



**UNIVERSITÀ
DEGLI STUDI
DI TRIESTE**

UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXVII CICLO DEL DOTTORATO DI RICERCA IN

Applied Data Science and Artificial Intelligence

**Training-Inference Resource Trade-offs for
Sustainable AI Deployment**

Settore scientifico-disciplinare: INF/01

**DOTTORANDO
Tomislav Šubić**

**COORDINATORE
PROF. Francesco Pauli**



Firmato digitalmente da Francesco
Pauli
Data: 29.10.2025 09:53:46 CET
Organizzazione: UNIVERSITA' DEGLI
STUDI DI TRIESTE/80013890324

**SUPERVISORE DI TESI
PROF. Stefano Cozzini**

Stefano
Cozzini
28.10.2025
15:27:02
GMT+01:00

ANNO ACCADEMICO 2024/2025

Abstract

AI and especially large language models, face critical deployment challenges as inference consumes substantially more energy than training over their operational lifecycle. This thesis investigates efficient inference methodologies through three complementary approaches. First, we explore neural network scaling laws and training-inference compute trade-offs, exploring how Chinchilla scaling laws can be adapted to account for inference costs. Second, we analyze computational patterns in transformer architectures to understand optimization targets. Third, we conduct empirical studies comparing quantized and full-precision models across multiple scales, demonstrating that certain quantization methods and techniques maintain performance while reducing computational requirements. We extend those studies by examining test-time scaling behaviors of model, seeing if smaller models can perform more efficiently than larger models if given extra compute at inference time. Our comprehensive analysis of weight and activation quantization establishes quantization as a fundamental technique for efficient deployment. This research provides both theoretical frameworks and practical methodologies for optimizing inference operations, enabling sophisticated language model deployment in resource-constrained environments while maintaining high performance.

Acknowledgments

I want to thank my family for their support and patience, their understanding during the long evenings and weekends spent on research made this possible.

I am grateful to Arctur for sponsoring my PhD studies and providing me with the opportunity to pursue this research. The company's commitment to innovation allowed me to explore practical applications of my work in real-world settings. My research on efficient LLM inference has directly informed numerous business decisions at Arctur, helping shape our AI strategy and service offerings.

This journey has been incredibly rewarding, not only for the knowledge gained but also for how it has prepared me to lead our growing AI efforts at Arctur. The intersection of research and practice has proven invaluable in building solutions that matter.

Contents

1	Introduction	2
1.1	The Economic and Strategic Case for Efficient Inference	2
1.2	Research Hypotheses	3
1.3	Scope	3
2	AI sustainability	5
2.1	The AI Energy and Environmental Landscape	5
2.2	The Cloud Computing Continuum	7
3	Scaling Laws and Compute Trade-offs	8
3.1	Training scaling	8
3.1.1	Inference-Aware Scaling	11
3.2	Inference scaling: Test-Time Compute	13
3.2.1	Parallel Scaling Law	14
3.2.2	Sequential Scaling: Budget Forcing	15
3.3	Computational Case Study:Qwen3 Model Family	15
4	AI Computing: Hardware, Parallelism, and Inference Bottlenecks	19
4.1	Resource Challenges for AI infrastructure	19
4.2	AI inference parallelization techniques	21
4.2.1	Data Parallelism	21
4.2.2	Model Parallelism	22
4.2.3	Model size	24
4.3	Computational Patterns in Transformer Inference	25
4.3.1	Transformers and Positional Encoding	25
4.3.2	Memory Hierarchy Optimization	33
4.4	Implications of Activation Distributions	34
4.4.1	Activation power law	35
4.4.2	Quantization Error Propagation	37
4.5	Resource Utilization	38
4.6	Latency vs throughput	39
5	Benchmarking and Evaluation Methodology	44
5.1	Performance metrics for LLMS	44
5.2	Benchmarks	45

6	Efficient Inference	48
6.1	Pruning	49
6.2	Quantization	50
6.2.1	Quantization-Aware Training	52
6.2.2	Numerical Formats in LLMs	53
6.2.3	Post-Training Quantization Methods	53
6.2.4	Quantization at various scales	59
6.2.5	In-Context Learning and Quantization	60
6.3	Prefill-Decode Disaggregation	61
7	Energy Efficiency of Quantized Models	64
7.1	Introduction	64
7.2	Experimental Design	65
7.2.1	GEMM Kernel impact on energy	66
7.3	Quantization Impact on Model Performance	67
7.3.1	Energy Efficiency Characteristics	69
7.3.2	Throughput analysis	71
7.4	Input/Output Length Optimization	72
7.5	Practical Implications	73
8	Inference Scaling	75
8.1	Introduction	75
8.2	Experimental Design	76
8.3	Results: Budget Scaling Analysis	78
8.4	Adaptive thinking flow	84
8.5	Hypothesis Evaluation	87
9	Efficient Production Inference	89
9.1	Deployment framework	91
10	Conclusion	94
	References	97

List of Figures

2.1	Model development phases over the AI system hardware life cycle. (a) At Facebook: power capacity breakdown of 10 : 20 : 70 for Experimentation, Training, and Inference. (b) End-to-end ML pipeline energy footprint of RM1: 31 : 29 : 40 for Data, Experimentation/Training, and Inference. (c) Despite carbon-free energy investments, data center electricity use reached over 7.17 million MWh in 2020. Adapted from Wu et al., “Sustainable AI: Environmental Implications, Challenges and Opportunities”.	6
4.1	Pipeline execution with gradient accumulation across micro-batches: forward passes on multiple micro-batches followed by backward phase for the entire group. Adapted from PipeDream [28]	23
4.2	Pipeline execution using PipeDream strategy with multiple concurrent forward passes from different micro-batches. Backward passes are prioritized over forward passes on each worker. Adapted from PipeDream [28]	24
4.3	The encoder-decoder structure of the Transformer architecture Adapted from [31]	26
4.4	Activation distribution measured on the MMLU dataset on Qwen3 models - without outliers.	36
4.5	Throughput at various batch sizes for Qwen3-4B	41
4.6	Latency metrics at various batch sizes for Qwen3-4B	42
4.7	Request latency at various batch sizes for Qwen3-4B	42
4.8	Latency vs throughput at various batch sizes for Qwen3-4B	43
6.1	Qwen3 quantization at various scales using GPTQ, AWQ, RTN	60
7.1	Model performance by size for 6 benchmarks - we evaluate zero-shot accuracy on ARC, MMLU, GSM8K, HumanEval, WinoGrande and TruthfulQA	68
7.2	Energy per output token for across model sizes quantized with various quantization methods	69

7.3	Energy per output token analysis for various model sizes using popular quantization methods.	70
7.4	Tokens per second throughput across model sizes using different quantization methods for context length 4000 tokens and output length 1000 tokens	71
7.5	Energy per output token for Qwen3-14B at various output lengths and with various quantization methods	72
7.6	Energy per output token analysis for increasing output tokens(left) and input tokens(right)	73
8.1	Qwen3 8B model performance on the GSM8K benchmark under various thinking budgets.	77
8.2	Qwen3 8B FP16 model zero-shot accuracy on the GSM8K benchmark sets a baseline for our evaluation. We see that the 14B model shows worse results than smaller models.	78
8.3	Qwen3 8B model performance on the GSM8K benchmark under various thinking budgets, without forced budget utilization.	79
8.4	Reasoning budget scaling analysis - results for various model sizes using increasing reasoning budgets on the GSM8K benchmark.	80
8.5	Energy-accuracy tradeoff at scaled model sizes.	81
9.1	Simplified diagram of Arctur's AI Infrastructure	90

List of Tables

3.1	Qwen3 Training Requirements at Practical H100 SXM Performance with Variable MFU	17
7.1	Quantization resiliency across Qwen3 model sizes for Metabench benchmark . . .	67
8.1	Accuracy gains and energy efficiency across model sizes for FP16 and INT8 from 500 to 2000 tokens reasoning budget	83
8.2	Fixed vs adaptive budget for Qwen3-4B on the GSM8K dataset	86

Chapter 1

Introduction

1.1 The Economic and Strategic Case for Efficient Inference

Arctur is a Slovenian research oriented company, focusing on high-performance computing and AI. Over the past decade, alongside development and services, Arctur relied on providing compute to customers - CPU and GPU clusters. The emergence of large language models has opened a new business dimension: organizations no longer want compute by the hour; they want AI capabilities and access to intelligence. This shift requires infrastructure providers to operate as an AI inference service provider maintaining always-on model endpoints, managing latency SLAs, and optimizing cost per token. Efficient inference is the main operational challenge that motivated this research. **Inference consumes substantially more energy and emits more carbon** over a model's lifecycle than training. Recent research has shown that up to 70% of ML infrastructure power is devoted to inference, with training accounting for only 30% [1]. For a model serving millions of requests and thousands of users, each individual inference operation is cheap, but the cumulative inference cost is significantly larger than the one-time training cost. For Arctur, managing this cost while providing competitive service quality requires a principled understanding of the efficiency trade-off space.

Artificial Intelligence has achieved remarkable milestones: defeating human champions in games such as Go through AlphaGo, generating stunning artwork with diffusion models, and powering versatile language systems like GPT that tackle a wide range of tasks. All these breakthroughs share a common reliance on massive computational resources [2]. In today's "post-Moore" era, this is pronounced even more through specialized hardware that doubles available compute at an unprecedented rate. The demand for compute is causing issues in the supply chain, raising concerns that a small group of companies will monopolize AI. Equally pressing are the environmental impacts of large-scale AI; data center energy usage and carbon emissions from hardware manufacturing represent significant sources of greenhouse gases [3]. Democratizing AI or **making it possible for everyone to create artificial intelligence systems**,

requires designing smaller, more efficient models and inference frameworks [4]. To achieve that, it is crucial to understand the trade-offs and relations between compute, model features, and inference techniques. The widespread deployment of large language models (LLMs) and their application to various use-cases has created an urgent need for efficient inference methodologies as these complex systems increasingly dominate the computational infrastructure.

This thesis addresses that challenge by investigating three levers that together define the efficiency frontier: 1. Model selection and scaling laws: which model size minimizes cost, given expected inference volume and required accuracy? 2. Quantization: how much precision can be removed without unacceptable accuracy loss? 3. Inference compute scaling — can compute-at-inference substitute for model size, and at what energy cost?

1.2 Research Hypotheses

The thesis is organized around two explicitly testable hypotheses.

Hypothesis 1 (H1): For equivalent FLOP budgets, a quantized model combined with test-time scaling achieves superior accuracy per unit compute compared to a larger unquantized model performing single-pass inference. This hypothesis rests on the intuition that the compute freed by quantization can be profitably reinvested into additional reasoning tokens rather than wasted.

Hypothesis 2 (H2): Smaller quantized models demonstrate greater relative performance improvement from test-time scaling than larger models. This follows from the observation that smaller models have lower baseline accuracy and thus a larger gap that extended reasoning can fill. If correct, it means that quantization and test-time scaling are complementary rather than independent optimizations where each amplifies the return on the other.

Both hypotheses are evaluated in Chapter 7 (quantization experiments) and Chapter 8 (inference scaling experiments), and jointly synthesised in the conclusion.

1.3 Scope

The thesis investigates inference exclusively, no training runs are performed. The Qwen model family was chosen because it provides identical architecture scaled across various parameter counts, enabling clean isolation of scaling effects. All experiments run on NVIDIA L40S GPUs with native FP8 and INT8 support the same hardware deployed in Arctur’s production inference cluster.

Four concrete questions structure the research: For compute- and energy-constrained deployment, is it better to use a large quantized model or a smaller unquantized model? How

much accuracy can be recovered by extending compute at inference time? What is the energy cost of test-time scaling relative to its accuracy benefit? Can adaptive compute allocation match fixed-budget peak accuracy at lower average cost?

The thesis is organized into ten chapters. Chapters 2 and 3 establish theoretical and contextual foundations. Chapter 3 reviews neural network scaling laws and discusses their implications for inference. Chapter 4 analyses the computational patterns within transformers to understand how they contribute to the inference cost. Chapter 5 discusses benchmarking and evaluation methods for LLMs. Chapter 6 covers efficient inference techniques. Chapters 7 and 8 present the two major studies. Chapters 9 and 10 draw impact and conclusions and outline future work.

The empirical component of this thesis presents comprehensive studies comparing model behavior across different scales, analyzing both quantized and original models through evaluation of various quantization methodologies. We demonstrate that quantized models can achieve comparable performance to full-precision models while reducing computational requirements. We conduct an in-depth analysis of quantization effects on both weights and activations, establishing quantization as a fundamental technique for achieving efficient AI model deployment. We extend these studies to account for **inference scaling**, using novel techniques to add compute at inference time and improve accuracy of smaller models. We explore how on-demand compute provisioning influences reasoning performance, utilization efficiency, and the cost-accuracy trade-off.

This research contributes to the growing understanding of efficient neural network deployment by providing both theoretical frameworks and practical methodologies for optimizing inference operations. The work demonstrates that careful application of quantization techniques, informed by scaling law principles and energy-aware optimization strategies, can enable the deployment of sophisticated language models in resource-constrained environments while maintaining the high performance required for practical applications.

Chapter 2

AI sustainability

2.1 The AI Energy and Environmental Landscape

The exponential growth of large language models and other large AI models has brought new capability, but it is coupled with significant environmental and energy cost. **Data centers and digital infrastructure account for up to 2.18% of global greenhouse gas emissions**, with US data centers producing 105 million tons of CO₂ equivalent annually - a number that is comparable to commercial air travel [5]. According to the International Energy Agency's Global Energy Review 2025, data centers alone were responsible for approximately 1.0% of energy-related CO₂ emissions in 2024, with total digital infrastructure, including data transmission networks, contributing up to 1.5% of global energy-related emissions [6]. While the academic community primarily focuses on the energy costs of training AI models, **inference - the deployment phase where models are used for actual tasks - accounts for up to 90% of the total energy consumption across a model's lifecycle** [7].

This asymmetry has important strategic and practical implications. A model that is trained once but queried a billion times will have its environmental and economic profile dominated entirely by the inference cost. To complicate matters further, the Jevons Paradox suggests that efficiency improvements tend to stimulate demand rather than reduce absolute consumption. A 2× reduction in inference cost may lead to 4× more queries, leaving total energy consumption unchanged or higher. This dynamic makes it insufficient to optimize inference in isolation; the entire deployment strategy model selection, quantization, and compute allocation - must be considered jointly.

The AI industry is beginning to acknowledge this reality, however, research on the energy efficiency of LLMs remains sparse, with only a handful of published studies examining real hardware energy draw during inference. Most sustainability discussions focus on training, which is the wrong phase to optimize for production systems.

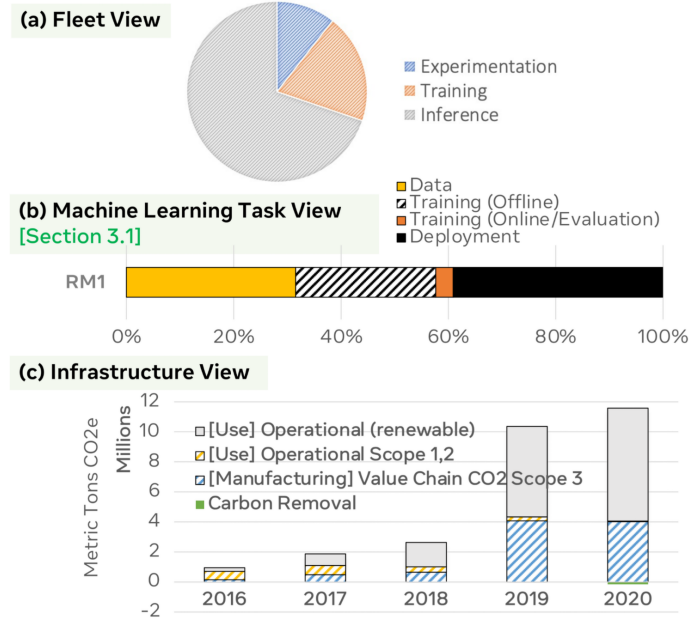


Figure 2.1: Model development phases over the AI system hardware life cycle. (a) At Facebook: power capacity breakdown of 10 : 20 : 70 for Experimentation, Training, and Inference. (b) End-to-end ML pipeline energy footprint of RM1: 31 : 29 : 40 for Data, Experimentation/Training, and Inference. (c) Despite carbon-free energy investments, data center electricity use reached over 7.17 million MWh in 2020. Adapted from Wu et al., “Sustainable AI: Environmental Implications, Challenges and Opportunities”.

A model goes through a lifecycle as does any other product. It consists of two distinct phases: training (data collection, pre-training, fine-tuning) and inference, where the model is run in production. Training a Recommendation model at Facebook leaves a carbon footprint of 45 US home’s annual energy use. During the model lifetime, experimentation and training account for only 30% of the models energy usage, while 70% represents inference and deployment costs [1]. Training and inference of a model are responsible for the **operational** carbon footprint, which is easily estimated. On the other hand we also consider the **embodied** carbon footprint, which represents the carbon emissions that came from manufacturing, transportation, material excavations and similar. The research confirmed that inference costs exceed training costs for deployed systems due to the multiplicative factor of repeated usage. Analysis of real-world deployment scenarios showed that inference energy consumption varies dramatically based on usage patterns, hardware efficiency, and optimization strategies.

Despite the critical environmental impact of large-scale inference, published research on the energy efficiency of LLMs remains limited, with only a handful of studies. From a model deployment perspective, research by [8] on energy consumption of various models across different hardware platforms shows variability in inference energy depending on prompt characteristics. However, they focused on serving models with the inference engines and frameworks that would not be suitable for production deployments.

2.2 The Cloud Computing Continuum

The Cloud Computing Continuum (CCC) describes a heterogeneous computational ecosystem spanning IoT sensors, edge devices, fog nodes, and centralized cloud infrastructure. The European Union has invested over €300 million within the Horizon Europe programme targeting this paradigm, recognizing that AI deployment requires models that can run across the full hardware spectrum. This heterogeneity creates a fundamental mismatch with modern LLM requirements. Edge devices operating in IoT deployments typically offer megabytes to a few gigabytes of memory, while frontier LLMs require tens to hundreds of gigabytes in full precision. Quantization can bridge this gap: a model requiring 4 GB at FP16 can operate within 200-400 MB after 4-bit compression, enabling deployment on hardware that would otherwise be entirely incapable of supporting it.

One of the most significant contributions of model compression to the Cloud Computing Continuum lies in enabling seamless model and service deployment across heterogeneous computational environments. Traditional AI deployment approaches require separate model variants optimized for specific hardware platforms, creating complex maintenance overhead and limiting deployment flexibility. Compressed models, particularly those using standardized quantization schemes, provide a unified representation that maintains computational compatibility across diverse hardware architectures.

Developing new European cloud systems in the era of data spaces and AI through comprehensive distributed architecture that spans from quantum computers to IoT sensors is one of the goals of these investments. The ability to deploy compressed models across this diverse infrastructure eliminates deployment complexity while ensuring consistent performance standards regardless of the specific hardware configuration [9].

Effective AI service deployment across heterogeneous platforms represents a critical challenge for digital autonomy. Current AI models require substantial computational resources that exceed the capabilities of edge devices, creating deployment barriers that limit the reach of intelligent applications across the computing continuum. This distributed computing vision encompasses everything from resource-constrained IoT sensors to high-performance quantum computers, creating what the European Commission terms the “cognitive computing continuum”. The challenge lies in making AI services operate seamlessly across this vast spectrum of computational capabilities without compromising functionality or efficiency [10].

Chapter 3

Scaling Laws and Compute Trade-offs

The training of large language models requires substantial computational resources, making it crucial to understand the relationship between model size, training data, and energy consumption. The current state of the art AI research is starting to challenge the idea that the more compute, data and parameters we have, that we are going to have a better and more intelligent model. During the years many have tried and some have successfully defined **empirical scaling laws for model training**. In this section, we focus on the optimal ratio between model parameters, training tokens and accuracy.

3.1 Training scaling

The Chinchilla scaling laws, introduced by [11], fundamentally changed the understanding on how to allocate computational resources in large language model training. Prior work, such as the scaling laws proposed by [12], suggested that increasing model size was the most effective approach to improve performance. However, the Chinchilla research demonstrated that for a fixed compute budget, optimal performance is achieved by balancing both model size and training data, rather than focusing solely on parameter count. In other words, **compute-optimal models require balanced scaling between model size and training data**.

By increasing training dataset size, model parameter count and computational resources, researchers could expect predictable improvements in model intelligence and accuracy. Through an extensive empirical study involving over 400 language models ranging from 70 million to 16 billion parameters trained on 5 to 500 billion tokens, the authors reveal that current large language models are significantly undertrained due to the field's focus on scaling model parameters while keeping training data constant at approximately 300 billion tokens. Their key finding contradicts earlier work by [12], which suggested that model size should grow faster than dataset size for optimal compute allocation.

Instead, the Chinchilla scaling laws propose that **for every doubling of model size,**

the number of training tokens should also be doubled, indicating that computational resources should be evenly distributed between parameters (N) and data (D). This principle was validated through the development of Chinchilla, a 70-billion parameter model trained on 1.4 trillion tokens that, despite using the same compute budget as the 280-billion parameter Gopher, achieved superior performance across downstream tasks including a state-of-the-art 67.5% accuracy on the MMLU benchmark. The implications of these scaling laws extend beyond training efficiency, as **compute-optimal models also substantially reduce inference costs and memory requirements**, making them more practical for deployment while maintaining superior performance.

The key insight from this research is that current large language models are significantly undertrained due to the focus on scaling model parameters while keeping training data constant at approximately 300 billion tokens. The optimal ratio established by Chinchilla research indicates that approximately 20 tokens should be used per model parameter for compute-optimal training.

The central formula proposed by Hoffmann et al. is:

$$L = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}$$

Where:

- L = cross-entropy loss (nats per token)
- E = irreducible loss representing the entropy of natural language (~ 1.69)
- N = number of model parameters
- A, B = scaling coefficients determined empirically (A 406.4, B 410.7)
- α, β = **scaling exponents fitted to the loss surface** (α 0.34, β 0.28)
- D = number of training tokens

The term $\frac{A}{N^\alpha}$ captures how model performance improves with increased parameters. The exponent $\alpha \approx 0.34$ indicates that doubling the model size provides a $2^{0.34} \approx 1.27x$ improvement in this loss component. Similarly, $\frac{B}{D^\beta}$ represents performance gains from additional training data. With $\beta \approx 0.28$, doubling the training tokens yields a $2^{0.28} \approx 1.21x$ improvement.

Separately, Hoffmann et al. also report exponents for how the *compute-optimal allocation itself* scales with total compute budget C constraint:

$$C = C_0 N D$$

Where:

- C = total compute budget (in FLOPs)
- C_0 = constant representing compute per parameter per token

- N = model parameters
- D = training tokens

To find the compute-optimal allocation, we minimize the loss function subject to a fixed compute budget, fitting $N_{opt} C^a$ and $D_{opt} C^b$ gives a 0.46, b 0.54 (they report a b 0.50). Because a and b are close to 0.5, doubling the compute budget increases both the optimal model size and optimal dataset size by roughly the same factor.

For optimal performance:

- $N_{opt} \propto C^{1/2}$
- $D_{opt} \propto C^{1/2}$

This means that **if the compute budget is doubled, both the effective model size and the amount of training data should also increase**, following this square-root scaling relationship between each factor and total compute.

The scaling law implies an optimal ratio of approximately **20 tokens per parameter** for compute-optimal training. As already mentioned, this contrasts the earlier Kaplan scaling laws, which suggested models should be much larger relative to their training data.

Prior to Chinchilla scaling laws, the common approach involved training larger models on fixed datasets. For comparison, a traditional approach might have involved training a 4 billion parameter model on 5 billion tokens, requiring:

- Compute: $C = 6 \times (4 \times 10^9) \times (5 \times 10^9) = 1.2 \times 10^{20} FLOPs$
- Same computational budget but suboptimal allocation

The Chinchilla approach achieves superior performance with the same compute budget by balancing model size and data more effectively, this compute-optimal approach provides several advantages:

- Improved model performance per unit of compute
- Reduced inference costs due to smaller model size
- Enhanced sample efficiency during training
- Lower memory requirements for deployment

The findings from Chinchilla scaling laws have significant implications for research planning. Organizations must allocate resources not only for model parameters but equally for data collection, curation, and processing. The 20:1 token-to-parameter ratio requires substantial investment in high-quality training data.

The computational requirements, while substantial, remain within the reach of academic institutions and smaller organizations for small models (up to 30B parameters). The ability to train such models on small clusters democratizes access to large language model research.

3.1.1 Inference-Aware Scaling

If we take an even deeper look at Chinchilla scaling laws, we find that it could reveal a fundamental trade-off between compute resources allocated to training versus inference. This offers significant flexibility in how computational budgets are distributed across the model lifecycle. This trade-off mechanism provides practitioners with strategic options for optimizing total compute costs based on deployment scenarios and usage patterns.

The scaling laws demonstrate that effectively, **compute can be reallocated between training and inference phases** while maintaining equivalent model performance. The laws can be interpreted that by varying the scaling policy alone, it's possible to achieve approximately 0.7 orders of magnitude (OOM) reduction in inference compute by spending 1.2 OOM more during training. This asymmetric exchange rate reflects the mathematical relationship between model parameters (N) and training data (D), where training compute scales as $TC = 6ND$ and inference compute scales as $IC = 2N$. In simpler terms, to achieve that we need to train a smaller parameter model for longer (on more data), so that during inference, a smaller model would spend less compute.

Beyond basic parameter-data scaling, multiple techniques enable more substantial trade-offs:

- **Overtraining approaches can save 1 OOM of inference compute** by investing 2 OOM additional training compute
- Resampling techniques allow 1.5 OOM increased inference compute to save 1 OOM in training compute
- Monte Carlo Tree Search (MCTS) methods enable bidirectional trade-offs of up to 1.6 OOM in either direction
- Specialized verification tasks can leverage up to 6 OOM additional inference compute to reduce training by 3 OOM

The scaling laws also suggest that models should not necessarily be trained to their lowest possible loss to remain compute-optimal. The trade-off relationship operates also in the opposite direction, enabling optimization depending on deployment requirements. When inference costs dominate (high-volume deployment scenarios such as widely-deployed consumer applications), we can invest additional training compute to reduce per-inference costs - for example by training a smaller model for longer. The optimal training strategy depends on anticipated inference

patterns. **When training budgets are constrained, additional inference compute can also compensate for smaller, undertrained models.**

As a general principle, each technique enables approximately 1 OOM of compute savings in either training or inference by increasing the other factor by somewhat more than 1 OOM. This relationship holds across various domains and model architectures, though the exact exchange rates vary by technique and application.

The most direct implementation of this trade-off involves strategically modifying the ratio between model parameters and training data. Rather than following the standard Chinchilla-optimal 20:1 token-to-parameter ratio, we can:

- Increase training data while reducing parameters to minimize inference costs for high-volume deployments
- Increase parameters while reducing training data when training compute is limited but inference efficiency is less critical
- Extend training duration beyond compute-optimal points to achieve better inference efficiency [13]

Research demonstrates that for models expecting high inference demand, significant compute reductions are achievable: a 30B-parameter Chinchilla-quality model expecting 10^{13} inference tokens can reduce total FLOPs by 28% by training a 13.6B parameter model on $2.84\times$ more data [13]. This approach maintains equivalent performance while dramatically reducing inference costs.

This trade-off framework fundamentally challenges the traditional approach of maximizing training efficiency in isolation, instead advocating for holistic optimization across the entire model lifecycle. The ability to allocate compute between training and inference phases provides researchers with powerful tools for optimizing total cost of ownership based on specific deployment scenarios and usage patterns.

Ultimately, an obvious flaw in the Chinchilla scaling laws is that they **do not account for inference compute**, which can be substantial for deployed models. This oversight leads to suboptimal resource allocation decisions when developing models for production deployment. The research by [13] addresses that issue and challenges the widely-accepted Chinchilla scaling laws by incorporating inference costs in language model optimization. This research demonstrates that real-world model deployment requires accounting for both training and inference expenses to achieve optimal costs distribution, a consideration that becomes increasingly important as language models are deployed at scale with billions of daily inference requests. Through extensive empirical validation involving 47 models of varying sizes and parameter counts, the

authors modify the Chinchilla scaling laws to minimize total FLOPs across the entire model lifecycle using the formula:

$$\operatorname{argmin} 6ND_{tr} + 2ND_{inf},$$

where training requires $6N$ FLOPs per token and inference requires $2N$ FLOPs per token. This reformulation explicitly accounts for the computational burden of serving models throughout their operational lifetime, addressing a significant gap in the original Chinchilla framework that assumed inference costs were negligible or constant. The research reveals that model quality continues to improve as token-to-parameter ratios scale to extreme ranges (up to 10,000 tokens per parameter), far beyond traditional Chinchilla-optimal ratios of approximately 20:1.

