows:

1. **Characterize reduced precision execution.** Evaluate TensorFlow AMP and TF-TRT inference optimization using VGG16, ResNet-50, and MobileNet_v1 using different datasets, image sizes, and batch sizes. Quantify the changes in accuracy, execution time, memory usage, inference latency, and throughput in training and inference stages.

2. **Measure multi-GPU training behavior.** Profile Data Parallelism (DP), DDP, FSDP-1, FSDP-2, and ZeRO stages 1–3 during supervised fine-tuning of 3B and 8B Llama models on a single-node, eight-GPU H100 system. Capture training time, memory usage, GPU utilization, power usage, temperature, and cumulative GPU energy.

3. **Energy aware multi-GPU training strategy selection.** Use Tokens/kJ together with memory requirements and time-to-train to identify configurations that are appropriate for different constraints. Also explains the cost associated with replication and training state sharding.

4. **Compare GPU platforms under real workloads.** Evaluate single-node, eight-GPU H100 and B200 configurations using DDP training across selected LLMs and VLMs. Relate training throughput and utilization to power consumption, total energy, TFLOPs/kW, and Tokens/kJ where it is applicable.

5. **Simulate AI workload measurements to infrastructure level.** Use measured platform values to inform workload placement and accelerator selection, and examine facility-level energy and electricity-cost projections under stated AI workload and simulated operation.

The objectives emphasize controlled workload comparisons and defined the measurement boundaries. In the distributed training study, the cumulative GPUs energy is obtained by integrating the sampled power across the GPUs. It is distinct from the energy of the whole node or facility, which includes additional components and overhead. Similarly, the facility-scale analysis in the accelerator study consists of modeled values rather than measurements from deployments containing thousands of nodes. The conclusions should be read with these boundaries in mind.

1.4 Dissertation Contributions

This dissertation contributes empirical framework and decision guidance in AI model execution, multi-GPU training, and GPU infrastructure. Its main contributions are as follows:

1. **A framework level characterization of mixed-precision training and post-quantized inference.** This dissertation evaluates AMP during training stage and FP32, FP16, and INT8 execution through TF-TRT during inference stage. By profiling multiple vision architectures and varying workload configurations, it characterizes the relationships among the model accuracy, training time, memory usage, inference latency, and the throughput. This contribution establishes the practical foundation for choosing the numerical precision and training configurations according to application requirements.

2. **A training-level methodology for evaluating multi-GPU LLM training efficiency.** The dissertation combines per-GPU telemetry with training measurements to characterize DP, DDP, FSDP, and ZeRO on an eight-GPU H100 platform. It also uses Tokens/kJ as a energy efficiency metric and integrates it with memory footprint and time-to-train design space. The contribution is the controlled, multidimensional evaluation methodology and it is used in the selection of multi-GPU strategies.

3. **An empirical decision framework for parallelism selection.** The multi-GPU training results demonstrate that no evaluated strategy dominates across all energy-memory-time objectives. FSDP and ZeRO-1/2 provide favorable combinations of runtime, energy efficiency, and memory usage in the configurations studied. ZeRO-3 provides better memory scalability but incurs additional runtime and cumulative energy costs, while DDP remains limited by replicated training states as model size increases. These findings help strategy selection according to whether memory capacity, energy consumption, or completion time is the dominant constraint.

4. **A workload-sensitive comparison of H100 and B200 GPU systems.** The same LLM and VLM workloads were trained with DDP on eight-GPU H100 and B200 nodes,

which keeps raw speed apart from energy efficiency. Every LLM trained faster on B200, but at lower TFLOPs/kW and Tokens/kJ. For the VLMs the outcome depended on the model; Vita-CLIP finished sooner on B200 and also used less energy in total. The comparison shows when a faster GPU generation turns into an energy gain and when it does not.

5. **Infrastructure guidance from modeled workload scenarios.** Node-level measurements are scaled up into modeled facility energy use and electricity cost. The projections show that choosing an accelerator involves the workload type, how busy the GPUs are, how much power they draw, and the limits the facility operates under. Taken over all three studies, the guidance rests on measured trade-offs rather than on any single performance number.

Together these contributions back the central argument of the thesis, namely that decisions on numerical precision, training strategy, and GPU infrastructure should be made per workload and judged on several metrics at once. The precision study shows how much the framework and the workload matter. The multi-GPU study carries the same reasoning over to energy, and the accelerator study tests it on two GPU generations. All findings hold for the models, configurations, software stacks, and measurement setup that were actually evaluated.

1.5 Dissertation Organization

The remainder of this dissertation is organized from numerical precision to multi-GPU training and GPU infrastructure as illustrated in Figure 1.1.

Chapter 2 gives the background and prior work that the three studies build on. It reviews GPU based deep learning, quantization and mixed precision, training and inference performance metrics, multi-GPU parallelism, and energy-efficiency measurement. It also introduces the distinction between performance metrics, cumulative energy consumption, and efficient work per unit of energy.

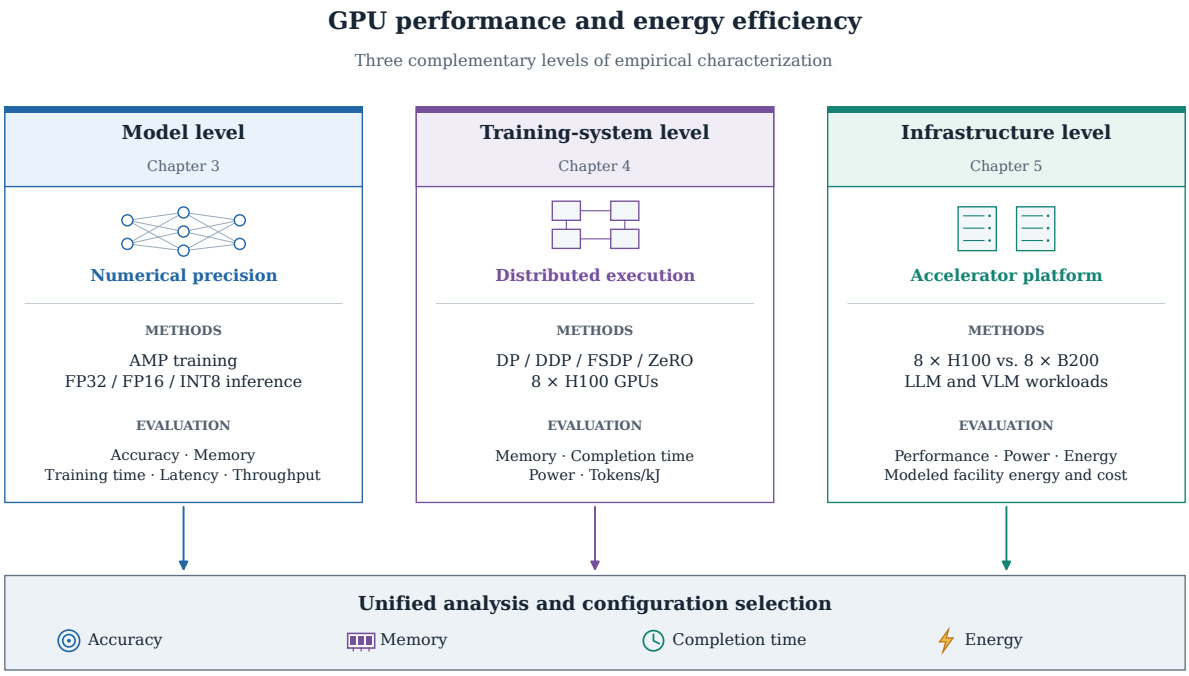


Figure 1.1: *Overview of the three research levels: model-level precision, multi-GPU training, and GPU infrastructure. Each study feeds into choosing a configuration when accuracy, memory, completion time, or energy is the limiting factor.*

Chapter 3 covers mixed-precision training and quantized inference on three vision models, beginning with how TensorFlow AMP and TF-TRT work and how the workloads were set up and profiled. The chapter analyzes accuracy, memory usage, training time, inference latency, and throughput across model architectures, datasets, image sizes, batch sizes, and the precision modes.

Chapter 4 turns to LLM training on multiple GPUs and the trade-offs it involves between energy, memory, and time. It presents the Tokens/kJ metric methodology and compares the DP, DDP, FSDP-1/2, and ZeRO stages 1–3 during fine-tuning on an eight-GPU H100 system. The chapter develops a design framework for selecting parallelism strategies under memory, energy, and completion-time conditions.

Chapter 5 compares the H100 and B200 GPU systems under distributed LLM and VLM training workloads. It analyzes the training performance, utilization, power, cumulative energy, and efficiency of work. The chapter then examines the implications of the measured behavior for accelerator selection and simulated facility-level energy and electricity costs.

Chapter 6 presents a unified analysis and discussion of the findings at the model-level, training-level, and infrastructure-level. It compares the objectives, develops guidance for workload-aware configuration, and examines the interactions that motivate integrated evaluation.

Chapter 7 concludes the dissertation by summarizing the research findings and contributions, discussing limitations, and identifying future directions in combined precision–parallelism–platform evaluation, multi-node scaling, and reproducible profiling.

Chapter 2

Background and Related Work

This chapter presents the concepts and prior research needed to understand the three studies developed in this dissertation. The discussion starts with GPU based deep learning and the distinction between training and inference. Next, it describes numerical precision, quantization, mixed-precision training, and inference optimization through TensorFlow–TensorRT (TF-TRT). Next, it reviews replicated and sharded multi-GPU training strategies, followed by performance and energy-efficiency metrics. Finally, it discusses workload-aware GPU infrastructure evaluation and identifies the research gaps addressed in Chapters 3–5.

2.1 GPU Based Deep Learning Systems

Deep neural networks learn parameterized representations from data and support applications ranging from image recognition to language processing[1–3]. Their execution involves tensor operations that can take advantage of the parallel computational resources of GPUs. However, the effective performance of a workload depends on the interaction between its operations, memory accesses, execution framework, and hardware resources. Consequently, a GPU's peak computational capability is an upper bound under particular conditions rather than a prediction of end-to-end application performance[9].

2.1.1 Training and Inference

Training consists of forward computation, loss evaluation, backpropagation, and parameter updates. In addition to parameters, the system stores intermediate activations, gradients, and optimizer states. These components can impose different memory requirements, depending on the optimizer, numerical precision, activation-management policy, and execution strategy. Inference uses trained parameters to produce output and generally excludes gradient computation and optimizer updates. Nevertheless, activations, framework buffers, and application-specific state still consume memory.

The distinction matters when evaluating optimization. A training configuration should support the numerical behavior required for learning, while an inference configuration should preserve sufficient accuracy and meet latency or throughput requirements for real-time applications. The precision study in Chapter 3 evaluates both stages using vision models. Chapters 4 and 5 focus on training workloads.

2.1.2 Workload Geometry and Resource Bottlenecks

The model architecture, batch size, and input dimensions influence the amount and shape of the computation. Increasing batch size can improve reuse and parallelism, but can also increase activation memory and may alter the optimization behavior of training. In vision models, image resolution influences intermediate tensor sizes. For transformer based language models, sequence length affects activation storage and attention computation. Different workload configurations can therefore encounter different bottlenecks even on the same hardware[7;9].

A compute-bound workload primarily limits performance through arithmetic execution, while a memory-bound workload is constrained by data movement across memory. Distributed execution introduces an additional communication dimension: synchronization and parameter exchange can delay computational execution unless they are sufficiently overlapped.

2.2 Numerical Precision and Quantization

Reduced precision execution changes the numerical representation used for computational tensors and operations. The intended advantages include small tensor memory and faster execution on compatible hardware. These benefits are conditional: conversion overhead, unsupported operators, kernel selection, and numerical sensitivity can reduce or reverse the expected improvement. Previous work on low-precision multiplication and vector quantization established important approaches to reducing the cost of neural network execution[13;14]. Practical deployment also requires evaluation of the frameworks that implement these approaches.

2.2.1 Floating-Point and Integer Formats

FP32 uses 32 bits per value, while FP16 and bfloat16 use 16 bits. FP16 allocates fewer exponent bits than bfloat16, giving the two formats different dynamic-range and precision properties. INT8 represents 256 integer values and requires a mapping between floating-point tensors and an integer range. The storage reduction for an individual tensor follows its bit width, but the reduction in total application memory is not necessarily proportional because other tensors and buffers can remain at higher precision[27;28].

Floating-point conversion and integer quantization should be distinguished. Both reduce representation cost, but integer quantization typically introduces an explicit scale and, in affine schemes, a zero-point. Mixed-precision training uses multiple floating-point formats to balance execution efficiency and numerical stability. It is not equivalent to quantization-aware training for integer inference.

2.2.2 Affine and Symmetric Quantization

A uniform affine quantizer maps a real value x into an integer value q as follows[27]:

$$q = \text{clip}\left(\text{round}\left(\frac{x}{s}\right) + z,\ q_{\min},\ q_{\max}\right), \tag{2.1}$$

where $s > 0$ is the scale, z is an integer zero-point, and $q_{\min}$ and $q_{\max}$ define the representable range. An approximate reconstructed value is

$$\hat{x} = s(q - z). \tag{2.2}$$

For a selected real interval $[a, b]$, a common scale is $s = (b - a)/(q_{\max} - q_{\min})$, with z chosen to map real zero into the integer domain. Rounding introduces discretization error, while clipping introduces error when a value falls outside the selected range.

In symmetric quantization, $z = 0$ and the range is centered on zero. For a restricted signed INT8 range of $[-127, 127]$, a common scale is $s = \max(|a|, |b|)/127$. Other integer-range conventions are possible, so the range and scale definition must be reported together. Per-tensor quantization uses one scale for a tensor, while per-channel quantization assigns separate scales to channels. Their accuracy and implementation costs depend on tensor distributions and hardware support[27].

2.2.3 Calibration and Post-Training Quantization

Post-training quantization (PTQ) converts an already trained model without repeating its original training procedure. Static activation quantization estimates ranges using representative calibration inputs before deployment. Dynamic quantization derives the selected activation ranges during execution. Dynamic range estimation can avoid a separate activation-calibration procedure but introduces runtime work; static calibration avoids that particular work during inference, but depends on how well calibration inputs represent deployment conditions[27].

Calibration is a methodological requirement rather than a conversion choice. Calibration samples represent the input distribution and evaluation data remain separate when reporting model quality. Neither PTQ nor a low bit width guaranties a particular accuracy change. The numerical error depends on the model, tensor distributions, quantizer configuration, and operator implementation.

2.2.4 Quantization-Aware and Mixed-Precision Training

Quantization-aware training (QAT) exposes a model to simulated quantization effects during training. Typical implementations use floating-point storage and fake quantization operations to emulate rounding and clipping effects in forward computation. The aim is to produce parameters better suited for post-quantized inference[27].

Mixed-precision training executes suitable operations in a lower floating-point precision and retains high precision for numerically sensitive components. Loss scaling can mitigate gradient underflow in FP16: the loss is multiplied by a scale before backpropagation, and gradients are unscaled before the optimizer update. The need for loss scaling depends on the numerical format and implementation. AMP automates precision selection within the framework, reducing the manual effort needed to configure mixed-precision execution[28].

2.3 Inference Optimization with TensorRT and TF-TRT

TensorRT is an inference optimization framework for NVIDIA hardware. TF-TRT integrates TensorRT execution into TensorFlow code by identifying compatible graph segments, converting supported operations, and building optimized engines. Unsupported operations remain in TensorFlow, producing a hybrid execution graph during inference[29;30].

The framework perform graph transformations and kernel selection, including operation fusion and the removal of unnecessary computation. FP32, FP16, and INT8 configurations provide different opportunities for acceleration, but a configured precision mode does not imply that every operator executes in that format. Unsupported operations, numerical requirements, and engine decisions can retain higher-precision computation. Therefore, application-level profiling is required to assess realized benefits in inference stage[29].

Engine construction and calibration also introduce costs that should be distinguished from steady state inference. Warm-up, input shapes, batch configuration, and timing boundaries affect latency and throughput in inference stage. The framework and precision choices

discussed here describe the experimental software context of Chapter 3.

2.4 Multi-GPU Training Strategies

Multi-GPU training distributes work and, depending on the method, training states across devices. Data parallelism assigns different input samples to workers, whereas model-parallel approaches divide model computation. This dissertation primarily studies replicated and sharded data-parallel execution. Tensor and pipeline parallelism provide additional scaling options, but are outside the evaluated strategy set.

2.4.1 Data Parallelism and Distributed Data Parallelism

DataParallel (DP) uses a single-process design to distribute inputs across devices and aggregate outputs and gradients. The primary device can incur additional memory and coordination costs. DP provides a useful baseline, but its execution structure differs from the one-process-per-GPU design used in DDP.

DDP maintains a model replica on each worker. Workers process local batches, compute gradients, and synchronize them through collective communication. Gradient synchronization can overlap with backpropagation, depending on gradient readiness and framework configuration[31]. DDP provides an effective replicated execution model when training states fit within each GPU's memory. However, adding workers does not divide the persistent parameter and optimizer-state storage required by each replica.

2.4.2 Fully Sharded Data Parallelism

Full-shard FSDP partitions parameters, gradients, and optimizer states across workers. Before a computation unit executes, the required parameter shards are gathered to form the parameters needed for that unit. Gradients are reduced and partitioned so that workers retain their assigned shards. Parameter materialization and resharding behavior depend on the implementation and configuration[32].

FSDP therefore reduces persistent state redundancy while introducing parameter communication and transient memory requirements. A worker still computes on its local input batch using materialized parameters; it does not merely compute a fractional slice of every layer because it stores a parameter shard. Wrapping granularity, prefetching, and the lifetime of gathered parameters affect execution behavior. FSDP-1 and FSDP-2 in Chapter 4 refer to the evaluated PyTorch implementation variants, rather than two universal sharding levels.

2.4.3 DeepSpeed ZeRO

ZeRO progressively partitions the training states that would otherwise be replicated in data-parallel training[33]. Stage 1 shards optimizer states. Stage 2 additionally shards gradients. Stage 3 also shards parameters and materializes them when required for computation. The stages offer different balances between persistent memory reduction and communication. Offloading can further change the memory hierarchy but should be reported separately because transferring states to CPU memory introduces additional execution considerations.

2.4.4 Parallelism and Sharding Layouts

DP uses a single-process DataParallel layout, whereas DDP uses replicated model state and synchronized gradients across multiple processes. Both FSDP variants use full sharding with CPU offloading disabled. ZeRO progressively partitions optimizer states, gradients, and parameters through stages 1, 2, and 3. Figures 2.1 and 2.2 illustrate the corresponding parallelism and state-sharding layouts. They describe state placement, rather than measured memory allocations. Figure 2.3 shows the per-GPU placement of training states for each strategy, together with the communication performed in each training step.