Their analysis reveals that for models expecting reasonably large inference demand ($\sim 1B$ requests), the optimal strategy is to train smaller models and for longer than Chinchilla-optimal configurations. Specific quantitative examples demonstrate that a 30B-Chinchilla-quality model expecting 10^{13} inference tokens can reduce total FLOPs by 28% by training a 13.6B parameter model on $2.84\times$ more data, while maintaining equivalent performance. Similarly, a Chinchilla-7B-quality model with 10^{11} inference tokens should use a 6B parameter model trained on $1.18\times$ more data to achieve optimal costs. These findings are particularly significant when considering real-world deployment scenarios where inference hardware typically operates at $50\times$ lower model FLOP utilization (MFU) compared to training infrastructure.

The findings suggest that inference providers should train models considerably longer - or use models that are smaller but trained longer, fundamentally shifting the paradigm from training-centric to deployment-centric optimization strategies. Another consideration here is that data acquisition and **dataset curation also incurs additional costs**, which the authors did not take into account.

The inference-aware scaling laws discussion provides the theoretical foundation for the empirical model selection experiments in Chapter 9.

3.2 Inference scaling: Test-Time Compute

Test-time scaling, also called long thinking or inference scaling, represents a paradigm shift in how computational resources are allocated during inference. Unlike traditional approaches that scale parameters or training data, test-time scaling leverages additional compute during inference to improve output quality without modifying learned weights. This approach has gained significant prominence following the success of OpenAI’s o1 model, which demonstrated that models can achieve substantial performance improvements on complex reasoning tasks by “thinking longer” during inference rather than simply scaling model size [14].

Empirical studies have established that test-time scaling follows predictable power-law relationships similar to traditional pre-training scaling laws. Optimally scaling test-time compute can be more effective than scaling model parameters; smaller models equipped with advanced inference algorithms can outperform models 14 times larger in FLOPs-matched evaluations. The fundamental insight underlying test-time scaling is captured by the principle that performance consistently improves with additional inference computation, exhibiting scaling behaviors that complement rather than replace pre-training scaling laws.

The mathematical foundation of test-time scaling can be understood through the lens of sampling and search algorithms. **Inference scaling laws exist, that demonstrate how cost-performance trade-offs emerge between model sizes and additional token generation during inference** [15]. Their analysis reveals that smaller models combined with sophisticated inference strategies often yield Pareto-optimal configurations, where a Llemma-7B model equipped with advanced tree search algorithms consistently outperforms Llemma-34B models across various inference strategies on mathematical benchmarks.

3.2.1 Parallel Scaling Law

Parallel scaling is a new approach that does not increase memory and latency, but increases computational load. It doesn't increase parameter count, but increases parallel computational load by generating multiple independent solution candidates and selects among them via voting or reward models. It was inspired by CFG (Classifier-Free Guidance) - a technique essential for diffusion models - which increases computation but eliminates the need for an external classifier by leveraging the diffusion model itself. A parallel stream in the context of the Parallel Scaling Law [16] refers to an independent computational pathway that processes a differently transformed version of the same input through the language model. The model processes the same input in multiple parallel lanes, each modified by a unique, learnable transformation. All streams use the same underlying model parameters (no model duplication or parameter increase). The outputs from all streams are dynamically aggregated-typically using a weighted sum, where the weights are computed by a small neural network based on the outputs themselves.

This approach allows the model to simulate the effect of a much larger model by increasing computational diversity, rather than parameter count, leading to significant improvements in efficiency and performance. The learnable prefixes are vectors or tokens (similar to those used in prefix-tuning) that are prepended to the input sequence. Each parallel stream has its own set of prefixes, which are trained to encourage diverse perspectives or representations of the input

These learnable transformations only add a small number of extra parameters (about 0.2%

per stream), so the approach is far more efficient than increasing the model size itself. This way we are scaling compute at inference time, where each stream acts as an expert in a MoE model, but using the same parameters in a sequential way.

3.2.2 Sequential Scaling: Budget Forcing

Budget forcing constrains models to generate extended reasoning chains before producing final answers, effectively trading inference latency for solution quality. **Iterative refinement** applies sequential improvement processes, where initial solutions undergo multiple rounds of enhancement guided by process reward models [17]. This technique requires no modifications to the underlying model architecture, weights, or training procedures. Implementation involves only prompt engineering and a custom logits processor that suppresses the end-of-thinking token until a minimum token threshold is reached and forces termination at a maximum budget, making it possible to apply to any existing language model. This facilitates rapid prototyping, systematic experimentation, and easy reproduction of results across different model families and sizes - crucial in today's fast pacing AI environment.

Budget forcing also operates within a single model instance, eliminating the need for additional GPU resources or complex distributed systems. This characteristic makes it particularly suitable for production environments with constrained computational resources. Unlike ensemble methods that may require $3-8\times$ the computational resources, budget forcing achieves scaling within the same hardware footprint as standard inference.

3.3 Computational Case Study:Qwen3 Model Family

The total computational cost for training transformer-based language models follows the standard formula:

$$C = 6PD$$

where:

- C represents the total compute in FLOPs
- P denotes the number of model parameters
- D indicates the number of training tokens

This formula accounts for both forward and backward passes during training, where the backward pass requires approximately twice the computation of the forward pass. The factor of 6 emerges from the mathematical operations required in transformer architectures, including matrix multiplications and attention computations. This chapter examines the computational

requirements for training a 1 billion parameter language model following the Chinchilla scaling laws.

Following those scaling laws, a 1 billion parameter model requires 20 billion training tokens for compute-optimal training. Training a 1 billion parameter language model following Chinchilla scaling laws requires 1.2×10^{20} FLOPs of computation and approximately 0.10 MWh of energy.

Using the compute formula $C = 6PD$:

- $C = 6 \times (1 \times 10^9) \times (20 \times 10^9)$
- $C = 1.2 \times 10^{20}$ FLOPs

The computational requirements, while substantial, remain accessible for academic research and smaller organizations.

Hardware Requirements

The NVIDIA H100 (SXM variant) delivers approximately 1,000 TFLOPs (1×10^{15} FLOPs per second) in BF16/FP16 for dense tensor core operations, which is typical for AI training computations (without assuming sparsity acceleration, as training is generally dense). This is the theoretical peak performance. However, real word - practical throughput measured through Model FLOPS Utilization (MFU) varies significantly across organizations:

- **Meta (LLaMA-3):** 38-41% MFU with 16,000 H100 GPUs
- **DeepSeek-v3:** 20-30% MFU (estimated)
- **Google PaLM:** ~37% MFU reported
- **Academic research:** Typically 10-25% MFU

The Model FLOPs Utilization (MFU) of ~35% reflects real-world efficiency constraints including communication overhead, memory bandwidth limitations, and other system inefficiencies typical in distributed training environments.

MFU decreases with scale due to communication overhead [18]:

- **8 GPUs:** Up to 63% MFU possible
- **64 GPUs:** ~40-50% MFU typical
- **1000+ GPUs:** Often below 30% MFU

Training a 1 billion parameter model requires approximately 22 GB of GPU memory, accounting for:

- Model weights: 4 bytes per parameter (FP32 or BF16 with mixed precision)
- Gradients: 4 bytes per parameter
- Optimizer states: 8 bytes per parameter (Adam optimizer)
- Activations and temporary memory: 8 bytes per parameter

This memory requirement fits comfortably within a single H100’s 80 GB capacity, eliminating the need for model parallelism across multiple GPUs for memory constraints.

Power Requirements

Each H100 GPU is rated at 700W, which it can reach during training operations. The energy consumption remains constant regardless of the number of GPUs used, as more GPUs complete training faster but consume proportionally more power during the shorter duration. When accounting for system overhead including cooling, networking, and supporting infrastructure, the effective power consumption increases by a 30% overhead factor.

From an environmental perspective, the carbon footprint depends on the energy source. Assuming a typical grid with approximately 0.25 kg CO₂ per kWh, a training run for a 1B parameter model would generate approximately 25 kg of CO₂ emissions, assuming it uses 100kWh.

Hardware Specifications:

- H100 SXM Theoretical Peak: **1,000 TFLOPS**
- Practical Performance (35% MFU): **350 TFLOPS**
- H100 SXM Power Consumption: **700W**

Table 3.1: Qwen3 Training Requirements at Practical H100 SXM Performance with Variable MFU

	Training	Training	1xSingle H100	8× H100	64× H100	Energy
Model	Tokens	FLOPs	(75% MFU)	(63% MFU)	(35% MFU)	(MWh)
0.6B	12.0B	4.32e+19	0.67 days	0.10 days	0.02 days	0.01
1.7B	34.0B	3.47e+20	5.35 days	0.80 days	0.18 days	0.12
4.0B	80.0B	1.92e+21	29.63 days	4.41 days	0.99 days	0.65
8.0B	160.0B	7.68e+21	118.52 days	17.64 days	3.97 days	2.59
14.0B	280.0B	2.35e+22	362.96 days	54.01 days	12.15 days	7.93
32.0B	640.0B	1.23e+23	1896.30 days	282.19 days	63.49 days	41.42

The results demonstrate **quadratic scaling** in computational requirements, where doubling model parameters increases training FLOPs by approximately 4×. This scaling pattern is

consistent with Chinchilla’s optimal allocation of compute between model size and training data.

Energy consumption ranges from **11.2 kWh for the 0.6B model to 41.42 MWh for the 32B model**. At typical data center electricity rates (€0.20/kWh), energy costs alone range from just ~€3 to over €8,000.

Ongoing research in training efficiency, including gradient compression, mixed precision training, and improved optimization algorithms, may further reduce computational requirements while maintaining model quality. The Chinchilla scaling laws assume uniform data quality. Recent research suggests that high-quality, curated datasets may require fewer tokens than the standard 20:1 ratio, potentially improving efficiency further.

Understanding these computational requirements is essential for planning research projects, estimating costs, and developing sustainable approaches to language model training. The Chinchilla scaling laws provide a framework for making informed decisions about resource allocation in the rapidly evolving field of large language model research.

Chapter 4

AI Computing: Hardware, Parallelism, and Inference Bottlenecks

4.1 Resource Challenges for AI infrastructure

The compute requirements required to train and serve AI models are poorly understood outside the technical AI community, leading to systematic underinvestment and misallocation. **Computational demands from AI have increased faster than hardware performance improvements**, creating a gap for organizations without access to cutting-edge infrastructure [19] [20]. NVIDIA's near-monopoly position in AI accelerators means that supply constraints and pricing dynamics directly affect the economics of inference providers like Arctur. As many others, Arctur runs a heterogeneous infrastructure consisting of various NVIDIA and non-NVIDIA AI Accelerators - devices with varying memory size and bandwidth, native FP8 and INT8 support, power rating and interconnects. This makes multi-GPU and high availability (HA) inference serving a constant quest for the most efficient setup.

By partitioning data and model parameters, and parallelizing the workload we can use more resources - multiple GPUs and multiple computing nodes. Determining the best way to assign data, parameters, and computations to the available processors and memory is a complex optimization problem. While designing training and inference algorithms, we need to keep a couple things in mind:

- Evenly distribute data across memory
- Balance computational load across all processing units
- Minimize communication overhead between memory and compute units
- Linear or better scaling of execution time with the number of processors, so that increasing computational resources proportionally reduces processing time.

AI can run on a wide variety of hardware, from regular CPUs which have proven to be capable of small local deployments, to specialized highly parallel vector processors with hardware

and instructions sets designed especially for AI and ML like IPUs (Intelligent Processing Units) or TPUs (Tensor Processing Units). The majority of researchers and industry use Graphical Processing Units (GPUs), which are commercially available (unlike TPUs) and strike a good balance between performance and cost. When designing these hardware solutions, the trade-off is between core and transistor count, transistor size and power consumption with physical properties like heat. Fundamentally, to boost a chip's performance, we need to add more transistors. However, to maintain efficiency without increasing the chip's physical size, these transistors must be made smaller. **As transistor sizes shrink and are packed closer together, power consumption and heat generation rise significantly.** These increases happen exponentially as the size of the transistors decreases linearly. With today's technology, we are **approaching the physical limits** of how small transistors can be made and how densely they can be arranged. Therefore for a similar performance gain, we focus on parallelization through multiple processor cores on lower clock frequency, which results in a lower power consumption. At a fixed power consumption, we can extract more compute power from many low frequency cores than from a few high frequency cores.

This interplay and these trade offs are very interesting also while optimizing compute requirements both for training and inferencing neural networks. When we scale a model, be it in training or inference, we aim to increase the number of computational units that are efficiently used - ideally with a linear increase in throughput. This is called *strong scaling*.

A major challenge in scaling LLMs comes from the computational and architectural limits of dense transformer models. When OpenAI moved from GPT-3 to GPT-4, their goal was to increase the model size by about 10x. The main obstacle was not just the model design, but the increasing cost of training. Dense transformer architectures, which are used in models like GPT-4, Meta LLaMA, TII Falcon, and MosaicML MPT, are widely adopted across the industry. While these models are powerful, their structure makes it difficult to scale efficiently.

Training these state-of-the-art LLMs often costs millions of dollars. For large tech companies, these costs are seen as capital investments, similar to other big technology projects. In general, spending more on compute resources leads to better model performance. However, the real limitation is often how quickly enough compute power can be made available for model development and testing, not just the financial cost, since the supply chains are stressed under high demand.

Despite the focus on training costs, the bigger challenge for scaling LLMs is the cost and complexity of inference - running the models to generate outputs, especially for practitioners and integrators. As models get larger, the resources needed to serve them in real time increase even

more. In some cases, inference costs can be several times higher than training costs, especially when accounting for model lifetime. This has led to growing interest in sparse model architectures, where only a portion of the model’s parameters are active during inference, reducing the need for compute and memory.

4.2 AI inference parallelization techniques

In the pursuit of efficient AI inference, parallelization has emerged as the fundamental strategy for overcoming computational bottlenecks. While individual matrix multiplications remain the atomic units of computation, the path to truly scalable inference lies in orchestrating these operations across multiple computing units. The challenge extends beyond simple parallel execution to encompass memory management, communication overhead, and dynamic workload distribution - each requiring sophisticated coordination mechanisms to maintain both correctness and performance.

The complexity of modern transformer architectures, particularly those serving millions of context tokens, has pushed traditional parallelization approaches to their limits [21]. Contemporary inference systems must navigate a multi-dimensional optimization space where memory bandwidth, computational intensity, and communication patterns interact in complex ways. The most effective solutions combine multiple parallelization strategies in what has become known as “3D parallelism” [22], where data, model, and pipeline parallelism work in concert to extract maximum performance from distributed hardware.

4.2.1 Data Parallelism

Data parallelism represents the most intuitive approach to distributed inference: replicate the complete model across multiple devices and partition the input batch accordingly. Each device processes its assigned subset of inputs independently, requiring synchronization only during gradient accumulation steps in training scenarios. For inference, this translates to processing multiple independent requests simultaneously, with each device maintaining its own copy of model parameters and KV cache [23].

Formally, given a batch of inputs $B = \{x_1, x_2, \dots, x_N\}$ and P parallel devices, each device i is assigned a subset $B_i \subset B$ such that $|B_i| \approx N/P$. Under perfect load balancing and identical hardware, the idealized total processing time satisfies

$$T_{\text{total}} = \max(T_1, T_2, \dots, T_P) \approx \frac{T_{\text{sequential}}}{P}$$

where T_i denotes the time taken by device i on its assigned subset and $T_{\text{sequential}}$ is the time to process the full batch on a single device.

However, data parallelism faces fundamental scalability limits. Memory requirements scale linearly with the number of devices since each must store the complete model. For a 70B parameter model requiring approximately 140 GB in half-precision, even modest parallelism degrees quickly exhaust available memory [24]. Moreover, the approach provides no benefit for single-request scenarios where latency (not throughput) is the primary concern.

4.2.2 Model Parallelism

When models exceed the memory capacity of individual devices, model parallelism becomes essential. It is the umbrella term for splitting one model across multiple devices, with each responsible for computing a subset of the network’s layers or operations [25].

The challenge lies in maintaining computational balance. Given L layers distributed across P devices, naive partitioning assigns $\frac{L}{P}$ layers to each device. However, memory consumption and computational intensity often vary significantly across layers, particularly between attention and feed-forward components. Advanced partitioning algorithms consider these asymmetries, using profiling data to optimize load distribution [26].

Communication overhead represents the primary bottleneck in model parallelism. Each device must transfer activation tensors to the next stage, creating a dependency chain that limits parallelism. For transformer models with activation dimensionality d_{model} and sequence length T , the communication volume per layer is $O(T \cdot d_{\text{model}})$. This scales poorly with sequence length, making model parallelism particularly challenging for long-context applications [21].

Tensor Parallelism

Tensor parallelism operates at the granularity of individual matrix operations, partitioning large matrix multiplications across multiple devices. This approach is particularly effective for transformer architectures, where attention mechanisms involve large matrix products that can be decomposed along specific dimensions. The attention mechanism in LLMs is an ideal target: query, key, and value projections can be partitioned column-wise across devices, with each device computing a subset of attention heads.

Formally, for an input matrix $\mathbf{X}$ and P devices, the projections can be written as

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q^{(1:P)}, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K^{(1:P)}, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V^{(1:P)}$$

where $\mathbf{W}_Q^{(i)}$, $\mathbf{W}_K^{(i)}$, and $\mathbf{W}_V^{(i)}$ denote the slices of the weight matrices assigned to device i , typically

corresponding to a subset of attention heads. Each device computes attention for its assigned heads independently, and communication is required only to aggregate the per-device outputs into a single combined representation.

Feed-forward networks follow a similar pattern, with the first linear layer partitioned column-wise and the second row-wise. For an input vector or matrix x ,

$$\text{FFN}(x) = \sigma(x\mathbf{W}_1^{(1:P)}) \mathbf{W}_2^{(1:P)}$$

where $\mathbf{W}_1^{(i)}$ and $\mathbf{W}_2^{(i)}$ are the corresponding partitions on device i . This arrangement minimizes communication by keeping intermediate activations local to each device until the final reduction or concatenation step.

However, as with other methods, tensor parallelism incurs significant communication overhead. Each partitioned operation requires an all-reduce communication to reconstruct the complete result, with volume proportional to the activation size. Recent work explores techniques to reduce this overhead, such as Sync-Point Drop (SPD), which selectively eliminates synchronization points where accuracy impact is minimal [27]. Tensor parallelism has emerged as the preferred method to use in production environments, and is included in major inference engines as the recommended way to split a model across multiple GPUs.

Pipeline Parallelism

Pipeline parallelism addresses memory limitations by distributing model layers across devices and processing multiple micro-batches simultaneously. Rather than waiting for each layer to complete before starting the next, the system maintains multiple requests in flight, with different micro-batches at different pipeline stages.

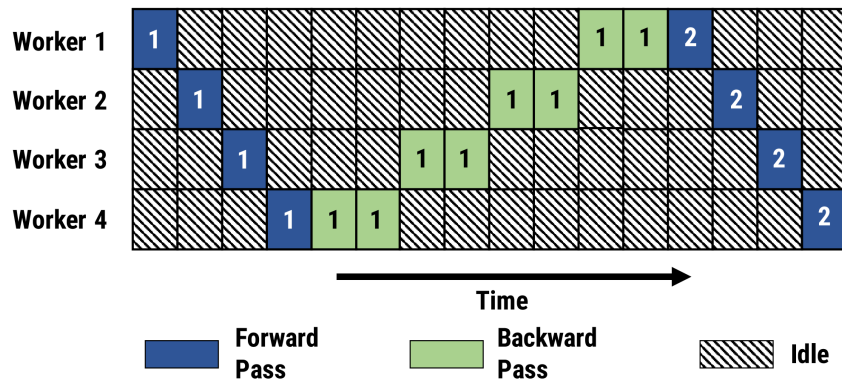


Figure 4.1: Pipeline execution with gradient accumulation across micro-batches: forward passes on multiple micro-batches followed by backward phase for the entire group. Adapted from PipeDream [28]

The mathematical foundation involves partitioning the model into S stages across P devices,

where each stage processes a micro-batch before passing results to the next stage. The total processing time becomes:

$$T_{\text{pipeline}} = T_{\text{startup}} + MB \cdot T_{\text{stage}} + T_{\text{drain}} \quad (3)$$

where B is the total batch size, M is the micro-batch size, and T_{stage} is the processing time for each stage. The startup and drain phases represent periods where not all devices are fully utilized, creating “pipeline bubbles” that reduce efficiency.

Modern pipeline parallelism systems employ sophisticated scheduling algorithms to minimize these bubbles. This led to the development of advanced scheduling strategies such as GPipe’s gradient accumulation across microbatches [29], PipeDream’s asynchronous execution with weight versioning and 1F1B (One Forward One Backward) scheduling that alternates forward and backward passes to minimize memory usage [28].

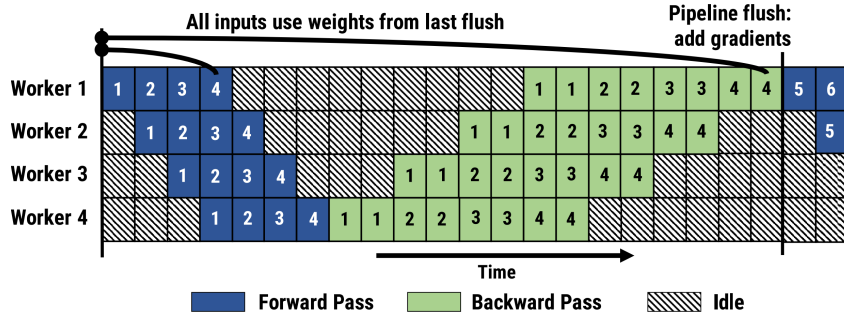


Figure 4.2: Pipeline execution using PipeDream strategy with multiple concurrent forward passes from different micro-batches. Backward passes are prioritized over forward passes on each worker. Adapted from PipeDream [28]

PP is less efficient than TP for latency (due to pipeline bubbles) but is the only viable option for multi-node inference over slower interconnects. For frontier models requiring more than eight GPUs, the standard configuration is TP within each node and PP between nodes (e.g., TP8PP2).

Expert Parallelism

Expert Parallelism (EP) is specific to Mixture-of-Experts (MoE) architectures. Each expert is assigned to a single GPU, with the expert router replicated across all GPUs. EP requires less synchronization than TP and scales well to multi-node deployments with limited interconnect bandwidth.

4.2.3 Model size

With all neural network models, the model size fundamentally refers to the number of parameters contained, measured in billions of trainable weights and activations that encode the

model’s learned knowledge. These parameters function as the “knobs and switches” that get fine-tuned during training, where each parameter represents a numerical value that determines how the model processes input tokens and generates subsequent outputs. Models like Gemma 2B, Mistral 7B, or GPT-4 (rumored to contain around 1.76 trillion parameters), the numerical designation directly corresponds to the parameter count rather than file size or computational requirements.

The relationship between parameter count and actual memory consumption is easy to predict and calculate - each parameter typically requires 2-4 bytes of storage depending on the precision format used. For instance, a 1 billion parameter model consumes approximately 3.73 GB of memory when stored in 32-bit precision (FP32) or 1.86 GB in 16-bit precision (FP16), creating a direct linear relationship between parameter count and hardware requirements. This becomes critical for deployment considerations, as larger models demand exponentially more computational resources, memory bandwidth, and specialized hardware to operate effectively.

Recent advances in model compression and efficiency techniques have challenged traditional scaling assumptions, with quantization methods enabling dramatic memory reductions by converting 32-bit parameters to 8-bit, 4-bit, or even lower precision formats while maintaining acceptable performance levels. Parameter-efficient fine-tuning (PEFT) approaches further optimize deployment by adapting only small parameter subsets (often less than 1% of total parameters) for specific tasks, achieving up to 90-95% reduction in training time and computational requirements [30].

From a practical deployment perspective, the choice of model size involves careful consideration of the performance-efficiency trade-off, where larger models generally provide superior output quality and contextual understanding but require substantial infrastructure investments and operational costs. Edge deployment scenarios often favor smaller, quantized models that can operate on consumer hardware, while cloud-based applications may justify the computational expense of larger models for maximum capability. The ongoing evolution of neural scaling laws and compression techniques continues to reshape these trade-offs, enabling more efficient utilization of large language models across diverse deployment environments and use cases.

4.3 Computational Patterns in Transformer Inference

4.3.1 Transformers and Positional Encoding

Transformers were introduced in the paper “Attention is all you need”, which took a new approach with the self-attention mechanism [31]. It enabled transformer models to process sequential data more effectively - models can evaluate and weigh relationships between all elements

in the input sequence simultaneously. This parallelism accelerates training and inference and allows transformers to capture long-range dependencies and intricate contextual relationships within data much better than previous architectures.

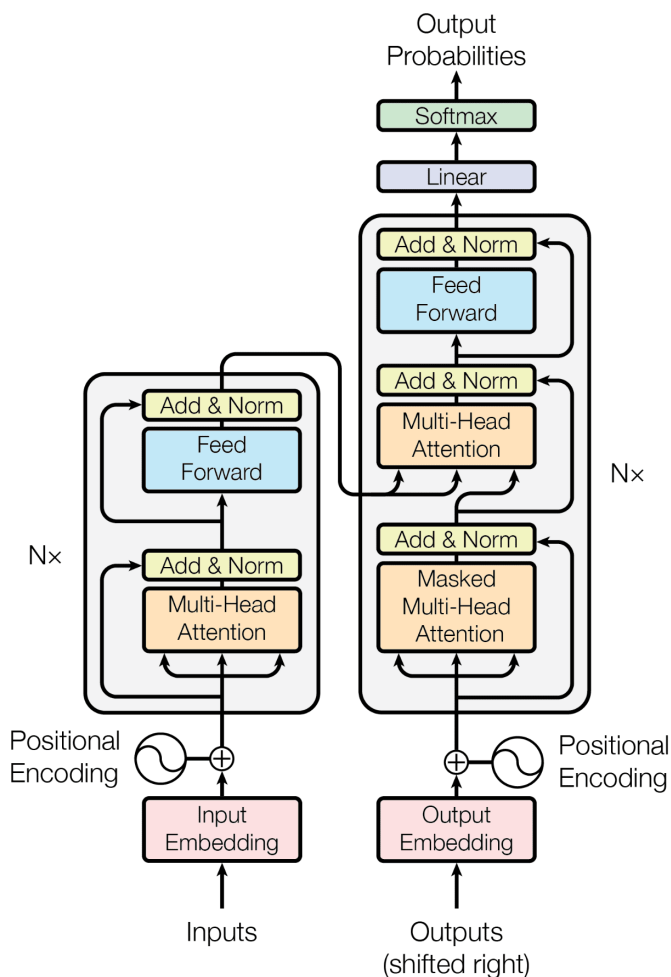


Figure 4.3: The encoder-decoder structure of the Transformer architecture Adapted from [31]

The transformer architecture is composed of stacks of encoders and decoders, each built from layers that include multi-head self-attention mechanisms, feed-forward neural networks, residual connections, and normalization layers. The encoder processes input data into a rich, contextual representation, while the decoder generates output sequences, making the architecture highly versatile for tasks such as machine translation, text generation and summarization. Crucially, transformers incorporate positional encodings to retain information about the order of sequence elements, compensating for the lack of inherent sequential processing.

Transformers have rapidly become the foundation of state-of-the-art models across natural language processing, but also in models focused on computer vision and speech recognition. The dominance of transformers in both research and real-world deployments is due to their

scalability, efficiency, and ability to model complex, context-dependent relationships in data. For these reasons, this work will focus on transformers, as they are the backbone of all current state-of-the-art models and agentic AI systems.