2.4.5 Memory and Communication Trade-offs

Let M_P, M_G, and M_O denote persistent parameter, gradient, and optimizer-state storage for an unsharded model. With G workers, idealized per-worker persistent memory is approxi-

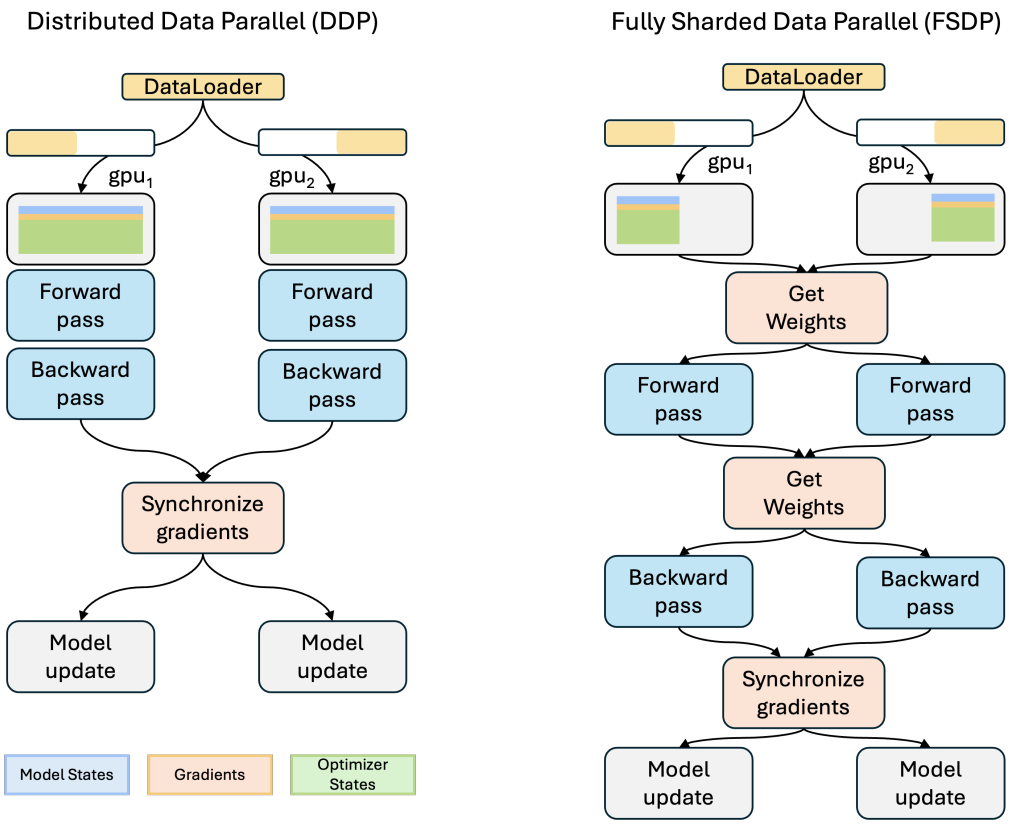


Figure 2.1: *Conceptual illustration of DDP and FSDP state layouts.*

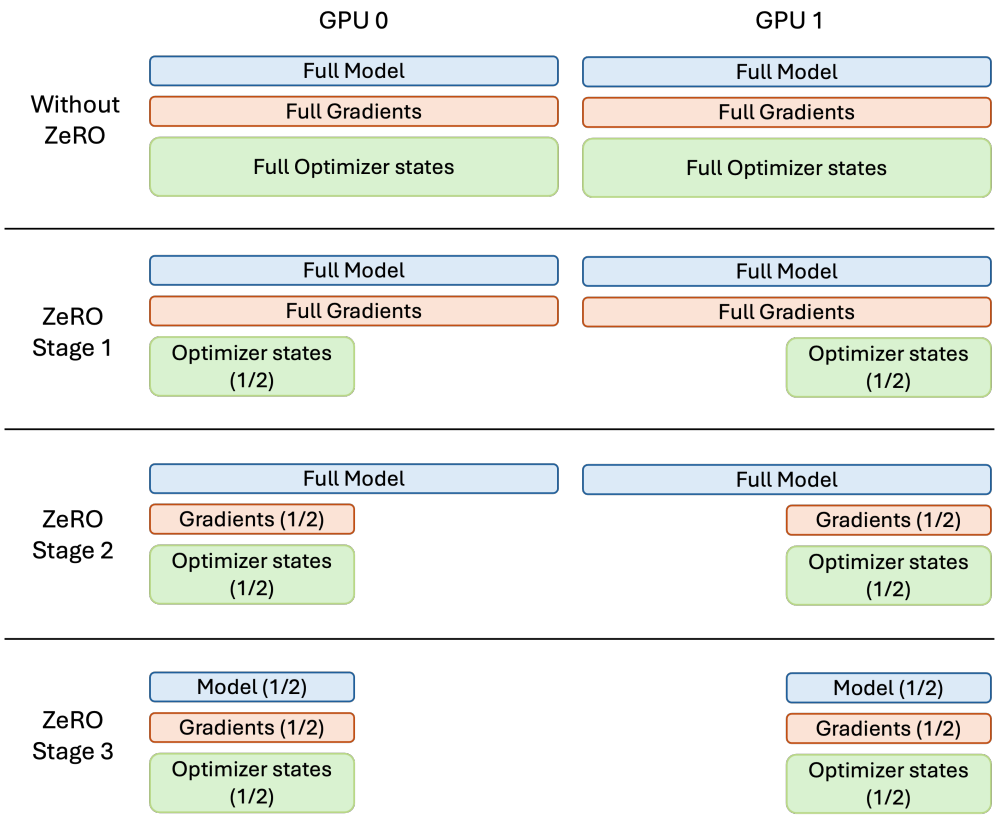


Figure 2.2: *Conceptual illustration of progressive state partitioning across ZeRO stages. The measured outcomes are reported in Chapter 4.*

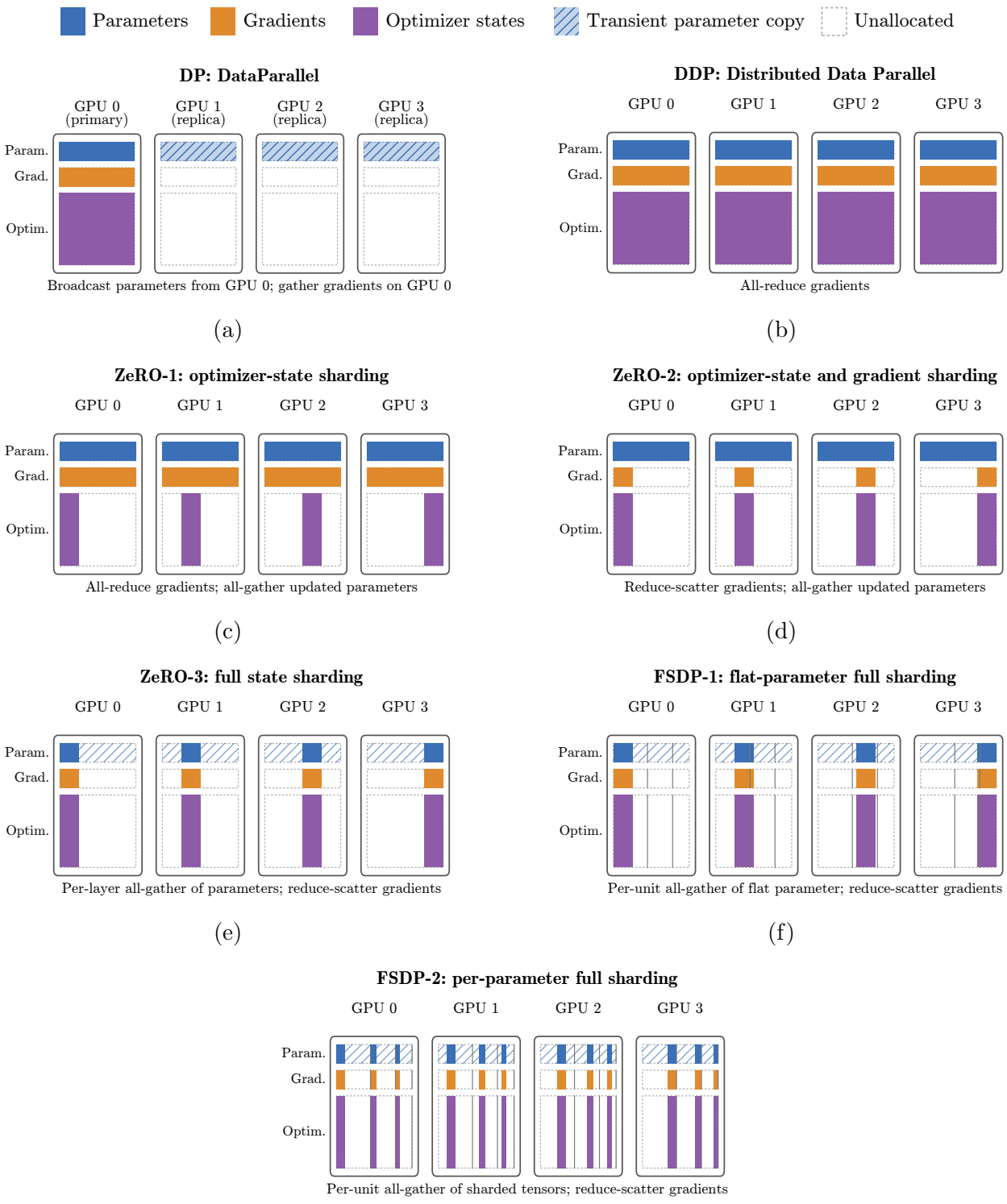


Figure 2.3: *Placement of parameters, gradients, and optimizer states on four GPUs for (a) DP, (b) DDP, (c)–(e) ZeRO stages 1–3, and (f)–(g) FSDP-1 and FSDP-2.*

mately

$$M_{\text{DDP}} \approx M_P + M_G + M_O, \tag{2.3}$$

$$M_{\text{ZeRO1}} \approx M_P + M_G + M_O/G, \tag{2.4}$$

$$M_{\text{ZeRO2}} \approx M_P + (M_G + M_O)/G, \tag{2.5}$$

$$M_{\text{full shard}} \approx (M_P + M_G + M_O)/G. \tag{2.6}$$

These expressions describe state partitioning, not peak allocated GPU memory. Activations, temporary parameter materialization, communication buffers, allocator behavior, and framework overhead must also be considered[32;33].

More aggressive sharding can enable workloads that exceed replicated memory capacity, but additional collectives can extend execution or change the overlap of communication and computation. Consequently, bandwidth characterization provides an important complement to memory analysis[8]. The most memory-efficient strategy is not necessarily the fastest or the most energy-efficient one.

2.5 Performance and Energy-Efficiency Metrics

2.5.1 Accuracy, Latency, Throughput, and Memory

Changes in numerical precision require measurements of the accuracy of the model and the performance of the execution. For classification workloads, accuracy reports the fraction of correctly classified samples according to a specified evaluation method. Latency measures completion time for a defined request or batch, and throughput expresses completed work per unit time.

Training throughput can be expressed as samples or tokens per second. Memory measurements should identify whether they describe allocated tensor memory, reserved framework memory, or device reported usage. Similarly, training duration requires a stated boundary, including whether model loading, initialization, warm-up, or evaluation is included or not.

These definitions are necessary for comparisons across frameworks and platforms[9].

2.5.2 Power and Cumulative Energy

Power is the rate of energy use measured in watts. Cumulative energy integrates power over time. For G participating GPUs and an execution interval $[0, T]$, GPU energy is

$$E_{\text{GPU}} = \sum_{g=1}^{G} \int_0^T P_g(t)\, \mathrm{d}t. \tag{2.7}$$

For sampled telemetry, a numerical approximation uses sample-specific time intervals:

$$E_{\text{GPU}} \approx \sum_{g=1}^{G} \sum_{i=1}^{N} P_g(t_i) \Delta t_i. \tag{2.8}$$

When power is in watts and time is in seconds, the result is in joules. The sampling cadence, sensor behavior, missing values, and measurement boundaries influence the estimate. Empirically calibrated power models provide a complementary approach when direct telemetry is incomplete, but their validity depends on the workloads and systems used for calibration[25].

A lower average power does not guaranty lower total energy. For completed work, a long run can consume more energy, despite drawing less instantaneous power. In contrast, fast execution can offset the higher power demand. Therefore, both duration and the power profile are required to understand the energy behavior[7].

2.5.3 Tokens per Kilojoule and Computational Efficiency

For N_{tok} processed training tokens and GPU energy E_{kJ}, the metric used in Chapter 4 is

$$\eta_{\text{tok}} = \frac{N_{\text{tok}}}{E_{\text{kJ}}}. \tag{2.9}$$

Token accounting must specify the treatment of padding, repeated epochs, and masked tokens, and use the same convention across compared configurations. Processing more tokens

itself is not evidence of better model quality. Tokens/kJ are most informative when comparing equivalent training work with comparable learning objectives, precision, and data handling. Previous training and inference efficiency studies motivate work-normalized energy evaluation[22–24].

For a fixed measurement boundary, tokens per second divided by average power in kilowatts is numerically equal to tokens per kilojoule. However, compute based and token based metrics emphasize different aspects of the work. TFLOPs/GPU describes a computational rate per accelerator. TFLOPs/kW normalize the rate by power.

2.6 GPU Infrastructure and Workload-Aware Evaluation

H100 and B200 represent the GPU platforms evaluated in Chapter 5. Differences in compute capability, memory resources, interconnects, and power behavior can affect their performance.

LLMs and VLMs provide complementary workload classes. Language models emphasize transformer computation, while selected video-oriented VLMs combine visual processing with language-image representations[34–36]. Their mix of operators can produce different utilization patterns. Attribution to a particular memory or compute bottleneck requires appropriate hardware profiling; utilization and power traces alone provide indirect evidence.

2.6.1 Thermal Conditions and Power Management

Thermal limits can affect clock behavior, while cooling system adds energy outside the accelerator limits. Studies of liquid-cooled and air-cooled H100 systems illustrate the need to report the operating environment when evaluating AI workload efficiency[37;38]. Power capping provides another control action: it limits the power of the device but can also change the runtime and the completed work per unit of energy[39;40]. Such controls should be evaluated against real AI workload instead of assumed to improve every metric.

2.6.2 Facility Energy and Operating Cost

Power usage effectiveness (PUE) relates the facility energy to the IT energy over a common interval:

$$\mathrm{PUE} = \frac{E_{\mathrm{facility}}}{E_{\mathrm{IT}}}. \tag{2.10}$$

The electricity cost can be estimated from the energy in kWh and the applicable rate. These operating costs are one component of total cost of ownership, which also includes acquisition and other lifecycle costs[26].

2.7 Related Work and Research Gaps

2.7.1 Reduced-Precision Model Execution

Low-precision multiplication, compressed representations, and adaptive numerical formats address the computing and storage cost of neural networks[13;14;16]. Mixed-precision training establishes mechanisms for preserving learning stability while accelerating suitable operations[28]. Quantization surveys organize PTQ, QAT, calibration, and quantizer-design choices[27]. TensorRT and TF-TRT provide an execution context in which these choices interact with graph optimization and kernel support[29;30].

This literature motivates the model-level question addressed in Chapter 3. The dissertation characterizes how existing precision and inference mechanisms behave across vision architectures and workload configurations.

2.7.2 Compression, Sparsity, and Framework-Level Acceleration

Quantization is one of several approaches to reducing the cost of neural network execution. Deep Compression combines pruning, trained quantization, and coding to reduce model representation requirements[41;42]. Structured sparsity targets groups of parameters rather than arbitrary individual weights[43], while Scalpel explicitly connects pruning decisions with underlying hardware parallelism[44]. Runtime pruning and convolutional kernel clustering

provide additional alternatives to reduce or restructure the work performed by vision models[45;46]. These approaches reinforce a main argument: a smaller representation or a low theoretical operation count does not specify the execution speed obtained from a particular framework and accelerator.

Training in reduced numerical precision introduces additional concerns beyond inference compression. Research on 8-bit training and the distributions of neural gradients tells how numerical representations interact with learning and gradient approximation[47;48]. Adaptive gradient quantization further considers quantization in data-parallel optimization[49]. These studies address mechanisms that differ from the AMP and TF-TRT configurations evaluated in Chapter 3. They provide context for separating numerical-method innovation from the dissertation's empirical assessment of existing framework mechanisms.

2.7.3 Distributed Training and State Sharding

DDP, FSDP, and ZeRO establish replicated and partitioned approaches to scaling training[31–33]. Their designs explain the relationships among state redundancy, parameter materialization, gradient synchronization, and memory capacity. Communication-oriented analysis further shows why bandwidth and overlap affect performance realized[8].

The methodological gap addressed in Chapter 4 is the connection between parallelism selection and a jointly measured energy–memory–time trade-off space. Existing scalability mechanisms do not, by themselves, determine which strategy will provide the best Tokens/kJ for a particular workload. Chapter 4 therefore compares the evaluated methods under a common fine-tuning setup and uses the resulting measurements to support constraint-aware strategy selection.

2.7.4 Workload Energy and Infrastructure Efficiency

Empirical node studies characterize differences between rated and observed GPU power during AI training[7;25]. Inference benchmarks show that the models and execution-engines also affect the normalized energy of work[22–24;50]. Performance models provide a complementary

framework for reasoning about compute, memory, and distributed execution[9]. The cooling, power-management, and life cycle studies extend the analysis to the operational infrastructure[26;37;39].

These studies establish substantial prior progress that the dissertation does not treat energy-aware AI as an unexplored area. The specific gap addressed in Chapter 5 is a workload sensitive comparison of the selected H100 and B200 configurations under a same DDP training protocol. Facility modeling further examines the operational implications under simulated scenario.

2.7.5 Distributed Infrastructure and Heterogeneous Execution

Training efficiency also depends on when workers execute and how they communicate. Geo-distributed training research examines training time duration and GPU utilization when communication spans geographically separated resources[51]. Zorse considers training efficiency on heterogeneous GPU cluster systems[52], while Rail-only examines network organization for large-scale model training[53]. These studies expand the execution problem beyond a homogeneous node: device imbalance, topology, and communication capability can affect whether additional resources improve the completed work per unit of time or not.

Infrastructure studies provide another but complementary perspective. Astral examines datacenter infrastructure for training at scale, and work on next-generation language model facilities considers the design of computing systems around growing model requirements[54;55]. Their scope differs from Chapter 4's controlled single-node strategy comparison. The dissertation uses a single node to study parallelism choices without experimentally varying geographic placement, network topology, or cluster heterogeneity. The results obtained in that setting motivate, instead of establish conclusions about broader deployments.

2.7.6 Inference Scheduling, Memory Hierarchy, and Power Control

Inference research also demonstrates why efficient hardware selection is a workload-aware decision. Studies of hybrid heterogeneous cluster systems and offline serving models examine how workload interacts with the differing energy characteristics of available hardware[56;57]. Predictive GPU throttling addresses another dimension by considering energy-aware execution for LLM serving[58]. These approaches complement GPU power capping. They change scheduling or operating policy.

Memory-system research further shows that serving large models depends on more than arithmetic capability. GPU-centric memory tiering on Grace Hopper systems examines how the memory hierarchy supports LLM serving[59]. Accelerator-level energy-cost modeling and datacenter TCO analysis with FP8 connect inference configuration with energy and economic objectives[60;61]. Their relevance to this dissertation is methodological: precision, memory movement, and hardware placement must be interpreted within an explicitly defined workload and cost limit. Their inference findings are not treated as experimental evidence for the training configurations evaluated in Chapters 4 and 5.

2.7.7 Position of the Dissertation

The three research chapters address complementary configuration decisions. Chapter 3 examines numerical precision and framework behavior for vision training and inference. Chapter 4 examines replicated and sharded execution for LLM fine-tuning. Chapter 5 examines GPU platform selection for LLM and VLM training. Their common contribution is an empirical approach to workload-aware selection using multiple metrics and explicit measurement boundaries.

2.8 Chapter Summary

This chapter established the numerical, multi-GPU training, and infrastructure concepts used throughout the dissertation. Reduced precision changes the representation and kernel opportunities. Replication and sharding in training, change memory and communication behavior GPU platform selection changes available resources and operating characteristics. Their effects must be evaluated through model accuracy, completed work time, memory, and energy under consistent definitions and settings. The following chapter applies these foundations to the characterization of mixed-precision training and quantized inference for vision models.

Chapter 3

Model-Level Characterization through Quantization

3.1 Introduction

This chapter investigates the first level of the dissertation's optimization framework: numerical precision and model execution. Deep learning systems must balance prediction quality with training time, memory requirements, and inference performance. Reduced-precision computation provides an opportunity to improve these objectives, but its practical benefits depend on the model, input dimensions, batch size, and execution framework[13;27;28]. Consequently, the availability of mixed-precision or quantization support is insufficient to establish the best configuration for a particular application.

The study characterizes TensorFlow automatic mixed precision (AMP) during training and TensorFlow–TensorRT (TF-TRT) during inference. Three vision architectures[42], VGG16, ResNet-50 and MobileNet_v1, are evaluated using CIFAR-10 and a Cats vs Dogs subset. Training is profiled with and without AMP, while inference is compared between the TensorFlow baseline and the TF-TRT FP32, FP16, and INT8 configurations. Measurements include accuracy, training time, memory usage, inference latency, and throughput.

The central question is how numerical precision and framework optimization affect the

performance–memory–accuracy trade-off space. The results demonstrate the importance of workload profiling for configuration selection, and provide the model-level foundation for the energy-aware distributed training study in Chapter 4.

3.2 Study Design and Optimization Workflow

The background mechanisms of AMP, quantization, calibration, and TF-TRT were introduced in Chapter 2. Here, AMP changes suitable training computations to lower floating-point precision while maintaining higher precision when needed. TF-TRT optimizes compatible segments of a trained TensorFlow graph for inference[29;30]. FP32 TF-TRT provides an important control: improvements over native TensorFlow in that mode can arise from framework and graph optimization without a reduction in numerical precision.

The experimental comparisons address three questions: whether AMP improves training time and memory usage; how the inference framework and precision settings affect latency and throughput; and whether the resulting configurations preserve classification accuracy. Batch-size sweeps are conducted separately for the two input resolutions. Because dataset and resolution change together, the comparisons do not independently isolate the causal effect of image size.

3.3 Methodology

The experimental setup is described below: the two datasets, the three CNN models, the GPU, the software stack, and the metrics used to judge each configuration.

3.3.1 Datasets

Two image-classification datasets, CIFAR-10 and Cats vs Dogs, were used to test how the different quantization approaches affect the selected deep learning models. CIFAR-10[62] holds 60K colour images of 32×32 pixels, spread evenly over 10 classes with 6K images

per class. Of these, 50K images form the training split, and the validation and test splits receive 5K images each. For this chapter, every CIFAR-10 image was resized to $48 \times 48 \times 3$ before training and optimization. The resizing was chosen with MobileNet_v1 in mind: at the native resolution this lightweight network trains so quickly, with or without AMP, that differences in training time are hard to observe. At $48 \times 48 \times 3$, the AMP and non-AMP training times can be compared meaningfully.

The second dataset is a subset of Cats vs Dogs[63], a two-class collection of colour images. The subset contains 3K images, balanced at 1.5K per class. Training uses 2K of them, and the validation and test sets have 500 images each. Images from this dataset were resized to $128 \times 128 \times 3$ for training and optimization.

3.3.2 Models

Three CNNs with quite different designs were trained and optimized, so that the effect of each optimization could be observed at both the training and the inference stage.

VGG16

VGG16 is a 16-layer CNN made of 13 convolutional layers with 3×3 kernels followed by 3 fully connected layers[64]. Every layer applies ReLU activation except the final one, which uses softmax.

MobileNet_v1

MobileNet_v1 was designed for mobile and embedded vision[65]. Its building block is the depthwise separable convolution, which keeps the network small and lowers both latency and computational cost on constrained devices. For this reason it is widely used for object detection, classification, and localization on mobile and embedded hardware.

ResNet-50

ResNet-50 belongs to the Residual Network family[66]. Skip connections let the network learn residual mappings, which counters the vanishing and exploding gradient problem and makes very deep models trainable. The network is built from convolutional layers and identity blocks and ends in a softmax layer.

3.3.3 Hardware

All experiments ran on an Nvidia Quadro RTX 4000[67], a Turing-generation GPU with compute capability 7.5. Compute capability identifies the feature set and general specifications that a GPU supports[68]. The card provides 2304 CUDA cores and 288 tensor cores, and the tensor cores handle quantized and mixed-precision arithmetic. Its 8GB of memory is GDDR6.

3.3.4 Software Environment and Experimental Configuration

The experiments use NVIDIA container release 22.03[69]. Training compares TensorFlow execution with and without automatic mixed precision (AMP), while inference compares TensorFlow–TensorRT (TF–TRT) execution in FP32, FP16, and INT8[29;30]. AMP and INT8 inference belong to different experimental stages. AMP changes the numerical computation used during training.

Table 3.1 summarizes the dataset and batch-size configurations. CIFAR-10 originally contains 32×32 images; the experimental input size in Table 3.1 reflects the resizing used in the study[62]. The Cats vs Dogs experiments use a balanced 3,000-image dataset[63]. For each model and dataset, the training comparison uses the listed training batches with and without AMP. The inference comparison uses the listed inference batches for each reported precision.

Table 3.1: *Documented dataset configurations for the quantization study.*

Setting	CIFAR-10	Cats vs Dogs subset
Classes	10	2
Training images	50,000	2,000
Validation images	5,000	500
Test images	5,000	500
Experimental input size	$48 \times 48 \times 3$	$128 \times 128 \times 3$
Training batch sizes	32, 64, 128, 256, 512, 1024	8, 16, 32, 64, 128, 256
Inference batch sizes	8, 16, 32, 64	8, 16, 32, 64

3.3.5 Metrics

Five metrics were used to compare the quantization techniques.

- **Accuracy**: the share of predictions that the classifier gets right, that is, correct predictions divided by all predictions, as in equation (3.1).

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{3.1}$$

- **Average training time**: the time needed to train a model for a fixed number of epochs. The AMP speedup is the ratio of the training time without AMP to the training time with AMP, given in equation (3.2).

$$\text{Speedup}_{\text{Training time}} = \frac{\text{Training time}_{\text{without AMP}}}{\text{Training time}_{\text{with AMP}}} \tag{3.2}$$

- **Memory usage in training**: the GPU memory occupied while the model trains. The saving from AMP is expressed as the memory reduction factor of equation (3.3).

$$\text{Reduction Factor}_{\text{Memory}} = \frac{\text{Memory}_{\text{without AMP}}}{\text{Memory}_{\text{with AMP}}} \tag{3.3}$$

- **Latency in inference**: the time the model needs to produce a prediction for a single input. The latency gain from quantization is expressed as the speedup in Equation

(3.4).

$$\text{Speedup}_{\text{Latency}} = \frac{\text{Latency}_{\text{without quantization}}}{\text{Latency}_{\text{with quantization}}} \tag{3.4}$$

- **Throughput in inference**: the number of images the model classifies per second. The gain is reported as the improvement factor (I.F.) of equation (3.5).

$$\text{Improvement Factor}_{\text{Throughput}} = \frac{\text{Throughput}_{\text{with quantization}}}{\text{Throughput}_{\text{without quantization}}} \tag{3.5}$$

3.4 Experimental Results

The following tables and figures present the experimental measurements. Training memory is reported in GB, inference throughput in images/s, and accuracy in percent. OOM denotes an infeasible memory configuration; N/A denotes an undefined comparison.

Table 3.2: *Training time and Speedup calculation with AMP for CIFAR-10 (image size:* $48 \times 48 \times 3$*)*

Batch Size	Training Time (s)								
	Without AMP			With AMP			Speedup		
	VGG16	ResNet-50	MBNet	VGG16	ResNet-50	MBNet	VGG16	ResNet-50	MBNet
32	480.14	659.36	253.27	265.44	711.65	367.09	1.81	0.93	0.69
64	373.22	403.44	138.8	187.99	360.74	192.23	1.99	1.12	0.72
128	310.14	300.74	94.49	143.61	193	110.48	2.16	1.56	0.86
256	291.32	269.95	78.14	121.49	115.46	57.29	2.40	2.34	1.36
512	256.16	257.65	74.6	112.28	89.54	37.87	2.28	2.88	1.97
1024	304.14	231.16	70.93	109.43	75.22	34.61	2.78	3.07	2.05

MBNet = MobileNet_v1

3.5 Analysis of Training Performance

3.5.1 Training Time and Batch Dependence

Tables 3.2 and 3.3 present the measured training durations and the corresponding speedup values. For CIFAR-10, the speedup of VGG16 increases from 1.81 at batch size 32 to 2.78

Table 3.3: *Training time and Speedup calculation with AMP for cats_and_dogs (image size:* $128 \times 128 \times 3$*)*

Batch Size	Training Time (s)								
	Without AMP			With AMP			Speedup		
	VGG16	ResNet-50	MBNet	VGG16	ResNet-50	MBNet	VGG16	ResNet-50	MBNet
8	114.81	120.95	59.83	63.51	130.59	70.84	1.81	0.93	0.84
16	78.94	86.82	41.96	50.19	72.84	39.25	1.57	1.19	1.07
32	70.79	69.94	35.29	45.4	44.89	22.28	1.56	1.56	1.58
64	72.61	63.52	33.85	41.92	37.53	16.26	1.73	1.69	2.08
128	79.8	61.87	33.45	42.36	33.99	21.15	1.88	1.82	1.58
256	OOM	OOM	36.19	49.12	31.5	15.09	N/A	N/A	2.40

OOM = Out of Memory, MBNet = MobileNet_v1, N/A = Not applicable

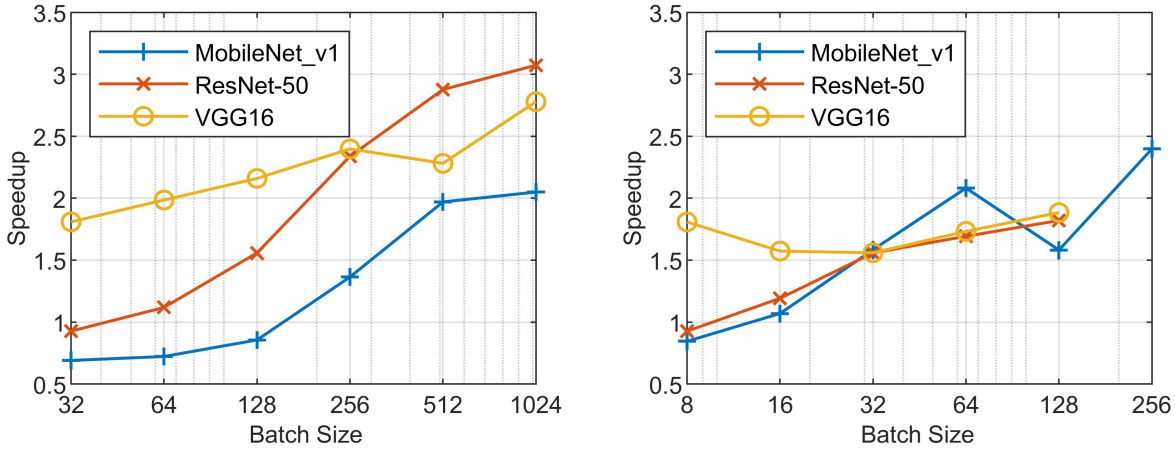


Figure 3.1: *Training time speedup comparison with AMP for CIFAR-10 (left) and CATS_VS_DOGS (right)*

Table 3.4: *Memory Reduction calculation with AMP for CIFAR-10 (image size:* $48 \times 48 \times 3$*)*

Batch Size	Memory Usage (GB)								
	Without AMP			With AMP			Memory Reduction Factor		
	VGG16	ResNet-50	MBNet	VGG16	ResNet-50	MBNet	VGG16	ResNet-50	MBNet
32	1.42	0.22	0.13	0.16	0.21	0.09	8.88	1.05	1.44
64	1.67	1.00	0.22	0.23	0.25	0.14	7.26	4.00	1.57
128	2.13	1.1	0.36	0.35	0.39	0.24	6.09	2.82	1.50
256	3.24	1.32	0.7	0.53	0.66	0.49	6.11	2.00	1.43
512	3.78	2.66	1.47	1.23	1.24	0.88	3.07	2.15	1.67
1024	5.79	5.23	2.79	1.81	2.34	1.77	3.20	2.24	1.58

MBNet = MobileNet_v1

Table 3.5: *Memory Reduction calculation with AMP for cats_and_dogs (image size:* $128 \times 128 \times 3$*)*

Batch Size	Memory Usage (GB)								
	Without AMP			With AMP			Memory Reduction Factor		
	VGG16	ResNet-50	MBNet	VGG16	ResNet-50	MBNet	VGG16	ResNet-50	MBNet
8	2.64	0.33	0.21	0.22	0.24	0.13	12.00	1.38	1.62
16	2.99	0.59	0.34	0.33	0.36	0.18	9.06	1.64	1.89
32	3.69	1.18	0.67	0.53	0.6	0.35	6.96	1.97	1.91
64	4.39	2.59	1.14	0.96	1.1	0.7	4.57	2.35	1.63
128	4.85	4.5	2.57	1.58	2.02	1.31	3.07	2.23	1.96
256	OOM	OOM	4.34	2.91	4.11	2.49	N/A	N/A	1.74

OOM = Out of Memory, MBNet = MobileNet_v1, N/A = Not applicable

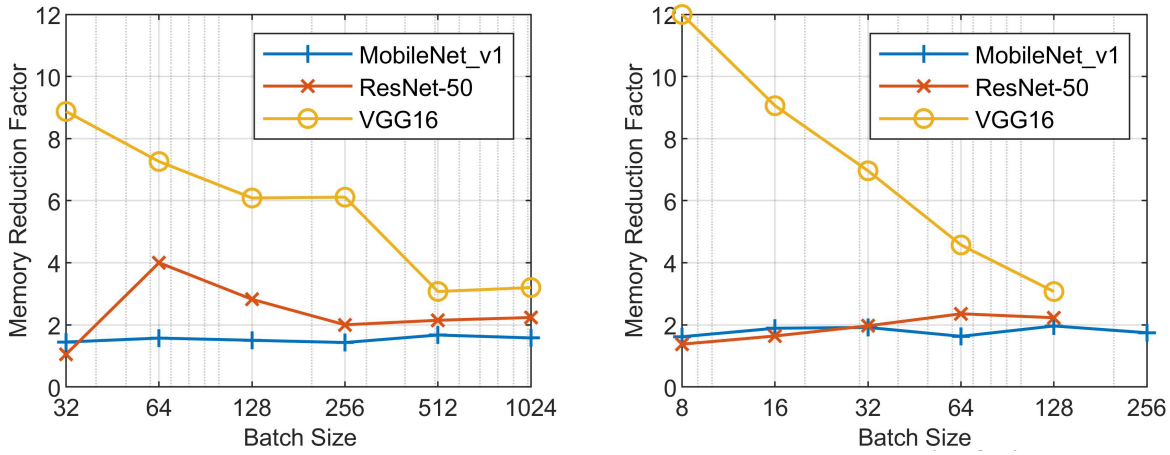


Figure 3.2: *Memory performance comparison with AMP for CIFAR-10 (left) and CATS_VS_DOGS (right)*

Table 3.6: *CIFAR-10: Latency speedup results with TF-TRT precision modes for different models*

Batch Size	Latency (ms)											
	VGG16				ResNet-50				MobileNet_v1			
	Base	FP32	FP16	INT8	Base	FP32	FP16	INT8	Base	FP32	FP16	INT8
8	39.3	4.7	1.5	1.2	47	4.2	1.9	1.9	39.8	1.9	1.4	1.3
16	39.4	5.6	1.5	1.4	47.9	4.2	2.1	1.9	39.8	1.9	1.4	1.4
32	38.1	7.9	1.7	1.5	48.1	4.6	2.1	2	37.3	1.9	1.5	1.4
64	55.2	14.2	3	1.8	60.2	6	2.2	2.1	45.7	2.2	1.6	1.5

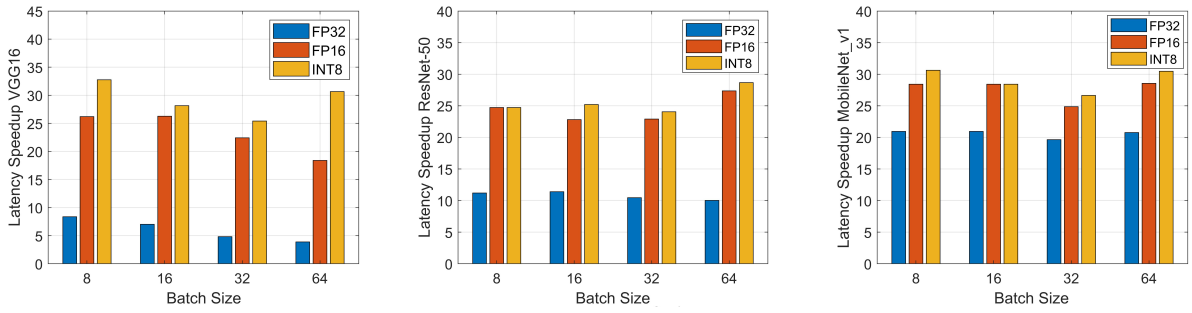


Figure 3.3: *CIFAR-10: Latency speedup comparison with TF-TRT for VGG16 (left), ResNet-50 (middle) and Mobilenet_v1 (right) models*

Table 3.7: *Cats vs Dogs: Latency speedup results with TF-TRT precision modes for different models*

Batch Size	Latency (ms)											
	VGG16				ResNet-50				MobileNet_v1			
	Base	FP32	FP16	INT8	Base	FP32	FP16	INT8	Base	FP32	FP16	INT8
8	47.8	10.6	2.8	2	49.1	4.6	2.6	2.1	40.8	2.2	1.9	1.8
16	51.9	19.1	5.1	3.1	51.4	7.9	3.3	2.8	41.2	2.8	2.3	2.3
32	54.2	37.7	10.1	5.6	52.8	15	5.2	3.5	43.2	4.8	3.1	3.2
64	83.3	76	19.6	11.2	77.3	28.9	9.1	6.3	58.6	9.2	4.7	3.9

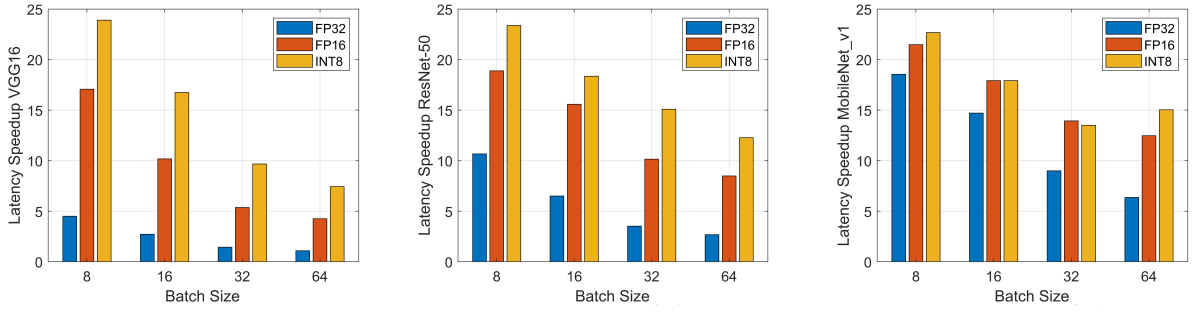


Figure 3.4: *Cats vs Dogs: Latency speedup comparison with TF-TRT for VGG16 (left), ResNet-50 (middle) and Mobilenet_v1 (right) models*

Table 3.8: *CIFAR-10: Throughput improvement factor results with TF-TRT precision modes for different models*

Batch Size	Throughput (images per second)											
	VGG16				ResNet-50				MobileNet_v1			
	Base	FP32	FP16	INT8	Base	FP32	FP16	INT8	Base	FP32	FP16	INT8
8	204	1689	5470	5455	170	1916	4105	4123	201	4286	5545	5947
16	406	2858	10554	10780	334	3771	7747	8301	402	8430	11271	11817
32	840	4026	18326	21065	666	6887	15254	16245	857	16890	20984	23040
64	1160	4491	21475	35960	1063	10618	30599	30975	1402	28950	40877	42037

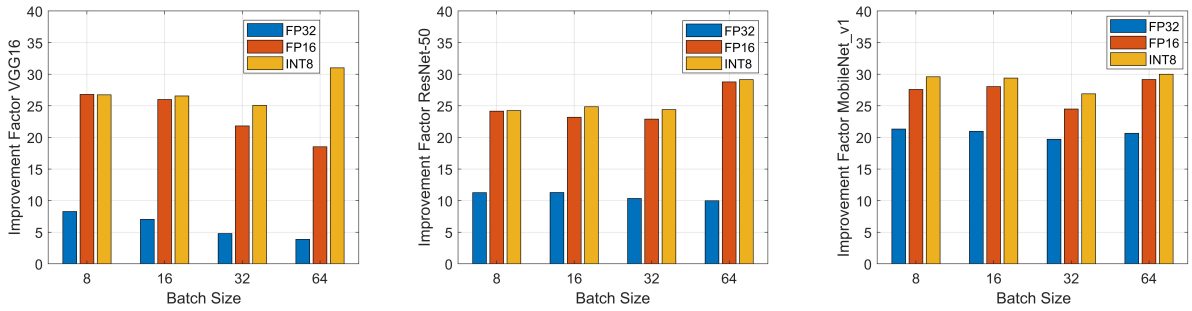


Figure 3.5: *CIFAR-10: Throughput improvement factor comparison with TF-TRT for VGG16 (left), ResNet-50 (middle) and Mobilenet_v1 (right) models*

Table 3.9: *Cats vs Dogs: Throughput improvement factor results with TF-TRT precision modes for different models*

Batch Size	Throughput (images per second)											
	VGG16				ResNet-50				MobileNet_v1			
	Base	FP32	FP16	INT8	Base	FP32	FP16	INT8	Base	FP32	FP16	INT8
8	167	757	2858	4081	163	1747	3086	3774	196	3563	4105	4470
16	308	839	3149	5224	311	2031	4859	5783	388	5788	7041	7108
32	591	848	3169	5711	606	2127	6154	9077	730	6716	10280	10024
64	768	842	3271	5718	828	2218	7033	10172	1093	6967	13670	16368

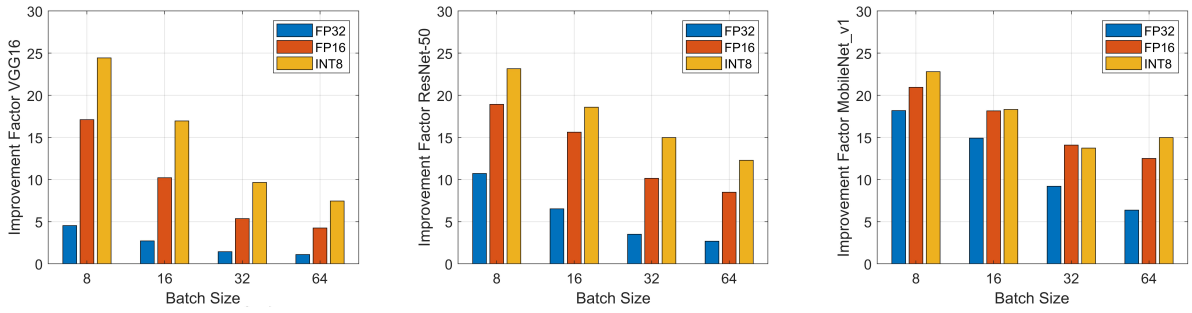


Figure 3.6: *Cats vs Dogs: Throughput improvement factor comparison with TF-TRT for VGG16 (left), ResNet-50 (middle) and MobileNet_v1 (right) models*

Table 3.10: *CIFAR-10: Accuracy for different models using the 22.03 Nvidia container*[69]

Model Accuracy				
Model	**TensorFlow FP32**	**TF-TRT FP32**	**TF-TRT FP16**	**TF-TRT INT8**
MobileNet_v1 without AMP	85.383	85.383	85.332	83.921
MobileNet_v1 with AMP	84.576	84.576	84.576	84.274
VGG16 without AMP	89.120	89.120	89.171	88.894
VGG16 with AMP	88.709	88.709	88.709	88.960
ResNet-50 without AMP	86.895	86.895	86.845	85.721
ResNet-50 with AMP	87.01	87.01	87.01	88.15

Table 3.11: *Cats vs Dogs: Accuracy for different models using the 22.03 Nvidia container*[69]

Model Accuracy				
Model	**TensorFlow FP32**	**TF-TRT FP32**	**TF-TRT FP16**	**TF-TRT INT8**
MobileNet_v1 without AMP	96.484	96.484	96.289	95.898
MobileNet_v1 with AMP	96.289	96.289	97.465	96.484
VGG16 without AMP	92.578	93.554	92.773	92.359
VGG16 with AMP	93.516	93.516	93.516	93.724
ResNet-50 without AMP	94.335	94.335	94.110	93.825
ResNet-50 with AMP	94.117	94.117	94.214	94.017

at batch size 1024, with variation between intermediate configurations. ResNet-50 achieves a speedup of 3.07 at batch size 1024, but ratio of 0.93 at batch size 32 shows slow execution with AMP. MobileNet_v1 similarly records ratios below one in batch sizes 32, 64 and 128 before reaching 2.05 at batch size 1024.