The evolution of positional encoding techniques demonstrates a progression from naive approaches to sophisticated methods that preserve both semantic and positional information without interference. Sinusoidal positional encoding, introduced in the original Transformer architecture [31], employs alternating sine and cosine functions with geometrically decreasing frequencies to create unique position representations that maintain linear relationships between positions. However, this additive approach “pollutes” semantic token embeddings with positional information, leading to the development of Rotary Positional Embedding (RoPE), which multiplicatively applies rotations to query and key vectors during attention computation rather than modifying input embeddings. RoPE leverages the geometric properties of dot products by encoding relative positional relationships through rotational transformations, enabling the model to capture positional dependencies without altering the semantic content encoded in vector norms. This multiplicative approach has proven superior in practice, being adopted by state-of-the-art models including LLaMA, and demonstrates better extrapolation capabilities for sequences longer than those encountered during training.

Transformer Inference

When a large language model is asked to generate text, every new token is produced by a strictly forward pass through the network; no gradients are formed, no weights are changed, and nothing is back-propagated. What makes this simple-sounding process possible is a set of linear algebra operations that have been engineered to execute in milliseconds. We present the whole process, from the moment characters enter the model to the moment the probability of the next token is returned, emphasizing what is really multiplied, why the famous “scaled dot-product” appears, and how clever caching avoids repeating work.

The first stage is **tokenization**. Incoming text is split into discrete symbols called tokens. Each token is represented as an integer index that selects a row from an embedding matrix $E \in \mathbb{R}^{V \times d_{\text{model}}}$.

A single embedding lookup can be written mathematically as multiplying a one-hot vector $x \in \mathbb{R}^V$ by E :

$$e = x^\top E$$

Because only one row is selected, this is implemented in practice as a gather rather than a

full matrix multiplication, but conceptually it is equivalent to a $(1 \times V)$ by $(V \times d_{\text{model}})$ matrix product.

A positional representation is then incorporated so that token order is preserved. In some models this is done by adding sinusoidal or learned positional embeddings, while in many modern transformers positional information is encoded with Rotary Position Embedding (RoPE), which applies a position-dependent rotation instead of simple addition.

The resulting embedding sequence $\mathbf{X} \in \mathbb{R}^{T \times d_{\text{model}}}$, where T is the current context length, enters the first transformer block. Each block then contains two computationally heavy sublayers: self-attention and a feed-forward network (FFN).

Inside **self-attention**, the model projects $\mathbf{X}$ in three different ways:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V$$

with $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d_{\text{model}} \times d_k}$.

Each projection is a matrix multiplication, costing roughly $2T d_{\text{model}} d_k$ floating-point operations (FLOPs).

The central energy-hungry step is then the scaled dot-product attention:

$$\mathbf{A} = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \right)$$

followed by another matrix multiplication, $\mathbf{A}\mathbf{V}$, to produce the attended representations.

The division by $\sqrt{d_k}$ keeps the variance of the dot products under control so that the softmax does not saturate. Without this scaling, the variance grows with d_k , which can push attention scores toward extreme values and lead to unstable gradients during training. Counting only the dominant matrix multiplications, the attention mechanism per head includes the products $\mathbf{Q}\mathbf{K}^\top$ and $\mathbf{A}\mathbf{V}$, each with cost proportional to the sequence length and head dimension. In autoregressive decoding with KV caching enabled, attention often becomes memory-bandwidth-bound because each new token must repeatedly read the accumulated key and value cache, while arithmetic work per step remains relatively small.

On modern GPUs such as the NVIDIA H100, these operations map efficiently onto tensor-core GEMM kernels, and optimized attention implementations can sustain very high throughput; however, performance during decoding is often limited more by memory traffic than by raw arithmetic capacity.

Self-attention is followed by the **feed-forward network**:

$$\text{FFN}(x) = \sigma(x\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{model}} \times 4d_{\text{model}}}$ and $\mathbf{W}_2 \in \mathbb{R}^{4d_{\text{model}} \times d_{\text{model}}}$.

In general terms, autoregressive transformers generate text through a process called *sampling*: given a sequence of tokens, the model produces a probability distribution over possible next tokens, from which a token is selected according to a chosen decoding strategy (e.g., greedy, top-k, or nucleus sampling). This process is then repeated iteratively, conditioning on the newly generated token each time.

Naively, this would require recomputing the entire sequence at every step, which is computationally inefficient. In practice, attention-based transformers avoid this by caching intermediate states, resulting in two distinct phases during inference: the prefill phase and the decode phase.

During the *prefill* phase, the model processes the entire input sequence in parallel through full forward passes of the transformer layers, including QKV projections, attention, and feed-forward networks. Because these operations have high arithmetic intensity, prefill is typically *compute-bound*, meaning performance is primarily limited by GPU FLOPS rather than memory bandwidth. This phase is highly parallelizable across tokens, leading to high GPU utilization and throughput, although latency can still grow with longer input sequences due to the quadratic cost of attention. During prefill, the KV cache is populated for all input positions, enabling incremental decoding in the subsequent phase.

After prefill, traditionally the model would generate tokens **autoregressively** in the *decode* phase- one token at a time. A critical optimization for autoregressive generation is the **KV cache**, which enables avoiding recomputation of attention over the entire prefix at every decoding step. This cuts the per-token complexity and makes autoregression fast enough for real-time use. Decode is *memory-bandwidth-bound* at low to moderate batch sizes, with compute sitting idle as weights are streamed from GPU VRAM.

Each newly generated token requires:

- **Sequential processing:** The model must wait for previous token before generating the next.
- **KV cache extension:** New key-value pairs are appended to the existing cache for every generated token.
- **Memory-bound operations:** Primarily reading from the large KV cache and model weights rather than raw FLOPs.
- **Lower GPU utilization:** Sequential nature limits parallelization
- **Longer duration:** Latency scales roughly linearly with the desired output length - with

throughput being significantly lower than prefill.

The KV cache stores keys and values for each token computed during prefill and updated during decode. Its size grows linearly with sequence length and batch size.

During interactive decoding only the new query vector for the last token must be computed; keys and values of all earlier tokens are identical across steps, so they are stored once and reused. Instead of recomputing the full $T \times T$ attention matrix every step, the model now multiplies a $1 \times d_k$ query by a cached $T \times d_k$ key matrix, then multiplies the resulting $1 \times T$ attention row by the cached $T \times d_v$ values. Memory footprint grows linearly with context length (two bytes per element gives about 0.5 GB for a 4k-token context in bfloat16), but the compute saved is enormous: after the prompt, attention complexity per additional token drops from $O(T^2)$ to $O(T)$. Well-tuned implementations overlap PCIe transfers of cached blocks with kernel launches or even partially recompute on GPU if bandwidth becomes the bottleneck.

The whole forward pass is essentially a deep pipeline of matrix multiplications with lightweight layer-normalization, bias addition and activation functions. GPUs exploit this regularity by fusing operations and keeping intermediate tiles in on-chip SRAM. For decades the hardware industry has pursued faster “multiply-accumulate” units; tensor cores now complete 8-bit dot products in one cycle, so lowering numerical precision is an obvious lever.

This asymmetry between prefill and decode has profound implications for inference optimization. Quantization benefits both phases differently: for prefill, lower-precision Tensor Cores double available FLOPS; for decode, lower-precision weights halve the memory bandwidth required per token. In practice, quantization from FP16 to FP8 or INT8 delivers 30–50% better performance for LLMs, not the theoretical $2\times$ maximum, due to dequantization overhead and other system factors.

Sparse and **activation-sparse** variants activate only a subset of FFN neurons or limit which tokens attend to which, reducing FLOPs by up to a factor of 3 at the same parameter count. Hardware-software co-designs approximate softmax and layer-norm in log space to shave latency on ASICs, while in-memory analog dot-product engines integrate storage and compute.

At the end of the stack a final linear layer projects the last hidden state back to vocabulary size V (another GEMM of shape $d_{\text{model}} \times V$), producing logits. Softmax turns logits into a probability distribution and a sampling rule picks the next token. That token’s index is appended to the context, its embedding is retrieved, and the whole cycle repeats. Because KV cache eliminates quadratic growth but not the linear dependence on sequence length, the wall-clock time per token gradually increases; batching multiple independent sequences helps fill GPU cores, whereas speculative decoding and streaming methods pipeline multiple passes

to hide this latency.

To clarify the compute budget: for a 32-layer decoder with $H = 32$ heads, $d_k = 128$, context $T = 2048$, computing attention for one query position against the full T -length context costs approximately $L \cdot H \cdot (2 \cdot d_k \cdot T)$ $32 \cdot 32 \cdot (2 \cdot 128 \cdot 2048)$ $0.54GFLOPs$. Critically, this per-row cost is the same whether that query is the last token of the prompt during prefill or a newly generated token during cached decoding, in both cases the query attends across the same T cached keys and values.

KV caching eliminates the need to repeat this computation for every earlier position at every generation step. Without caching, producing the token at position $T + n$ would require recomputing attention for all $T + n$ positions from scratch which is a cost that grows as $O((T + n)^2)$ with the length of the generated sequence. With caching, each new token requires only this one $O(T)$ row, so cumulative attention cost over a generation of n tokens grows as $O(n \cdot T)$ rather than $O(n^2)$, linearly rather than quadratically.

The FFN remains constant at about $32 \cdot 2 \cdot 4096 \cdot 1024 \approx 270$ MFLOPs per token, regardless of caching, since it has no sequence-length dependence. Consequently, once the prompt is consumed, the FFN dominates compute while attention dominates memory. Designers exploit this by using higher precision in FFN to preserve accuracy and more aggressive quantisation in attention to save bandwidth.

From a software perspective the whole inference engine boils down to four high-performance kernels: GEMM (for every matrix multiplication), bias-activation fusion (GELU or SiLU), layer-norm, and softmax. Frameworks like FasterTransformer and vLLM bind them together with custom CUDA graphs to avoid overheads. The result is that a forward pass through a 70B parameter model can stream tens of tokens per second on a single H100 card while using less than half the card’s power budget, thanks to the alignment of mathematical structure, numerical tricks, and hardware pathways painstakingly described above.

In summary, inference in modern LLMs is a sequence of matrix multiplications whose shapes stay fixed once the model is configured. By projecting inputs into queries, keys and values, scaling the dot product, caching as much as possible, and running optimized fused kernels, practitioners turn terabytes of floating-point arithmetic into conversational response times. Understanding these steps demystifies the “magic” of large language generation and provides the foundation for every future optimization.

KV Cache Optimizations

The **Key-Value (KV) cache** represents a critical component in transformer inference memory hierarchy optimization. Recent advances have focused on multiple dimensions of KV cache efficiency, including dynamic compression, prefetching strategies, and cache-aware scheduling. KeyFormer introduces a **dynamic KV cache compression** framework that intelligently discards unnecessary tokens during inference. The technique achieves $2.1\times$ reduction in inference latency and $2.4\times$ improvement in token generation throughput while maintaining model accuracy within 99-99.9% of baseline performance. The methodology employs multi-dimensional token importance metrics at head-level granularity, enabling fine-grained memory allocation decisions [32].

Researchers propose new methods proposing L2 cache-oriented **asynchronous prefetching** specifically designed for transformer KV cache management. The approach strategically schedules idle memory bandwidth resources to proactively prefetch required KV cache data into GPU L2 cache during active compute cycles. Experimental results demonstrate significant improvements in memory bandwidth utilization by enabling L2 cache hits for high-speed KV cache loading, effectively hiding HBM access latency [33]

Advanced caching strategies incorporate token-level importance scoring mechanisms that consider **both semantic relevance and cache state**. LOOK-M achieves minimal performance degradation with fixed KV cache budgets by implementing look-once optimization principles, reducing decoding latency from 28.55 ms/token to 18.22 ms/token with 5% cache retention [34].

The key-value cache presents unique challenges for parallelization due to its dynamic nature and substantial memory requirements [21]. Unlike static model parameters, KV cache grows linearly with sequence length and varies across different requests, making traditional parallelization strategies inadequate.

KV Cache Parallelism (KVP) addresses these challenges by distributing cache management across multiple devices. Rather than storing the complete cache on a single device, the system partitions cache entries based on sequence position or attention head, with each device responsible for a subset of the cache.

During attention computation, devices must aggregate contributions from all partitions, requiring communication proportional to the cache size. LayerKV extends this concept by introducing layer-wise cache management [35]. Rather than treating the cache as a monolithic structure, the system manages cache entries independently for each transformer layer. This enables fine-grained memory control and supports dynamic cache offloading based on memory

pressure and service level objectives.

Sequence Parallelism

Sequence parallelism represents a specialized form of parallelism targeting the sequence dimension of transformer inputs. Rather than processing the entire sequence on a single device, the approach partitions sequences across devices, with each responsible for computing attention and feed-forward operations on its assigned tokens.

The key insight is that attention computation can be decomposed along the sequence dimension without sacrificing accuracy. For a sequence of length T distributed across P devices, each device processes T/P tokens. However, attention mechanisms require access to the complete key and value matrices, necessitating careful coordination between devices.

Ring-Attention addresses this challenge through a circular communication pattern. Devices arranged in a ring topology iteratively exchange key and value matrices, allowing each to compute attention scores for all sequence positions. The total communication volume remains $O(T \cdot d_{model})$, but the computation is distributed across multiple devices.

4.3.2 Memory Hierarchy Optimization

FlashAttention is an IO-aware exact attention algorithm that tiles Q/K/V into blocks fitting in on-chip SRAM, reducing the number of HBM reads/writes and making attention memory usage linear rather than quadratic in sequence length, achieving significant speedups across various transformer architectures [36]. Recent extensions include FlashAttention-2 and FlashAttention-3, which further optimize memory access patterns for specific hardware architectures.

Other optimization research, like sparse attention mechanisms addresses both computational complexity and memory bandwidth limitations. Fixed-size memory representations and efficient attention lookup mechanisms reduce the memory bandwidth requirements while maintaining model performance. However, these approaches often require architectural modifications that limit their applicability to existing pretrained models.

Neural network-based approaches for learning and predicting memory access patterns show promise for prefetching optimization. Hashemi et al. demonstrate that LSTM-based prefetchers consistently outperform traditional hardware prefetchers in terms of precision and recall across challenging benchmark datasets. This approach treats memory access prediction as a sequence modeling problem, leveraging temporal dependencies in access patterns [37].

Modern AI accelerators increasingly rely on HBM technologies to address memory bandwidth limitations. However, even with HBM integration, memory bandwidth remains the pri-

mary bottleneck for large model inference. This finding highlights the need for algorithmic innovations that reduce absolute memory bandwidth requirements rather than solely relying on hardware improvements.

4.4 Implications of Activation Distributions

Neural network activations represent the intermediate computational states that emerge as data flows through layers of a deep neural network. These activation values, produced by applying nonlinear transformation functions to the weighted sum of inputs at each neuron, capture the learned internal representations that enable neural networks to solve complex tasks. Understanding and analyzing these activations provides critical insights into model behavior, performance characteristics, and optimization opportunities. Neurons or layers with high activation spikes are considered *hot*, while other, low activation neurons and layers are considered to be *cold*.

Activation patterns reveal how effectively a network captures relevant features from input data. High-quality representations exhibit clear separation between different classes or concepts, while poor representations show overlapping or poorly structured activation patterns. By examining activation distributions and clustering properties, researchers can assess whether a model has learned meaningful representations [38].

Performance issues often manifest differently across network layers. Early layers may struggle with feature extraction, while later layers might fail at classification or regression tasks. Activation analysis enables layer-specific diagnosis, allowing practitioners to identify which parts of the network contribute most to performance bottlenecks. Activation sparsity patterns reveal how efficiently a network uses its representational capacity. Models with extremely sparse activations may be underutilizing their parameters, while models with dense activations across all layers might benefit from regularization. This can inform architecture design decisions and capacity allocation.

Different layers exhibit varying sensitivity to quantization. Some layers maintain stable activation patterns under quantization, while others show significant degradation. Understanding these layer-specific sensitivities enables **adaptive quantization strategies** where more sensitive layers receive higher precision allocation. This selective approach optimizes the trade-off between model size and accuracy.

Effective quantization also **requires understanding the range and distribution of activation values**. Layers with wide dynamic ranges or skewed distributions may require different quantization schemes than those with narrow, symmetric ranges. Activation analysis

reveals these distribution characteristics, informing the choice of quantization parameters such as scale factors and zero points.

Modern quantization methods rely on calibration datasets to determine optimal quantization parameters. Activation analysis guides this calibration process by identifying which input samples produce representative activation patterns and which layers require careful parameter tuning. This ensures that quantized models maintain performance on real-world data distributions.

However, the primary challenge in neural network quantization is minimizing accuracy loss while maximizing compression. Activation analysis provides direct measurement of how quantization affects internal representations. By comparing activation patterns between full-precision and quantized models, we can quantify the representational degradation introduced by reduced precision. Quantization errors can compound as they propagate through network layers. Activation analysis enables tracking how small errors in early layers amplify into larger deviations in later layers, providing insights into error propagation mechanisms and guiding mitigation strategies.

4.4.1 Activation power law

Neural activations in deep transformer models do not remain Gaussian in later layers; instead, their magnitudes tend to follow a **heavy-tailed power-law distribution**. Concretely, if we let A denote the absolute value of a hidden-state activation, then for large x :

$$\Pr(A > x) \propto x^{-\alpha} \quad \text{for some exponent } \alpha > 0$$

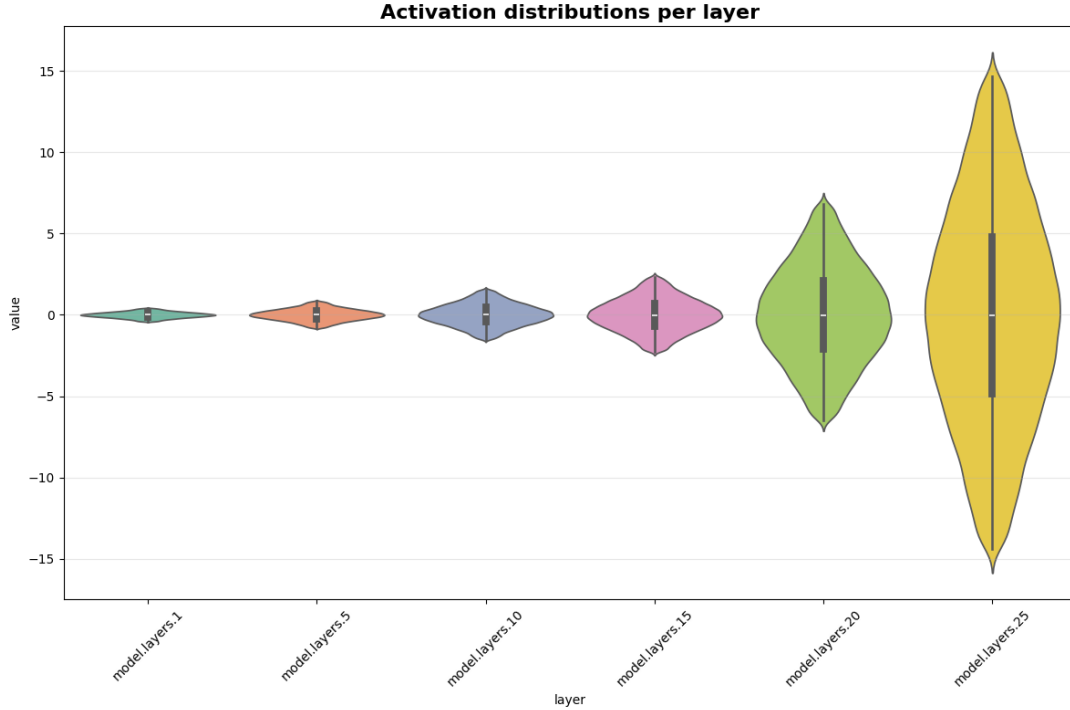


Figure 4.4: Activation distribution measured on the MMLU dataset on Qwen3 models - without outliers.

Current research suggests that some of the most important weights are found in earlier layers - identified by high activation spikes [39]. This may lead us to the conclusion that earlier layers are more important, but that is counter intuitive. As we can observe on Figure 4.4, heavy tails appear in later layers. Unlike Gaussian or exponential decay, power-law tails imply a non-negligible probability of very large activations, corresponding to rare but strong feature responses in later layers [40]

There is no single “typical” activation scale in the tail - doubling x reduces the tail probability by a constant factor $2 - \alpha$ regardless of x . Empirical studies show α decreases modestly with depth (e.g. from 3 in mid-layers to 2 in deeper layers), reflecting progressively heavier tails as representations specialize [41]

This implies that Power-law tails increase quantization error, since extreme activations far from zero incur large rounding errors under fixed-range quantization.

Power-law activation behavior arises from two interacting factors:

1. Successive layer transformations and residual additions effectively multiply or add heavy-tailed inputs, propagating and reinforcing tail heaviness.
2. Softmax-weighted token mixing can amplify certain token features disproportionately, creating rare large spikes in hidden-state values.

Recognizing power-law tails guides quantization and regularization:

- We should use quantization schemes that allocate more representational range to capture tail values rather than symmetric clipping around zero.
- Sample calibration data that includes rare high-activation events to set scales and clipping thresholds appropriately should be used.
- During training or fine-tuning, we can apply light clipping to extreme activations to temper the heaviest tails without disrupting overall representation quality.

Transformer activations in deeper layers exhibit a **power-law distribution**, with tail exponent α decreasing with depth. This scale-free heavy-tailed behavior critically impacts quantization design, necessitating tail-aware strategies to maintain model fidelity.

4.4.2 Quantization Error Propagation

In the previous sections, we have shown that efficient deployment of large-scale transformer models in production environments often relies on reducing numerical precision of weights and activations to lower bit-width formats. While the fundamentals of quantization are well understood, the subtle dynamics of how quantization errors propagate through the layers and interact with evolving activation distributions remain underexplored.

In each linear transformation, the output element is perturbed by rounding error when weights are quantized. Although this error occurs only once per layer, errors **compound** through successive layers: a quantization perturbation at layer l becomes part of the input to layer $l + 1$, where it is further transformed and potentially amplified. Layers with larger spectral norms or wider activation ranges act as amplifiers, often leading to near-exponential growth of cumulative error in deeper layers [42].

To quantify how errors accumulate, one attaches forward hooks at the output of each transformer block in both the full-precision model and its quantized counterpart. By capturing hidden-state activations under identical inputs, we can compute their difference at each layer.

Empirical measurements consistently show three regimes of error growth:

- **Initial transformer blocks exhibit minimal perturbation**, owing to both residual connections and layer normalization that dampen errors. Distributions are narrow and near-Gaussian, reflecting general feature extraction.
- Error growth accelerates near the model’s midpoint, where representations shift from general feature extraction to task-specific encodings, increasing activation magnitudes.

Positive skew emerges as channels specialize for particular input patterns.

- In the final layers, cumulative error often follows an exponential law, indicating that **layers with high dynamic range and large Lipschitz constants are critical error-sensitive bottlenecks**. Heavy tails and increased variance indicate rare but strong activations corresponding to high-level semantic features.

Understanding these trends supports **layer-adaptive quantization**, where more robust early layers use lower precision uniformly, while later layers receive finer-grained or mixed precision, this, combined with activation-aware quantization techniques should lead to best results - when adapting to specific use-cases. Visualizing flattened activations using violin or box plots highlights these trends, showing gradual widening of distributions and the emergence of heavy tails in deeper layers.

4.5 Resource Utilization

Modern AI inference workloads face fundamental performance bottlenecks that are increasingly determined not by computational throughput, but by the rate at which data can be transferred from memory to processing units - in other words, by hardware capabilities. This is particularly pronounced in large language models (LLMs) and transformer architectures, and to better understand the effect, we use *memory bandwidth utilization* as a critical performance metric for inference optimization:

$$\text{Memory BW Utilization} = \frac{\text{Achieved Bandwidth}}{\text{Peak Memory Bandwidth}} \times 100\%$$

Even with highly capable compute units, the rate at which model weights and activations can be moved from high-bandwidth memory (HBM) to the computation units often directly limits overall inference speed. Energy consumption during inference is dominated by memory access [43].

The concept of the “memory wall” describes the growing imbalance between computational capabilities and memory system performance in modern hardware architectures. While GPU compute performance has increased dramatically, with advances enabling specialized tensor cores and reduced precision arithmetic, memory bandwidth has not scaled proportionally, creating a fundamental bottleneck for neural network inference workloads [44]

In most practical LLM inference deployments, where token-by-token generation with small batches dictates most of the inference load, the arithmetic intensity is low, making performance memory-bound. This means that the maximum theoretical tokens/sec output directly scales

with how many bytes/sec of model parameters and key-value (KV) cache can be delivered from memory to compute units.

There are a few methods we can use for reduced memory footprint and better runtime performance of a model. Increasing the batch size for example, will move the problem to be closer to a CPU-bound problem rather than a memory-bound problem.

To assess how close actual inference performance approaches this theoretical bound, the **Model Bandwidth Utilization** (MBU) metric is introduced:

$$\text{MBU} = \frac{B_{\text{achieved}}}{B_{\text{peak}}}$$

Where **Achieved Bandwidth** is

$$B_{\text{achieved}} = \frac{\text{Model Parameter Size} + \text{KV Cache Size}}{\text{Time Per Output Token}}$$

Empirical research shows well-optimized LLM serving systems can attain MBUs of 50-70%, though this decreases with factors such as excessive model sharding across devices, sub-optimal memory access patterns, or communication overhead between distributed accelerators.

Calculating the maximum theoretical inference rate for a large model on any hardware platform starts from an accounting of sustainable memory bandwidth and model size. The theoretical peak performance and MBU together offer a principled basis for predicting, benchmarking, and optimizing LLM inference - guiding both architectural choices and system-level optimizations needed to approach hardware limits in production.

The key difference is that memory bandwidth utilization is a general hardware metric showing raw bandwidth usage, while MBU is specifically designed to measure how efficiently LLM inference workloads are utilizing available memory bandwidth by accounting for the actual data movement requirements of loading model weights during token generation.

4.6 Latency vs throughput

Inference systems face a fundamental trade-off between latency (per-user response speed) and throughput (total tokens generated per second across all users). The two metrics pull in opposite directions: lower latency requires small batch sizes that leave GPU compute underutilized, while higher throughput requires large batches that increase per-user queue time and latency. For Arctur's service offerings, both dimensions matter depending on the use case. Interactive chat applications require low time-to-first-token (TTFT) and high inter-token throughput (tokens per second per user). Batch document processing applications prioritize total throughput

with looser latency requirements. Serving strategies must be configured separately for these workloads, ideally with separate inference deployments using different batch size targets.

When smaller models like Qwen3-8B are distributed across multiple GPUs, we observe lower MBU as we are moving smaller memory chunks in each GPU.

For comprehensive throughput benchmarking, the tests include **input and output token lengths** of 128, 256, 512, 1024, and 2048 tokens. These cover the full spectrum from short interactions to document processing scenarios.

Different applications require distinct input/output ratios:

- **Translation tasks:** Similar input and output lengths (~500-2000 tokens each)
- **Content generation:** Short input (~100 tokens), long output (~1000 tokens)
- **Summarization:** Long input (~1000 tokens), short output (~100 tokens)
- **Reasoning models:** Short input (~100 tokens), very long output (1000-10000 tokens)

For general benchmarking, **1024 input × 1024 output** tokens provides a balanced baseline that reflects many real-world scenarios. The following results are a mix of the various input and output lengths, with ~1000 requests per benchmark, each request having an average of 543 input tokens and 150 output tokens.