For Cats vs Dogs, VGG16 benefits from AMP in all settings with a non-AMP baseline. ResNet-50 is slow with AMP at batch size 8, but fast at the larger batch sizes. MobileNet_v1 reaches a reported speedup of 2.40 at batch size 256. These results show that architecture and batch configuration affect whether lower precision execution produces a training timing benefit.

3.5.2 Memory Usage and Workload Feasibility

Tables 3.4 and 3.5 show reduced memory usage with AMP. VGG16 exhibits the large improvement ratios at small batch sizes: 8.88 for CIFAR-10 at batch size 32 and 12.00 for Cats vs Dogs at batch size 8. At larger batch sizes, its reduction factor declines, while other architectures show different profiles.

A useful outcome is the change in workload size. At batch size 256 for Cats vs Dogs, VGG16 and ResNet-50 encounter out-of-memory conditions without AMP but execute with AMP on the same 8 GB GPU. An unavailable baseline is recorded as OOM.

Selecting a batch size therefore requires combined consideration of training timing, memory, and learning behavior. The large feasible batch is not automatically the best training choice, and the study does not establish equal convergence outcomes across batch sizes. The practical contribution is a characterization of execution feasibility and performance under the evaluated configurations.

3.6 Analysis of Inference Performance

3.6.1 Framework Optimization and Numerical Precision

The latency and throughput tables retain four inference configurations: native TensorFlow, TF-TRT FP32, FP16, and INT8. For example, CIFAR-10 MobileNet_v1 at batch size 8 has reported latency values of 39.8 ms for the baseline and 1.9 ms for TF-TRT FP32. Further reductions to 1.4 ms with FP16 and 1.3 ms with INT8 are smaller than the baseline to FP32 variation.

Reduced precision produces additional gains in other settings. For CIFAR-10 VGG16 at batch size 64, latency falls from 14.2 ms in TF-TRT FP32 to 3.0 ms in FP16 and 1.8 ms in INT8. The corresponding throughput increases from 4,491 to 21,475 and 35,960 images/s. For Cats vs Dogs VGG16 at batch size 64, the respective latency values are 76.0, 19.6, and 11.2 ms. These observations demonstrate substantial precision-dependent differences within the optimized framework, while preserving the distinction between framework effects and precision effects.

INT8 does not outperform FP16 in every reported case. For Cats vs Dogs MobileNet_v1 at batch size 32, FP16 throughput is 10,280 images/s compared to 10,024 images/s for INT8. CIFAR-10 VGG16 at batch size 8 also reports slightly higher FP16 throughput. Such cases reinforce the need to measure the application rather than assume a strict performance ordering from bit width alone.

3.6.2 Batch Size and Interpretation of Latency

The reported throughput generally increases with the inference batch size, while the absolute latency and relative improvements vary between architectures and input configurations. Throughput is appropriate for aggregate image processing, whereas response-time requirements must be evaluated using the appropriate timing boundary for the application.

3.7 Accuracy across Training and Inference Configurations

Tables 3.10 and 3.11 report classification accuracy in percent for models trained with and without AMP. FP16 frequently preserves or closely approximates the corresponding TensorFlow accuracy, while INT8 changes vary by architecture and training configuration. CIFAR-10 MobileNet_v1 without AMP changes from 85.383% to 83.921% with INT8, a decrease of 1.462 percentage points. The AMP-trained counterpart changes from 84.576% to 84.274%, a decrease of 0.302 percentage points. These within-model differences should be distinguished from comparisons between separately trained AMP and non-AMP models.

Some configurations report increases in accuracy. CIFAR-10 ResNet-50 trained with AMP has 87.01% TensorFlow accuracy and 88.15% INT8 accuracy, while Cats vs Dogs MobileNet_v1 trained with AMP has 96.289% TensorFlow accuracy and 97.465% FP16 accuracy, possible reason is quantization sometimes helps reduce overfitting.

3.8 Implications for Model-Level Optimization

The study supports three practical conclusions. First, precision selection should be evaluated together with the model and batch setting. AMP reduces memory usage but does not accelerate every small batch configuration. Second, framework optimization and numerical precision are distinct contributors to inference performance, as demonstrated by the TF-TRT FP32 control. Third, a desirable execution settings must preserve sufficient model accuracy as well as satisfy memory and performance requirements.

These findings establish the model-level foundation for the dissertation's broader system-level evaluation. They do not imply measured energy savings. The study profiles accuracy, memory, and execution performance without reporting direct GPU energy measurements. Chapter 4 extends the evaluation to energy and Tokens/kJ, allowing execution strategies to be compared against explicit energy, memory, and time constraints.

3.9 Scope and Limitations

The experimental findings are limited to the evaluated vision models, two dataset configurations, Quadro RTX 4000 hardware, and the reported TensorFlow/TF-TRT framework. They do not generalize the broader behavior of LLM quantization, newer accelerator generations, or different inference engines.

3.10 Chapter Summary

This chapter characterized model-level optimization with AMP training and TF-TRT inference across VGG16, ResNet-50, and MobileNet_v1. The results show that benefits of execution depend on architecture, input configuration, batch size, and precision mode. AMP can reduce memory and enable infeasible training batches. Accuracy remains a important component of configuration selection. The following chapter moves from numerical precision to training-level optimization, examining how multi-GPU parallelism changes efficient work per unit energy with memory usage and time-to-train.

Chapter 4

Training-Level Characterization through Energy-Aware Multi-GPU Execution

4.1 Introduction

Chapter 3 showed that numerical precision choices affect the performance, memory usage, and model accuracy. This chapter advances to the second level of the dissertation: the multi-GPU training system. For large language models (LLMs), the distribution of computation between GPUs introduces choices in replication, state sharding, and communication. These settings not only affect training duration and memory capacity, but also the energy requirements[8;31–33].

The research question is how multi-GPU parallelism changes the energy–memory–train-to-time trade-off space of supervised fine-tuning. The study evaluates Data Parallelism (DP), Distributed Data Parallelism (DDP), two Fully Sharded Data Parallelism (FSDP) implementations, and DeepSpeed ZeRO stages 1,2,3 on a single node system containing eight NVIDIA H100 GPUs. Two Llama model are used under a common data-processing pipeline.

Tokens per Kilojoule (Tokens/kJ) is used as a energy efficiency metric. The evaluation of Tokens-per-energy has precedents in LLM efficiency research[22–24]; the contribution here is its integration with runtime, memory, and synchronized telemetry to evaluate and support the selection of parallelism. No new sharding algorithm is proposed here. What the chapter offers is a controlled set of measurements and a decision framework built on them.

4.2 Parallelism and Evaluation Design

The methods were introduced in Chapter 2. DDP replicates training states, FSDP shards parameters, gradients, and optimizer states, and ZeRO techniques progressively partition these training states[31–33]. On paper, each scheme implies a certain memory footprint, but the memory and energy seen at run time also depend on parameter materialization, communication buffers, and scheduling. For that reason every training run was profiled from start to finish, and no method was chosen from its partitioning scheme alone.

The principal comparison maintains the workload and platform settings consistent while only vary the training strategy. Identical token counts support within the model energy comparisons, but the 3B and 8B models do not perform equal arithmetic work per token. Across model sizes, Tokens/kJ shows the execution cost for the selected models.

4.3 Methodology

This section describes the model architectures, dataset processing pipeline, training settings, hardware, and profiling instrumentation used to evaluate system behavior across multiple multi-GPU training strategies for LLM supervised fine-tuning (SFT).

4.3.1 Model Architecture

We evaluated two instruction-tuned decoder-only transformer models: *Llama-3.2-3B-Instruct* (3.2B parameters) and *Llama-3.1-8B-Instruct* (8.0B parameters)[70]. Both models are full

parameters fine-tuned with no parameter freezing or LoRA adapters. Full-parameter SFT avoids introducing adapter effect or parameter freezing as additional experimental factors.

Tokenization is performed using the official LLaMA tokenizer, which supports chat-style multi-turn format and the LLaMA end-of-turn token `<|eot_id|>` to standardize training data across all runs.

4.3.2 Dataset and Preprocessing

Training uses a 3,200 sample subset of the Alpaca dataset[71], a widely adopted dataset for instruction following SFT. Each example is transformed into a chat style message sequence with explicit `user` and `assistant` roles. The instruction and optional input fields are merged into the user prompt, while the output field becomes the part of assistant response.

Across 10 epochs of training, a total of 335,370 tokens per epoch are processed, resulting in 3,353,700 tokens per full run. All multi-GPU methods see identical token counts, data order, and identical preprocessing to ensure a controlled comparison. The token budget of 335,370 tokens per epoch is measured with the Llama 3 tokenizer. Because Llama-3.2-3B-Instruct and Llama-3.1-8B-Instruct share this tokenizer, both models process the same 3,353,700 tokens per run, so tokens/kJ is comparable across model sizes and training strategies.

4.3.3 Training Framework

Fine-tuning runs through the TRL *SFTTrainer*, a wrapper that TRL provides for instruction tuning. Training parameters are defined via *SFTConfig* with the following settings:

- **Batch Size:** Per-GPU batch size of 4, yielding the global batch size of 32 across 8 GPUs in training.
- **Chat Template:** The official LLaMA chat template is used in each message sequence.
- **Checkpointing:** Disabled for all runs to avoid I/O overhead and maintain identical comparison.

All experiments log metrics through Weights & Biases (wandb), including training loss, throughput, GPU utilization, and power statistics.

4.3.4 Hardware and Software Setup

Experiments were conducted on a single node, multi-GPU system with the following configuration:

- 8× NVIDIA H100 SXM5 GPUs (80 GB HBM3 each), interconnected via NVLink and NVSwitch
- 168 CPU cores, 1.8 TB of system RAM, and 12.5 TB of local NVMe storage
- Ubuntu Linux host, NVIDIA driver and CUDA toolkit versions held fixed and same across all runs, with PyTorch, TRL, Accelerate, and DeepSpeed versions pinned to identical releases

Pinning the OS, driver, CUDA, and framework stack across runs is intended to minimize software-side confounders and to ensure that observed differences in power, memory, and thermal behavior reflect the parallelism strategy rather than environment drift.

All training methods are orchestrated using the *accelerate* framework. Environment variables, GPU visibility mappings, and process counts were kept the same for every strategy. The following methods were evaluated:

- **DP:** Single-process data parallelism via PyTorch *DataParallel*, with a master GPU aggregating gradients.
- **DDP:** Multi-process replicated training with synchronous *all-reduce* across 8 GPUs.
- **FSDP-1:** PyTorch FSDP v1 with full sharding of parameters, gradients, and optimizer states.
- **FSDP-2:** PyTorch FSDP v2 API with full sharding. CPU offloading is disabled in both variants.

- **ZeRO-1/2/3:** DeepSpeed memory-optimized training with progressively aggressive sharding of optimizer states, gradients, and parameters.

4.3.5 Profiling and Instrumentation

System-level metrics were collected using a combination of per-GPU telemetry, framework-level logging, and external monitoring:

- **nvitop:** Real-time sampling of per-GPU power draw (W), HBM usage (GB), temperature (°C), SM utilization (%), and memory bandwidth at a 5 s interval.
- **nvidia-smi:** Logged per-GPU memory use, power limits, fan speeds, and P-state changes throughout each run.
- **wandb:** Aggregation of training and system metrics, enabling synchronized visualization of compute, thermal, and energy behavior against wall-clock time.

Telemetry scripts run concurrently with training and share synchronized timestamps so that compute events can be tightly aligned with system-level effects.

4.3.6 Energy Computation

Because several of the evaluated strategies exhibit highly non-uniform power profiles, we compute total GPU energy as the time-integrated sum of instantaneous per-GPU power across all $G = 8$ devices:

$$E_{\text{total}} = \sum_{g=1}^{G} \int_0^T P_g(t)\,\mathrm{d}t,$$

which we approximate numerically from the sampled telemetry as

$$E_{\text{total}} \approx \sum_{g=1}^{G} \sum_{i=1}^{N} P_g(t_i)\,\Delta t,$$

where $P_g(t_i)$ is the instantaneous power of GPU g at sample time t_i, Δt is the sampling interval (nominally 5 s) , and N is the number of samples collected over the training run of duration T. The result is expressed in kilojoules, $E_{\mathrm{kJ}} = E_{\mathrm{total}}/1000$.

4.3.7 Measurement Protocol and Scope

Each configuration is executed under identical software, hardware, and environmental conditions, with the same random seed, data set ordering, and framework versions. For the main results, we report the telemetry collected from multiple training runs per configuration; short pilot runs were used beforehand to confirm that steady-state power and thermal behavior are stable across repeated executions on this platform.

The measurements are also deliberately scoped to a single-node, multi-GPU setting. Behaviors tied to multi-node interconnects, fabric contention, and cross-rack thermal coupling are outside the scope of this study, and are discussed qualitatively where relevant.

4.3.8 Configuration Summary

Table 4.1 summarizes the configuration of the distributed-training experiments. Their scope is full-parameter supervised fine-tuning on a single eight-GPU H100 node, with the summed energy of the eight GPUs as the energy boundary. The model and dataset sources are identified by the existing LLaMA and Alpaca references[70;71]. Examples use explicit user and assistant roles, with instruction and optional input combined into the user prompt.

In this study, checkpointing was disabled to avoid I/O overhead. This refers to checkpoint-file writing; it does not establish the activation-checkpointing policy. Driver, CUDA, PyTorch, TRL, Accelerate, and DeepSpeed releases were held fixed across runs. Section 2.4.4 illustrates the parallelism and sharding layouts of the evaluated strategies.

Table 4.1: *Documented configuration of the distributed-training experiments.*

Component	Reported setting
Models	Llama-3.2-3B-Instruct and Llama-3.1-8B-Instruct
Fine-tuning	Full parameters; no LoRA adapters or parameter freezing
Dataset	Alpaca subset, official LLaMA chat template
Training duration	10 epochs
Token budget	335,370 per epoch; 3,353,700 per run
Batch size	4 per GPU; 32 globally across eight GPUs
Training framework	TRL SFTTrainer and SFTConfig; Accelerate orchestration
GPU hardware	Eight H100 SXM5 GPUs, 80 GB HBM3 each; NVLink/NVSwitch
Host resources	168 CPU cores; 1.8 TB RAM; 12.5 TB local NVMe
Telemetry	nvitop every 5 seconds; nvidia-smi; Weights & Biases
Repetitions	Three independent runs per evaluated configuration
Energy boundary	Summed energy of the eight GPUs

4.4 Results

The measurements below come from fine-tuning the 3B and 8B LLaMA models with each multi-GPU strategy on an 8× NVIDIA H100 SXM5 node. The analysis covers training time, GPU utilization, memory consumption, thermal and power behavior, total energy usage, and energy efficiency expressed in Tokens/kJ. All metrics are derived from the synchronized telemetry described in Section 4.3.

4.4.1 Metric Definitions

GPU utilization, memory usage, and temperature were sampled every 5 s using *nvitop* and *nvidia-smi*. The reported Power value in the tables refers to the time-averaged per-GPU power over the full training run. Total GPU energy across the node E_{kJ} is obtained from the time-integrated, per-GPU formulation defined in Section 4.3.6, which sums the sampled instantaneous power across all 8 GPUs and integrates over the full training duration.

Each experiment processes 10 epochs of training data, equivalent to 3,353,700 tokens. Energy efficiency is therefore reported as

$$\mathrm{Tokens/kJ} \;=\; \frac{3{,}353{,}700}{E_{\mathrm{kJ}}}.$$

Table 4.2: *3B model measurements across training strategies: (a) performance and (b) GPU telemetry. Values are means with variation over three independent runs. GPU telemetry and memory values are averaged across the eight devices; energy is integrated across all participating GPUs.*

(a) Performance measurements

Method	Memory (GB)	Time (s)	GPU energy (kJ)	Tokens/kJ
DP	39.8 ± 0.25	810 ± 9.50	2772.3 ± 42.3	1210 ± 18.0
DDP	63.6 ± 0.32	270 ± 3.49	739.3 ± 14.6	4537 ± 90.0
FSDP-1	22.2 ± 0.19	277 ± 3.55	580.9 ± 11.4	5773 ± 114.0
FSDP-2	19.1 ± 0.15	274 ± 4.26	590.8 ± 12.8	5676 ± 123.0
ZeRO-1	21.4 ± 0.10	226 ± 3.02	490.7 ± 9.6	6834 ± 134.0
ZeRO-2	20.1 ± 0.19	224 ± 2.85	490.0 ± 11.1	6844 ± 155.0
ZeRO-3	24.5 ± 0.20	542 ± 4.22	1029.9 ± 20.1	3256 ± 63.0

(b) Telemetry measurements

Method	GPU util. (%)	Mem. util. (%)	Temp. (°C)	Power (W)
DP	72.43 ± 7.46	13.15 ± 2.17	51.76 ± 1.41	427.95 ± 4.18
DDP	79.11 ± 6.12	42.62 ± 5.89	45.07 ± 1.22	341.91 ± 5.12
FSDP-1	72.05 ± 5.23	18.73 ± 3.67	42.56 ± 1.84	262.16 ± 3.92
FSDP-2	76.46 ± 4.46	21.41 ± 4.49	43.22 ± 2.05	269.55 ± 4.09
ZeRO-1	62.02 ± 5.99	22.68 ± 3.18	43.26 ± 1.47	271.49 ± 3.89
ZeRO-2	65.73 ± 4.16	24.13 ± 3.89	43.59 ± 1.88	273.47 ± 5.10
ZeRO-3	79.38 ± 5.81	11.55 ± 2.08	43.07 ± 1.10	237.53 ± 4.24

4.4.2 3B Model Results

Table 4.2 summarize the system behavior observed during 3B model training run. Training times vary across multi-GPU methods. DeepSpeed ZeRO-1 and ZeRO-2 record the shortest runtimes at 226 s and 224 s, respectively, followed by DDP at 270 s and two FSDP variants at 274–277 s. ZeRO-3 requires long (542 s), while single-process Data Parallelism (DP) shows slowest performance at 810 s.

GPU Utilization. GPU utilization shows a clear separation between fully replicated and sharded techniques. DDP reaches large utilization (79.11%), potentially reflecting an execution pattern dominated by compute instead of communication. Sharded methods such as FSDP 1/2 and ZeRO 1/2 operate in the 62–76% range, which may reflect variable schedules of computation and synchronization. Memory use follows directly from how each algorithm

Table 4.3: *8B model measurements across training strategies: (a) performance and (b) GPU telemetry. Values are means with variation over three independent runs. GPU telemetry and memory values are averaged across the eight devices; energy is integrated across all participating GPUs.*

(a) Performance measurements

Method	Memory (GB)	Time (s)	GPU energy (kJ)	Tokens/kJ
FSDP-1	35.3 ± 0.17	326 ± 4.10	911.8 ± 15.9	3678 ± 64.0
FSDP-2	35.8 ± 0.20	318 ± 3.86	983.4 ± 17.5	3411 ± 61.0
ZeRO-1	46.6 ± 0.16	361 ± 5.17	923.3 ± 16.0	3633 ± 63.0
ZeRO-2	38.5 ± 0.46	354 ± 5.02	926.8 ± 20.3	3619 ± 79.0
ZeRO-3	37.9 ± 0.11	560 ± 4.27	1360.0 ± 22.0	2466 ± 40.0

(b) Telemetry measurements

Method	GPU util. (%)	Mem. util. (%)	Temp. (°C)	Power (W)
FSDP-1	73.77 ± 5.29	35.51 ± 4.18	43.72 ± 1.99	349.63 ± 4.23
FSDP-2	81.49 ± 6.10	42.35 ± 3.92	47.47 ± 1.05	386.57 ± 5.01
ZeRO-1	69.47 ± 3.24	32.27 ± 3.12	43.60 ± 0.99	319.69 ± 3.12
ZeRO-2	71.56 ± 4.12	33.30 ± 4.87	46.03 ± 1.32	327.25 ± 5.47
ZeRO-3	83.75 ± 6.89	26.83 ± 4.42	45.81 ± 1.89	303.24 ± 4.33

stores training state. DP and DDP consume 40–64 GB per GPU due to full model replication, whereas FSDP 1/2 and ZeRO 1/2 operate with small footprints (19–22 GB). ZeRO-3 uses 24.5 GB. Buffering and transient runtime states may contribute to its higher measured footprint, but the study telemetry does not isolate their individual costs.

Thermal Behavior. No 3B run showed thermal problems. FSDP-1/2 ran coolest, at 42–43°C, and the DP run was the warmest at 51.76°C. Average per-GPU power spanned 237–428 W across methods, consistent with their different balance of computation and communication.

Energy Consumption and Energy Efficiency. Total GPU energy across the node shows the strongest divergence across methods. FSDP 1/2 and ZeRO 1/2 consume the least energy (490–591 kJ), reflecting the combination of moderate per GPU power and short runtime. DDP requires 739 kJ, while DP incurs the highest energy cost at 2,772 kJ. ZeRO-3 requires 1,030 kJ, which is associated with its extended runtime.

Energy efficiency, measured as Tokens/kJ, varies by more than 5× across methods. ZeRO-

1 and ZeRO-2 achieve the high efficiency at 6,834 and 6,844 Tokens/kJ. FSDP-1 and FSDP-2 follow closely at 5,773 and 5,676 Tokens/kJ. DDP reaches 4,537 Tokens/kJ, while DP results only 1,210 Tokens/kJ. ZeRO-3 delivers 3,256 Tokens/kJ, illustrating that its memory savings come at a meaningful cost in overall energy efficiency.

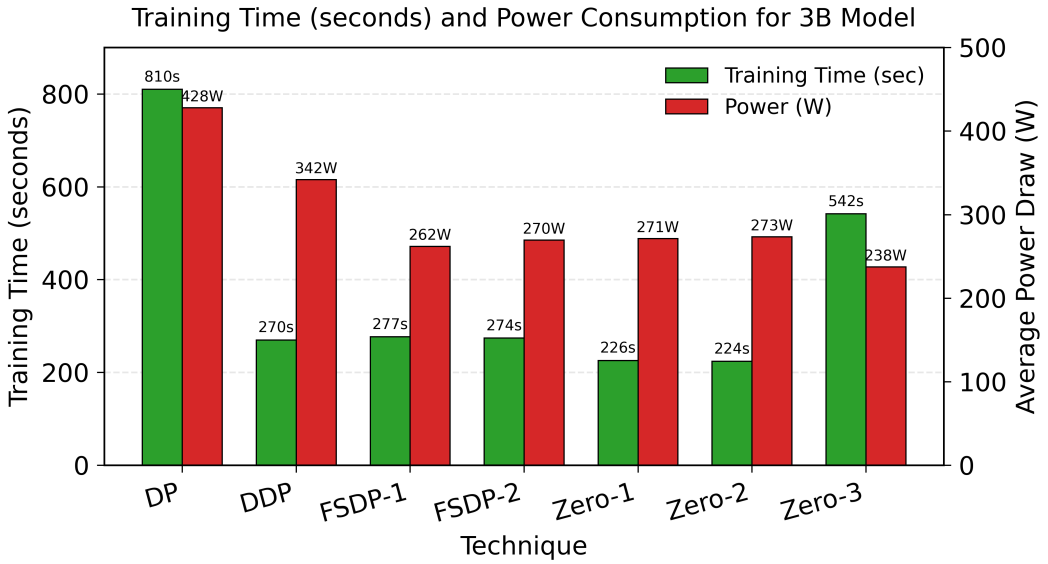


Figure 4.1: *Training time and average per-GPU power draw for the 3B model across multi-GPU training techniques. Average per-GPU power is shown for descriptive comparison.*

4.4.3 8B Model Results

Table 4.3 present results for the 8B model, evaluated under FSDP and ZeRO configurations; DP and DDP are omitted because their per-GPU memory replication exceeds available HBM capacity. FSDP-1 and FSDP-2 record the fastest runtimes at 326 s and 318 s. ZeRO-1 and ZeRO-2 follow at 361 s and 354 s, while ZeRO-3 again records the longest training time at 560 s, due to its high communication requirements. The 8B DP and DDP configurations were infeasible under the memory constraints. Their absence from the performance comparison is due to feasibility result and should not be represented as a zero throughput or zero energy.

GPU Utilization. GPU utilization spans 69–84%. FSDP-2 and ZeRO-3 reach the highest utilization (81.49% and 83.75%), potentially aided by efficient parameter sharding and overlap of communication with computation. Per-GPU memory usage ranges from 35–46 GB for

FSDP and ZeRO-1/2, with ZeRO-3 settling at 37.9 GB.

Thermal Behavior. Thermal behavior for the 8B model remains tightly clustered: all methods fall between 43–47°C. ZeRO-3 drew the least average power per GPU (303 W) and FSDP-2 the most (387 W).

Energy Consumption and Energy Efficiency. For the 8B runs, FSDP 1/2 and ZeRO 1/2 each used 912–984 kJ of GPU energy across the node; ZeRO-3 used 1,360 kJ. Corresponding energy-efficiency values are: FSDP-1 at 3,678 Tokens/kJ, ZeRO-1 at 3,633 Tokens/kJ, ZeRO-2 at 3,619 Tokens/kJ, FSDP-2 at 3,411 Tokens/kJ, and ZeRO-3 at 2,466 Tokens/kJ.

4.4.4 Cross-Model Energy–Performance–Memory Trade-offs

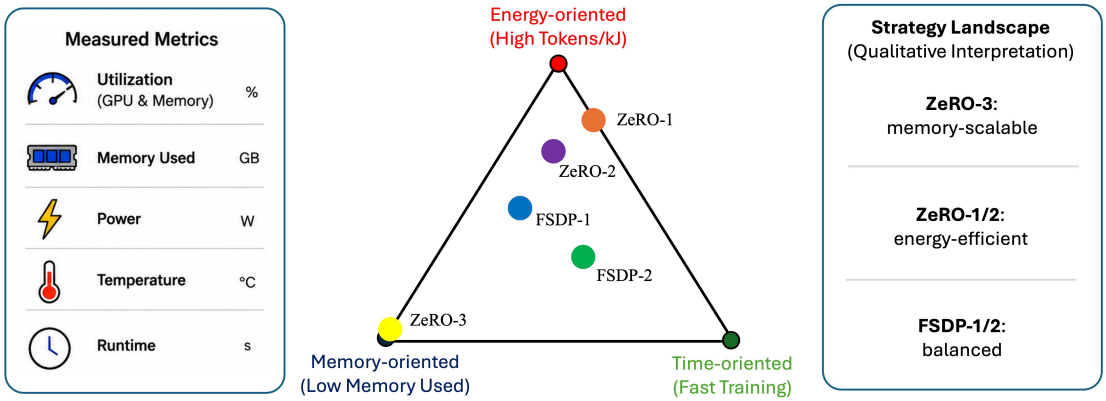


Figure 4.2: *Normalized energy–memory–time trade-off visualization for the 8B model. Coordinates combine min–max normalized Tokens/kJ, memory, and time using L1 normalization. The figure is a relative visualization of the reported objectives, not a Pareto-optimality test or an independent composite efficiency score.*

At both model sizes, FSDP and ZeRO-1/2 land in the favorable part of the energy–performance–memory trade-off space. The best Tokens/kJ came from ZeRO 1/2 on the 3B model and from FSDP-1 on the 8B model. These methods ran quickly with moderate memory footprints and low cumulative energy.

ZeRO-3 sits elsewhere. Its sharding lets memory scale with the model, yet in these settings it ran long and accumulated high energy. A central trade-off of multi-GPU training is visible here: minimizing instantaneous resource demand or maximizing GPU utilization

does not necessarily minimize total energy.

DDP held up well on the 3B model. Because it replicates the full model on every GPU, though, the 8B model did not fit in per-GPU memory. DP trailed the newer distributed strategies in both runtime and energy efficiency. Tokens/kJ also fell as the model grew: FSDP-1 went from 5,773 Tokens/kJ at 3B to 3,678 Tokens/kJ at 8B, since the larger model needs more compute and energy for each token it processes.

Choosing a parallelism strategy is thus a problem with several objectives. Runtime, GPU utilization, memory footprint, and average power each describe part of the behavior, and cumulative energy and Tokens/kJ show what those interactions cost in energy. On the single-node H100 configuration tested, ZeRO-1/2 and FSDP struck the best balance of energy efficiency, training time, and memory use. That conclusion holds for this hardware and software environment. Whether it carries over to multi-node fabrics, other accelerator generations, and broader workloads is not tested in this dissertation and is left for future study.

4.5 Discussion

4.5.1 GPU Utilization Trends

The GPU utilization measurements reveal differences in how each parallelism strategy balances the computation and communication. These differences do not translate directly into energy efficiency rankings. For the 3B model, ZeRO-3 and DDP achieve the high utilization at 79.4% and 79.1%, respectively. DDP combines high utilization with short runtime, but its full model replication results in high VRAM usage and prevents execution of the evaluated 8B model on this platform.

FSDP-1 and FSDP-2 sustain stable and high utilization levels (72–76% for 3B; 74–81% for 8B). By sharding parameters and gradients across GPUs, these methods reduce per-device memory pressure while maintaining comparatively efficient execution.

For the 3B model, ZeRO-3 reaches 79.4% utilization yet requires more than twice the

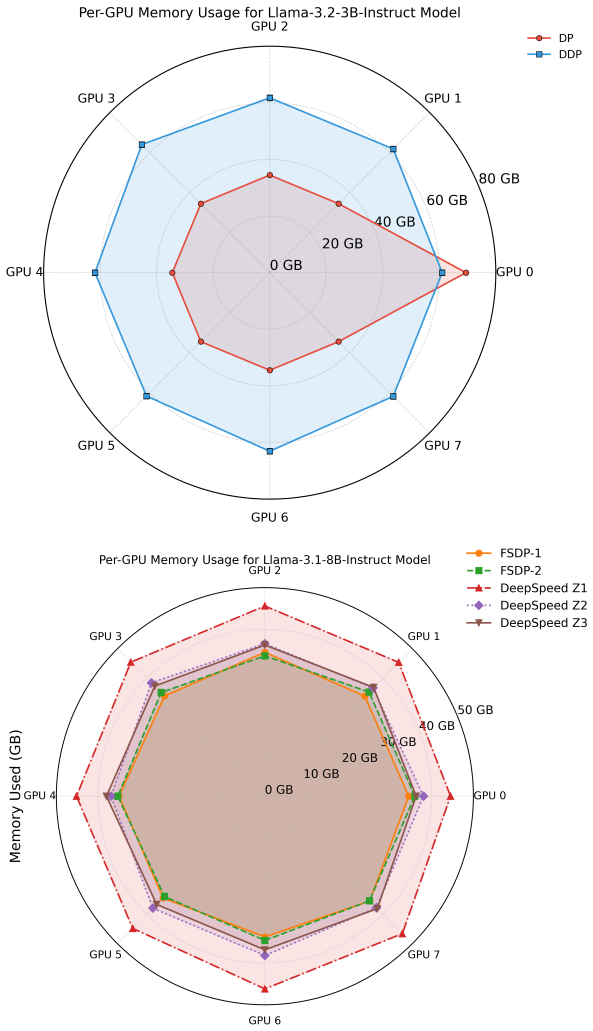


Figure 4.3: *Octagonal GPU memory comparison across (a) DP vs. DDP memory distribution for a 3B LLaMA and (b) multi-GPU training techniques for an 8B LLaMA.*

runtime of ZeRO 1/2 and achieves substantially lower Tokens/kJ. A similar pattern appears at 8B, where ZeRO-3 achieves the highest utilization (83.75%) but the low energy efficiency among the evaluated methods. A high utilization figure, then, says little on its own about whether a run is productive or energy-efficient.

4.5.2 Power and Energy Implications

Instantaneous power and cumulative energy did not rank the methods in the same way. DP exhibits the high energy cost for the 3B model (2,772 kJ at the node level), driven combined

by its long runtime and high average per-GPU power. DDP brought node energy down to 739 kJ, helped by both a shorter run and lower average power.

FSDP-1 and FSDP-2 used 581–591 kJ, with neither power nor runtime pushing the total up. The lowest 3B figures, about 490 kJ each, belonged to ZeRO-1 and ZeRO-2. ZeRO-3 drew comparatively little power per GPU and still needed 1,030 kJ, because its run lasted so much longer.

At 8B the spread among FSDP-1/2 and ZeRO-1/2 was narrow, 912–984 kJ, and ZeRO-3 again stood apart at 1,360 kJ. FSDP-1 had the best Tokens/kJ at this size, with ZeRO 1/2 just behind. The consistency of this broader pattern across the evaluated model sizes supports its use as an input to the decision framework in Section 4.5.5.

4.5.3 Training Duration

The training time results expose the compute–communication trade-offs across methods. For the 3B model, ZeRO-1 and ZeRO-2 are fast at 226 s and 224 s, respectively, followed by DDP at 270 s and FSDP 1/2 at 274–277 s. ZeRO-3 requires long runtime at 542 s, due to its more aggressive state sharding, additional communication and parameter management overhead associated with this settings.

For the 8B model, FSDP 1/2 record short runtimes at 326 s and 318 s, followed by ZeRO-1/2 at 361 s and 354 s. ZeRO-3 again requires the long runtime at 560 s. These results show that the preferred strategy changes with model scale: ZeRO 1/2 provide the strongest time-to-train performance for 3B, while FSDP 1/2 are fast for 8B. More broadly, the runtime penalty associated with more aggressive sharding reinforces the need to consider time-to-train combined with memory and energy.

4.5.4 Memory Behavior

Memory usage also shows the differences across training methods, as highlighted in Figure 4.3. DP and DDP incur significant memory costs due to model replication, with DDP reaching 63.6 GB per GPU for the 3B model. FSDP and ZeRO each shard state, though not

in the same way. FSDP 1/2 used about 19–22 GB at 3B and 35–36 GB at 8B; ZeRO-3 used 24.5 GB and 37.9 GB.

ZeRO-3 shards parameters more aggressively by design, yet its measured footprint also includes framework buffers, metadata, and runtime state. Its advantage is better described as the ability to scale memory with model size than as the smallest footprint in every case. That advantage was paid for with longer runtime and more cumulative energy.

4.5.5 A Decision Framework for Energy-Aware Strategy Selection

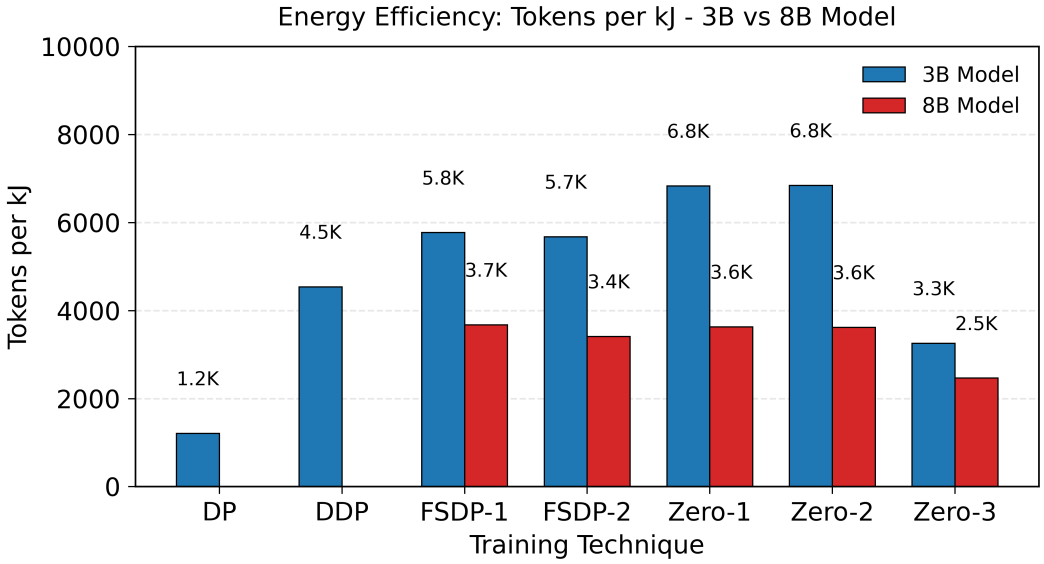


Figure 4.4: *Energy efficiency comparison (Tokens per kJ) across multi-GPU training techniques for 3B and 8B LLMs, computed against total GPU energy across the node across 8 GPUs. Higher is better.*

No strategy in these measurements was best on memory, energy, and time-to-train at once. Which one to prefer depends on the constraint that matters most for a deployment. We consolidate these observations into a practitioner-oriented decision framework:

- **If memory scalability is the binding constraint:** Full-shard FSDP and ZeRO-3 partition all three persistent state categories. Compare their measured peak-memory feasibility; the reported FSDP footprints are lower here. ZeRO-3 was built for memory

scaling, and here it paid for that with longer runtime and more cumulative energy, so it makes most sense when getting the model to fit is the first requirement.

- **If energy budget is the binding constraint:** On the 3B model, ZeRO-1/2 gave the most tokens per kilojoule (6,834–6,844 Tokens/kJ). On the 8B model FSDP-1 led with 3,678 Tokens/kJ, and ZeRO-1/2 were close behind at 3,619–3,633 Tokens/kJ.

- **If a balanced operating point is needed** across memory efficiency, runtime, and energy cost, FSDP and ZeRO-1/2 provide the strongest overall trade-offs across the evaluated configurations. These methods avoid the runtime and energy penalties observed with ZeRO-3 while maintaining substantially lower memory requirements than replicated approaches.

- **If time-to-train is the dominant constraint:** ZeRO-1/2 provide the shortest runtimes for the 3B model (224–226 s), while FSDP-1/2 are fastest for the 8B model (318–326 s). DDP was close behind on 3B, but because it replicates the model, its memory footprint limits it.

- **DP** serves only as a lower bound and is not recommended for deployment. On the 3B model it was the slowest and least energy-efficient configuration, at only 1,210 Tokens/kJ.

This framework reflects the behavior observed on a single-node, 8× H100 SXM5 platform with a fixed software stack, and the specific Tokens/kJ values should be interpreted as method-level tendencies rather than universal constants. Multi-node fabrics, other accelerator generations, more model scales, and broader workloads have not been tested and are left for future work. The method itself, which judges a strategy on memory, cumulative energy, and time-to-train together instead of on one system metric, does not depend on this platform.

4.6 Measurement Boundaries and Interpretation

The energy boundary in this chapter consists of the eight participating GPUs. Host CPU, system RAM, storage, network equipment outside the devices, and facility cooling are not included. When this chapter speaks of energy across the node, it means the GPU energy summed within that node; whole-server and facility energy were not measured. This distinction is consistent with the separation of device and facility boundaries established in Chapter 2[7;25].

Reported GPU utilization and thermal behavior support a descriptive comparison, but do not establish causal kernel-level explanations. Communication overlap, stalls, and tensor-core activity would require more detailed profiling. The chapter's evidence concerns execution efficiency for a fixed fine-tuning workload.