We analyze three primary latency components:

- $TTFT = \text{Time to First Token}$
- $TPOT = \text{Time per Output Token}$
- $\text{Total Request Latency} = TTFT + (n \times TPOT)$ where n is number of output tokens

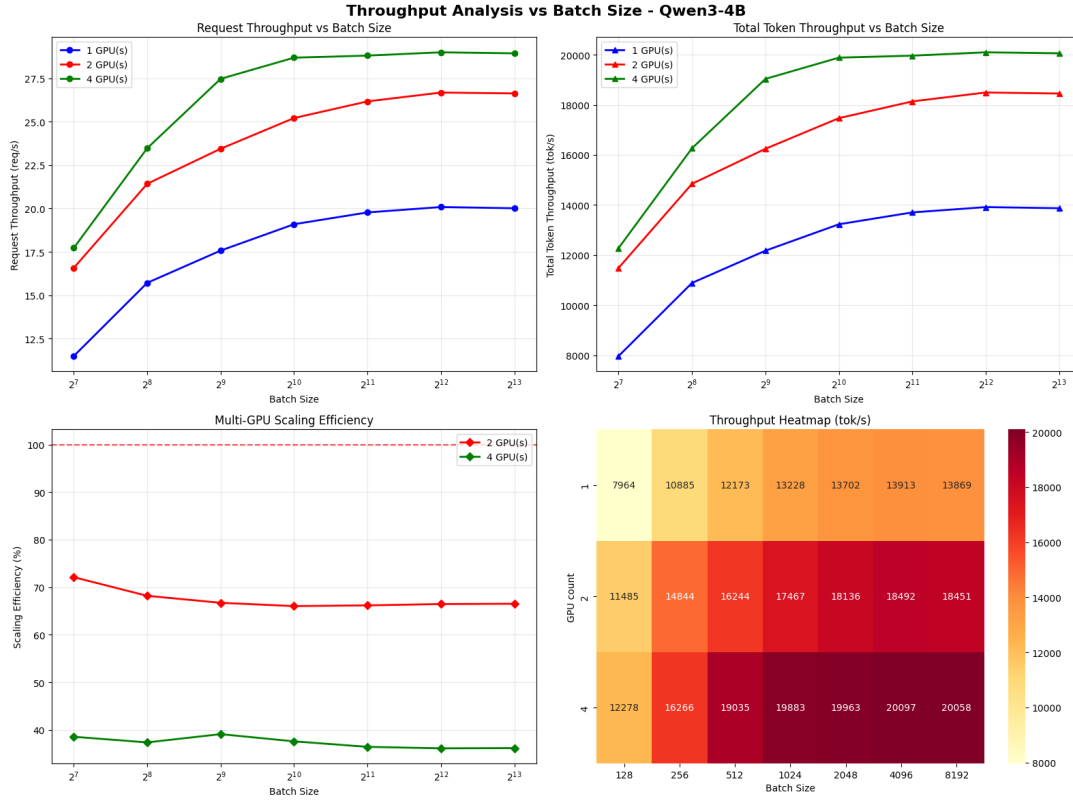


Figure 4.5: Throughput at various batch sizes for Qwen3-4B

The results show throughput increases dramatically with batch size until hardware saturation. Batch size 1024 can achieve **2x higher throughput** than batch size 128, this is even more pronounced for batch size 1. However, testing with batch size 1 is not really relevant for real world scenarios.

Scaling efficiency measures how effectively additional GPUs contribute to performance compared to perfect linear scaling.

$$\text{Scaling Efficiency (\%)} = \frac{\text{Actual Speedup}}{\text{Ideal Speedup}} \times 100$$

Where:

- **Actual Speedup** = Multi-GPU Throughput ÷ Single GPU Throughput
- **Ideal Speedup** = Number of GPUs (perfect linear scaling)

While **efficiency decreases** with more GPUs, **absolute performance** still increases, so the optimal choice depends on whether we prioritize cost-effectiveness and energy efficiency or raw performance (throughput).

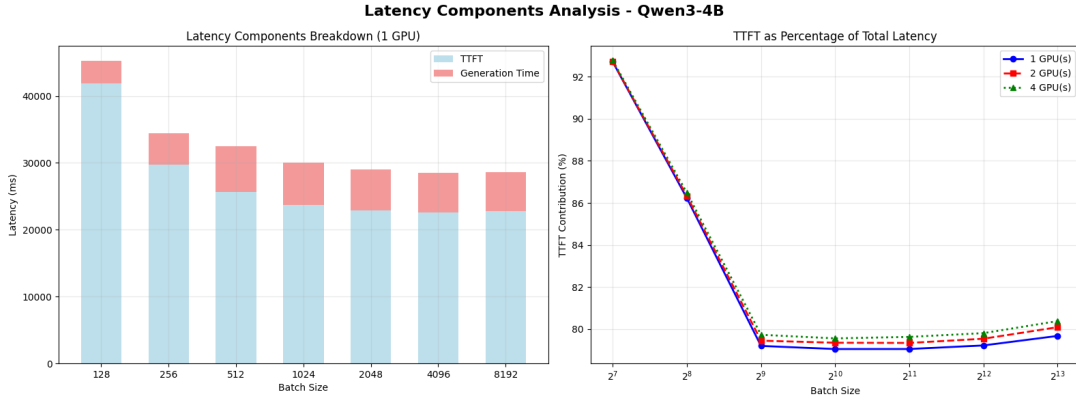


Figure 4.6: Latency metrics at various batch sizes for Qwen3-4B

As the **batch size increases**, **TTFT (Time to First Token)** decreases, which could be perceived as counter intuitive. We observe that for Qwen3-4B, TTFT drops from 41,924ms to 22,790ms (-45.6%). Larger batches spread the fixed computational overhead across more requests, reducing per-request latency. Hardware factors also influence this; small batches underutilize GPU parallel processing capabilities, while larger batches better saturate GPU compute resources. Matrix operations, with optimized GEMM kernels used in modern inference engines, are more efficient with bigger tensors.

One particular inference trait that influences memory bandwidth utilization, is that **model parameters are loaded once per batch**, regardless of batch size. The KV cache operations also become more efficient, as memory access patterns improve with larger batch processing.

While **TTFT decreases**, the results also show that **TPOT (Time per Output Token)** **increases**; Qwen3-4B TPOT increases from 22.0ms to 38.8ms (+76.3%).

This creates the **throughput-latency trade-off**: Smaller batch sizes show worse (higher) TTFT and better (lower) TPOT, with lower throughput, while larger batch sizes generate with higher throughput.

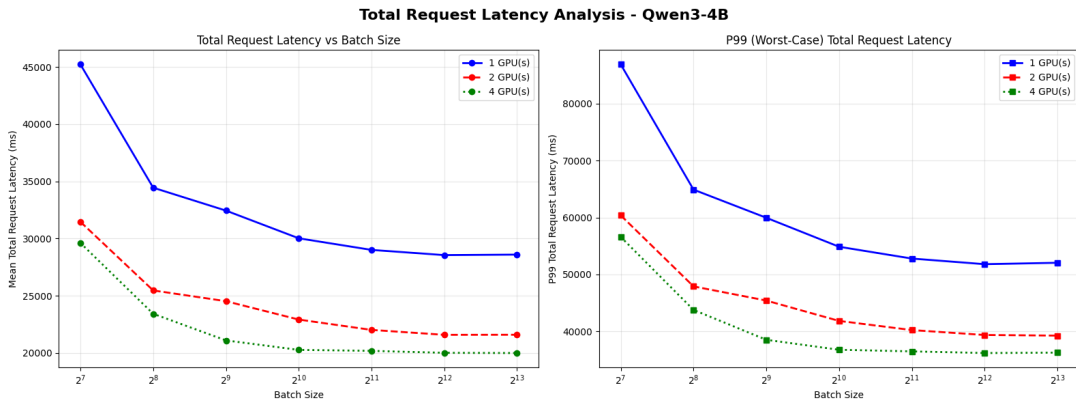


Figure 4.7: Request latency at various batch sizes for Qwen3-4B

For **total generation time**, larger batches may still be slower due to increased TPOT, creating another trade-off between **user-perceived latency** (TTFT) and **total processing latency**.

We see that request latency increases with batch size for total request time, but TTFT can decrease. Total Request Latency (also called **End-to-End Request Latency**) is the complete time from when a request is submitted until the final response token is received.

Request latency decreases with batch size, but reaches a plateau after batch size 1024 on 4 GPUs. Shared inference services typically pick a balanced batch size. Users hosting their own models should decide the appropriate latency/throughput trade-off for their applications. In some applications, like chatbots, low latency for fast responses is the top priority. In other applications, like batched processing of unstructured PDFs, we might want to sacrifice the latency to process an individual document to process all of them fast in parallel. Our benchmarks show that there is no incentive to use smaller batch sizes for general tasks.

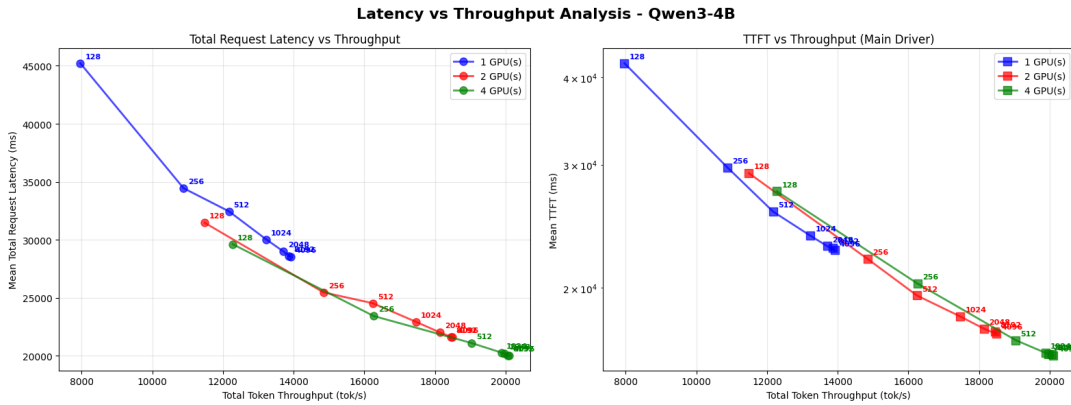


Figure 4.8: Latency vs throughput at various batch sizes for Qwen3-4B

Under certain conditions, when we have a computational overhead, and output sequences remain relatively short, increasing batch size can simultaneously improve both latency and throughput - this has significant implications for production inference optimization strategies.

Chapter 5

Benchmarking and Evaluation Methodology

5.1 Performance metrics for LLMS

Meaningful evaluation of LLM performance requires separating two distinct measurement concepts: intelligence benchmarks that assess what a model knows and can reason about, and performance benchmarks that assess how fast and efficiently it serves those capabilities. Both dimensions matter for production deployment but they measure fundamentally different things. Latency metrics include: + **Time to First Token (TTFT)** measures the latency from request submission to the generation of the first output token, primarily reflecting the efficiency of the prefill phase where the entire prompt is processed in parallel + **Tokens Per Second (TPS or t/s)**, driven by memory-bandwidth-bound decode quantifies the rate of token generation during the decode phase + **End-to-end latency** which represents total request time including network and queue

Latency distributions are right-skewed, so P90 and P99 percentiles are more informative than means, usually outliers disproportionately affect user experience. Throughput metrics should include total tokens per second across all concurrent requests, which determines the maximum user load a deployment can sustain. Energy metrics like **energy per output token** in Joules, measured at the GPU level are the primary efficiency measure of this thesis, measured using the Zeus energy measurement framework [45].

These metrics are interdependent and often exhibit trade-offs; for instance, optimizations that improve TTFT through larger batch sizes may increase memory usage, while techniques like quantization can reduce memory requirements at the potential cost of generation quality, making comprehensive evaluation essential for deployment decisions.

5.2 Benchmarks

The evaluation of large language models needs to leverage a suite of standardized benchmarks, each designed to measure distinct aspects of model capability. No single benchmark captures all dimensions of performance; instead, a multi-benchmark approach reveals strengths, limitations, and trade-offs relevant to specific applications. Below, key benchmarks that are used in the thesis are described in detail.

Language Understanding and Reasoning

MMLU (Massive Multitask Language Understanding) comprises 57 multiple-choice tasks drawn from professional and academic subjects-ranging from US history and law to biology and calculus. It assesses both **factual recall** and **specialized reasoning**. Questions often require the model to interpret domain-specific terminology, apply procedural knowledge (e.g., solving algebraic equations), or evaluate counterfactual scenarios in social sciences. High MMLU scores indicate that a model encodes a broad spectrum of structured knowledge and can perform discrete reasoning steps, but they do not directly measure generative fluency or conversational coherence. Because MMLU questions have clear, correct answers, performance is less sensitive to variation in prompt phrasing, making it a reliable gauge of stored knowledge across domains.

HellaSwag is constructed to foil models that rely on superficial syntactic cues by pairing each context with four plausible but incorrect continuations alongside one correct continuation. Tasks require deep **commonsense reasoning** and understanding of real-world physics, object interactions, and human behavior. According to evaluation studies, models that excel on HellaSwag demonstrate robustness to adversarial distractors and a capacity for inference beyond language patterns.

ARC (AI2 Reasoning Challenge) includes science exam questions at the middle- and high-school level, presented in both multiple-choice and open-ended formats. Questions test **applied reasoning**, such as interpreting experimental results, understanding physical laws, or inferring chemical properties. The open-ended subset in particular assesses generative explanation ability: models must articulate concise answers without relying on answer choices. ARC performance correlates with the model’s ability to integrate factual knowledge with deductive logic, mirroring the investigative reasoning required in scientific problem solving.

TruthfulQA consists of questions crafted to expose **hallucination** and **misinformation** tendencies, including ambiguous or leading prompts that can induce erroneous model outputs. It measures the **factual reliability** of generated answers and penalizes confidently incorrect or misleading content. Models that perform well on TruthfulQA exhibit stronger

alignment with factual sources and reduced susceptibility to biases in prompt phrasing.

Mathematical and Analytical Reasoning

GSM8K (Grade School Math 8K) has become the standard benchmark for evaluating **mathematical reasoning capabilities** in LLM. This dataset contains 8,500 grade-school level math word problems requiring multi-step reasoning and arithmetic operations. Quantization research consistently demonstrates that mathematical reasoning tasks are particularly sensitive to precision reduction, with GSM8K scores often showing more substantial degradation than general language understanding tasks.

Code Generation and Programming

HumanEval stands as the definitive benchmark for assessing code generation capabilities in quantized models. This dataset contains 164 hand-written programming problems with corresponding test cases, evaluated using the pass@k metric. Research shows that code generation tasks exhibit particular sensitivity to quantization, with notable performance drops observed at 4-bit and below quantization levels. It assesses an LLM's **code synthesis** capability, including its understanding of programming constructs, algorithmic logic, and API usage. High performance on this benchmark indicates that a model has internalized programming patterns and can produce functionally correct, maintainable code, which is critical for applications in software engineering assistance and automated code generation.

Language Modeling and Perplexity

WikiText-2 serves as the standard perplexity evaluation dataset, providing consistent baselines for measuring language modeling degradation. Perplexity measurements on WikiText-2 offer quantitative insights into how quantization affects the model's ability to predict text continuations, with studies showing strong correlations between perplexity increases and downstream task performance degradation. **WinoGrande** challenges models with pronoun resolution tasks requiring commonsense knowledge. Studies indicate that WinoGrande performance remains relatively stable under quantization, suggesting that coreference resolution capabilities are robust to precision reduction. **TruthfulQA** evaluates the truthfulness and factual accuracy of model responses, serving as a hallucination detection benchmark. Research indicates that quantized models often show increased susceptibility to generating false information, making TruthfulQA a crucial evaluation metric [46].

Interpreting Benchmark Results

In the benchmark overview above, individual datasets and benchmarks are introduced to clarify which aspects of LLM capability are most relevant for this thesis: general knowledge and reasoning (MMLU, HellaSwag, ARC), factual reliability and hallucination resistance (TruthfulQA), mathematical reasoning (GSM8K), coding ability (HumanEval), and language modeling quality (WikiText-2, WinoGrande). By aligning benchmark selection with deployment goals and by critically examining prompt sensitivity, cultural biases, and dataset coverage, researchers and practitioners can more accurately predict real-world LLM behavior and guide further model development. Contemporary evaluation practice, however, increasingly relies on composite suites that integrate many such tasks into a unified framework. To produce this multi-dimensional evaluation in a reproducible way, some experiments use **Metabench**, a meta-benchmark that aggregates these standard datasets under a common evaluation interface, scoring methodology, and reporting format. Metabench is chosen because it subsumes most of the benchmarks discussed above, allowing the thesis to preserve the conceptual separation of capabilities (knowledge recall, reasoning, commonsense, coding, and hallucination) while avoiding fragmented, benchmark-specific pipelines. This unified suite makes it possible to trace how quantization affects different capability clusters within a single evaluation run, and to compare models across applications (scientific reasoning vs. coding assistance, etc.) using consistent metrics.

Chapter 6

Efficient Inference

The computational analysis of Chapter 4 established that LLM decode is memory-bandwidth bound. A Qwen3-14B model in FP16 requires 28 GB of VRAM and generates approximately 217 GB/s of memory traffic per forward pass at batch size 1 which saturates the NVIDIA L40S's 864 GB/s bandwidth at approximately 4 concurrent decodes. This bottleneck motivates the compression techniques discussed in this chapter: if model weights can be represented in fewer bits, the same memory bandwidth delivers more computation per second, directly improving throughput and reducing energy per output token. Compression can be achieved using two strategies: **structural compression** (pruning), which removes entire neurons, attention heads, or layers, producing smaller architectures that require less computation regardless of hardware; and **numerical compression** (quantization), which reduces the bit-width of existing parameters, allowing the same computation to proceed faster on hardware with native low-precision support. Both strategies involve accuracy trade-offs that must be carefully measured.

Modern generative AI models sizes range from millions to trillions of parameters. Assuming a standard 32-bit floating point representation of each parameter, we come to the conclusion that memory requirements scale up to terabytes. That, and other computational demands of LLMs limits their usability - local deployments on consumer GPUs or on edge devices seem impossible. Several model compression and optimization techniques have emerged as viable solutions, each with its own set of trade-offs:

- **Pruning** reduces model size by removing some weights out of the model completely. The techniques focus on weights that do not have a big impact on model performance - for example *zero* or *close-to-zero* weights.
- **Knowledge distillation** a special case of model compression, that transfers learned weights from a larger to a smaller model. This allows the smaller model to be smaller and lighter, and *some times* of increased performance
- **Decomposition** is a method that tries to decompose big weight matrices into other -

smaller matrices. For example, using SVD (singular value decomposition) we can approximate a $N \times N$ matrix with two $N \times k$ matrices.

- **Quantization** is a method that reduces the precision of weights and activations - from 32 and 16-bit floating points to formats like 8-bit integers or more recently 8-bit floating point.

6.1 Pruning

Pruning essentially removes unused parameters in neural networks, the idea behind it is that modern neural network are usually overparametrized. Some studies have shown that we can reduce the amount of parameters by 90% without significant model performance [47]. This aligns also with the Lottery ticket hypothesis which states that “A dense, randomly-initialized neural network contains a subnetwork that, when trained in isolation, can match the performance of the original network.” [48] . Weights or neurons are simply *dropped* after training or during training, and while the pruning of weights alone will reduce memory footprint it will not reduce compute time in GPUs due to the registers being filled with the same neurons as pre-pruning.

It is infeasible to retrain or iteratively prune models at the scale of hundreds of billions of parameters. Thus, methods in iterative pruning and lottery ticket hypothesis [48] can only be applied to smaller-scale models.

If the underlying hardware and inference engine can efficiently exploit this sparsity (e.g., by skipping computations for pruned weights), the number of operations per token can be reduced, leading to **higher token generation rates (tokens/sec)** and more efficient use of memory bandwidth. Studies in structured and block-wise pruning of transformer weights have shown measurable speedups and throughput improvements, particularly when the pruning results in compute-friendly sparsity patterns and the inference system is optimized for sparse matrix multiplication [49].

However, on general-purpose GPUs, the actual **throughput gains from unstructured pruning may be limited unless specialized sparse compute libraries or architectures are used**, which is usually not the case. Structured pruning (removing whole heads, channels, or blocks) tends to yield more consistent improvements because it aligns better with hardware parallelism, leading to faster inference and increased total throughput.

Structured pruning methods have been shown to disproportionately harm in-context learning (ICL) performance compared to other capabilities . Applying magnitude-based pruning to transformer models, in-context learning abilities deteriorate more rapidly than general language modeling performance, suggesting that ICL relies on specific network struc-

tures that are vulnerable to pruning [50]. Experiments on GPT-style models show that even modest pruning ratios (20-30%) could **reduce few-shot learning accuracy by 40-60%** while maintaining reasonable perplexity scores.

Gradient-based pruning approaches reveal similar patterns. Pruning based on gradient magnitudes preferentially removes parameters that are crucial for ICL, particularly those involved in cross-attention mechanisms and later transformer layers [51]. ICL performance on tasks like few-shot classification and reasoning dropped significantly faster than zero-shot performance as pruning intensity increased.

SparseGPT is a post-training pruning method specifically designed to compress large language models from the GPT family efficiently and accurately without requiring any retraining. Traditional pruning methods require retraining to recover lost accuracy, while SparseGPT can prune models to at least 50% sparsity in a single pass without any fine-tuning [52]. While this is a promising technique - the language models they used for evaluation - BLOOM and OPT are known to be seriously undertrained by Chinchilla scaling laws. **If a model is undertrained - a lot of the weights serve no purpose.**

These recent findings, together with the fact that we rarely see any improvement in throughput, suggest that pruning may not be mature enough for production use. This thesis focuses primarily on quantization as the practical compression technique. Pruning remains relevant for edge/IoT deployments with extremely constrained memory, where quantization alone cannot achieve the required model footprint.

6.2 Quantization

Quantization converts model parameters from their native high-precision representation (typically BF16 or FP16, using 16 bits per value) to a lower-precision format, reducing memory footprint and enabling faster computation on hardware with native low-precision support, without altering the model structure. We focus on **post-training quantization (PTQ)**, which directly quantizes LLMs - sometimes using calibration datasets to optimize scale factors without retraining. Standard PTQ methods typically round weights to the nearest level to keep runtimes feasible for large models [53] [54] [55]. Quantization can be used to reach various goals; be it to increase the numerical range and increase accuracy, as a test to drive hardware development, or reducing the memory footprint of a model and improve throughput by sacrificing accuracy.

INT4 quantization for models where both weights and activations are quantized to 4-bit integers (W4A4) can dramatically speed up (8.5x speedup) and shrink transformer models like BERT and BART with **little to no accuracy loss, but are not yet suitable for decoder-**

only models like GPT due to significant quality degradation [56].

It is important to understand that **4-bit quantized models perform inference de-quantization** and then computing. The weights are stored in compressed 4-bit format but are dequantized to FP16/BF16 for actual mathematical operations. The GPUs don't have capabilities to do INT4 computations. This approach achieves significant memory savings and bandwidth improvements while maintaining computational accuracy, though it doesn't eliminate the need for higher-precision arithmetic during inference. This can have a significant computational overhead which needs to be accounted for.

Memory bandwidth is often the biggest source of latency in single-prompt inference, as for the inference process to complete, we need to pass the whole model from VRAM to the GPU processors, and for each generated token - quantization dramatically reduces this transfer overhead. At BF16 precision a 14 billion-parameter model like Qwen3-14B occupies about 29.5 GB and requires roughly 34 ms per token just to stream its weights. Converting the model to INT8 cuts its size to around 15.7 GB, halving the transfer time to approximately 18 ms per token. Pushing further to a highly optimized 4-bit format reduces the footprint to about 9 GB and brings per-token transfer times down to just over 10 ms.

The decrease in memory-transfer latency should outweigh any added compute overhead from dequantization, making it exceptionally well suited for real-time language model deployment.

Modern hardware can execute integer adapted GEMM kernels directly, so several inference toolchains now keep both weights and activation tensors in INT8 throughout the forward pass instead of inflating them back to FP16/BF16 on every layer. The computational decisions between inference engines and quantized models represent a careful orchestration of hardware-accelerated computation, memory management, and on-the-fly data transformation. Understanding this relationship requires examining how inference engines perform computations, where quantization mechanisms have their advantages, and the critical differences between dynamic and static quantization approaches. **Quantization can be very useful when it can make use of specialized hardware features.**

Inference engines include specialized kernels for various quantization formats:

- **Marlin kernels** for GPTQ and AWQ,
- **CUTLASS kernels** for INT8/FP8, and custom implementations for other formats.

These kernels perform **fused dequantization and computation**, avoiding the overhead of separate dequantization steps.

We differentiate between two different PTQ types:

- **Static quantization** typically delivers **2-3x better performance** than dynamic approaches
- **Dynamic quantization** may provide **better accuracy** at the cost of inference speed. Applies quantization only at inference, balancing speed and accuracy

For production deployment, **static quantization** is generally preferred due to predictable performance characteristics.

The **Marlin Linear Kernel** is a family of highly optimized matrix-multiplication kernels designed for mixed-precision inference of large language models (LLMs). It is integrated into inference engines like vLLM, it targets the key bottleneck of LLM inference - the heavy memory traffic of loading model weights - by combining low-precision storage with fused computation [57].

Marlin is adapted to the GPTQ quantization methods, weights are stored in low precision (e.g., INT4) with per-group or per-column FP16 scaling factors while activations remain in FP16. During inference, Marlin fuses dequantization and Tensor-Core math in a single pipeline stage, avoiding separate weight-load and dequantize steps. Modern GPUs often have a FLOP-to-byte ratio of 100-200, meaning they can compute much faster than they can fetch data from memory. By reducing each weight from FP16 to INT4, Marlin achieves a theoretical $4\times$ reduction in memory movement, which directly translates into up to $\sim 4\times$ speedups for batch-sizes up to the **memory-optimal batch** (typically 16-32 tokens) before the computation becomes the bottleneck.

6.2.1 Quantization-Aware Training

Empirical studies show that the impact of using different precisions for weights, activations, and attention is systematic and can be captured by scaling laws, implying that pure BF16 or very low precision (e.g. FP4) training alone is not optimal [58]. Quantization-aware training (QAT) and mixed-precision schemes recover most of the accuracy lost by naive post-training quantization [59], with late-stage QAT (roughly 90% of the budget in BF16/full precision, $\sim 10\%$ in QAT) typically outperforming both PTQ and training entirely at the target quantized precision [60]. Recent open-weight models such as Gemma-3 adopt this strategy, integrating QAT into the training recipe rather than treating quantization as a purely post-hoc deployment step. Several frontier labs even release QAT-quantized models directly; GPT-OSS in MXFP4 and Kimi K2 Thinking in INT4 are notable examples.

6.2.2 Numerical Formats in LLMs

Large language models, and other deep learning models have billions of numerical values stored as their parameters. Choosing how each of these values is encoded onto the hardware directly determines memory footprint, cache pressure, bandwidth demand, and ultimately attainable performance. Neural network weights show distributions where most values cluster near zero, with occasional large outliers. Low-precision formats must be designed to preserve this distribution faithfully.

The two primary numeric types used in LLM quantization are:

- Integer formats (INT8, INT4): fixed precision with linear spacing across the representable range. Simple to implement but limited dynamic range (outlier values are poorly represented).
- Floating-point formats (FP8, FP16, BF16): variable precision using sign, exponent, and mantissa bits, providing higher dynamic range. FP8 E4M3 (4 exponent bits, 3 mantissa bits) is the standard format for H100 and L40S FP8 inference.

6.2.3 Post-Training Quantization Methods

Post-training quantization converts finished model weights to a new precision by computing scale factors and validating quality through calibration. Scale factors map the low-precision representation back to the original value distribution. The granularity of scale factor application - per-tensor, per-channel, or per-block - determines the quality-performance trade-off: finer granularity preserves accuracy but introduces storage and computation overhead.

Round to nearest

Round-to-nearest (RTN) quantization is a straightforward yet effective method that transforms high-precision weights into lower-precision representations by applying a scaling factor followed by rounding operations. The process begins by calculating a scaling factor S using the formula. $S = \frac{\max(r)}{2^{b-1}-0.5}$ where $\max(r)$ represents the maximum absolute value in the input tensor and b is the target bit precision. Each weight value r is then divided by this scaling factor and rounded to the nearest integer using the formula $RTN(r) = \text{Int}(r/S)$ where the Int function performs both round-to-nearest and truncation operations to fit the target range (e.g., $[-128, 127]$ for INT8). To mitigate the impact of outliers that can skew the scaling factor and reduce quantization resolution, RTN is typically applied in groups of g values (commonly $g=128$) rather than to entire tensors, requiring separate scaling factors for each group but providing better

accuracy. The rounding process employs **round-to-nearest-even**, which rounds tie cases to the nearest even value to reduce systematic bias in quantization. This approach does not require calibration and can be applied on-the-fly during model loading.

K-Quants

K-Quants is a popular quantization methods offering Small, Medium and Large varieties, for example: Q4_K_M is a medium sized K-quantized model down to 4bit. The main difference is that while classical quantization uses fixed size blocks (typically 32 weights per block) with one scale per block, k-Quants quantization uses blocks and superblocks, with per-block and per superblock scales - this allows more granular scaling. In k-Quants, also the scaling factors can be quantized.

The S, M, L indicate block and superblock size:

- Large (L) has larger blocks or superblocks which means fewer scaling factors per total weights - more compressed.
- Medium (M) is a balanced block size with a good trade off and is most commonly used.
- Small (S) smaller blocks and superblocks, they preserve better accuracy but are larger and slower to inference.