4.7 Implications for the Dissertation

The study demonstrates that a configuration with high GPU utilization or low average power can consume more cumulative energy. The favorable measured operating points depend on model size: ZeRO 1/2 leads the 3B energy efficiency comparison, while FSDP-1 has the high reported Tokens/kJ for 8B.

This also complements Chapter 3's finding that low precision does not guaranty the fast execution at every batch size. Across both levels, selecting an efficient configuration requires measurements of the workload profile with operational constraints. Chapter 5 extends the same reasoning to infrastructure-level by comparing H100 and B200 GPU platforms under distributed LLM and VLM workloads. That comparison examines whether newer hardware capability improves efficient work per unit energy, instead of assuming that fast accelerator implies better energy efficiency[9;26].

4.8 Chapter Summary

This chapter characterized the energy–memory–time trade-offs of multi-GPU LLM fine-tuning on an eight-GPU H100 system. The measured results show that no strategy provides the best value for every objective. Replicated execution is constrained by memory, while state sharding offers different balances between capacity and execution overhead. Tokens/kJ exposes cumulative energy differences that utilization, average power, or runtime alone cannot describe. The resulting decision framework supports workload-specific parallelism selection and establishes the training-system foundation for the infrastructure comparison in the next chapter.

Chapter 5

Infrastructure-Level Characterization through Workload-Aware GPU Evaluation

5.1 Introduction

The preceding chapters examined model-level numerical precision and training-level parallelism. This chapter completes the dissertation's progression by evaluating infrastructure-level choices. The architecture of GPU, the sustained power, and the operating environment influence whether a workload converts additional hardware capability into faster execution and more useful work per unit of energy. Accelerator selection therefore requires end-to-end measurement rather than a comparison of peak specifications alone[7;9;25].

The study compares single-node configurations containing eight NVIDIA H100 or eight NVIDIA B200 GPUs. Five large language models (LLMs) and three vision–language models (VLMs) are evaluated using a common Distributed Data Parallelism (DDP) approach. Unlike Chapter 4's full-parameter LLM fine-tuning experiments, the LLM workload in this chapter uses QLoRA. This difference is essential when interpreting the model feasibility and energy-efficiency values across chapters.

The central research question is whether platform-level improvements in execution performance also improve energy efficiency under evaluated workloads. A second question concerns how the resulting power profiles influence facility-scale operating scenarios. Cooling enters this analysis through the operating context and assumed power usage effectiveness (PUE), not through an independently controlled cooling experiment. The contribution is a workload-sensitive platform comparison and an explicitly scoped infrastructure analysis[6;26].

5.2 Methodology and Experimental Setup

The comparison uses two nodes, one with 8× NVIDIA H100 GPUs and one with 8× NVIDIA B200 GPUs, and records performance, power, and computational efficiency on each. Only actual LLM and VLM training jobs were benchmarked, all run with Distributed Data Parallel (DDP), so that the measurements describe multi-GPU behavior as it occurs in practice. For every workload and node, the reported values come from one sustained training run with a fixed seed. Figure 5.1 shows which metrics were measured directly and which were derived.

5.2.1 Training Workloads

Eight models were benchmarked. The five LLMs are Mistral-7B-v0.3[72], LLaMA-3.1-8B[70], Mistral-NeMo-Base-2407[73], Gemma-2-27B[74], and Qwen2.5-32B[75]; the three VLMs are X-CLIP[34], EVL[35], and ViTA-CLIP[36]. No hyperparameter differed between the H100 and B200 runs. Every job ran in BF16, which matters when reading the results, since an achieved TFLOPs figure would mean something different at another precision.

LLM Training Setup

Supervised fine-tuning of the LLMs used Alpaca[71] as training data. Fine-tuning was parameter-efficient, using Quantized Low-Rank Adaptation (QLoRA) with rank $r = 128$. A single epoch was run with 4 samples per GPU, which gives a global batch of 32 across the 8 GPUs under DDP.

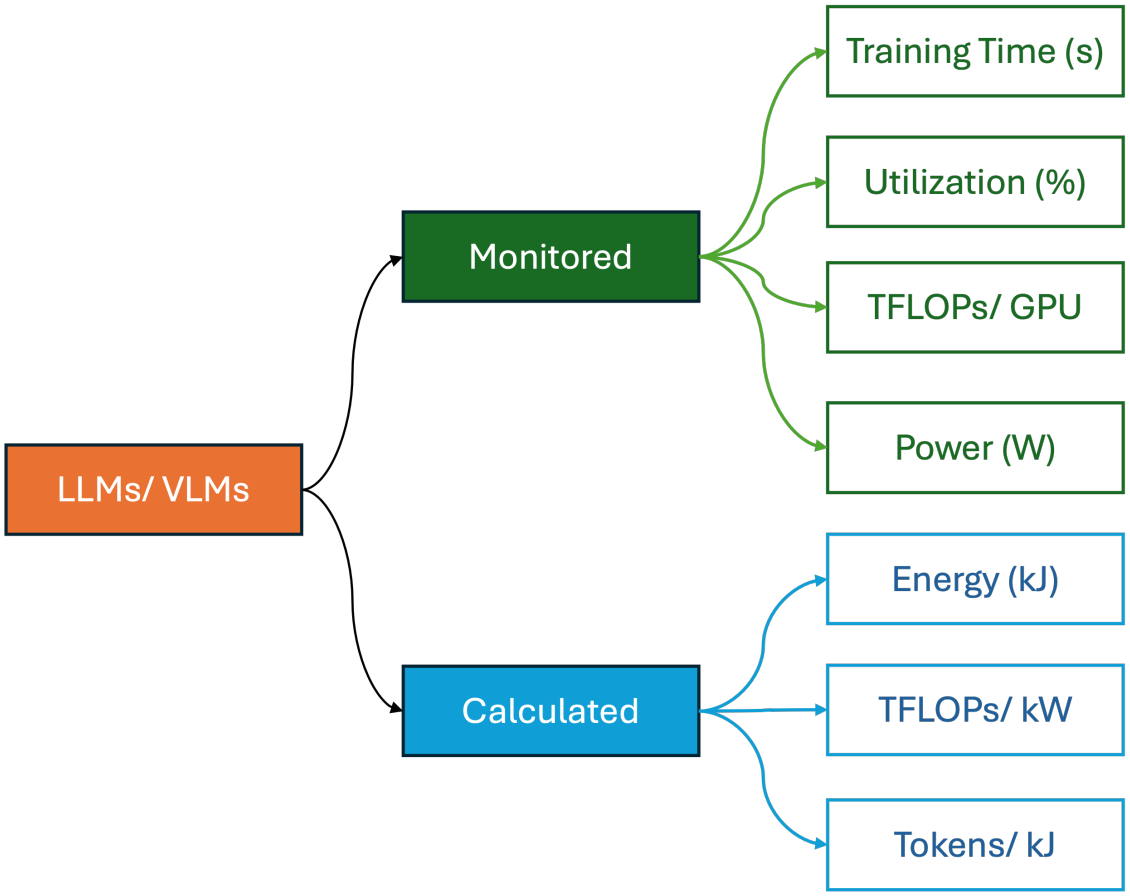


Figure 5.1: *Metrics classification based on measurement and calculation in this study.*

VLM Training Setup

The VLMs were trained on UCF101[76] for 50 epochs, with 4 clips per GPU (a global batch of 32) and 16 frames per clip. Both the H100 and the B200 node ran these jobs under 8-way DDP with exactly the same hyperparameters.

The settings for both platforms are collected in Table 5.1. With QLoRA, only the low-rank adapters receive gradient updates; the base weights remain in quantized form throughout. A given model saw the same seed and hyperparameters on H100 and on B200, so its two runs can be set side by side directly.

Table 5.1: *Documented workload settings for the H100–B200 comparison.*

Setting	LLM workloads	VLM workloads
Models	Mistral-7B-v0.3, Llama-3.1-8B, Mistral-NeMo, Gemma-2-27B, Qwen2.5-32B	X-CLIP, EVL, ViTA-CLIP
Dataset	Alpaca	UCF101
Execution	Eight-way DDP	Eight-way DDP
Computation precision	BF16	BF16
Training epochs	1	50
Batch size	4 per GPU; 32 globally	4 per GPU; 32 globally
Additional setting	QLoRA rank 128	16 frames per sequence
Runs per configuration	One sustained run	One sustained run

5.2.2 Measurement and Profiling Methodology

Telemetry was gathered for the whole of every run, both per GPU and for the node as a whole. GPU-side readings came from *NVIDIA-SMI* (version 570.195.03); on the node side, *ipmitool* polled power every 20 seconds.

Four quantities were kept in the logs:

- **GPU Utilization:** Live compute utilization of each GPU, showing whether it was saturated.
- **GPU Power Draw:** Power per GPU, read both as momentary values and as run averages; performance per watt is computed from these.

- **Estimated FLOP rate:** Effective TFLOPs/GPU over the run, taken from runtime FLOP estimators; hardware instruction counters were not read.
- **Training Time:** Elapsed wall-clock time of the complete training job (Figure 5.1).

5.2.3 Benchmarking Objective

Because both nodes were measured in the same way, sustained multi-GPU performance can be compared directly between the two architectures. The comparison covers the variation of training time across workloads, how busy and how efficient the GPUs were, and the power and energy spent per token. TFLOPs/GPU and TFLOPs/kW are also reported, together with the behavior of each architecture under realistic LLM and VLM training.

The protocol provides an empirical basis for comparing the evaluated platforms under matched workload settings. Complete reproduction also requires the corresponding software and training configuration files.

All experiments use Distributed Data Parallel (DDP), which keeps the comparison between the H100 and B200 nodes controlled. Larger models are often trained in production with sharded strategies such as Fully Sharded Data Parallel (FSDP) and ZeRO, but these were deliberately left out here. Sharding brings its own variables (optimizer-state partitioning, parameter partitioning, memory offloading, and altered communication patterns), and any of them could blur a hardware-level comparison of the two GPU generations. DDP therefore serves as one common baseline against which throughput, utilization, power draw, and energy efficiency are compared on the two systems. Sharded execution and multi-node deployment are left for future work; within a single node, FSDP and ZeRO were already studied in Chapter 4.

Table 5.2: *LLM comparison for the evaluated H100 and B200 configurations: (a) execution and (b) energy. Energy is the average per-GPU energy.*

(a) Execution measurements

Model	Node	Time (s)	GPU util. (%)	TFLOPs/GPU
Mistral-7B-v0.3	H100	2702	95.8	35.19
Mistral-7B-v0.3	B200	2370	96.6	46.56
Llama-3.1-8B	H100	2466	94.9	35.12
Llama-3.1-8B	B200	2191	96.2	40.64
Mistral-NeMo	H100	3827	95.8	35.71
Mistral-NeMo	B200	3323	97.1	42.85
Gemma-2-27B	H100	8393	92.7	39.17
Gemma-2-27B	B200	7111	98.5	50.96
Qwen2.5-32B	H100	9738	93.4	37.42
Qwen2.5-32B	B200	8302	98.70	48.7

(b) Energy measurements

Model	Node	Power (W/GPU)	Energy (kJ/GPU)	TFLOPs/kW	Reported tokens/kJ
Mistral-7B-v0.3	H100	437.8	1182.9	80.4	5634
Mistral-7B-v0.3	B200	579.8	1374.1	80.3	4850
Llama-3.1-8B	H100	438.1	1080.3	80.1	5217
Llama-3.1-8B	B200	580.4	1271.6	70.0	4432
Mistral-NeMo	H100	444.5	1701.1	80.3	3385
Mistral-NeMo	B200	604.3	2008.0	70.9	2867
Gemma-2-27B	H100	467.7	3925.4	83.8	1517
Gemma-2-27B	B200	616.3	4382.5	82.7	1358
Qwen2.5-32B	H100	458.8	4467.7	81.6	1270
Qwen2.5-32B	B200	605.1	5023.5	80.5	1129

Table 5.3: *VLM comparison for the evaluated H100 and B200 configurations: (a) execution and (b) energy. Energy is the average per-GPU energy.*

(a) Execution measurements

Model	Node	Time (s)	GPU util. (%)	TFLOPs/GPU
X-CLIP	H100	1503	58.6	0.377
X-CLIP	B200	2094	56.1	0.376
EVL	H100	14530	60.0	0.119
EVL	B200	16749	66.1	0.119
Vita-CLIP	H100	91654	97.4	0.106
Vita-CLIP	B200	47911	78.5	0.106

(b) Energy measurements

Model	Node	Power (W/GPU)	Energy (kJ/GPU)	TFLOPs/kW	Reported tokens/kJ
X-CLIP	H100	408.5	613.9	0.923	99414
X-CLIP	B200	400.6	838.8	0.938	72759
EVL	H100	290.0	4213.7	0.410	14483
EVL	B200	279.0	4673.0	0.426	13060
Vita-CLIP	H100	536.9	49209.0	0.198	1240
Vita-CLIP	B200	518.5	24841.8	0.205	2456

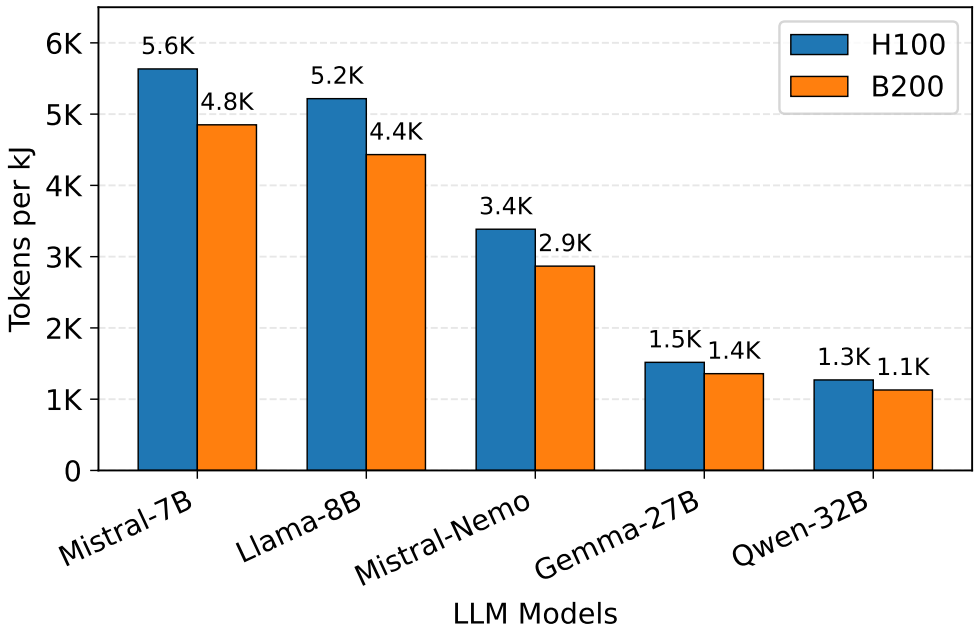


Figure 5.2: *Tokens-per-kilojoule comparison for the five LLM workloads. Values are normalized using average per-GPU energy. Token counts are model-specific because each model uses its own tokenizer; values are comparable across platforms within a model only.*

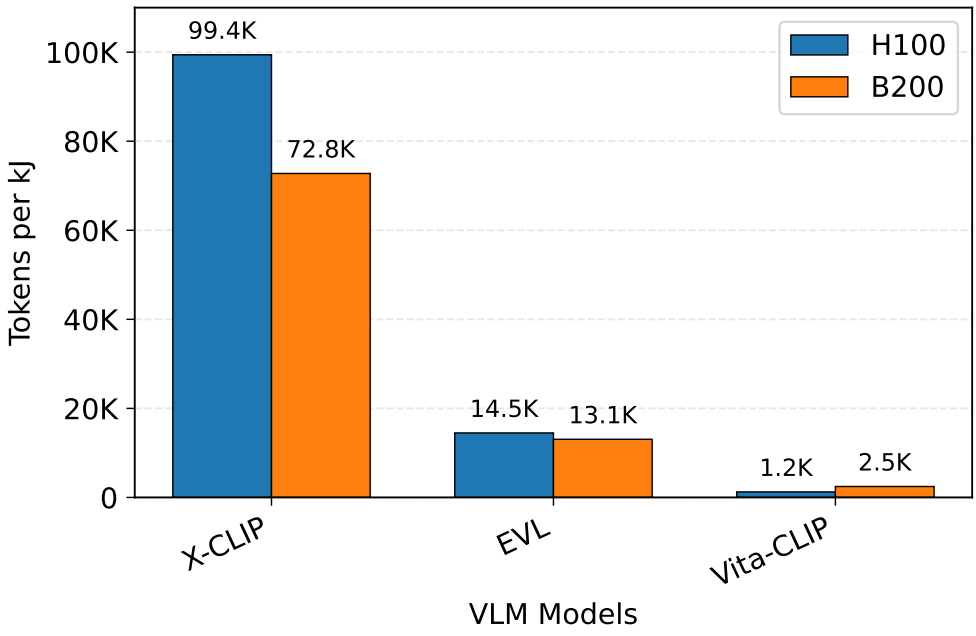


Figure 5.3: *Token-efficiency comparison for the three VLM workloads. VLM token or frame-work accounting must be specified before comparing these absolute values with LLM token efficiency.*

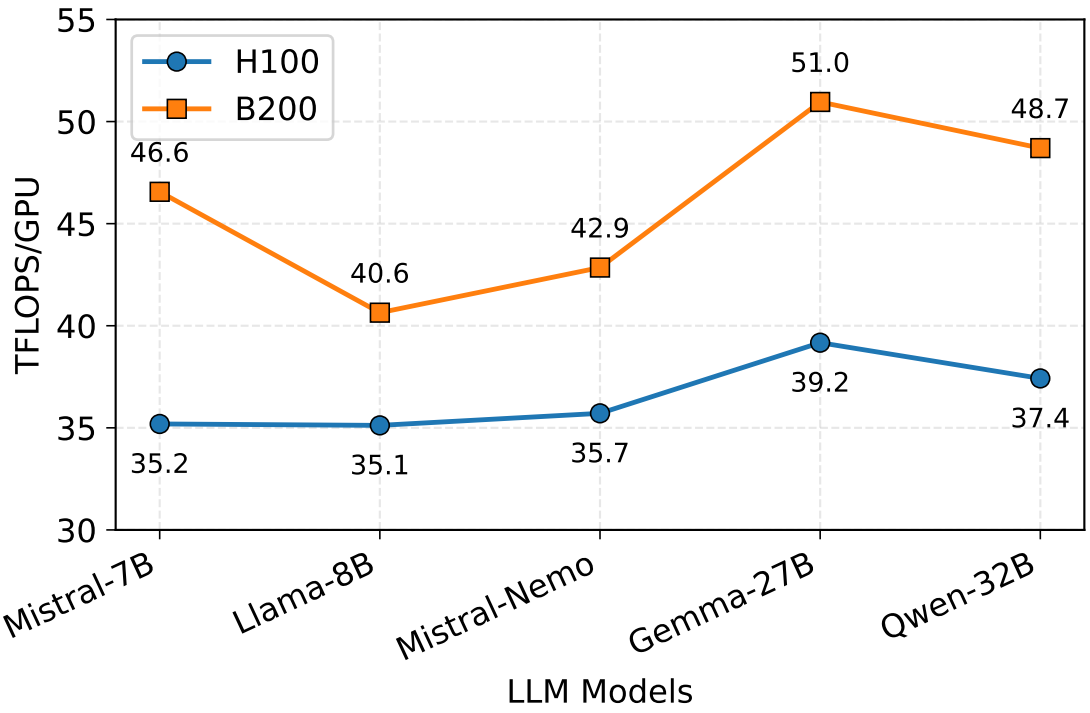


Figure 5.4: *Estimated TFLOPs/GPU comparison for LLM training on H100 and B200. Estimates are based on the runtime FLOP-counting methodology described in this chapter.*

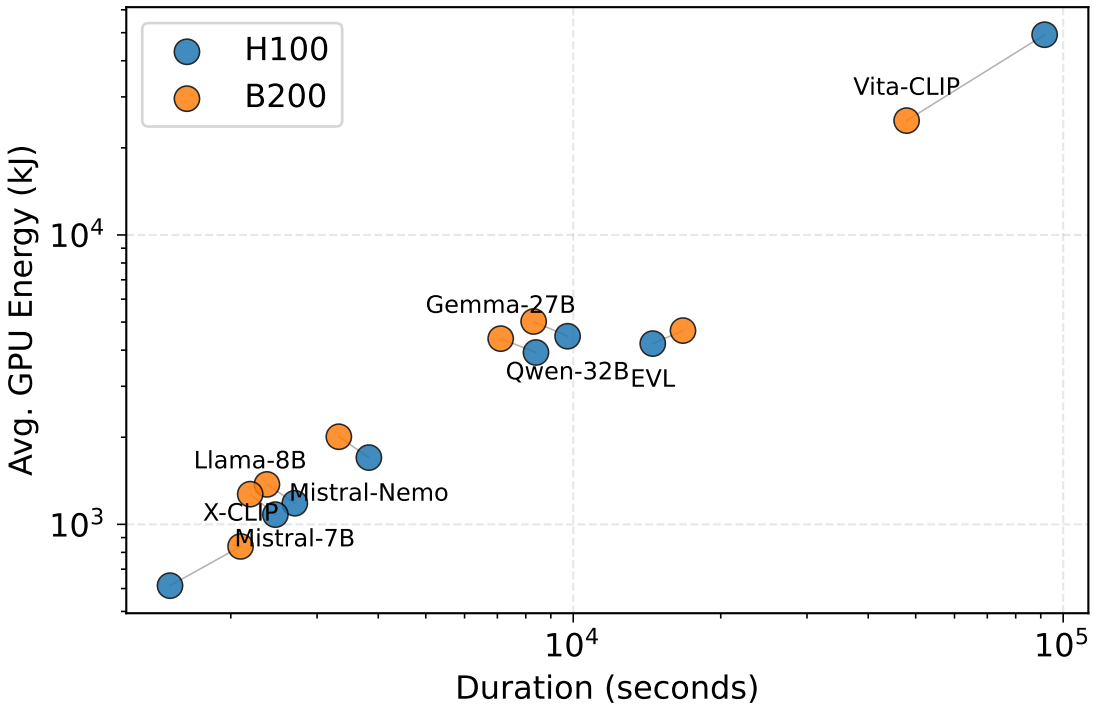


Figure 5.5: *Training duration and energy comparison across the evaluated H100 and B200 GPU configurations. Energy values follow the per-GPU boundary used in the workload table.*

5.3 Measured Performance and Energy Results

5.3.1 LLM Performance and Energy Trade-offs

Table 5.2 show faster reported execution on B200 for every evaluated LLM. Mistral-7B duration decreases from 2,702 s to 2,370 s, while Llama-3.1-8B decreases from 2,466 s to 2,191 s. Gemma-2-27B decreases from 8,393 s to 7,111 s, and Qwen2.5-32B decreases from 9,738 s to 8,302 s. These correspond to reductions in elapsed time of approximately 11–15% across the five workloads. Percentage reduction in duration and multiplicative speedup are distinct: the latter is obtained by dividing H100 time by B200 time.