Because of the many options in the variations(Small, Medium, Large) and the bit number - k-Quants offer more efficient and clear tradeoffs. A 13B model in its full 32-bit format, requires about 48 GB of memory - while its 16-bit representation requires about 24GB of memory. Reducing this model to 4-bit quantization would shrink it to around 6 GB. In traditional 4-bit quantization (Q4), we might see noticeable drops in performance, particularly for tasks requiring creativity or precise language understanding. However, by using k-Quants, such as a Q4_K_M variant, we can get a model that still fits within the 6GB range but retains significantly more accuracy and performance.

SmoothQuant (W8A8)

SmoothQuant enables accurate 8-bit weight and 8-bit activation inference for LLMs by addressing the fundamental challenge of systematic activation outliers that typically degrade quantization performance. The method applies a mathematically equivalent transformation using scaling factors to redistribute quantization difficulty from activations to weights, where a linear layer computation $Y = XW$ is reformulated as $Y = (X \cdot \text{diag}(s)^{-1}) \cdot (\text{diag}(s) \cdot W)$

allowing the activation scaling to be absorbed into preceding layers while the weight scaling is pre-computed offline. The scaling factors are determined using calibration data and controlled

by a hyperparameter that balances the quantization difficulty between weights and activations, with typical values ranging from 0.4 to 0.9 for simultaneous weight-activation quantization. This approach has demonstrated effectiveness across major transformer architectures including OPT, BLOOM, GLM, MT-NLG, and LLAMA families, enabling significant memory reduction and computational acceleration while preserving model accuracy. The key innovation lies in its ability to smooth the distribution of both weights and activations without requiring retraining, making it particularly valuable for deploying large language models in resource-constrained environments where both memory bandwidth and computational efficiency are critical [61].

AWQ (W4A16)

Activation-aware Weight Quantization (AWQ) is a post-training quantization method that enables efficient low-bit weight-only quantization for LLMs by leveraging the key insight that not all weights contribute equally to model performance, with protecting merely 1% of salient weights significantly reducing quantization error [62]. The technique identifies critical weight channels by analyzing activation distributions rather than weight magnitudes, recognizing that quantization error is proportional to input activation magnitude and that weights multiplied by large activation values are more sensitive to quantization. AWQ employs a mathematically equivalent scaling transformation where salient weight channels are scaled up to protect them from quantization damage, with the scaling factors determined through offline calibration using activation statistics without requiring backpropagation or model reconstruction. This activation-aware approach allows the method to maintain model accuracy while achieving aggressive quantization to 4 bits, demonstrating good generalization across different domains and modalities compared to traditional quantization schemes. The method has proven particularly effective for instruction-tuned language models and multi-modal models, offering substantial memory reduction and computational acceleration while preserving the model’s ability to perform well on diverse benchmarks including language modeling, coding, and mathematical reasoning tasks.

EXL2 and EXL3

EXL2 and EXL3 are quantization formats developed to enable efficient deployment of LLMs on consumer GPUs. EXL2 introduced an optimized method for quantizing model weights, leveraging flexible mixed-bitrate strategies to achieve improvements in trade-offs between memory usage, throughput, and model fidelity during inference. This demonstrated reductions in VRAM requirements and improvements in inference speed compared to other quantization

schemes. EXL3 advances this methodology by incorporating recent research from multidimensional codebook quantization and randomized Hadamard transforms, offering higher compression ratios and improved conversion efficiency. Empirical results have shown that EXL3 can maintain coherent model output at lower bitrates, thereby reducing hardware demands while preserving inference quality. Unlike resource-intensive methods, both EXL2 and EXL3 emphasize practicality, requiring minimal preprocessing and computational resources for quantization. Collectively, these formats contribute to the broader effort to make LLM inference more accessible and cost-efficient, while retaining competitive performance benchmarks relative to existing quantization techniques.

GGUF

GGUF (GGML Universal Format) is a container and quantization format designed to enable efficient storage and deployment of large language models, particularly for local inference on consumer hardware, including CPUs and hybrid CPU-GPU configurations. The GGUF format streamlines model packaging by storing weights, metadata, and model architecture information in a single file, optimizing for rapid loading and broad cross-platform compatibility. Quantization in GGUF entails converting full-precision floating-point weights to reduced bit-width representations (most commonly 2- to 8-bit), using block-wise strategies that balance compression with fidelity by applying scale and offset parameters within grouped weights. Advanced methods such as k-quantization further refine this process by dynamically allocating precision to weights or layers deemed more important, minimizing accuracy loss at lower bitrates. GGUF also supports flexible hybrid deployment, allowing layers to be offloaded between CPU and GPU to accommodate hardware constraints. While GGUF lags behind highly optimized GPU-specific formats (such as EXL2 and EXL3) in terms of inference speed and VRAM efficiency, it excels in portability, ease of use, and adaptability, serving as the dominant format for local and edge-oriented LLM deployments where accessibility and resource efficiency are prioritized over maximum throughput.

GPTQ

GPTQ [54] leverages approximate second-order information from the Hessian matrix (which contains second derivatives of the error with respect to the weights). The algorithm quantizes the weights of each layer one at a time, focusing on minimizing the error in the layer's output rather than just the difference between original and quantized weight.

To do this, it uses second-order information: specifically, the Hessian matrix of the loss

with respect to the weights. For a linear layer, the Hessian H is computed as $H = 2XX^T$, where X is a matrix of input activations from a calibration dataset. But the Hessian can be very large and dense, especially for large language models so inverting it is computationally expensive. Interestingly, the algorithm uses the Cholesky decomposition as a key step to efficiently and accurately compensate for quantization errors during the quantization of large language models.

This allows the algorithm to make more informed decisions about which weights are most sensitive to quantization and how to best round them. It does not require access to the original training data or retraining the model, only a small set of representative calibration data.

The GPTQ quantization framework begins with a formal problem definition that frames weight quantization as an optimization challenge. The core objective is expressed through the optimization problem $\operatorname{argmin} \|WX - X\|^2$, where W represents the full precision weights at layer l , $\hat{W}$ denotes the corresponding quantized weights, and X represents the calibration dataset inputs at that layer. This formulation essentially seeks to find quantized weight values that minimize the squared difference between outputs produced by the original full precision weights and those generated by the quantized weights when processing the same input data. In practical terms, the optimization problem can be interpreted as “finding quantized weight values such that the model’s output behavior remains as similar as possible to the original full precision model’s behavior.”

The GPTQ solution methodology builds upon the Optimal Brain Quantizer (OBQ) technique from 2023, which itself extends the classical Optimal Brain Surgeon approach from 1993.

The calibration dataset serves as the foundation for computing the Hessian matrix, which guides the weight update process during quantization. Since the Hessian is computed using the mean of inputs to each layer, the quality of the calibration data directly impacts the quantization process. **The calibration dataset should be representative of the model’s expected inference workload** to ensure that the quantization optimization accurately reflects real-world usage patterns. Similar to traditional machine learning principles, concepts of overfitting and underfitting apply to the calibration process - insufficient or non-representative calibration data may lead to suboptimal quantization that degrades performance on the target domain, while overly specific calibration data might not generalize well to diverse inference scenarios. The dataset’s impact on downstream performance extends beyond basic inference accuracy to affect instruction-tuning tasks and domain-specific applications, making careful calibration dataset selection crucial for maintaining model quality across different deployment contexts.

Methods like GPTQ and AWQ have demonstrated the ability to quantize LLM weights to 4-bit precision with minimal performance degradation.

Vector Quantization

Traditional quantization methods often use uniform, scalar quantization (each weight quantized independently). Vector quantization is a state-of-the-art technique where groups of weights are quantized together, and increases the dimensionality of these groups. This higher-dimensional quantization grid better matches the true distribution of weights, improving representational accuracy for a given model size. VQ uses a non-uniform number system in multiple dimensions to aggressively reduce LLM footprint. VQ reduces model footprint, but fails to improve token rate because of the overhead for decoding VQ back to a native format. GPTVQ extends the GPTQ quantization algorithm to vector quantization. It interleaves quantization of columns with updates to unquantized weights, using the Hessian of the per-layer output reconstruction error to minimize quantization noise [63]. The method includes tunable parameters such as quantization dimensionality, codebook size, and group/block sizes, with ablation studies showing the impact of these choices on accuracy and compression.

AQLM AQLM achieving 2-2.5 bit quantization for large language models with better performance than GPTQ. Even though the method is based on additive quantization, AQLM [64] still uses vector quantization. The method does extreme compression down to 2bit with improved accuracy over uniformly quantized models at same bit width - however there is no support for it in inference engines.

1-bit LLMs

Large language models have recently been responsible for many advancements in AI. Because of their vast application potential, industry and academia have measured energy requirements, performance and scaling laws specifically for LLMs, sometimes taking the experiments to the extremes. The paper “The Era of 1-bit LLMs” by [65] discusses a new approach by quantizing LLM down to 1.58 bits, or 3 values for weights: $\{-1, 0, 1\}$ and 8 bits for activations, calling it BitNet 1.58, since their starting point was BitNet, which only has 1 bit for weights with values $\{-1, 1\}$. They claim similar zero-shot performance to LLaMA on a range of language tasks like ARC, HellaSwag, Winogrande. The conclusion is that at increased model size (70B parameters) the accuracy is the same, with great improvements in latency and memory efficiency. The methods discussed in this paper cannot be used for post-training quantization since models are trained from scratch.

TT decomposition

Tensor Train (TT) decomposition is a method for representing high-dimensional tensors (multi-way arrays) in a compressed, efficient format. It is particularly useful for reducing the storage and computational complexity of large-scale data in scientific computing, machine learning, and data compression. The TT decomposition expresses a high-dimensional tensor as a sequence (or “train”) of 3-dimensional tensors called **TT-core**

The most common algorithm for computing the TT decomposition is based on a sequence of **truncated Singular Value Decompositions (SVDs)**, sometimes called the **TT-SVD** or **Hierarchical SVD**

TT decomposition can represent huge tensors with exponentially fewer parameters if the TT-ranks are low.

Recent studies have developed hybrid algorithms that integrate TT decomposition with quantization. For example, the QuaDCNN method combines TT decomposition with weight quantization to compress deep convolutional neural networks (CNNs). In this approach: **TT decomposition** is first applied to decompose large weight tensors into a series of smaller TT-cores, drastically reducing the number of parameters.

Quantization is then applied to the elements of these TT-cores, reducing the memory requirements even more. Tucker and **CP** may outperform TT for lower-order tensors or when interpretability/flexibility is needed

Adaptive, compute budget-aware methods like LWIQ, which combine TT decomposition with quantization and automatic rank selection, can outperform standard TT decomposition. For example, LWIQ achieved 1.3-1.8% higher accuracy than TT at similar or higher compression rates on ResNet-32, indicating that integrating quantization and adaptive rank selection yields better results. Combining TT with other decompositions (e.g., TT+CP, TT+Tucker) or with quantization can achieve multiplicative compression, often outperforming any single method alone [66]

6.2.4 Quantization at various scales

One of the questions that is left unanswered is how scaling a neural network influences the accuracy and performance of its quantized version. The ratios are not well understood; could a 10B parameter model represented in FP16 have the same accuracy as a quantized version of the scaled 20B parameter model? Which of these models would have better performance and energy efficiency? How much does a model need to be scaled and in what way, to achieve the same accuracy in its quantized form? While an interesting topic, quantization of scaled models was

only mentioned by a couple of studies , with the conclusion that classical quantization techniques degrade heavily after 6.7B parameter LLM models because of the occurrence of outliers in them. Degradation-free quantization even up to 350M parameters is poorly understood [53].

When quantizing a model, we lose accuracy - but larger models with more parameters should compensate for the quantization loss better than smaller models. Models become harder to quantize as they are overtrained on lots of data, more pretraining data can be actively harmful. Here we see the Qwen3 model at various model sizes quantized using different methods and evaluated on the MMLU benchmark.

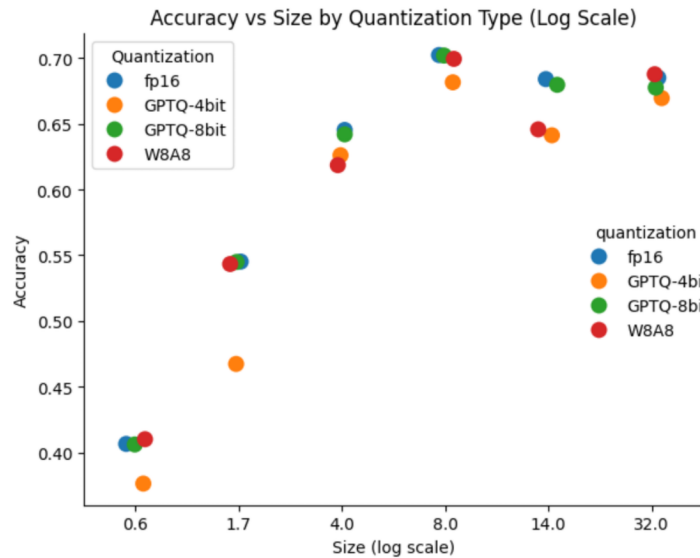


Figure 6.1: Qwen3 quantization at various scales using GPTQ, AWQ, RTN

Finding the sweet spot is a delicate task with considerations in compute cost and performance. Our test show that 8-bit quantization is the ideal balance. A 32B model quantized to 8bit could have better accuracy than a 13B full precision model for some tasks.

6.2.5 In-Context Learning and Quantization

Language models with billions of parameters have demonstrated impressive performance and adaptability with prompt engineering, few shot learning - also called in-context-learning, which is performing a task by conditioning input-output examples only, without changing model parameters [67] This provides a low cost alternative to fine-tuning and RAG systems, but with lower quality.

Although ICL outperformed zero-shot inference on many tasks, there is still little understanding on how it exactly works. ICL turns out to be a pattern recognition procedure, rather than learning. The input-output label pairs that are given to the model with context, serve as a guidance to retrieve similar examples it has been exposed to during training.

However, ground truth demonstrations are not required - replacing labels with random labels in the input-output pairs barely hurts performance on a range of classification and multi-choice tasks over 12 different models. Language models can still retain up to 95% of performance gains by using either the inputs only or the label set only if the right format is used. [68]

Part of the reason ICL works could be due to the so called *induction heads* [69]. Induction heads are specific attention heads - implemented via a “circuit” or two attention heads in transformer language models [70]. The first head is a “previous token head” which copies information from the previous token into the next token, while the second head (the actual “induction head”) uses that information to find tokens preceded by the present token. The effect they have is that together they can copy or complete patterns. Although the research was done on small models with simplified architecture - two layer attention-only models it still gives us an idea on what is happening in large language models with more complex architectures.

Contextual sparsity in large language models (LLMs) refers to a dynamic, input-dependent sparsity pattern where only a subset of model components, such as attention heads or neurons, are activated or used for a particular input, rather than statically pruning the same components across all input. This is distinct from traditional (static) sparsity, where specific parameters are permanently removed or deactivated regardless of the input. Small, input-dependent sets of attention heads and MLP layers can produce approximately the same output as a dense model [71].

A critical gap exists in understanding how quantization affects complex reasoning capabilities. Mainstream quantization methods can cause up to 32.39% accuracy degradation on mathematical reasoning tasks [72].

6.3 Prefill-Decode Disaggregation

As introduced in Chapter 4, LLM inference consists of two distinct phases. During prefill, the model processes the complete input prompt in a single highly parallel forward pass. This phase is usually compute-bound. During decode, the model generates output tokens autoregressively, normally one token per step. Some architectures can generate multiple tokens per step using multiple-token prediction (MTP), but autoregressive decode remains the common serving model. Decode is usually constrained by memory bandwidth and KV-cache capacity rather than by raw compute throughput.

In conventional inference serving, prefill and decode are colocated on the same GPU resources. This causes interference between the phases: large prefill batches can delay the per-token latency of active decode requests, while the continuous GPU-memory use of decode re-

quests, especially for the KV cache, limits the batch size available for prefill. Prefill-decode (P/D) disaggregation separates the two phases onto dedicated GPU pools. The prefill pool computes the prompt representation and produces the KV cache, which is then transferred to a decode worker that continues token generation. This allows each pool to use an independent parallelism strategy, batch size, and hardware configuration.

DistServe showed that this separation can eliminate prefill-decode interference and allow separate optimization of resource allocation and parallelism for the two phases and achieved up to 7.4x more requests under the same latency constraints, or met up to 12.6x tighter service-level requirements, than the colocated baselines. The exact benefit depends on the workload, latency targets, GPU configuration, and the cost of transferring the KV cache between the prefill and decode pools. This changes the problem from optimizing one shared inference deployment into a scheduling and resource-allocation problem. The system must determine how many resources to assign to prefill and decode, select suitable parallelism configurations for each pool, and transfer the KV cache with sufficiently low latency and high bandwidth. The optimal allocation depends strongly on workload characteristics. Workloads with long prompts, such as retrieval-augmented generation and document analysis, require relatively more prefill capacity. Interactive chat workloads with short prompts and long generated responses require relatively more decode capacity. [73]

Once prefill and decode run on physically separate hardware, they can also use different numerical precisions, model configurations, or accelerator types. Uniform quantization applies a single precision to the whole model, even though the phases have different computational properties and error sensitivity. This may be suboptimal in a disaggregated design.

Recent work suggests that prefill is more suitable for quantization than decode. A prefill quantization error changes the internal representation of a fixed input context once. In contrast, decode operates autoregressively: each generated token becomes part of the context for later steps, so numerical errors may alter later token probabilities and accumulate over a long generation. Mix-Quant uses quantized prefill with higher-precision decode. Its decode-only quantization configuration showed a weaker quality-efficiency trade-off than prefill-only quantization in the evaluated settings. This result should not be interpreted as universal, because quantization tolerance depends on the model family, bit-width, prompt length, decoding strategy, and target task. [74]

P/D disaggregation and phase-aware quantization create a further optimization space for production inference. A deployment can select the number and type of prefill and decode workers, their hardware, the model precision used in each phase, and the KV-cache transfer

format. These choices should be evaluated jointly using end-to-end metrics: time to first token (TTFT), time per output token (TPOT), throughput under compute constraints, model quality, and energy consumption per request.

Chapter 7

Energy Efficiency of Quantized Models

This Chapter integrates the paper Quantization Methods for Energy-Efficient LLM Deployments (ICAIR 2025).

7.1 Introduction

Beyond model compression, energy efficiency is becoming a crucial consideration. Recent benchmarking shows significant variability in energy profiles across quantization methods. This suggests substantial opportunities for developing quantization methods optimized specifically for energy efficiency. Some research shows that lower-precision models consume less energy, crucial for edge and mobile deployments [55]. However, the research is scarce, and details about the quantization specifics, inference kernels or how energy measurements are collected are usually not provided.

Larger models are more resilient to quantization accuracy degradation and some even suggest there are scaling laws to confirm it [75]. However, a lot of quantization research focuses their evaluations on the Facebook OPT model family, which is now outdated and is considered to be undertrained. OPT-175B was trained on 300 billion tokens, yet the compute-optimal training for a 175 billion-parameter model would require roughly 1.4 trillion tokens according to the Chinchilla scaling laws [11]. If a model is undertrained, a lot of weights are irrelevant, which has an effect on quantization accuracy and resilience.

Energy consumption in neural network inference depends on multiple factors including model size, quantization strategy, and hardware implementation. Recent work has established that quantization can achieve substantial energy savings, with studies reporting up to 37% energy reduction without accuracy loss in Convolutional Neural Networks [76].

The relationship between model parameters and energy consumption follows predictable patterns, with larger models requiring proportionally more energy per inference operation. However, quantization can significantly alter this relationship, enabling energy-efficient deployment

of larger models, but the magnitude of these effects is not clearly understood. Energy profiling can reveal that quantization methods show significant variability in energy consumption despite similar compression ratios [77].

7.2 Experimental Design

The quantization experiments are designed to jointly evaluate accuracy, throughput, and energy efficiency across the Qwen3 model family and six quantization methods.

Hardware: NVIDIA L40S GPUs (48 GB GDDR6, 864 GB/s memory bandwidth), native INT8 and FP8 support. Models up to 14B run on a single GPU; the 32B model requires two GPUs for FP16, GPTQ, and AWQ variants.

Models: Qwen3-0.6B, 1.7B, 4B, 8B, 14B, 32B. The uniform architecture scaling allows isolation of model size effects without architectural discrepancies.

Quantization methods:

- **FP16 (Baseline):** Original 16-bit floating-point precision
- **GPTQ 4-bit:** 4-bit quantization using GPTQ algorithm, activations in FP16
- **GPTQ 8-bit:** 8-bit quantization using GPTQ algorithm, activations in FP16
- **AWQ:** Activation-aware weight quantization, weights in INT8, activations in FP16
- **FP8 W8A8:** 8-bit floating-point quantization, weights and activations RTN
- **INT8 A8W8:** 8-bit integer weight and activation RTN

Metrics:

1. Benchmark accuracy on six benchmarks and MetaBench composite score;
2. Throughput (tokens/second) at context length 4000, output length 1000
3. Energy per output token (Joules), measured via the Zeus GPU-level energy framework [78], averaged over 100 requests per configuration.

The evaluation benchmark frameworks was discussed in Chapter 5. Accuracy metrics include benchmark scores across all evaluation tasks, enabling assessment of quantization impact on model capabilities. Benchmarks were done using the LM Evaluation Harness [79]. The experiments include measurements across different context lengths (300, 500, 1000, 4000 and 8000 tokens) and output lengths (100, 500, 1000, 4000 and 8000). For each input-output length combination, the experiments were repeated 5 times and the results averaged - to avoid any unrelated power usage discrepancies and interference.

The Qwen3 model family was selected for our experiments because it offers an ideal balance of scalability, performance, and ecosystem support. It is available at various parameter sizes,

which enables studying inference scaling and quantization trade-offs, allowing us to evaluate both tiny and large configurations under a unified architecture. At Arctur, the Qwen model family is used extensively across a number of products and services; to deliver AI-enabled tools like chat and RAG, as a model-as-a-service offering and to fine-tune for specific customer use-cases. Choosing the most efficient model size and quantization type for a use-case is the first step in any of those implementations. Qwen3’s release includes optimized inference kernels and first-class support for reasoning tasks, with built-in “thinking mode” APIs that enable implementing test-time scaling and chain-of-thought prompts. Empirically, Qwen3 has demonstrated strong zero-shot and few-shot reasoning accuracy across benchmarks like GSM8K and MMLU, making it a robust choice for exploring compute-for-reasoning paradigms.

The NVIDIA L40S GPU offers a good cost balance for AI infrastructure providers like Arctur: its 350W power consumption rating and standard dual-slot PCIe form factor let it slot into normal HPC servers and racks without specialized cooling or power delivery. It works well for container passthrough and lets workloads isolate cleanly per container in Kubernetes or SLURM setups. Its support for FP32, required by scientific simulation software is another reason why it is deployed in Arctur’s heterogeneous HPC environment. The card’s power draw and performance are also similar to Arctur’s own co-designed MACx AI accelerator, making it a useful reference point for the same rack and orchestration stack during the MACx AI development phase [80].

FP8 W8A8 quantization utilizes 8-bit floating-point representation, offering a balance between compression and accuracy preservation. This approach has gained adoption in modern hardware accelerators designed for machine learning workloads [81]

All models were quantized locally under the same conditions and standard calibration datasets using the **llm-compressor** library [82]. Inference was measured via an offline vLLM inference server.

7.2.1 GEMM Kernel impact on energy

Not all devices support all quantization formats natively, understanding compatibility and software implementations of quantization algorithms and inference engines is key to making sure we have an optimal setup that uses energy efficiently. Power consumption of GEMM (General Matrix-Matrix Multiplication) operations on GPUs can vary by up to 40% depending solely on the content and pattern of the input data even when matrix sizes and arithmetic workloads are identical [83]. Matrices filled with larger or more uniform numerical values (i.e., small sets of repeated numbers or sorted values) tend to lower GPU power usage. Inference engines like

vLLM automatically select the most appropriate GEMM kernel according to the model architecture and quantization - if that kernel is available. Increasing the input data sparsity generally reduces power, but for already sorted data, some sparsity patterns can paradoxically increase power use.

7.3 Quantization Impact on Model Performance

Table 7.1 presents accuracy retention rates (fraction of FP16 baseline accuracy preserved) on MetaBench across model sizes. The results confirm that larger models are systematically more resilient to quantization: the 32B model retains 98.3% accuracy with AWQ compared to 90.1% for the 0.6B model. FP8 and INT8 achieve near-lossless performance at 14B and above.

For some benchmarks, data demonstrates that larger models consistently maintain higher accuracy retention rates when subjected to aggressive quantization methods. Retention refers to the fraction of a model’s original (FP16) performance that remains after it has been quantized.

Table 7.1: Quantization resiliency across Qwen3 model sizes for Metabench benchmark

Model size	AWQ	GPTQ-4bit	GPTQ-8bit	INT8	FP8
0.6B	90.07%	91.25%	97.02%	100.94%	93.52%
1.7B	91.84%	95.83%	100.13%	95.01%	97.86%
4B	95.32%	96.73%	99.53%	97.90%	98.50%
8B	96.36%	97.08%	99.12%	97.43%	95.58%
14B	98.05%	100.15%	99.18%	98.05%	100.56%
32B	98.28%	97.43%	100.02%	99.99%	100.97%

A counter intuitive finding appears where quantized models (INT8, FP8, and GPTQ-8bit at certain scales) show accuracy above 100% of the FP16 baseline. This can be attributed to regularization effects of quantization noise, which can improve generalization on held-out benchmarks when models are in regimes where FP16 training slightly overfits to training distribution characteristics [84]. Full-precision models with excess capacity can memorize training-set idiosyncrasies, producing sharp, narrow minima that generalize poorly to new or shifted data. Quantization basically introduces controlled noise into weights and activations, which disrupts this fine-grained memorization and biases the optimizer toward flatter, wider minima [85]. The same principle applies to other regularization techniques like *dropout* or weight noise injection: a controlled degradation during training/inference prevents the network from over-relying on precise, sensitive weight configurations, trading a small amount of raw capacity for improved

robustness [86].

This regularization interpretation is consistent with the observation that the effect is largest at smaller model scales (0.6B INT8 at 100.94%) and disappears at larger scales where overparameterization provides enough redundancy.

For Metabench benchmark, AWQ, GPTQ-4bit and FP8 show scaling patterns where models are more resilient to quantization as they grow in size. Parameter redundancy at larger model sizes keeps the quantization error to a minimum. Interestingly, GPTQ-8bit and INT8 show better resiliency at smaller model sizes.

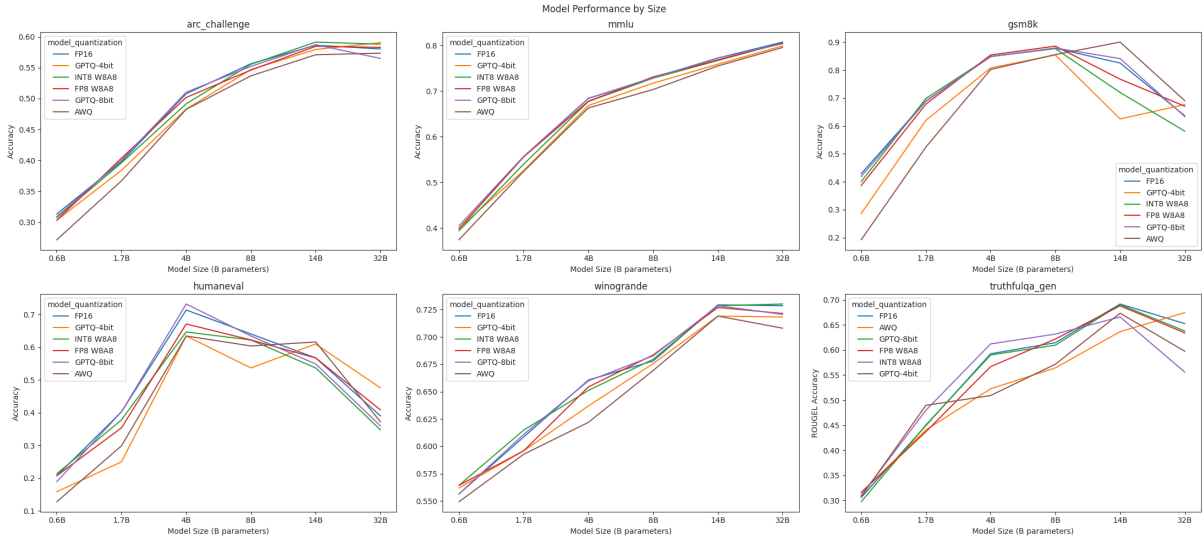


Figure 7.1: Model performance by size for 6 benchmarks - we evaluate zero-shot accuracy on ARC, MMLU, GSM8K, HumanEval, WinoGrande and TruthfulQA

Some specific tasks like coding or mathematical reasoning are more sensitive to quantization than others [59]. We can also see that here, mathematical reasoning tasks (GSM8K) show particular sensitivity to quantization effects. Smaller models experience pronounced degradation, while larger models, especially Qwen3-14B, have large discrepancies between quantization methods, AWQ even performing significantly better than the FP16 baseline.