B200 also records higher GPU utilization and higher estimated TFLOPs/GPU for these LLMs. Mistral-7B rises from 35.19 to 46.56 TFLOPs/GPU, an increase of approximately 32%. Gemma-2-27B rises from 39.17 to 50.96 TFLOPs/GPU. GPU utilization increases by 0.8 percentage points for Mistral-7B and 5.8 percentage points for Gemma-2-27B. Percentage-point changes should be distinguished from relative percentage changes in utilization.

The higher computational rates coincide with higher average GPU power. For example, Mistral-7B increases from 437.8 W to 579.8 W per GPU. Its shorter execution does not fully offset the power increase: reported per-GPU energy rises from 1,182.9 kJ to 1,374.1 kJ. The same direction is observed for all five LLMs. As a result, every LLM has a lower reported tokens/kJ on B200. TFLOPs/kW also drops on B200, although for Mistral-7B the rounded values, 80.4 and 80.3, are almost identical.

For the QLoRA workloads tested, then, B200 finishes training sooner and spends more GPU energy doing so.

The five LLMs use different tokenizers, with vocabularies of approximately 32K tokens (Mistral-7B-v0.3), 128K (Llama-3.1-8B), 131K (Mistral-NeMo, Tekken), 151K (Qwen2.5-32B), and 256K (Gemma-2-27B). The same Alpaca text therefore produces a different number of tokens for each model, and one token covers a different amount of text. For this reason, tokens/kJ is compared only within each model across H100 and B200, where the tokenizer and token count are identical. Differences in tokens/kJ between models reflect both computational cost and tokenization, and are not interpreted as differences in efficiency. A task-level comparison across models would require a tokenizer-independent unit, such as training examples per kilojoule.

5.3.2 VLM Performance and Workload Dependence

Table 5.3 show a different pattern across VLMs. X-CLIP takes 1,503 s on H100 and 2,094 s on B200, approximately 39% longer on B200. Its measured GPU utilization decreases from 58.6% to 56.1%, rather than increasing. Average GPU power also decreases slightly, from 408.5 W to 400.6 W, but per-GPU energy rises from 613.9 kJ to 838.8 kJ because of the longer execution interval.

EVL runs similarly longer on B200: 16,749 s compared to 14,530 s on H100, an increase of approximately 15%. On B200 its utilization went up, from 60.0% to 66.1%, and its power went down, from 290.0 W to 279.0 W. Per-GPU energy still grew from 4,213.7 kJ to 4,673.0 kJ. In both X-CLIP and EVL, lower power did not mean less energy to finish the job.

Table 5.4: *Facility summary of annual energy consumption and cost metrics under moderate and high GPU cluster loads for H100 and B200 nodes. (PUE: Power Usage Effectiveness)*

Metric	**Moderate Load (2,000 Nodes)**	**High Load (5,000 Nodes)**
Low-/High-Intensity Workload Mix	50% / 50%	15% / 85%
Avg. Node Power - H100 (W)	3,253	3,574
Avg. Node Power - B200 (W)	3,696	4,379
Annual Energy - H100 (GWh)	70.4	185.0
Annual Energy - B200 (GWh)	76.6	217.8
PUE (H100)	1.25	1.20
PUE (B200)	1.20	1.15
Electricity Rate (USD/kWh)	\$0.10	\$0.13
Annual Energy Cost - H100	\$7.04M	\$24.05M
Annual Energy Cost - B200	\$7.66M	\$28.31M
Annual Cost Overhead (B200 vs. H100)	+\$0.62M	+\$4.26M

Vita-CLIP reverses the energy outcome. Duration falls from 91,654 s to 47,911 s, a reduction of approximately 47.7% and a speedup of approximately 1.91. Average GPU power decreases from 536.9 W to 518.5 W, while per-GPU energy falls from 49,209.0 kJ to 24,841.8 kJ. Reported token efficiency rises from 1,240 to 2,456. Its utilization is lower on B200, 78.5% rather than 97.4%, reinforcing the observation from Chapter 4 that utilization alone is not an adequate measure of productive execution.

5.4 Facility-Level Energy and Electricity-Cost Scenarios

The study extends measured workload power profiles to two hypothetical deployment scenarios: 2,000 eight-GPU nodes under a moderate workload mix and 5,000 nodes under a high-intensity mix. These projections examine annual electricity requirements under fixed operating assumptions. They are distinct from the directly measured single-node runs and from a fixed-completed-work comparison.

5.4.1 Scenario Assumptions

The moderate scenario assigns 50% of workload activity to lower-intensity jobs and 50% to higher-intensity jobs. The study reports weighted average node powers of 3,253 W for H100 and 3,696 W for B200, 99% uptime, and PUE assumptions of 1.25 and 1.20. The electricity rate is \$0.10/kWh. The high-load scenario assigns 85% of activity to higher-intensity workloads and 15% to lower-intensity workloads. Reported weighted node powers are 3,574 W and 4,379 W, with PUE values of 1.20 and 1.15 and an electricity rate of \$0.13/kWh. The study cites electricity-rate information and cooling infrastructure context[77;78].

The study additionally assumes a 10% power-delivery loss. These PUE values are scenario assumptions, not measured facility outcomes. Different assumed cooling overheads contribute to the modeled platform comparison. A generic estimate using average IT power $\bar{P}_{\mathrm{IT}}$ in kW, N nodes, active hours H, and assumed PUE is

$$E_{\mathrm{facility,kWh}} \approx N \bar{P}_{\mathrm{IT}} H \,\mathrm{PUE}, \qquad C = r E_{\mathrm{facility,kWh}}, \tag{5.1}$$

where C is the annual electricity cost and r is the electricity rate.

5.4.2 Projected Energy and Cost

Table 5.4 reports 70.4 GWh annually for the moderate H100 scenario and 76.6 GWh for B200. Priced at \$0.10/kWh, these come to \$7.04 million and \$7.66 million, so B200 costs \$0.62 million more per year. The gap widens in the high-load scenario, where the totals are 185.0 GWh and 217.8 GWh and the rounded costs \$24.05 million and \$28.31 million, about \$4.26 million apart. How large the gap becomes depends on operating time, workload mix, device power, and facility overhead.

As a check, 70.4 GWh at \$0.10/kWh gives \$7.04 million, and 217.8 GWh at \$0.13/kWh gives about \$28.31 million, matching the rounded costs. This confirms only the conversion from energy to cost. It does not validate the power projection itself, and total cost of ownership lies outside its scope. Running the same number of nodes for the same hours also

does not mean that both platforms complete the same amount of work.

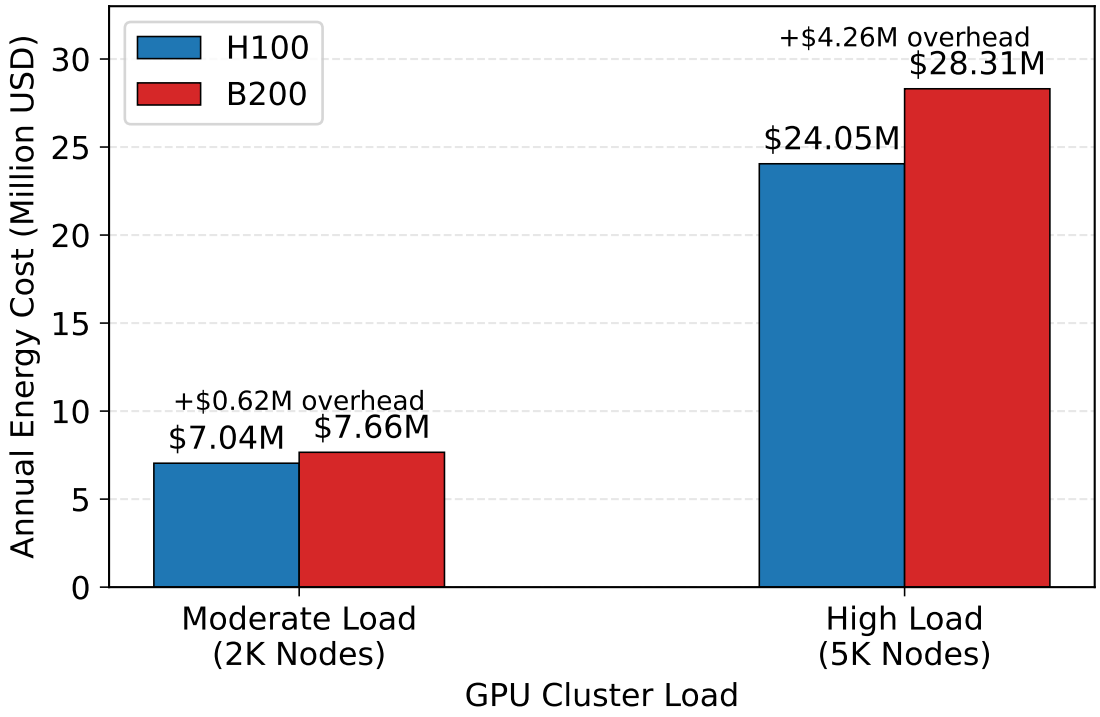


Figure 5.6: *Projected annual electricity costs under the moderate- and high-load scenarios. The values follow from the assumed workload mix, power, PUE, uptime, and electricity rates. No facility of this size was measured.*

5.5 Workload-Aware Infrastructure Selection

Which platform to choose depends on the binding constraint. If completion time matters most, B200 ran every tested LLM configuration faster. If GPU energy for the same LLM run matters most, H100 used less. The VLMs split. Vita-CLIP was both faster and lower in energy on B200, whereas X-CLIP and EVL did better on H100 on both counts in the reported configurations.

Memory feasibility, software support, available power, and operating requirements also enter the decision. Nothing in these results ranks one accelerator above the other for every model and metric, and neither GPU utilization nor theoretical FLOPs would have picked the better platform on its own. Chapter 4 reached the same conclusion for training strategies;

here it applies to the hardware.

5.6 Scope, Limitations, and Future Evaluation

The LLM runs in this chapter use QLoRA, whereas Chapter 4 fine-tuned all parameters. Results from the two chapters come from different controlled comparisons and are not combined.

Several gaps remain. The energy boundary and the count of processed work differ between chapters and should be aligned. FLOP estimates need checking against hardware counters, host and GPU power should be reported separately, and repeated runs are needed to measure variation. Running FSDP and ZeRO on both accelerators would tie this chapter more closely to Chapter 4. Power-cap sweeps, controlled cooling, more workload settings, and multi-node runs would show how far the present results generalize.

5.7 Chapter Summary

H100 and B200 systems were compared here on the same LLM and VLM workloads. B200 trained every LLM faster but used more GPU energy. Among the VLMs, Vita-CLIP did better on B200 in both time and energy, and X-CLIP and EVL did better on H100. The facility scenarios then translated these per-run differences into annual electricity cost.

With this chapter, the empirical work at all three levels is complete: precision in Chapter 3, parallelism in Chapter 4, and the accelerator platform here. At each level, the efficient choice could be identified only by measuring the workload against the objectives of the deployment. Chapter 6 synthesizes these findings across the three optimization levels. Chapter 7 presents the final conclusions and future research directions.

Chapter 6

Unified Analysis and Discussion

6.1 Purpose and Scope of the Unified Analysis

Chapters 3, 4, and 5 were carried out as separate studies, each with its own hardware and its own variable. Numerical precision was varied on a single GPU, the parallelism strategy on an eight-GPU H100 node, and the platform itself, H100 or B200, under a fixed DDP setup. Read together, they bear on a practical question for anyone configuring a GPU job, namely whether measurements of this kind can say which configuration to choose when accuracy, speed, memory, and energy point in different directions. Some effects recurred across the studies and others held only where they were measured; both are examined below.

Because the workloads, hardware, precision settings, and energy boundaries were not the same, the three studies are not pooled into one experiment. Chapter 7 covers contributions, limitations, and future work.

6.2 Shared Findings across Optimization Levels

6.2.1 A Common Principle of Workload-Aware Selection

In every study, efficiency had to be measured on the workload; it could not be read off the configuration. Turning on AMP in Chapter 3 made several small-batch runs slower. With

the 3B model in Chapter 4, ZeRO-3 sharded more training state than any other strategy, but ZeRO-1 and ZeRO-2 finished the same 3,353,700 tokens with less energy. Chapter 5 gives the sharpest example. Mistral-7B ran at 437.8 W per GPU on H100 and at 579.8 W on B200, and the shorter B200 run was not short enough to make up the difference; per-GPU energy went from 1,182.9 kJ to 1,374.1 kJ.

Each level has its own explanation for results like these. With AMP, tensors are stored differently and different kernels run, but for small batches on ResNet-50 and MobileNet_v1 this did not translate into faster training. Going from DDP to FSDP or ZeRO trades replicated state for extra communication and for transient buffers on each GPU. B200 changes compute, memory, and power relative to H100 in one step, and a faster GPU can also expose communication or data-delivery limits that were less visible on the older platform[8;9]. Every study moved only one of these factors, so how they combine has not been measured. That is left for future work.

6.2.2 Multi-Objective Efficiency

Four numbers are attached to every configuration in this dissertation. The first is accuracy, because a fast model that classifies poorly is of no use; Chapter 3 reported it next to its latency and throughput figures for that reason. Memory decides whether the run can start at all; the 8B DP and DDP runs of Chapter 4 never got past this point. Completion time matters under a deadline. Energy is what the finished job cost. None of the four can be inferred from the others.

The order of checks follows from this. Quality and memory come first and act as filters, and time and energy are compared only among the configurations that pass them. The 8B model in Chapter 4 is an example. With DP and DDP out, FSDP-1 and FSDP-2 were left. Of those two, FSDP-1 reached the higher Tokens/kJ while FSDP-2 finished first. A team working to a deadline would likely take FSDP-2. One working under an energy budget would take FSDP-1, and each choice would be right for its own job.

6.3 Comparison of Experimental Scope and Evidence

Table 6.1 sets out what each study covered. Read across its rows, the variable that changed is different in every study, and so is the hardware, which ranges from a single Quadro RTX 4000 to eight-GPU B200 nodes. Pooling the numbers into one efficiency ranking would therefore mix unlike things.

Table 6.1: *Experimental scope and evidence across the dissertation's three optimization levels.*

Dimension	Model level	Training-system level	Infrastructure level
Workloads	Vision training and inference	Full-parameter LLM fine-tuning	QLoRA LLM and VLM training
Varied choice	AMP and TF-TRT precision	DP, DDP, FSDP, ZeRO	H100 versus B200
Hardware scope	Single Quadro RTX 4000	Eight-GPU H100 node	Eight-GPU H100 and B200 nodes
Energy evidence	No direct energy measurement	Summed energy across eight GPUs	Average per-GPU energy; facility scenarios
Quality evidence	Classification accuracy	Fixed processed workload; no equal-quality convergence claim	Fixed training setup; no equal-quality convergence claim
Decision supported	Precision and batch configuration	Parallelism under resource constraints	Workload-aware platform selection

The table also settles a point that could otherwise confuse a reader. In Chapter 4, full-parameter training of the 8B model did not fit under DP or DDP. Chapter 5, though, ran larger LLMs under DDP without trouble. That was possible because of QLoRA. The base weights stay quantized and only small low-rank adapters are trained, so far less trainable state sits on each GPU than in full-parameter training.

6.4 Practical Configuration-Selection Framework

The findings can also be read as a working order for choosing a deployment configuration. It begins with a definition of useful work and quality: what one unit of work is (tokens, in Chapters 4 and 5), what inputs the task sees, and how far accuracy may drop. Feasibility comes next. A run that does not fit in GPU memory, or that the software stack cannot

support at the chosen precision, strategy, and platform, is dropped at this point, as DP and DDP were for the 8B model in Chapter 4. For the candidates that remain, runtime, throughput, memory, and power are recorded over the same interval, in the way the telemetry of Chapters 4 and 5 was collected. Among those that finish in time, the one with the least cumulative energy is preferred. Only then is the result scaled to the host and the facility with the power capacity and tariffs of the site; Chapter 5 did this when it projected 70.4 GWh a year for the moderate H100 scenario. Nothing in this sequence goes beyond the methods of Chapters 3–5. Its use lies in fixing what has to be measured, and when, before a configuration is chosen.

6.5 Summary

What the unified analysis shows, in the end, is that the needs of the application should decide the optimization. No study here varied precision, parallelism, and platform together. Such a study remains to be done, and Chapter 7 returns to it along with the contributions and limitations of the dissertation.

Chapter 7

Conclusions and Future Research

7.1 Dissertation Overview

This dissertation studied the performance and resource efficiency of GPU-based AI systems at three levels. The first study looked at reduced precision and execution frameworks for vision models, in both training and inference. The second turned to multi-GPU LLM training. There, GPU energy was measured directly and reported as Tokens per Kilojoule (Tokens/kJ), next to memory usage and completion time. The third study ran LLM and VLM workloads on H100 and B200 platforms and compared their performance and energy. Its results were then used in facility-level scenarios to estimate what each accelerator would cost to operate.

One question ran through all three studies, namely whether measurements of reduced precision, multi-GPU execution, and accelerator performance can guide configuration decisions when accuracy, runtime, memory, and energy have to be balanced. In the experiments, no optimization method or hardware platform came out ahead on every workload. Lower precision reduced memory but did not always shorten execution time. The gains from state sharding, and from moving to a newer GPU, also depended on the workload.

The main result of the dissertation is therefore a set of empirical findings for workload-aware configuration. Since the studies used different experimental conditions and addressed different parts of GPU-based AI systems, each is interpreted on its own terms before the

three are compared. None of the experiments varied numerical precision, parallelism, and GPU generation at the same time. Chapter 6 compared the findings across levels; this chapter summarizes the contributions and limitations and outlines future research.

7.2 Summary of Research Findings

7.2.1 Model Level Optimization: Quantization and Mixed Precision

Chapter 3 examined TensorFlow automatic mixed precision (AMP) during training and TensorFlow–TensorRT (TF-TRT) during inference using VGG16, ResNet-50, and MobileNet_v1. The study evaluated classification accuracy, training time, memory usage, inference latency, and throughput across different batch sizes and two dataset settings. The practical effects of reduced precision depended on the model architecture and on the characteristics of the workload.