Code generation capabilities (HumanEval) demonstrate similar scaling performance. The evaluation suggests that complex coding and mathematical abilities in large language models degrades even without quantization – possibly due to overthinking. The GPTQ-8bit and AWQ quantization methods even performing better than the FP16 baseline.

In language understanding tasks (MMLU, ARC Challenge and WinoGrande), quantization error and scaling seem to be as expected - with consistent errors which reduce and stabilize as we scale the model.

Factual accuracy (TruthfulQA) shows consistent degradation patterns across model sizes, suggesting that truthfulness preservation is more dependent on quantization method than model

scale.

To sum up, general understanding benchmarks (MMLU) scale approximately linearly with model size, while mathematical reasoning (GSM8K) and code generation (HumanEval) show different behavior with performance decreasing at some larger model sizes. This pattern motivates the test-time scaling experiments in Chapter 8.

7.3.1 Energy Efficiency Characteristics

Our experiments show that energy per output token varies primarily with model size ($8\times$ variation across 0.6B to 32B) while quantization method contributes a secondary factor (up to $1.7\times$ variation at fixed model size). This result tells us that the most impactful optimization decision is model size selection, with quantization as a complementary efficiency lever.

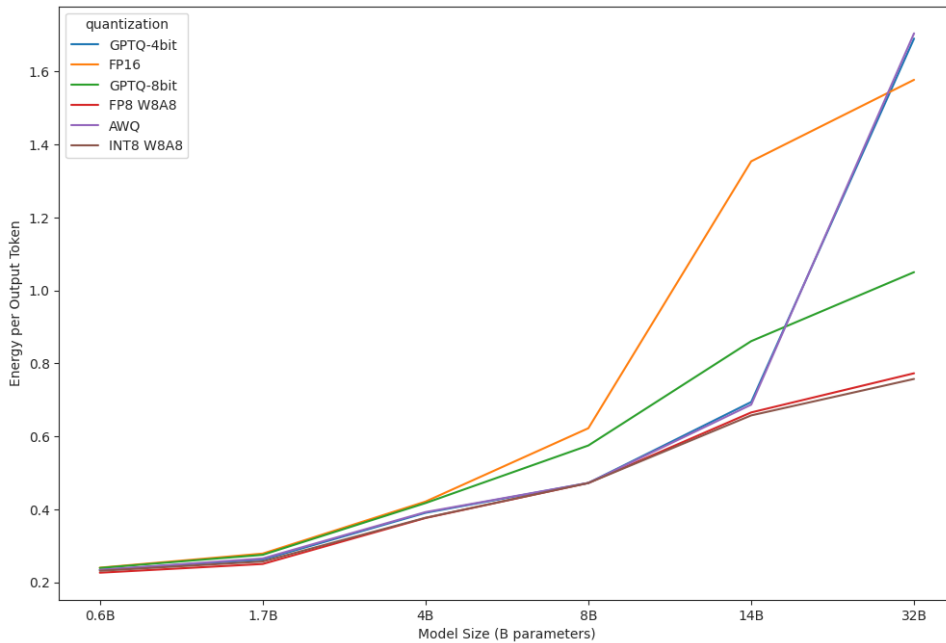


Figure 7.2: Energy per output token for across model sizes quantized with various quantization methods

The 32B model with GPTQ 4-bit quantization consumes 1.69 J per output token compared to 1.58 J for the FP16 baseline, while the FP8 quantized model consumes only 0.77 J, representing an **improvement of more than 2x in energy efficiency** for the same accuracy. INT8 and FP8 quantized models perform consistently better than other methods, and their energy efficiency improves on larger model sizes. This is mostly due to the native hardware support of INT8 and FP8 on modern GPUs which reduces not only the memory footprint, but also the amount of computations necessary.

The 32B model sizes quantized to FP16, GPTQ-4bit, GPTQ-8bit and AWQ had to be run on 2 GPUs. Although we ran experiments with various context lengths and output lengths, the

result were consistent to the ones shown in Figure 2. This improvement enables deployment of larger models within similar energy budgets as smaller unquantized models – where a 32B FP8 models energy requirements can be compared to a 8B unquantized models energy requirements. Input length is not significant for performance but important for hardware requirements: The addition of 512 **input** tokens increases latency less than the production of 8 additional **output** tokens in LLM models. However, the need to support long inputs can make models harder to serve.

Overall latency scales sub-linearly with model size, on the same hardware, larger models are slower, but the speed ratio won't necessarily match the parameter count ratio.

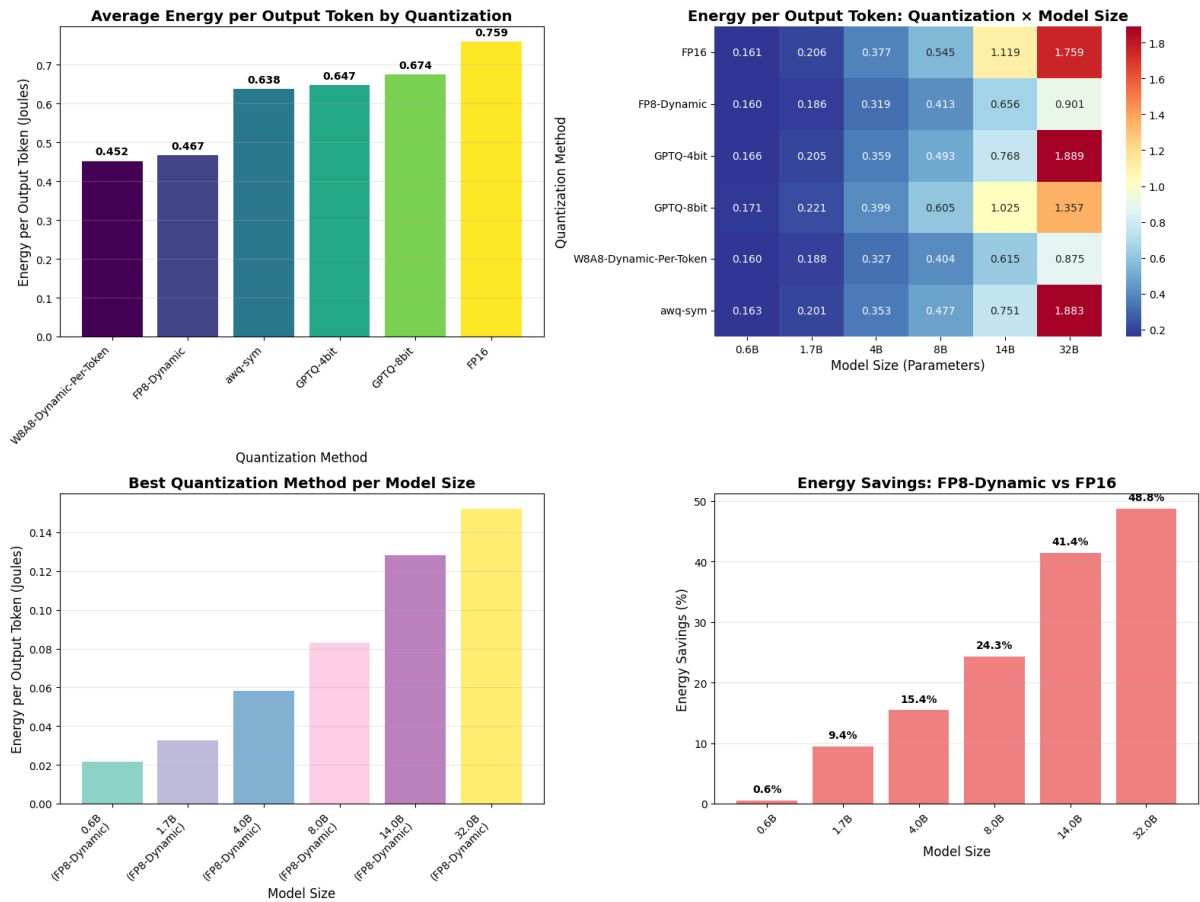


Figure 7.3: Energy per output token analysis for various model sizes using popular quantization methods.

The efficiency difference between best and worst quantization is only **1.7x**. This is much smaller than the 8.6x difference between model sizes, indicating that **model size selection** has far greater energy impact than quantization method.

INT8 and **FP8** consistently outperform across all model sizes, showing better scaling properties (5.5-5.6x energy increase vs 10.9-11.4x for others) and statistically significant improvements over FP16 (40.5% and 38.5% respectively)

Although we see INT8 has lower average energy per output token across all model sizes, FP8 dominates individual configurations, where achieves the best energy efficiency for every single model size. The efficiency **increases with model size**, showing that quantization becomes more critical for larger models.

Quantization method rankings for energy per output token efficiency:

1. **W8A8-Dynamic-Per-Token**: 0.452 J/token (most efficient)
2. **FP8-Dynamic**: 0.467 J/token (consistently best across all model sizes)
3. **AWQ-sym**: 0.638 J/token
4. **GPTQ-4bit**: 0.647 J/token
5. **GPTQ-8bit**: 0.674 J/token
6. **FP16**: 0.759 J/token (least efficient)

7.3.2 Throughput analysis

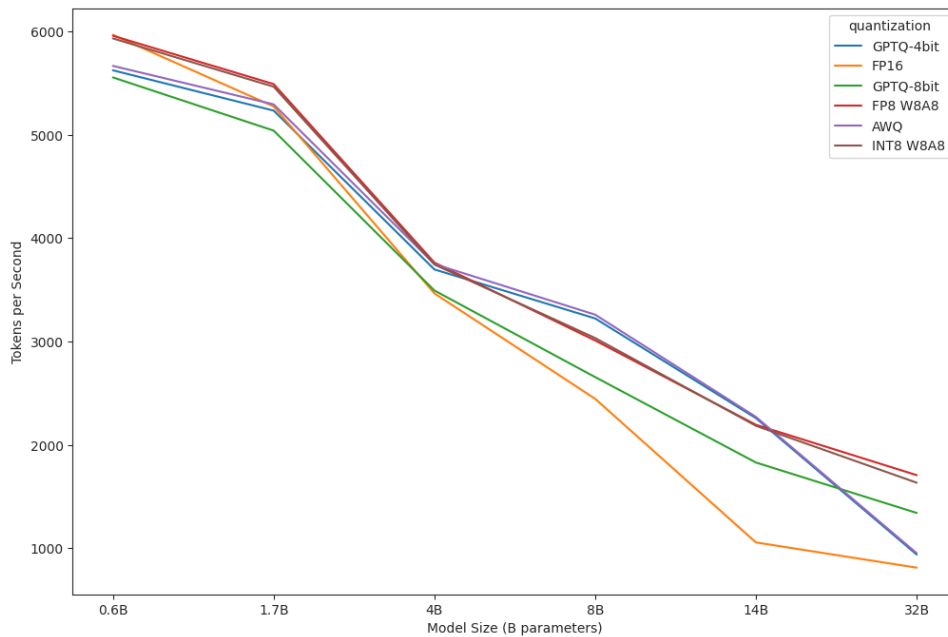


Figure 7.4: Tokens per second throughput across model sizes using different quantization methods for context length 4000 tokens and output length 1000 tokens

Inference throughput improvements scale predictably with quantization. Smaller model sizes perform well on modern GPUs in any format, while larger models show better throughput performance with quantization across all quantization methods.

FP8 and INT8 quantization provide approximately 2x throughput improvement on larger models compared to FP16 baseline across all context length. GPTQ-4bit and AWQ show almost identical throughput performance as expected, since both of them use 4-bit representation of weights with 16-bit activations.

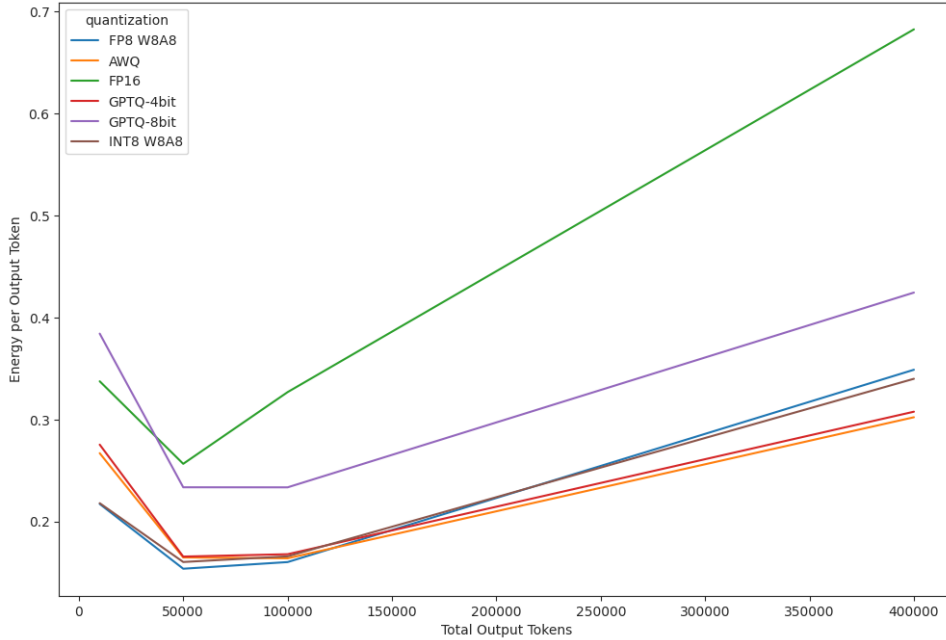


Figure 7.5: Energy per output token for Qwen3-14B at various output lengths and with various quantization methods

For a 14B Qwen3 model with input tokens set to 500, we see a strong correlation between inference energy and output token length - the more output tokens are generated - energy per output token rises - for all quantization models alike. However, the full precision model shows significantly more energy use per output token.

7.4 Input/Output Length Optimization

Input length strongly correlates with energy per output token ($r=0.607$), while output length shows weak negative correlation ($r=-0.077$). This suggests input processing has higher overhead than output generation. When processing input tokens, the transformer models must compute **self-attention across the entire input context simultaneously**. The computational complexity follows $O(T^2d + Td^2)$ where T is the sequence length - this is shown perfectly with the data from the 4B model, between input length and energy consumption. Each additional input token must attend to **all previous tokens**, creating a quadratic explosion in computation, consistent with the autoregressive decode bottleneck identified in Chapter 4..

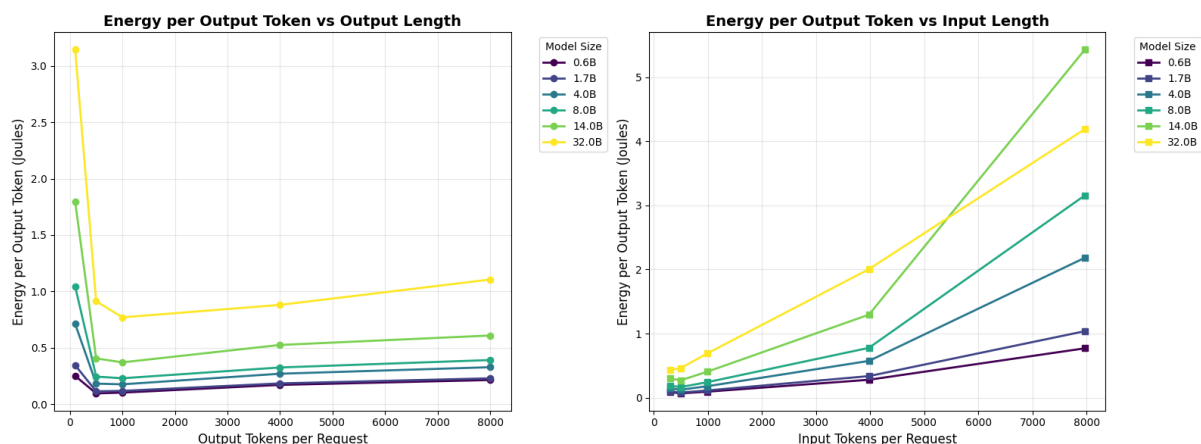


Figure 7.6: Energy per output token analysis for increasing output tokens(left) and input tokens(right)

The strong positive correlation ($r=0.889$) between input length and energy per output token in the 4B model data reveals that **input processing is the energy bottleneck**. Each additional input token adds $\sim 0.00032\text{J}$ per output token because the model must recompute attention weights across the expanded context and maintain larger key-value caches in memory.

This explains why the most efficient configuration uses minimal input (297 tokens) while output length has minimal impact on per-token efficiency. The transformer architecture inherently favors scenarios with concise prompts and longer generations for optimal energy efficiency.

Counter-intuitively, **longer outputs are more energy efficient per token**. Generating 100,000 tokens requires only 0.308 J/token , while 10,000-token outputs consume 1.216 J/token . This suggests significant overhead costs that amortize over longer generations. Fixed preprocessing costs (model loading, context encoding) are amortized and GPU resources are utilized more efficiently.

7.5 Practical Implications

The quantization tolerance of larger models fundamentally challenges conventional deployment strategies. Our findings suggest that deploying a quantized large model often provides better performance than deploying a smaller unquantized model within similar resource constraints.

This is a crucial distinction for production deployments: 4-bit methods fit larger models in VRAM but do not improve tokens-per-second throughput. For Arctur's inference service, this implies a tiered deployment strategy:

- Throughput-optimised tier (high-volume, latency-tolerant): INT8 or FP8 models on large parameter counts
- Memory-optimised tier (large model in constrained VRAM): GPTQ-4bit or AWQ

- Accuracy-optimised tier (quality-critical): GPTQ-8bit or FP8 with conservative component selection

GPTQ-8bit quantization emerges as the preferred method for applications requiring high accuracy preservation with moderate efficiency gains. GPTQ methods offer excellent compression ratios with good accuracy preservation, making them suitable for memory-constrained deployments. The GPTQ-4-bit variant has better accuracy across benchmarks than AWQ, while being on par with energy-efficiency and throughput.

4-bit quantization offers substantial **energy reduction on ARM cores**, but requires careful accuracy evaluation and optimized kernels that dequantize to 8-bit integers [87]. Our results show that **the dequantization overhead is significant** for inferencing 4-bit quantized models on modern GPUs, especially as we increase the number of parameters. **GPTQ-4bit performs worse than GPTQ-8bit in some cases**, suggesting that aggressive quantization doesn't always translate to energy efficiency.

FP8 and INT8 W8A8 provide maximum throughput improvements and maximum energy efficiency with simple implementation requirements, making it attractive for high-volume inference scenarios where slight accuracy degradation is acceptable. FP8 quantization can provide significant energy savings with minimal accuracy loss. The ability to deploy larger, more capable models within similar energy budgets as smaller unquantized models represents a paradigm shift in deployment strategy optimization. This finding suggests that quantization enables “scaling up” rather than “scaling down” for energy-constrained scenarios. For a balanced accuracy/performance trade-off, 4B or 8B models with FP8 quantization offer good efficiency ratios. If there is native hardware support, always use FP8 quantization, it's the most consistent winner across all model sizes. INT8 is slightly better on average but may have implementation complexity. For production environments, FP16 should be avoided, as it incurs a 40% energy penalty with minimal accuracy benefit.

The data clearly shows that smaller models with advanced quantization methods provide the most energy-efficient text generation, with diminishing returns as model size increases beyond 8B parameters.

For energy-conscious deployment:

1. **Prioritize small models:** 0.6B-4B range offers best energy/capability balance
2. **Optimize for longer outputs:** Batch multiple requests or generate longer sequences
3. **Minimize input overhead:** Keep prompts concise when possible
4. **Use advanced quantization:** FP8 consistently outperforms across all sizes

Chapter 8

Inference Scaling

This Chapter integrates the paper [Balancing Compute in LLM Inference: Model Selection, Quantization, and Test-Time Scaling \(EuroMLSys 2026\)](#).

8.1 Introduction

The preceding chapters have established fundamental relationships between model scale, quantization methods, and energy efficiency in large language models. Our analysis of scaling laws revealed how quantization effectiveness varies with model size, while energy-aware compression formats demonstrate significant computational savings without proportional accuracy degradation. Building upon these findings, this section explores a novel application of quantization-enabled compute savings: strategic reallocation to test-time scaling methods that enhance reasoning performance through increased inference computation.

Inference scaling represents a new approach from the traditional training-time compute allocation. Rather than investing computational resources in larger model parameters or extended training, inference (also called test-time) scaling allocates additional compute to improve solution quality for individual queries. This approach is particularly relevant given our previous findings that quantization can reduce inference costs while maintaining 95%+ accuracy, creating substantial computational headroom for more sophisticated reasoning processes.

Test-time scaling can be achieved with several approaches, each with distinct computational characteristics and effectiveness patterns. They are classified into sequential, where computations depend on previous tokens computed (like budget forcing - long thinking), and parallel, where multiple computations are run independently and the results are gathered at the end (like best-of-N).

These methods alter the inference economics established in our energy efficiency analysis. Traditional inference operates under fixed computational budgets determined by model architecture and quantization settings. Test-time scaling can introduce dynamic budget allocation,

where computational investment scales with problem difficulty rather than model capacity.

The integration of quantization savings with test-time scaling creates new optimization opportunities. The efficiency gains from quantization can be directly translated to additional computational capacity for parallel candidate generation or deeper reasoning chain exploration. Specifically, the bandwidth savings from quantizing a 32B parameter model enables generation of 3-5 times additional thinking tokens or parallel candidates within equivalent energy budgets.

8.2 Experimental Design

The test-time scaling experiments build directly on the quantization findings of Chapter 7. The experimental framework employs compute-equivalent comparisons across different allocation strategies: rather than simply comparing accuracy at fixed parameter counts, configurations are compared at equivalent energy budgets to assess whether the compute freed by quantization can be profitably reinvested into reasoning tokens.

Models evaluated: Qwen3-0.6B, 1.7B, 4B, 8B, 14B and 32B in FP16 and INT8

For evaluation we use the GSM8K mathematical reasoning benchmark in a zero-shot setting. Mathematical reasoning benefits most from chain-of-thought generation due to the multi-step nature of arithmetic and algebra. Factual retrieval tasks show minimal benefit from extended reasoning and are not the primary evaluation target.

We implement budget forcing through a custom logits processor that suppresses the end-of-thinking token until a minimum token threshold is reached, then forces termination at a maximum budget. A 70% utilization threshold prevents premature termination. Temperature is set to 0.3 with top-p 0.9 during reasoning; final answers use greedy decoding. Reasoning budgets of 500, 1000, 2000, and 4000 tokens are evaluated.

Since budget-forcing is a novel method, standard open-source frameworks or implementations do not exist, so we have to implement our own. The implementation consists of three core components working in sequence. We develop a new logits processor, which intercedes during each token generation step, receiving the model’s logit distribution and modifying it according to budget constraints. We employ a generation process with token-level control where the model can only terminate the thinking process if it stays between *min_tokens* and *max_tokens*.

We add thinking token tags (`<think>` and `</think>`), which serve as **explicit delimiters** that create a **structured reasoning phase** separate from the final answer generation. These are **standard vocabulary tokens**, not special model control tokens like `<|endoftext|>`. The model learns to recognize their semantic meaning through training data patterns.

A complete budget-forced prompt creates this token sequence:

Question: What is 15×23 ?

<think>

Step 1: Break down the multiplication...

Step 2: Calculate systematically...

...thinking process...

</think>

Final Answer: 345

To investigate how allocating explicit reasoning time affects problem-solving accuracy, each question is answered in two stages:

1. **Reasoning Phase:** The model is prompted with the problem and constrained to generate up to a fixed number of “thinking” tokens encapsulated within <think>...</think> tags. During this phase, the model produces detailed step-by-step derivations, forced to continue until a specified minimum token threshold and only allowed to conclude reasoning once it reaches the maximum token budget.
2. **Answer Phase:** The complete reasoning text is appended to the conversation under <think> tags, and the model is asked to output the final numerical answer in a second chat call. This two-step procedure ensures that all compute allocated to reasoning is explicitly consumed before answer generation.

Initial tests show that thinking improves accuracy on reasoning tasks, but models do not use the whole thinking budget out of the box, which leads to stagnation in accuracy performance.

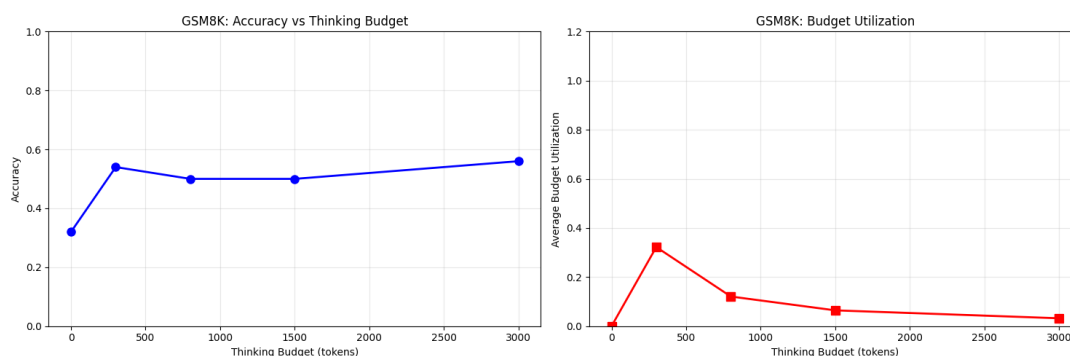


Figure 8.1: Qwen3 8B model performance on the GSM8K benchmark under various thinking budgets.

This research builds upon our quantization scaling law findings that showed smaller models maintain better quantization robustness. We predict this robustness extends to test-time scenarios, where quantized models may compound effectiveness more favorably.

8.3 Results: Budget Scaling Analysis

We choose the GSM8K mathematical reasoning benchmark, as thinking and reasoning characteristics mostly benefit such benchmark - and not factual knowledge benchmarks. We compare accuracy against a zero-shot baseline without explicit reasoning. The quantization evaluation done in previous sections were using a 5-shot setup, whereas here we use a zero-shot setting to have more control over the reasoning process.

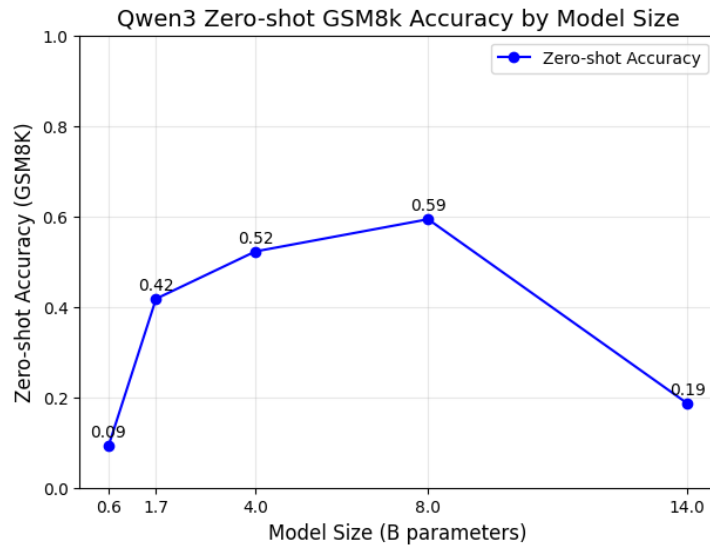


Figure 8.2: Qwen3 8B FP16 model zero-shot accuracy on the GSM8K benchmark sets a baseline for our evaluation. We see that the 14B model shows worse results than smaller models.

A range of reasoning budgets was tested: 500, 1000, 2000, and 4000 tokens - chosen to span from minimal to extensive reasoning time relative to typical GSM8K chain-of-thought lengths. For each budget, a minimum threshold of 70% of the budget was enforced using prompt engineering and *min_tokens* parameter before allowing early termination, which helped high utilization of allocated reasoning tokens.

For comparison, the model was also prompted directly for each answer without any `<think>` tags or reasoning constraints, using greedy decoding. This represents the **zero-shot accuracy** reflecting the model’s default problem-solving capability which can be seen in Figure 8.3.

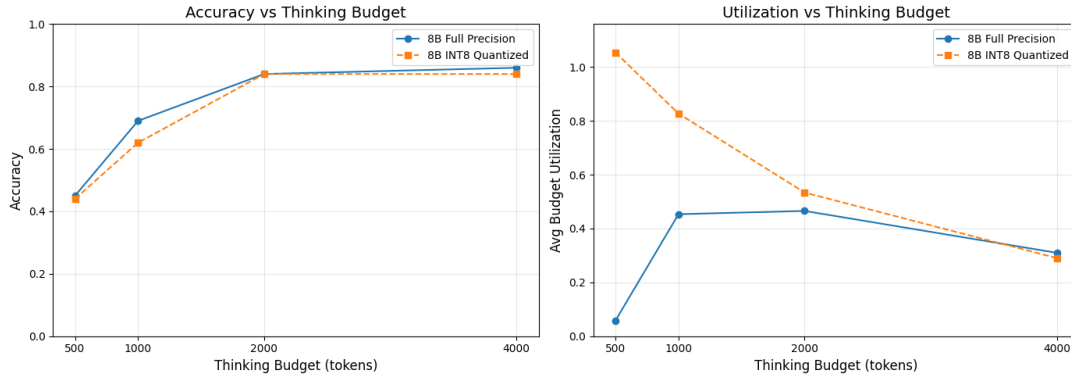


Figure 8.3: Qwen3 8B model performance on the GSM8K benchmark under various thinking budgets, without forced budget utilization.

Budget utilization is the fraction of the reasoning budget actually consumed, measured as the number of tokens generated between `<think>` and `</think>` divided by the maximum reasoning budget.

In Figure 8.4 we can see GSM8K accuracy under various reasoning budgets for different model sizes. The 4B model achieves the highest accuracy (93%) at 4000 tokens, outperforming the larger 14B model (82%). At moderate budgets (2000 tokens), the 4B model (90%) still exceeds the 14B model’s best result. The 14B model shows poor performance at low budgets (13% at 500 tokens), indicating that larger models may underperform when reasoning chains are limited. Even with strict budget forcing techniques, the LLMs tend to not utilize all the given budget, especially beyond the 2000 token mark. The largest accuracy gains occur in the step from 500 to 1000 reasoning tokens (e.g., +0.39 for 0.6B FP16, +0.43 for 14B FP16), while the 2000 to 4000 step yields at most +0.09. For most models, accuracy remains the same or even slightly regresses at budget=4000, suggesting that beyond 2000 tokens the model either produces redundant reasoning or it doesn’t utilize the whole budget. For small models (0.6B), INT8 matches or slightly exceeds FP16 at most budgets, indicating the precision reduction is within the model’s noise floor. For mid-range models (4B, 8B), INT8 reduces accuracy by 2-5%. The 32B model shows the opposite pattern where INT8 outperforms FP16, which can be attributed to regularization effects. These findings suggest that deployment strategies should prioritize medium-sized models with moderate reasoning budgets to achieve optimal energy efficiency without sacrificing mathematical reasoning performance, challenging the conventional wisdom that larger models are inherently superior for complex reasoning tasks.

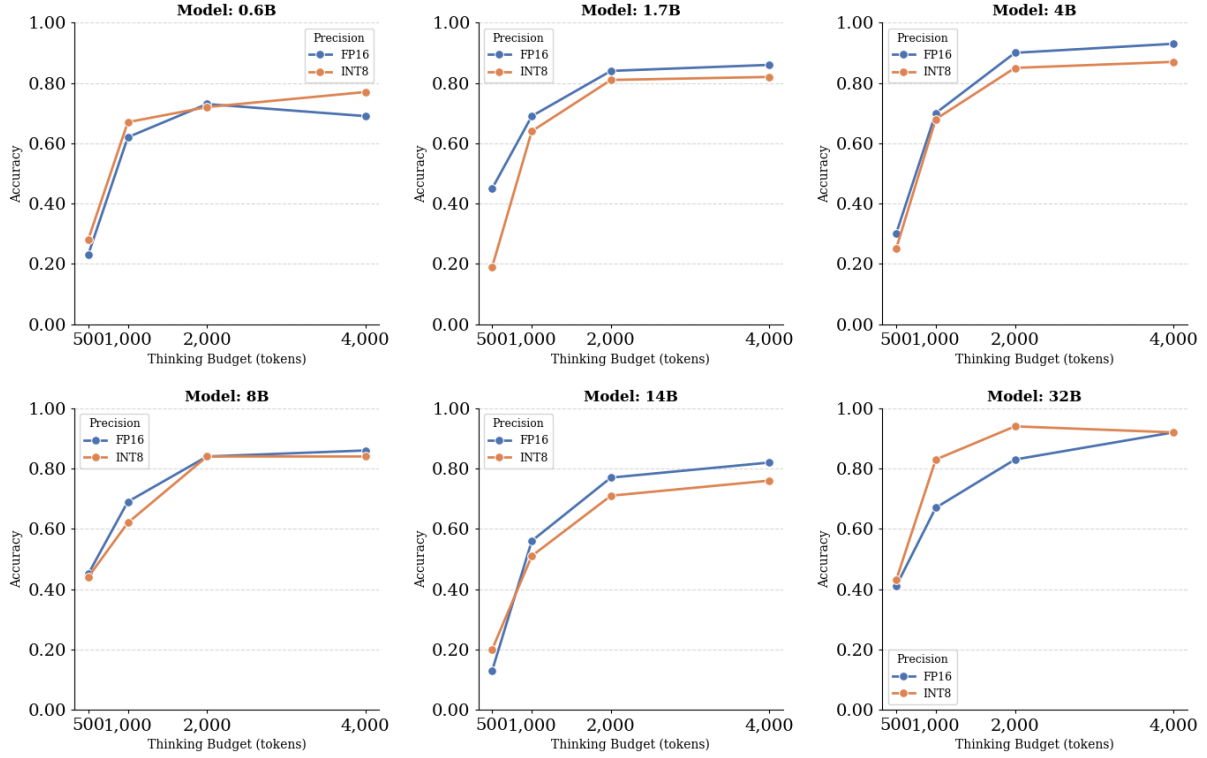


Figure 8.4: Reasoning budget scaling analysis - results for various model sizes using increasing reasoning budgets on the GSM8K benchmark.

These results reveal that **smaller models with extended reasoning budgets can achieve superior performance while requiring significantly less computational resources than larger models with limited reasoning**. This analysis examines the computational trade-offs between model size and inference-time reasoning budget across five model scales (0.6B to 32B parameters) on the GSM8K mathematical reasoning benchmark.

These results challenge the assumption that larger models are inherently superior. We demonstrate that the 4B model consistently outperforms all larger variants, achieving 93.0% accuracy on GSM8K with a 4000-token reasoning budget. This represents a 11 percentage point improvement over the 14B model (82.0% accuracy) while using 71% less GPU memory (8GB vs 28GB). The 4B model also outperforms the 8B model by 7 percentage points while requiring half the memory footprint. The energy cost per query is 204.5 J for the 4B model versus 518.7 J for the 14B model at maximum budget.

At moderate reasoning budgets (2000 tokens), the 4B model achieves 90.0% accuracy, which exceeds the performance of both 8B and 14B models even with their maximum reasoning budgets. This suggests that architectural efficiency and training quality matter more than raw parameter count for mathematical reasoning tasks.

Memory Efficiency: The analysis reveals substantial memory savings opportunities. To achieve approximately 80% accuracy, deployments can choose between a 14B model requiring

28GB of memory or a 1.7B model with extended reasoning requiring only 3.4GB - an $8.2\times$ reduction in memory usage. The smaller model actually achieves higher accuracy (86.0% vs 82.0%) while using dramatically fewer resources.

Parameter Efficiency: When targeting 70% accuracy, our results show that a 0.6B model with a 4000-token reasoning budget (69.0% accuracy, 1.2GB memory) can match a 14B quantized model with a 2000-token budget (71.0% accuracy, 14GB memory). This represents a 96% reduction in model parameters while requiring only 51% additional reasoning tokens.

Compute-Memory Trade-offs: The most efficient configuration for practical deployment is the 1.7B model with a 4000-token reasoning budget, achieving 86.0% accuracy with a computational efficiency of 0.506 accuracy points per billion parameters. This configuration provides 95% of the performance of the best 4B model while using 58% less memory.

As shown in previous sections, larger models use more energy, so each reasoning token is less efficient.

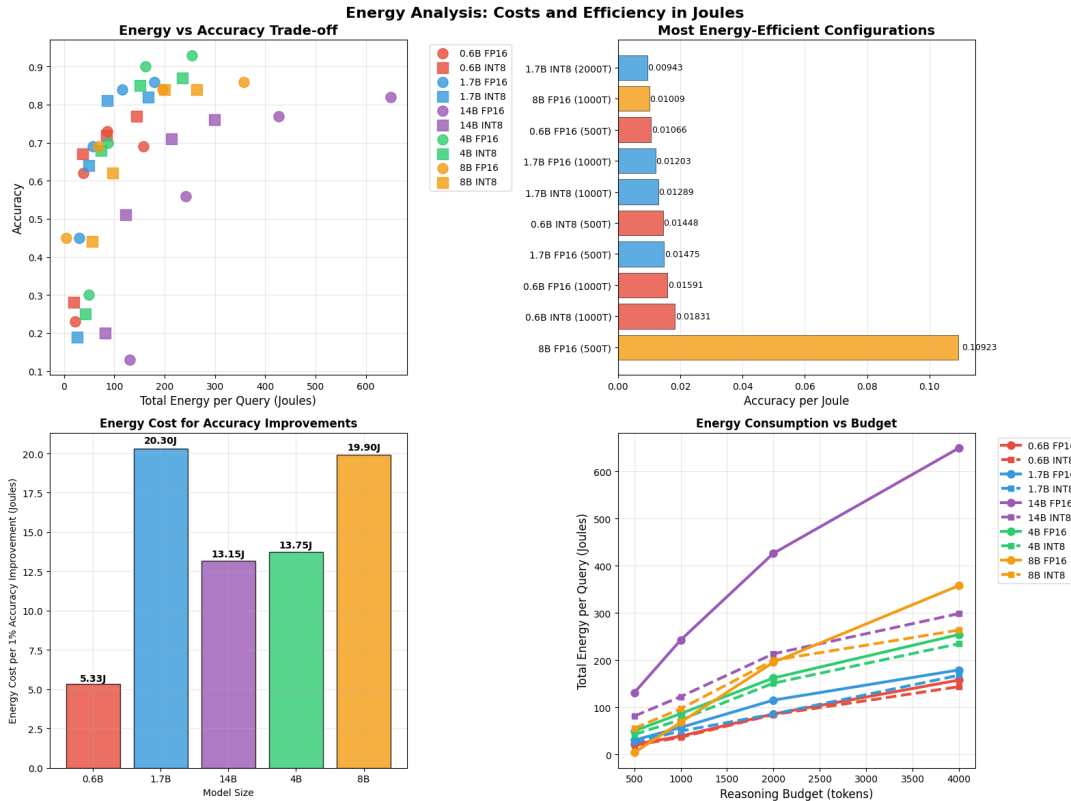


Figure 8.5: Energy-accuracy tradeoff at scaled model sizes.

We choose the INT8 quantization method for comparative analysis: this method has shown the best energy savings in our previous analysis, while also being suitable for heterogeneous hardware deployment.

Figure 8.5 illustrates how quantization and test-time scaling reshape the conventional scaling

intuition by flattening the Pareto frontier between accuracy and energy across model sizes. For each parameter scale, the INT8 configurations lie consistently to the left of their FP16 counterparts, meaning that at fixed accuracy they consume less energy per query, while at fixed energy budget they can reach higher GSM8K accuracy once the reasoning budget is increased. This effect is particularly pronounced for the 0.6B and 1.7B models, where INT8 both improves final accuracy and increases the accuracy-gain-per-Joule over FP16, confirming that compute freed by quantization can be profitably reinvested into longer chain-of-thought reasoning.

For deployment scenarios prioritizing memory efficiency, quantized smaller models with extended reasoning budgets offer better trade-offs. A 0.6B INT8 model with 4000 tokens achieves 77.0% accuracy using only 0.6GB of memory demonstrating that inference-time computation can compensate for reduced model capacity.

For deployments with severe memory limitations (less than 4GB), the optimal strategy is using sub-2B models with extended reasoning budgets. A 1.7B FP16 model with 2000-4000 token budgets achieves 84.0-86.0% accuracy while fitting within typical consumer GPU memory constraints.

When inference latency is critical, larger models with minimal reasoning budgets provide faster time-to-first-token. However, our results suggest this strategy sacrifices accuracy, the 14B model with a 500-token budget achieves only 13.0% accuracy, making it unsuitable for mathematical reasoning tasks.

The optimal configuration for balancing accuracy, memory usage, and reasoning cost is the 4B FP16 model with a 2000-token budget, achieving 90.0% accuracy with 8GB memory requirements and moderate reasoning overhead (1045 tokens per query).

These findings challenge the conventional scaling paradigm in language model deployment. Rather than deploying the largest possible model with minimal reasoning, we should prioritize deploying smaller, well-trained models with extended reasoning budgets. This approach can yield better task performance while dramatically reducing hardware and energy requirements, potentially making advanced reasoning capabilities accessible on consumer-grade hardware.

The results suggest that test-time scaling through increased reasoning budgets may be more sample-efficient than pre-training scaling for mathematical reasoning tasks. This has significant implications for both model development strategies and deployment architectures, particularly in resource-constrained environments where computational efficiency is paramount.

Table 8.1: Accuracy gains and energy efficiency across model sizes for FP16 and INT8 from 500 to 2000 tokens reasoning budget

Model	Precision	Acc Gain	Δ Energy (J)	Acc Gain / Δ J
0.6B	FP16	+0.50	0.0444	11.27
0.6B	INT8	+0.44	0.0464	9.49
1.7B	FP16	+0.39	0.0482	8.09
1.7B	INT8	+0.62	0.0294	21.07
4B	FP16	+0.60	0.0593	10.12
4B	INT8	+0.60	0.0621	9.66
8B	FP16	+0.39	0.0682	5.72
8B	INT8	+0.40	0.0818	4.89
14B	FP16	+0.64	0.2339	2.74
14B	INT8	+0.51	0.0925	5.51
32B	FP16	+0.42	0.4870	0.86
32B	INT8	+0.51	0.1430	3.57

Table 8.1 clearly shows that smaller quantized models provide the best energy efficiency for reasoning improvements, while the 4B model achieves optimal absolute performance. The most striking result is the 1.7B INT8 configuration, which achieves the highest efficiency ratio of 21.07 accuracy gain per Joule gaining 0.62 accuracy points for only 0.029 J of additional energy. This is a direct consequence of INT8’s tightly bounded energy scaling at low parameter counts, confirming H2: smaller quantized models demonstrate greater relative improvement from test-time scaling.

By analyzing the reasoning budgets and measuring accuracy, we characterize the trade-off between compute spent on explicit reasoning and problem-solving performance. The zero-shot baseline establishes the lower bound of accuracy with no enforced reasoning. Improvements above this baseline demonstrate the value of dedicating dedicated compute to structured, explicit chain-of-thought generation:

- At low budgets (500 tokens), we expect modest gains if the model’s inherent reasoning capacity suffices with brief chains.
- Mid-range budgets (1000-2000 tokens), where additional reasoning time yields smaller incremental accuracy improvements.
- High budgets (4000 tokens) test whether very extensive reasoning can recover solutions

on the most challenging problems, at the cost of increased compute and latency.

This methodology provides a way to evaluate how **inference-time compute allocation** via budget forcing can systematically enhance or plateau a model’s reasoning performance compared to its baseline zero-shot accuracy.

Our analysis suggests several key hypotheses regarding the interaction between model size and test-time scaling effectiveness. **Smaller models exhibit superior test-time scaling efficiency** due to their lower baseline computational requirements, enabling more aggressive budget forcing, but also parallel sampling and iterative refinement within fixed compute budgets. Conversely, **larger models require less additional reasoning time** to achieve equivalent performance levels, as their increased parametric knowledge reduces the computational search space required for accurate solutions.

The quantization-test-time scaling interaction introduces additional complexity, as different quantization strategies may exhibit varying compatibility with specific scaling approaches. This analysis contributes to the broader understanding of inference economics by quantifying how computational resource allocation decisions impact both accuracy and efficiency in practical deployment scenarios. The results will inform deployment strategies for production systems where the balance between model sophistication, inference efficiency, and solution quality determines overall system effectiveness and economic viability.

8.4 Adaptive thinking flow

We explore a novel approach - adaptive thinking, which allocates inference-time compute proportionally to problem difficulty, ensuring that simple queries consume minimal resources while complex tasks receive sufficient reasoning capacity. By first assessing a problem’s complexity and then mapping that rating to a token budget (e.g., 100-10 000 tokens), this approach dramatically lowers overall energy consumption compared to fixed-budget chains of thought: trivial arithmetic problems expend only a few hundred tokens, whereas multi-concept reasoning leverages larger budgets. This selective compute allocation not only reduces latency and power draw on GPU clusters but also preserves accuracy by preventing under-thinking on hard tasks. Moreover, demonstrating that pre-trained large language models can self-assess complexity and perform controlled, multi-step reasoning highlights their latent meta-reasoning abilities.

The Adaptive Thinking Budget Allocation algorithm addresses the challenge of efficiently allocating computational resources for reasoning tasks. Rather than using fixed token budgets for all problems, this approach dynamically adjusts the thinking time based on problem complexity.

The algorithm operates in five sequential phases. First, it performs complexity assessment by prompting the model to rate the mathematical problem on a scale from 1 to 10, where 1 represents basic arithmetic and 10 indicates highly complex multi-step reasoning problems. This self-assessment allows the model to evaluate the reasoning demand of each task.

Second, the algorithm maps the complexity score to a thinking budget using linear interpolation. Problems rated as complexity 1 receive 500 tokens for reasoning, while complexity 10 problems receive 4,000 tokens - to stay in line with maximum budget for the fixed budget forcing approach. This creates a proportional relationship between perceived difficulty and allocated computational resources.

Third, the reasoning generation phase uses the allocated budget to produce step-by-step mathematical reasoning. The model generates detailed explanations and calculations within the token constraints, using a higher temperature setting (0.7) to encourage diverse reasoning approaches.

Fourth, the algorithm enforces budget utilization to ensure efficient resource usage. If the model consumes less than 70% of its allocated thinking budget, additional verification text is appended to encourage more thorough reasoning. This prevents premature termination and maximizes the use of available computational resources.

Finally, the algorithm generates the final response by combining the original question with the complete reasoning chain, formatted within thinking tags. The model then produces a concise final answer using deterministic generation (temperature 0.0) to ensure consistency.

This adaptive approach offers several advantages over fixed budgeting strategies. Simple problems we would ideally avoid unnecessary computational overhead, while complex problems receive sufficient resources for thorough analysis. The self-assessment mechanism enables automatic scaling without manual parameter tuning for different problem types. Additionally, the utilization enforcement ensures that allocated resources are fully utilized, leading to more comprehensive reasoning outputs.

Algorithm 1: Adaptive Thinking Budget Allocation

Input: Query prompt q , model M

Output: Final response r , complexity c , budget b , utilization u

Phase 1: Complexity Assessment;
 $prompt_c \leftarrow \text{"Rate complexity 1-10: " } + q$;
 $c \leftarrow \text{extract_complexity}(M(prompt_c))$;
if $c \notin [1, 10]$ **then**
 $c \leftarrow 5$;
end

Phase 2: Budget Allocation;
 $b \leftarrow \lfloor 500 + (c - 1) \times \frac{4000}{9} \rfloor$;

Phase 3: Reasoning Generation;
 $reasoning \leftarrow M(\text{system_prompt} \parallel q, b, \tau = 0.7)$;
 $used \leftarrow \text{count_tokens}(reasoning)$;

Phase 4: Utilization Enforcement;
 $u \leftarrow used/b$;
if $u < 0.7$ **then**
 $reasoning \leftarrow reasoning \parallel \text{padding}$;
end

Phase 5: Final Answer;
 $r \leftarrow M(q \parallel \text{"<think>" } \parallel reasoning \parallel \text{"</think>"})$;
return r, c, b, u ;

The algorithm returns four key outputs: the final response, the complexity assessment, the allocated budget, and the utilization rate. These metrics enable analysis of the relationship between problem complexity, resource allocation, and reasoning quality in mathematical problem-solving tasks.

Table 8.2: Fixed vs adaptive budget for Qwen3-4B on the GSM8K dataset

Strategy	Accuracy (%)	Avg. Tokens
Fixed 2000 tokens	90.0	1810
Fixed 4000 tokens	93.0	3420
Adaptive (500–4000)	91.0	1950

Table 8.2 shows that adaptive budgeting achieves 91% accuracy with an average budget of 1992 tokens - close to the 93% of the fixed 4000-token budget while using 43% fewer reasoning

tokens on average. Simple arithmetic problems consume 500–800 tokens, while more complex multi-step problems use the full budget. This reduces average latency and energy consumption per query without requiring external classifiers or additional model calls. Truly adaptive compute allocation remains difficult to implement reliably in current LLMs. Recent work on token-budget-aware reasoning confirms our findings that models often fail to respect tight length constraints: when prompted with an extremely small budget, actual thinking traces can become even longer than under more generous limits, a phenomenon termed *token elasticity* that breaks simple proportional budget control [88]. Survey results and empirical studies on adaptive and controllable test-time compute suggest that estimating instance difficulty on-the-fly is noisy and error prone, which our results confirm, so dynamic policies frequently misallocate compute by overthinking easy queries and underthinking genuinely hard ones [89]. Even methods that explicitly train models to be budget-aware through control tokens, reinforcement learning, or external budget guidance predictors improve adherence only in expectation and typically require additional engineering and tuning effort [90]. As a consequence, deployed systems rarely achieve strict budget utilization close to 100% across heterogeneous workloads.

Across model sizes, the results also show that there is no linear relationship between energy and accuracy once test-time scaling is enabled. Medium-sized models such as Qwen3-4B seem to have an optimal configuration where extended reasoning budgets push them to near-peak accuracy while consuming less energy than much larger FP16 models with shorter budgets. The smallest INT8 models with long budgets, approach or match the accuracy of the largest FP16 configuration at a fraction of its energy cost, echoing recent FLOPs-aware analyses that treat parameter count and test-time compute as exchangeable levers on the cost–accuracy.

8.5 Hypothesis Evaluation

H1 confirmed: The 4B INT8 model with 2000-token budget (90% accuracy, ~192 J) outperforms the 14B FP16 model with low reasoning budget (13%-82% accuracy, 518 J baseline) at $2.7\times$ lower energy. Within equivalent energy requirements, quantized models combined with test-time scaling consistently achieve superior accuracy per Joule compared to larger unquantized single-pass models.

H2 confirmed: Smaller quantized models show greater relative improvement from test-time scaling. The 1.7B INT8 model achieves 21.07 accuracy gain per Joule (budget 500→2000), compared to 2.74 for 14B FP16 - a $7.7\times$ difference in efficiency. The 0.6B model improves from 23% (FP16 zero-shot) to 77% (INT8, 4000 tokens) which is a 54-percentage-point gain that no larger model approaches in relative terms within the same compute budget.

The finding is that model downsizing combined with extended reasoning budgets provides orders of magnitude greater energy savings than quantization alone. Quantization serves as a valuable complementary technique that can provide additional efficiency gains once the optimal model size is identified. The deployment paradigm shift this implies: scale down models, scale up reasoning rather than deploying the largest available model.

Chapter 9

Efficient Production Inference

The experimental findings of Chapters 7-8 and the state-of-the-art overview in previous chapters establish the theoretically optimal configurations for AI inference. This chapter addresses how those configurations are put into practice in a production inference service at Arctur, drawing on the inference engineering principles documented in the earlier chapters. Arctur follows a policy of hosting everything in-house: relying on third parties, especially those based in other countries, removes guarantees around data control, availability, and long-term cost.

A production inference stack consists of three layers:

1. The runtime layer, responsible for maximizing the performance of individual model instances on one or more GPUs.
2. The infrastructure layer, managing scaling, routing, and capacity across multiple GPU nodes.
3. The developer experience layer, providing the API surface and monitoring tools that users interact with.

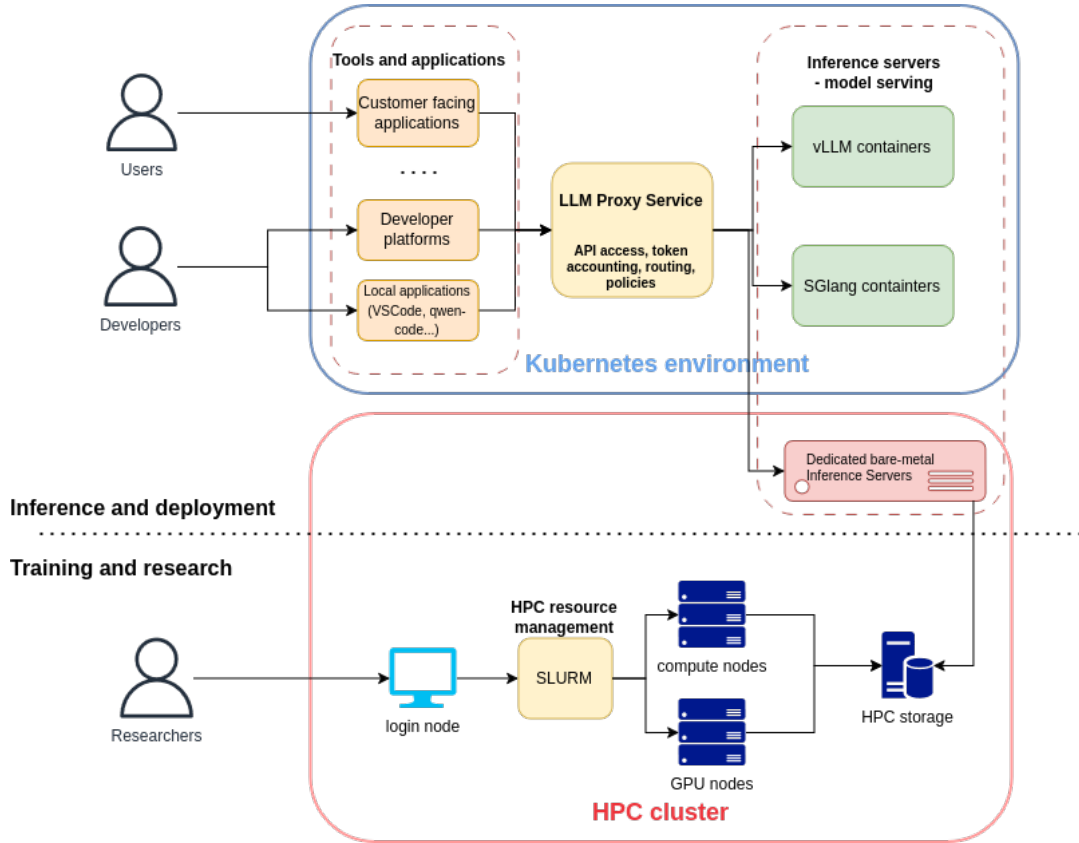


Figure 9.1: Simplified diagram of Arctur's AI Infrastructure

Arctur also runs a separate training and research environment, used by researchers, data scientists, and machine learning engineers to build datasets and data pipelines, train models, and produce research outputs. This environment is set up as a classical SLURM cluster with powerful CPU-only compute nodes alongside heterogeneous GPU nodes. The GPU nodes include both NVIDIA GPUs and specialized MACx AI accelerators. A high-performance storage system supports data processing and stores models and outputs. Nothing in this environment is virtualized; everything runs on bare metal to achieve maximum performance with minimal overhead.

The inference and deployment environment gives users and software developers efficient access to AI tools and services, while allowing Arctur to manage them efficiently. It supports user-facing applications as well as developer-facing platforms.

The inference serving stack combines several inference engines and routers, including vLLM, SGLang, and a small number of Ollama containers. Different models perform best on different engines, and each engine offers specific features required by certain applications. Dedicated bare-metal servers equipped with specialized AI accelerators run optimized models to meet high-throughput, low-latency requirements. These accelerators natively support certain quantization formats. For every new model, Arctur runs the benchmarking pipeline established in this thesis

to determine the most energy-efficient model configuration that meets the requirements.