AMP reduced training memory usage in the evaluated configurations and allowed VGG16 and ResNet-50 to run a Cats vs Dogs batch-size configuration that otherwise failed with out-of-memory (OOM) errors. Training-time improvements were less consistent. Some ResNet-50 and MobileNet_v1 configurations with smaller batch sizes ran more slowly with AMP, while larger configurations achieved substantial speedups. Memory requirements and execution time therefore need to be evaluated together, since a gain in one did not produce a proportional gain in the other.

The inference experiments separated the effect of execution-framework optimization from that of precision reduction. TF-TRT in FP32 already outperformed native TensorFlow, so graph and execution optimizations provided benefits before any reduction in numerical precision. Dropping to FP16 or INT8 usually helped further. INT8, however, did not always beat FP16. Accuracy also shifted from one training setting to another, so a faster inference configuration is worth having only if the accuracy it delivers is acceptable.

The study characterizes existing precision optimizations at the framework level. It does

not propose a new quantization technique, and it did not measure energy. Its findings support selecting precision and batch size through workload-specific profiling in which accuracy, execution time, and memory requirements are considered together.

7.2.2 Training-Level Optimization: Distributed Execution and Tokens/kJ

Chapter 4 moved from single-model execution to the full training system. The workload was full-parameter supervised fine-tuning of Llama at 3B and 8B, run on a single node with eight H100 GPUs. Seven setups were compared (DP, DDP, two FSDP implementations, and ZeRO stages 1–3), and every one of them read its data through the same pipeline. Sampled power from all eight GPUs was integrated to obtain GPU energy, and Tokens per Kilojoule (Tokens/kJ) was computed by dividing the processed tokens by that energy.

No single metric described training performance on its own; GPU utilization, average power, runtime, and memory usage each showed something different. With the 3B model, ZeRO-1 and ZeRO-2 came out on top for both reported Tokens/kJ and runtime, and FSDP did well on memory and energy. DDP's runtime was close to the leaders at the price of much higher memory use, while DP finished last on speed and on energy efficiency. Moving to 8B reshuffled the order: FSDP-1 had the best reported Tokens/kJ, FSDP-2 the shortest training time, and DP and DDP no longer fit in GPU memory.

ZeRO-3 shows why neither average power nor the amount of state a method shards on paper is a safe guide. Although ZeRO-3 shards parameters, gradients, and optimizer states, it did not reach the most favorable operating point in the evaluated configurations. Its longer runtime produced higher cumulative GPU energy consumption than FSDP and ZeRO-1/2, and its measured memory footprint was larger than that of FSDP. Memory feasibility therefore depends on the memory required during execution, including transient runtime state, as well as on the degree of state sharding.

The principal contribution of this study is a controlled, multidimensional methodology for measuring distributed training strategies, which also gives an empirical basis for choosing

a parallelism technique according to workload requirements. Tokens-per-energy metrics have been used in previous research[22;23]; this dissertation applies Tokens/kJ to the comparison of parallelism strategies and reports it alongside runtime, memory usage, and GPU energy consumption.

7.2.3 Infrastructure-Level Optimization: GPU Platforms and Operating Conditions

Chapter 5 extended the evaluation to the infrastructure level by comparing eight-GPU H100 and B200 platforms across five LLM and three VLM workloads, with the LLM experiments using QLoRA under DDP. Workload duration, GPU utilization, power consumption, and energy usage were measured, and FLOPs and energy efficiency were estimated from these measurements. The measurements were also used in simulated scenarios to estimate the effect of the hardware choice at the scale of a full facility.

On every evaluated LLM workload, B200 finished sooner than H100 and reached a higher estimated TFLOPs/GPU. It also used more GPU energy per run, so its Tokens/kJ was lower on each of these workloads. In these experiments, speed and computational throughput did not bring better energy efficiency.

The VLM workloads did not follow one pattern. On B200, X-CLIP and EVL ran longer and used more energy, even though their average GPU power was lower. Vita-CLIP behaved differently. It finished approximately 1.91 times faster on B200, used substantially less energy, and reported lower GPU utilization. GPU utilization, hardware generation, and model family would not have predicted these results, which is why accelerator selection has to rely on measurements of the workload itself.

Annual electricity use was then estimated for several workload mixes, node counts, operating hours, electricity rates, and PUE values. These scenarios show how differences in hardware performance and energy can grow into differences in operating cost at scale. They are models built on stated assumptions, not measurements from a real facility.

7.3 Dissertation Contributions

Five contributions come out of this work:

1. Measurements of mixed-precision training and quantized inference as TensorFlow and TF-TRT implement them, covering accuracy, speed, and memory for three vision architectures over a range of batch sizes.

2. A way of profiling multi-GPU LLM training at the system level, in which GPU telemetry, cumulative GPU energy, token counts, memory, and completion time are recorded together so that configurations can be ranked on efficiency.

3. Measurement-based guidance on when replicated parallelism is the better choice and when sharded parallelism is, depending on the workload.

4. An H100 versus B200 comparison on LLM and VLM training that keeps speed and energy efficiency apart and finds that the platform which comes out ahead depends on the workload.

5. A cross-level reading of the model, training, and infrastructure results, which makes the case for workload-aware evaluation and states the measurement limits that bound each result.

All of this is empirical work. No new quantization algorithm, distributed optimizer, or GPU architecture is introduced, and no claim is made to a universal efficiency model. The dissertation instead shows how techniques that already exist can be measured and chosen between, using observed performance and energy together with the limits of the system at hand.

7.4 Future Research Directions

7.4.1 Joint Precision, Parallelism, and Platform Evaluation

The most direct extension is one experiment in which numerical precision, training strategy, and GPU platform are varied together on a shared workload. It would show whether a precision setting that works well with one sharding policy, or on one architecture, still works well with another. It would also show whether these interactions shift the energy–memory–time design space enough to change which configuration ranks first.

7.4.2 Multi-Node Scaling and Communication

A second extension is to move from one node to several, with the network topology and bandwidth characterized explicitly. Tensor and pipeline parallelism could then be tested against the replicated and sharded data-parallel strategies used in this work. The measurements would need to separate communication, computation, and their overlap, and to include networking energy whenever it lies within the chosen system boundary[8;9].

7.4.3 Power and Thermal Operating Policies

Power caps, frequency settings, and controlled cooling conditions add a further dimension to workload-aware evaluation. Sweeping these settings against training or inference performance targets would identify operating points that meet completion-time and quality constraints with less cumulative energy. Lowering instantaneous power alone does not guarantee this outcome[39;40].

7.5 Concluding Remarks

Making GPU-based deep learning efficient involves decisions at several levels of the system. Precision affects how the model executes, and the multi-GPU strategy affects how training state is stored and exchanged. The accelerator platform, in turn, sets the compute resources

and power available to the workload. Across the three studies, the benefit of each choice depended on the workload being executed and on the objectives that mattered for it.

The combined evidence supports workload-aware optimization grounded in measured quality, feasibility, completion time, and energy. It also shows that consistent accounting is needed as conclusions move from the model to the training system and on to the facility. These findings offer a practical basis for evaluating and building AI systems that use computing resources more efficiently.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *nature*, 521(7553): 436–444, 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 60(6):84–90, 2017.

[3] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

[4] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep Learning Based Recommender System: A survey and New Perspectives. *ACM computing surveys (CSUR)*, 52(1):1–38, 2019.

[5] R. Bommasani, D. Hudson, E. Adeli, R. Altman, S. Arora, S. Arx, M. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. Koh, M. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. Li, X. Li, T. Ma, A. Malik, C. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. Niebles, H. Nilforoshan, J. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. Roohani, C. Ruiz, J. Ryan, C. Ré, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. Srinivasan, A. Tamkin, R. Taori,

A. Thomas, F. Tramèr, R. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. Xie, M. Yasunaga, J. You, M. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. On the opportunities and risks of foundation models. 2021. doi: 10.48550/arxiv.2108.07258.

[6] Lynn H. Kaack, Priya L. Donti, Emma Strubell, George Kamiya, Felix Creutzig, and David Rolnick. Aligning artificial intelligence with climate change mitigation. *Nature Climate Change*, 12(6):518–527, 2022. ISSN 1758-6798. doi: 10.1038/s41558-022-01377-7. URL https://doi.org/10.1038/s41558-022-01377-7.

[7] Imran Latif, Alex C Newkirk, Matthew R Carbone, Arslan Munir, Yuewei Lin, Jonathan Koomey, Xi Yu, and Zhihua Dong. Single-node power demand during ai training: Measurements on an 8-gpu nvidia h100 system. *IEEE Access*, 2025.

[8] Bagus Hanindhito, Bhavesh Patel, and Lizy K John. Bandwidth characterization of deepspeed on distributed large language model training. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 241–256. IEEE, 2024.

[9] Joyjit Kundu, Wenzhe Guo, Ali BanaGozar, Udari De Alwis, Sourav Sengupta, Puneet Gupta, and Arindam Mallik. Performance modeling and workload analysis of distributed large language model training and inference. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*, pages 57–67. IEEE, 2024.

[10] Eshed Ohn-Bar and Mohan Manubhai Trivedi. Looking at Humans in the Age of Self-driving and Highly Automated Vehicles. *IEEE Transactions on Intelligent Vehicles*, 1 (1):90–104, 2016.

[11] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end Learning for Self-driving Cars. *arXiv preprint arXiv:1604.07316*, 2016.

[12] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1527–1533. IEEE, 2017.

[13] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training Deep Neural Networks with Low Precision Multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[14] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[15] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the Speed of Neural Networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[16] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K Bansal, William Constable, Oguz Elibol, Scott Gray, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Carey Kloss, Ruby J Pai, and Naveen Rao. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[17] Muhammad Ali Shafique, Arslan Munir, and Joonho Kong. Deep learning performance characterization on gpus for various quantization frameworks. *AI*, 4(4):926–948, 2023. ISSN 2673-2688. doi: 10.3390/ai4040047. URL https://www.mdpi.com/2673-2688/4/4/47.

[18] Areej Mehboob, Muhammad Ali Shafique, Muhammad Usman Qadeer, and Hamza Farooq. Beyond accuracy: How quantization affects coding agents, 2026. URL https://arxiv.org/abs/2611.02368.

[19] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Re-

won Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

[20] Nate Rattner and Tom Dotan. The ai spending boom in charts. *The Wall Street Journal*, September 2024. URL https://www.wsj.com/tech/ai/artificial-intelligence-investing-charts-7b8e1a97. Tech Section.

[21] Muhammad Ali Shafique, Imran Latif, Hayat Ullah, Alex C. Newkirk, and Arslan Munir. Scalable and energy-efficient ai: System-level profiling of nvidia gpu clusters for distributed llm training. *AI*, 7(7), 2026. ISSN 2673-2688. doi: 10.3390/ai7070232. URL https://www.mdpi.com/2673-2688/7/7/232.

[22] Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. From words to watts: Benchmarking the energy costs of large language model inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9. IEEE, 2023.

[23] Jared Fernandez, Clara Na, Vashisth Tiwari, Yonatan Bisk, Sasha Luccioni, and Emma Strubell. Energy considerations of large language model inference and efficiency optimizations. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32556–32569, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1563. URL https://aclanthology.org/2025.acl-long.1563/.

[24] Chenxu Niu, Wei Zhang, Yongjian Zhao, and Yong Chen. Energy efficient or exhaustive? benchmarking power consumption of llm inference engines. *ACM SIGENERGY Energy Informatics Review*, 5(2):56–62, 2025.

[25] Alex C. Newkirk, Jared Fernandez, Jonathan Koomey, Imran Latif, Emma Strubell,

Arman Shehabi, and Constantine Samaras. Empirically-calibrated h100 node power models for accurate ai training energy estimation. *Environmental Research: Energy*, 2 (4):045016, 2025.

[26] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, and Ricardo Bianchini. Rearchitecting datacenter lifecycle for ai: A tco-driven framework. *arXiv preprint arXiv:2509.26534*, 2025.

[27] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

[28] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018.

[29] Nvidia. Accelerating inference in tensorflow with tensorrt user guide, . URL https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html.

[30] Nvidia. Nvidia tensorrt, . URL https://developer.nvidia.com/tensorrt.

[31] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020. URL https://arxiv.org/abs/2006.15704.

[32] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023. URL https://arxiv.org/abs/2304.11277.

[33] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory

optimizations toward training trillion parameter models, 2020. URL https://arxiv.org/abs/1910.02054.

[34] Bolin Ni, Houwen Peng, Minghao Chen, Songyang Zhang, Gaofeng Meng, Jianlong Fu, Shiming Xiang, and Haibin Ling. Expanding language-image pretrained models for general video recognition. In *European conference on computer vision*, pages 1–18. Springer, 2022.

[35] Ziyi Lin, Shijie Geng, Renrui Zhang, Peng Gao, Gerard De Melo, Xiaogang Wang, Jifeng Dai, Yu Qiao, and Hongsheng Li. Frozen clip models are efficient video learners. In *European Conference on Computer Vision*, pages 388–404. Springer, 2022.

[36] Syed Talal Wasim, Muzammal Naseer, Salman Khan, Fahad Shahbaz Khan, and Mubarak Shah. Vita-clip: Video and text adaptive clip via multimodal prompting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23034–23044, 2023.

[37] Imran Latif, Muhammad Ali Shafique, Hayat Ullah, Alex C. Newkirk, Xi Yu, and Arslan Munir. Cooling matters: Benchmarking large language models and vision-language models on liquid-cooled versus air-cooled h100 gpu systems. *IEEE Transactions on Cloud Computing*, 14(3):1645–1656, 2026. doi: 10.1109/TCC.2026.3700971.

[38] Bharath Ramakrishnan, Cam Turner, Husam Alissa, Dennis Trieu, Felipe Rivera, Luke Melton, Muralikrishna Rao, Sruti Chigullapalli, Tatek Getachew, Vladimir Prodanovic, et al. Understanding the impact of data center liquid cooling on energy and performance of machine learning and artificial intelligence workloads. *Journal of Electronic Packaging*, 147(2):021003, 2025.

[39] Dan Zhao, Siddharth Samsi, Joseph McDonald, Baolin Li, David Bestor, Michael Jones, Devesh Tiwari, and Vijay Gadepally. Sustainable supercomputing for ai: Gpu power capping at hpc scale. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, pages 588–596, 2023.

[40] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrier, Nithish Mahalingam, and Ricardo Bianchini. Polca: Power oversubscription in llm cloud providers. *arXiv preprint arXiv:2308.12908*, 2023.

[41] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[42] Hayat Ullah, Muhammad Ali Shafique, Abbas Khan, and Arslan Munir. Dvfl-net: A lightweight distilled video focal modulation network for spatio-temporal action recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 36(3):2952–2967, 2026. doi: 10.1109/TCSVT.2025.3615909.

[43] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning Structured Sparsity in Deep Neural Networks. *Advances in neural information processing systems*, 29, 2016.

[44] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism. *ACM SIGARCH Computer Architecture News*, 45(2):548–560, 2017.

[45] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime Neural Pruning. *Advances in neural information processing systems*, 30, 2017.

[46] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. Clustering Convolutional Kernels to Compress Deep Neural Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 216–232, 2018.

[47] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. *Advances in neural information processing systems*, 31, 2018.

[48] Brian Chmiel, Liad Ben-Uri, Moran Shkolnik, Elad Hoffer, Ron Banner, and Daniel Soudry. Neural gradients are near-lognormal: improved quantized and sparse training, 2020.

[49] Fartash Faghri, Iman Tabrizian, Ilia Markov, Dan Alistarh, Daniel M Roy, and Ali Ramezani-Kebrya. Adaptive gradient quantization for data-parallel sgd. *Advances in neural information processing systems*, 33:3174–3185, 2020.

[50] Mauricio Fadel Argerich and Marta Patiño-Martínez. Measuring and improving the energy efficiency of large language models inference. *IEEE Access*, 12:80194–80207, 2024. doi: 10.1109/ACCESS.2024.3409745.

[51] Palak, Tella Rajashekhar Reddy, Bhaskar Kataria, Rohan Gandhi, Karan Tandon, Debopam Bhattacherjee, and Venkata N. Padmanabhan. Improving training time and gpu utilization in geo-distributed language model training, 2025. URL https://arxiv.org/abs/2411.14458.

[52] Runsheng Guo, Utkarsh Anand, Khuzaima Daudjee, and Rathijit Sen. Zorse: Optimizing llm training efficiency on heterogeneous gpu clusters. *Proceedings of Machine Learning and Systems*, 8:1569–1582, 2026.

[53] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. Rail-only: A low-cost high-performance network for training llms with trillion parameters. In *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 1–10. IEEE, 2024.

[54] Qingkai Meng, Hao Zheng, Zhenhui Zhang, ChonLam Lao, Chengyuan Huang, Baojia Li, Ziyuan Zhu, Hao Lu, Weizhen Dang, Zitong Lin, et al. Astral: A datacenter infrastructure for large language model training at scale. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pages 609–625, 2025.

[55] Jesmin Jahan Tithi, Hanjiang Wu, Avishaii Abuhatzera, and Fabrizio Petrini. Scaling intelligence: Designing data centers for next-gen language models. *arXiv preprint arXiv:2506.15006*, 2025.

[56] Grant Wilkins, Srinivasan Keshav, and Richard Mortier. Hybrid heterogeneous clusters can lower the energy consumption of llm inference workloads. In *Proceedings of the*

15th ACM International Conference on Future and Sustainable Energy Systems, pages 506–513, 2024.

[57] Grant Wilkins, Srinivasan Keshav, and Richard Mortier. Offline energy-optimal llm serving: Workload-based energy models for llm inference on heterogeneous systems. *ACM SIGENERGY Energy Informatics Review*, 4(5):113–119, 2024.

[58] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Petros Vavaroutsos, Sotirios Xydis, and Dimitrios Soudris. throttll'em: Predictive gpu throttling for energy efficient llm inference serving. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1363–1378. IEEE, 2025.

[59] Woohyung Choi, Jinwoo Jeong, Hanhwi Jang, and Jeongseob Ahn. Gpu-centric memory tiering for llm serving with nvidia grace hopper superchip. *IEEE Computer Architecture Letters*, 2025.

[60] Robin Geens, Man Shi, Arne Symons, Chao Fang, and Marian Verhelst. Energy cost modelling for optimizing large language model inference on hardware accelerators. In *2024 IEEE 37th International System-on-Chip Conference (SOCC)*, pages 1–6. IEEE, 2024.

[61] Jiwoo Kim, Joonhyung Lee, Gunho Park, Byeongwook Kim, Se Jung Kwon, Dongsoo Lee, and Youngjoo Lee. An inquiry into datacenter tco for llm inference with fp8. *arXiv preprint arXiv:2502.01070*, 2025.

[62] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[63] Jeremy Elson, John (JD) Douceur, Jon Howell, and Jared Saul. Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization. In *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc., October 2007. URL https://www.microsoft.com/en-us/research/publication/asirra-a-captcha-that-exploits-interest-aligned-manual-image-categorization/.

[64] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[65] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[67] Nvidia. Real time means real change nvidia quadro rtx 4000, . URL https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/quadro-rtx-4000-data-sheet-us-nvidia-830682-r6-web.pdf.

[68] Nvidia. Compute capability 7.x, . URL https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capability-7-x.

[69] Nvidia. Contents of the tensorflow container, . URL https://docs.nvidia.com/deeplearning/frameworks/tensorflow-release-notes/rel_22-03.html.

[70] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[71] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[72] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

[73] Mistral AI Team. Mistral nemo: Our new best small model. https://mistral.ai/news/mistral-nemo, July 2024. URL https://mistral.ai/news/mistral-nemo. Accessed March 2025.

[74] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, et al. Gemma 2: Improving open language models at a practical size, 2024. URL https://arxiv.org/abs/2408.00118.

[75] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[76] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[77] U.S. Energy Information Administration (EIA). Electric power monthly. https://www.eia.gov/electricity/monthly/epm_table_grapher.php?t=epmt_5_6_a, 2025.

[78] Supermicro. Comparison of air-cooled versus liquid-cooled nvidia gpu systems: Liquid-cooled gpu servers reduce power consumption and increase performance. White paper, October 2025. URL https://www.supermicro.com/white_paper/white_paper_SMCI_Air_VS_Liquid-Cooled_NVIDIA_GPU_Systems.pdf.

Statement on the Use of Generative Artificial Intelligence

Generative artificial intelligence tools, including ChatGPT and Claude, were utilized during preparation of this dissertation. AI Tools were used to assist with language editing, grammar correction, Latex formatting and consistency checks. All AI-assisted content was reviewed and verified by the author. The research contributions, data processing, empirical methods, interpretation of results, and conclusions were the responsibility of the author.