An LLM proxy service allows Arctur to manage these various inference engines through a single OpenAI-compatible API endpoint. It handles access control via API keys, usage accounting, and model routing (directing requests across multiple instances of the same model behind one endpoint). This configuration supports automatic quantization-format detection, continuous batching, PagedAttention-based KV cache management, Prefill-decode disaggregation, and speculative decoding, all of which are directly relevant to the efficiency findings presented in this thesis.

All containerized applications run inside a Kubernetes environment, which provides high availability, redundancy, and simplified scaling and resource management. Each layer and each stack presents distinct optimization challenges.

9.1 Deployment framework

We propose a framework that guides through essential evaluation phases before deploying large language models in production environments, enabling optimal trade-offs between energy consumption, computational resources, and performance requirements:

Step 1: Resource Assessment and Constraint Analysis requires practitioners to benchmark target hardware capabilities, measure baseline energy consumption patterns, and establish hard constraints for memory utilization, latency requirements, and energy budgets, ensuring deployment feasibility within operational limits before model selection.

Step 2: Configuration Exploration systematically evaluates candidate model configurations across three primary dimensions: model size, quantization precision, and reasoning budget allocation through controlled experimentation that measures accuracy, energy consumption per query, memory footprint, and inference latency for each configuration combination. Our empirical validation demonstrates energy savings potential ranging from 60-73% when smaller quantized models with extended reasoning budgets replace larger alternatives while maintaining equivalent accuracy.

Step 3: Performance-Energy Trade-off Analysis implements cost-benefit evaluation using the optimization function:

$$TotalCost = \alpha \times Energy + \beta \times Latency + \gamma \times Memory + \delta \times Accuracy_Loss$$

where deployment-specific weight parameters ($\alpha + \beta + \gamma + \delta = 1$) reflect organizational priorities and enable objective comparison across configuration alternatives based on measured performance metrics rather than theoretical estimates.

Step 4: Workload-Adaptive Resource Allocation Strategy develops dynamic scal-

ing policies that adjust reasoning budgets, quantization levels, or model selection based on query complexity assessment, enabling systems to minimize energy consumption during low-complexity tasks while maintaining performance for demanding queries through intelligent resource escalation.

Step 5: Validation and Continuous Optimization establishes monitoring frameworks for production deployment that track energy efficiency metrics, performance degradation, and resource utilization patterns, enabling data-driven refinement of configuration choices and early detection of deployment inefficiencies.

This framework provides a practical methodology for organizations to achieve substantial energy savings through evidence-based deployment optimization rather than intuition-driven model selection.

We have shown that we can achieve consistent energy savings using quantization:

- **FP8 quantization** can achieve up to **2x energy improvement** (32B model: 1.69J FP16 vs 0.77J FP8 per token)
- **INT8 quantization** can achieve **10-15% average energy savings** across model sizes.

32B FP8 model achieves energy efficiency comparable to 8B unquantized FP16 model.

Model size selection has **8x greater energy impact** than quantization method choice (only 1.7× difference between best/worst quantization methods).

We have found that we can achieve substantial **energy savings through strategic model downsizing**, for example:

- **73% energy reduction:** 1.7B INT8 @ 4000T (141.7J, 82% accuracy) vs 14B FP16 @ 4000T (518.7J, 82% accuracy)
- **2.7x less energy:** 4B FP16 @ 2000T (162.3J, 90% accuracy) vs 14B model (582.4J, 82% accuracy)
- **8.2x memory reduction:** 1.7B model (3.4GB, 86% accuracy) vs 14B model (28GB, 82% accuracy)

Practical deployment depends on scenarios with specific computational resource availability:

- **Edge devices:** 0.6B INT8 + 1000T budget = 67% accuracy, 36.6J per query
- **Workstations:** 1.7B FP16 + 2000T budget = 84% accuracy, 148.9J per query
- **Production:** 4B FP16 + 2000T budget = 90% accuracy, 162.3J per query

Reasoning budget scaling follows diminishing returns - increasing from 500 to 2000 tokens provides substantial gains (+44 percentage points for 0.6B model), but 2000 to 4000 tokens yields minimal improvement at higher cost.

The **counterintuitive finding** that 4B models can outperform 14B models (93% vs 82% accuracy) while using 71% less memory suggests that **architectural efficiency and training quality matter more than raw parameter count**.

Inference-time computation can effectively substitute for parameter scaling, enabling:

1. **60-73% energy savings** through strategic model downsizing
2. **Quantization provides additional 10-40% energy reduction** on top of downsizing benefits
3. **Extended reasoning budgets compensate for reduced model capacity** effectively

The optimization strategy with biggest impact is **model downsizing combined with extended reasoning budgets**, which provides **orders of magnitude greater energy savings** than quantization alone. Quantization serves as a valuable complementary technique that can provide additional efficiency gains once optimal model size is selected.

Practitioners should “**scale down models and scale up reasoning**” rather than deploy the largest available model - a paradigm shift with major implications for sustainable AI deployment.

Chapter 10

Conclusion

This thesis addresses one of the most critical challenges facing the AI industry today: the sustainable deployment of large language models in resource-constrained environments. Through four complementary research approaches: scaling law analysis, computational optimization, empirical quantization studies, and adaptive test-time scaling, we have demonstrated that efficient inference is a critical problem for AI inference providers and other AI practitioners to understand, and that it is essential for the democratization of AI technology.

Our research reveals several fundamental insights that enhance the understanding of optimal model deployment strategies. Most significantly, we deepened our understanding of scaling laws, demonstrating that compute can be reallocated between training and inference. This enables practitioners to train smaller models for longer periods to achieve dramatically reduced inference costs, particularly valuable for high-volume deployment scenarios where inference demands dominate operational expenses.

The comprehensive energy efficiency analysis across the Qwen3 model family (0.6B to 32B parameters) provides empirical evidence that quantized models can achieve more than $2\times$ energy efficiency improvement while maintaining performance. Perhaps most practically significant, we demonstrated that larger models (32B) with FP8 quantization can match the energy requirements of smaller unquantized models (8B), fundamentally altering the cost-benefit calculus for model selection.

Our activation distribution analysis confirms that transformer models exhibit power-law patterns in neuron activation, providing foundations for targeted quantization strategies. This insight enables more sophisticated approaches to mixed-precision deployment, where critical “hot” neurons retain higher precision while “cold” neurons can be aggressively quantized without performance degradation.

The test-time scaling findings reveal that inference-time compute can serve as an effective substitute for model scale. Rather than deploying larger models for all tasks,

organizations can now use smaller, more efficient base models and selectively apply additional compute only when problem complexity demands it. This approach fundamentally changes the economics of AI deployment by enabling dynamic resource allocation that matches computational investment to task difficulty, rather than over-provisioning for worst-case scenarios.

These findings converge to provide clear guidance for optimal model selection and deployment strategies. For organizations facing high inference demand, scaling laws suggest training smaller, more efficient models for extended periods rather than pursuing maximum parameter counts. The energy efficiency results demonstrate that quantization is no longer a compromise but a strategic advantage, enabling deployment of larger models within limited energy budgets. Organizations should calculate optimal configurations based on expected inference volumes and available hardware: a model anticipating 10^{13} inference requests should use a smaller parameter model trained on $2.84\times$ more data to achieve 28% total FLOP reduction. **Combined with adaptive test-time scaling, this strategy enables deployment of highly efficient base models that can dynamically scale performance when needed, optimizing both baseline efficiency and peak capability.**

For edge deployment scenarios, our findings indicate that aggressive quantization methods like INT8 enable sophisticated models to operate on resource-constrained hardware while maintaining acceptable performance.

Energy-aware design principles must be integrated into every stage of model development, from initial scaling decisions through quantization strategy selection to deployment configuration. The findings enable organizations to make data-driven decisions about model selection, quantization methods, and inference-time scaling strategies based on specific use cases and constraints.

Future research opportunities emerge at the intersection of these optimization strategies. Energy-aware architecture design represents a fundamental research opportunity that extends beyond quantization to model architecture itself. Future architectures specifically designed for efficient inference could integrate the energy-accuracy trade-offs we have characterized into their fundamental computational patterns, **while also incorporating native support for adaptive thinking time allocation and complexity-aware compute scaling.**

These research directions directly enable companies to pursue optimized AI deployment strategies while maintaining sustainability objectives. The quantified trade-offs between training investment and inference efficiency provide clear frameworks for organizations to optimize their AI infrastructure investments. **The demonstration that test-time scaling can sub-**

stitute for model scale enables us to deploy smaller, more sustainable base models while maintaining the capability to handle complex tasks through adaptive compute allocation.

The sustainable deployment of large language models is not only feasible but presents opportunities for improved performance and reduced operational costs. **The integration of scaling law principles, energy-aware optimization, advanced quantization techniques, and adaptive test-time scaling provides a comprehensive framework for efficient AI deployment that balances capability, sustainability, and accessibility.**

The path forward requires continued integration of efficiency considerations into every aspect of AI development, from fundamental architecture design through deployment optimization. **The frameworks established here, provide the analytical guidelines necessary for navigating this evolving landscape, enabling both research advancement and practical deployment success in the era of applied artificial intelligence.** The demonstration that smaller models can achieve competitive performance through intelligent inference-time compute allocation fundamentally changes the scaling paradigm from “bigger is better” to “smarter allocation is better,” opening new pathways for sustainable and accessible AI deployment.

References

- [1] C.-J. Wu *et al.*, “Sustainable AI: Environmental Implications, Challenges and Opportunities.” arXiv, Jan. 09, 2022. doi: 10.48550/arXiv.2111.00364.
- [2] S. Hooker, “The hardware lottery,” *Commun. ACM*, vol. 64, no. 12, pp. 58–65, Nov. 2021, doi: 10.1145/3467017.
- [3] U. Gupta *et al.*, “Chasing Carbon: The Elusive Environmental Footprint of Computing.” arXiv, Oct. 28, 2020. doi: 10.48550/arXiv.2011.02839.
- [4] M. Riedl, “AI Democratization in the Era of GPT-3,” *The Gradient*, 2020. <https://thegradient.pub/ai-democratization-in-the-era-of-gpt-3/> (accessed July 19, 2023).
- [5] G. Guidi *et al.*, “Environmental Burden of United States Data Centers in the Artificial Intelligence Era.” arXiv, Nov. 14, 2024. doi: 10.48550/arXiv.2411.09786.
- [6] “Global Energy Review 2025 – Analysis,” *IEA*, Mar. 24, 2025. <https://www.iea.org/reports/global-energy-review-2025> (accessed Sept. 10, 2025).
- [7] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, “Compute and Energy Consumption Trends in Deep Learning Inference,” *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, Apr. 2023, doi: 10.1016/j.suscom.2023.100857.
- [8] E. J. Husom, A. Goknil, L. K. Shar, and S. Sen, “The Price of Prompting: Profiling Energy Use in Large Language Models Inference.” arXiv, July 04, 2024. doi: 10.48550/arXiv.2407.16893.
- [9] A. Montero Fernández *et al.*, “A catalyst for European cloud services in the era of data spaces, high-performance and edge computing: NOUS,” in *Proceedings of the 4th Eclipse Security, AI, Architecture and Modelling Conference on Data Space*, Nov. 2024, pp. 1–9. doi: 10.1145/3685651.3686660.
- [10] “Investing in Cloud, Edge and the Internet of Things | Shaping Europe’s digital future.” <https://digital-strategy.ec.europa.eu/en/policies/iot-investing> (accessed Sept. 29, 2025).

-
- [11] J. Hoffmann *et al.*, “Training Compute-Optimal Large Language Models.” arXiv, Mar. 29, 2022. doi: 10.48550/arXiv.2203.15556.
 - [12] J. Kaplan *et al.*, “Scaling Laws for Neural Language Models.” arXiv, Jan. 22, 2020. doi: 10.48550/arXiv.2001.08361.
 - [13] N. Sardana, J. Portes, S. Doubov, and J. Frankle, “Beyond Chinchilla-Optimal: Accounting for Inference in Language Model Scaling Laws.” arXiv, Apr. 14, 2025. doi: 10.48550/arXiv.2401.00448.
 - [14] C. Snell, J. Lee, K. Xu, and A. Kumar, “Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters.” arXiv, Aug. 06, 2024. doi: 10.48550/arXiv.2408.03314.
 - [15] Y. Wu, Z. Sun, S. Li, S. Welleck, and Y. Yang, “Inference Scaling Laws: An Empirical Analysis of Compute-Optimal Inference for Problem-Solving with Language Models.” arXiv, Mar. 03, 2025. doi: 10.48550/arXiv.2408.00724.
 - [16] M. Chen *et al.*, “Parallel Scaling Law for Language Models.” arXiv, May 15, 2025. doi: 10.48550/arXiv.2505.10475.
 - [17] N. Muennighoff *et al.*, “s1: Simple test-time scaling.” arXiv, Feb. 03, 2025. doi: 10.48550/arXiv.2501.19393.
 - [18] Q. Chen *et al.*, “Lins: Reducing Communication Overhead of ZeRO for Efficient LLM Training,” in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*, June 2024, pp. 1–10. doi: 10.1109/IWQoS61813.2024.10682856.
 - [19] N. Thompson, K. Greenewald, K. Lee, and G. F. Manso, “The Computational Limits of Deep Learning,” June 2023. doi: 10.21428/bf6fb269.1f033948.
 - [20] “A blueprint for building national compute capacity for artificial intelligence,” OECD Publishing, Paris, OECD Digital Economy Papers 350, Feb. 2023. Accessed: July 09, 2025. [Online]. Available: https://www.oecd.org/en/publications/a-blueprint-for-building-national-compute-capacity-for-artificial-intelligence_876367e3-en.html
 - [21] A. Agrawal *et al.*, “Medha: Efficiently Serving Multi-Million Context Length LLM Inference Requests Without Approximations.” arXiv, June 18, 2025. doi: 10.48550/arXiv.2409.17264.
 - [22] S. Smith *et al.*, “Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model.” arXiv, Feb. 04, 2022. doi: 10.48550/arXiv.2201.11990.

-
- [23] Y. Sheng *et al.*, “S-LoRA: Serving Thousands of Concurrent LoRA Adapters.” arXiv, June 05, 2024. doi: 10.48550/arXiv.2311.03285.
 - [24] B. Zhao *et al.*, “FairKV: Balancing Per-Head KV Cache for Fast Multi-GPU Inference.” arXiv, May 17, 2025. doi: 10.48550/arXiv.2502.15804.
 - [25] Y. Wei *et al.*, “Communication-Efficient Model Parallelism for Distributed In-Situ Transformer Inference,” in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2024, pp. 1–6. doi: 10.23919/DATE58400.2024.10546617.
 - [26] J. Du *et al.*, “Co-Designing Transformer Architectures for Distributed Inference With Low Communication,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 4, pp. 717–730, Apr. 2025, doi: 10.1109/TPDS.2024.3521582.
 - [27] H.-B. Kim, D. Hoang, A. Kundu, M. Samragh, and M. Cho, “SPD: Sync-Point Drop for Efficient Tensor Parallelism of Large Language Models.” arXiv, June 01, 2025. doi: 10.48550/arXiv.2502.20727.
 - [28] D. Narayanan *et al.*, “PipeDream: Generalized pipeline parallelism for DNN training,” Oct. 2019. Available: <https://www.microsoft.com/en-us/research/publication/pipedream-generalized-pipeline-parallelism-for-dnn-training/>
 - [29] Y. Huang *et al.*, “GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism.” arXiv, July 25, 2019. doi: 10.48550/arXiv.1811.06965.
 - [30] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, “Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey.” arXiv, Sept. 16, 2024. doi: 10.48550/arXiv.2403.14608.
 - [31] A. Vaswani *et al.*, “Attention Is All You Need.” arXiv, Dec. 05, 2017. doi: 10.48550/arXiv.1706.03762.
 - [32] M. Adnan, A. Arunkumar, G. Jain, P. J. Nair, I. Soloveychik, and P. Kamath, “Keyformer: KV Cache Reduction through Key Tokens Selection for Efficient Generative Inference.” arXiv, Apr. 06, 2024. doi: 10.48550/arXiv.2403.09054.
 - [33] Y. Dong, Y. Miao, W. Li, X. Zheng, C. Wang, and F. Lyu, “Accelerating LLM Inference Throughput via Asynchronous KV Cache Prefetching.” arXiv, Apr. 08, 2025. doi: 10.48550/arXiv.2504.06319.
 - [34] M. Federici *et al.*, “Efficient LLM Inference using Dynamic Input Pruning and Cache-Aware Masking.” arXiv, Apr. 03, 2025. doi: 10.48550/arXiv.2412.01380.
 - [35] Y. Xiong *et al.*, “LayerKV: Optimizing Large Language Model Serving with Layer-wise KV Cache Management.” arXiv, Oct. 09, 2024. doi: 10.48550/arXiv.2410.00428.

-
- [36] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness.” arXiv, June 23, 2022. doi: 10.48550/arXiv.2205.14135.
 - [37] M. Hashemi *et al.*, “Learning Memory Access Patterns.” arXiv, Mar. 06, 2018. doi: 10.48550/arXiv.1803.02329.
 - [38] L. Wang, C. Wang, Y. Li, and R. Wang, “Explaining the Behavior of Neuron Activations in Deep Neural Networks,” *Ad Hoc Networks*, vol. 111, p. 102346, Feb. 2021, doi: 10.1016/j.adhoc.2020.102346.
 - [39] M. Yu, D. Wang, Q. Shan, C. Reed, and A. Wan, “The Super Weight in Large Language Models.” arXiv, Nov. 11, 2024. doi: 10.48550/arXiv.2411.07191.
 - [40] J. Fernando and G. Guitchounts, “Transformer Dynamics: A neuroscientific approach to interpretability of large language models.” arXiv, Feb. 17, 2025. doi: 10.48550/arXiv.2502.12131.
 - [41] S. Nadipalli, “Layer-Wise Evolution of Representations in Fine-Tuned Transformers: Insights from Sparse AutoEncoders.” arXiv, Feb. 23, 2025. doi: 10.48550/arXiv.2502.16722.
 - [42] Y. Arai and Y. Ichikawa, “Quantization Error Propagation: Revisiting Layer-Wise Post-Training Quantization.” arXiv, Jan. 13, 2026. doi: 10.48550/arXiv.2504.09629.
 - [43] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding.” arXiv, Feb. 15, 2016. doi: 10.48550/arXiv.1510.00149.
 - [44] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, “AI and Memory Wall.” arXiv, Mar. 21, 2024. doi: 10.48550/arXiv.2403.14123.
 - [45] “ml-energy/zeus.” ML.ENERGY, Sept. 09, 2025. Accessed: Sept. 18, 2025. [Online]. Available: <https://github.com/ml-energy/zeus>
 - [46] S. Lin, J. Hilton, and O. Evans, “TruthfulQA: Measuring How Models Mimic Human Falsehoods.” arXiv, May 08, 2022. doi: 10.48550/arXiv.2109.07958.
 - [47] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both Weights and Connections for Efficient Neural Networks.” arXiv, Oct. 30, 2015. doi: 10.48550/arXiv.1506.02626.
 - [48] J. Frankle and M. Carbin, “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks.” arXiv, Mar. 04, 2019. doi: 10.48550/arXiv.1803.03635.
 - [49] T. D. K, T. Fischer, and C. Biemann, “Large Language Models Are Overparameterized Text Encoders.” arXiv, Oct. 18, 2024. doi: 10.48550/arXiv.2410.14578.

-
- [50] H. Gao, F. Zhang, W. Jiang, J. Shu, F. Zheng, and H. Wei, “On the Noise Robustness of In-Context Learning for Text Generation.” arXiv, Oct. 24, 2024. doi: 10.48550/arXiv.2405.17264.
 - [51] Z. Zhang *et al.*, “Does your LLM truly unlearn? An embarrassingly simple approach to recover unlearned knowledge.” arXiv, Oct. 21, 2024. doi: 10.48550/arXiv.2410.16454.
 - [52] E. Frantar and D. Alistarh, “SparseGPT: massive language models can be accurately pruned in one-shot,” in *Proceedings of the 40th International Conference on Machine Learning*, July 2023, vol. 202, pp. 10323–10337.
 - [53] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, “LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale.” Nov. 10, 2022. doi: 10.48550/arXiv.2208.07339.
 - [54] E. Frantar, S. Ashkboos, T. Hoefer, and D. Alistarh, “GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers.” arXiv, Mar. 22, 2023. doi: 10.48550/arXiv.2210.17323.
 - [55] J. Hasan, “Optimizing Large Language Models through Quantization: A Comparative Analysis of PTQ and QAT Techniques.” arXiv, Nov. 09, 2024. doi: 10.48550/arXiv.2411.06084.
 - [56] X. Wu, C. Li, R. Y. Aminabadi, Z. Yao, and Y. He, “Understanding INT4 Quantization for Transformer Models: Latency Speedup, Composability, and Failure Cases.” arXiv, May 30, 2023. doi: 10.48550/arXiv.2301.12017.
 - [57] E. Frantar, R. L. Castro, J. Chen, T. Hoefer, and D. Alistarh, “MARLIN: Mixed-Precision Auto-Regressive Parallel Inference on Large Language Models.” arXiv, Aug. 21, 2024. doi: 10.48550/arXiv.2408.11743.
 - [58] T. Kumar *et al.*, “Scaling Laws for Precision.” arXiv, Nov. 30, 2024. doi: 10.48550/arXiv.2411.04330.
 - [59] J. Lee, S. Park, J. Kwon, J. Oh, and Y. Kwon, “Exploring the Trade-Offs: Quantization Methods, Task Difficulty, and Model Size in Large Language Models From Edge to Giant.” arXiv, June 04, 2025. doi: 10.48550/arXiv.2409.11055.
 - [60] Z. Liu *et al.*, “ParetoQ: Scaling Laws in Extremely Low-bit LLM Quantization.” arXiv, Feb. 04, 2025. doi: 10.48550/arXiv.2502.02631.
 - [61] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, “SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models.” arXiv, Mar. 29, 2024. doi: 10.48550/arXiv.2211.10438.
 - [62] J. Lin *et al.*, “AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration.” arXiv, July 18, 2024. doi: 10.48550/arXiv.2306.00978.

-
- [63] M. van Baalen *et al.*, “GPTVQ: The Blessing of Dimensionality for LLM Quantization,” *arXiv.org*, Feb. 23, 2024. <https://arxiv.org/abs/2402.15319v2> (accessed June 08, 2025).
 - [64] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, and D. Alistarh, “Extreme Compression of Large Language Models via Additive Quantization,” *arXiv.org*, Jan. 11, 2024. <https://arxiv.org/abs/2401.06118v4> (accessed June 08, 2025).
 - [65] S. Ma *et al.*, “The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits.” *arXiv*, Feb. 27, 2024. doi: 10.48550/arXiv.2402.17764.
 - [66] S. Luo, M. Liu, P. Sun, Y. Yu, S. Ren, and Y. Bai, “An Adaptive Tensor-Train Decomposition Approach for Efficient Deep Neural Network Compression,” *arXiv.org*, Aug. 02, 2024. <https://arxiv.org/abs/2408.01534v1> (accessed June 15, 2025).
 - [67] T. B. Brown *et al.*, “Language Models are Few-Shot Learners.” *arXiv*, July 22, 2020. doi: 10.48550/arXiv.2005.14165.
 - [68] S. Min *et al.*, “Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?” *arXiv*, Oct. 20, 2022. doi: 10.48550/arXiv.2202.12837.
 - [69] N. Elhage *et al.*, “A mathematical framework for transformer circuits,” *Transformer Circuits Thread*, 2021.
 - [70] C. Olsson *et al.*, “In-context learning and induction heads,” *Transformer Circuits Thread*, 2022.
 - [71] Z. Liu *et al.*, “Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time.” *arXiv*, Oct. 26, 2023. doi: 10.48550/arXiv.2310.17157.
 - [72] Z. Li *et al.*, “Quantization Meets Reasoning: Exploring LLM Low-Bit Quantization Degradation for Mathematical Reasoning.” *arXiv*, Feb. 19, 2025. doi: 10.48550/arXiv.2501.03035.
 - [73] Y. Zhong *et al.*, “DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving,” 2024, pp. 193–210. Accessed: Aug. 11, 2026. [Online]. Available: <https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin>
 - [74] “Mix-Quant: Quantized Prefilling, Precise Decoding for Agentic LLMs.” <https://arxiv.org/html/2605.20315v1> (accessed Aug. 11, 2026).
 - [75] T. Dettmers and L. Zettlemoyer, “The case for 4-bit precision: k-bit Inference Scaling Laws.” *arXiv*, Feb. 28, 2023. doi: 10.48550/arXiv.2212.09720.

-
- [76] J. Klhufek, M. Safar, V. Mrazek, Z. Vasicek, and L. Sekanina, “Exploring Quantization and Mapping Synergy in Hardware-Aware Deep Neural Network Accelerators,” in *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*, Apr. 2024, pp. 1–6. doi: 10.1109/DDECS60919.2024.10508920.
 - [77] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, “Green AI.” arXiv, Aug. 13, 2019. doi: 10.48550/arXiv.1907.10597.
 - [78] J. You, J.-W. Chung, and M. Chowdhury, “Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training,” 2023, pp. 119–139. Accessed: Sept. 18, 2025. [Online]. Available: <https://www.usenix.org/conference/nsdi23/presentation/you>
 - [79] L. Sutawika *et al.*, “EleutherAI/lm-evaluation-harness: v0.4.3.” Zenodo, July 01, 2024. doi: 10.5281/zenodo.12608602.
 - [80] T. Ilijaš, T. Pahor, and T. Šubić, “Innovative Expansion of HPC Infrastructure for Scalable AI Inference Using MACx GPUs,” Apr. 2026. doi: 10.25365/phaidra.765.
 - [81] M. van Baalen *et al.*, “FP8 versus INT8 for efficient deep learning inference.” arXiv, June 15, 2023. doi: 10.48550/arXiv.2303.17951.
 - [82] Red Hat AI and vLLM Project, “LLM Compressor.” Aug. 2024. Accessed: Aug. 31, 2025. [Online]. Available: <https://github.com/vllm-project/llm-compressor>
 - [83] T. Gregersen, P. Patel, and E. Choukse, “Input-Dependent Power Usage in GPUs.” arXiv, Sept. 26, 2024. doi: 10.48550/arXiv.2409.18324.
 - [84] Y. Fu, Q. Yu, M. Li, V. Chandra, and Y. Lin, “Double-Win Quant: Aggressively Winning Robustness of Quantized Deep Neural Networks via Random Precision Training and Inference,” in *Proceedings of the 38th International Conference on Machine Learning*, July 2021, pp. 3492–3504. Accessed: June 22, 2026. [Online]. Available: <https://proceedings.mlr.press/v139/fu21c.html>
 - [85] S. Javed, H. Le, and M. Salzmann, “QT-DoG: Quantization-Aware Training for Domain Generalization,” in *Proceedings of the 42nd International Conference on Machine Learning*, Oct. 2025, pp. 26981–27004. Accessed: Aug. 09, 2026. [Online]. Available: <https://proceedings.mlr.press/v267/javed25a.html>
 - [86] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014, Accessed: Aug. 09, 2026. [Online]. Available: <http://jmlr.org/papers/v15/srivastava14a.html>

-
- [87] D. Gope, D. Mansell, D. Loh, and I. Bratt, “Highly Optimized Kernels and Fine-Grained Codebooks for LLM Inference on Arm CPUs.” arXiv, Dec. 23, 2024. doi: 10.48550/arXiv.2501.00032.
 - [88] T. Han, Z. Wang, C. Fang, S. Zhao, S. Ma, and Z. Chen, “Token-Budget-Aware LLM Reasoning,” in *Findings of the Association for Computational Linguistics: ACL 2025*, July 2025, pp. 24842–24855. doi: 10.18653/v1/2025.findings-acl.1274.
 - [89] M. A. Alomrani *et al.*, “Reasoning on a Budget: A Survey of Adaptive and Controllable Test-Time Compute in LLMs.” arXiv, July 02, 2025. doi: 10.48550/arXiv.2507.02076.
 - [90] H. Wen *et al.*, “BudgetThinker: Empowering Budget-aware LLM Reasoning with Control Tokens.” arXiv, Aug. 29, 2025. doi: 10.48550/arXiv.2508.17196.