

Overview of Memory-Efficient Architectures for Deep Learning in Real-Time Systems [†]

Bilgin Demir ¹, Ervin Domazet ^{1,*} and Daniela Meckaroska ²

¹ Department of Computer Engineering, International Balkan University, 1000 Skopje, North Macedonia; bilgin.demir@ibu.edu.mk

² Faculty of Computer Science & Engineering, University of Information Science and Technology “St. Paul the Apostle”, 6000 Ohrid, North Macedonia; daniela.meckaroska@uist.edu.mk

* Correspondence: ervin.domazet@ibu.edu.mk

[†] Presented at the International Conference on Electronics, Engineering Physics and Earth Science (EEPES 2025), Alexandroupolis, Greece, 18–20 June 2025.

Abstract

With advancements in artificial intelligence (AI), deep learning (DL) has become crucial for real-time data analytics in areas like autonomous driving, healthcare, and predictive maintenance; however, its computational and memory demands often exceed the capabilities of low-end devices. This paper explores optimizing deep learning architectures for memory efficiency to enable real-time computation in low-power designs. Strategies include model compression, quantization, and efficient network designs. Techniques such as eliminating unnecessary parameters, sparse representations, and optimized data handling significantly enhance system performance. The design addresses cache utilization, memory hierarchies, and data movement, reducing latency and energy use. By comparing memory management methods, this study highlights dynamic pruning and adaptive compression as effective solutions for improving efficiency and performance. These findings guide the development of accurate, power-efficient deep learning systems for real-time applications, unlocking new possibilities for edge and embedded AI.

Keywords: deep learning; memory-efficient architectures; real-time data processing; low-power devices; quantization; parameter pruning



Academic Editor: Grigor Mihaylov

Published: 4 September 2025

Citation: Demir, B.; Domazet, E.; Meckaroska, D. Overview of Memory-Efficient Architectures for Deep Learning in Real-Time Systems. *Eng. Proc.* **2025**, *104*, 77. <https://doi.org/10.3390/engproc2025104077>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The rise of artificial intelligence (AI) in important sectors like self-driving cars, telemedicine, and maintenance forecasting emphasizes the need for processing data in real time at the highest level. Deep learning (DL) is one subfield of AI that has shown amazing results in various difficult tasks. Despite these capabilities, DL models face significant computational and memory limitations when deployed on low-end devices and real-time systems.

This study looks at how progress in DL architecture can help overcome these limitations. It emphasizes model compression, quantization, and improved data utilization as memory-efficient design techniques. By synthesizing important and influential works, this paper discusses perspectives for further research aimed at improving the efficiency-accuracy ratio of the models in restricted environments. In this regard, Section 2 discusses the approaches and techniques, whereas the results are listed in Section 3. Section 4 provides the implications and discussions of the optimization methods. The paper is concluded in Section 5 with proper future considerations.

Recent extensive reviews like Kaur et al. [1] consider the Processing-in-Memory (PIM) architectures as one of the most important innovations for improving memory usage in deep learning frameworks. By placing processing elements in proximity to memory, PIM mitigates the data movement bottleneck, latency, and energy use. This method works synergistically with model compression and quantization techniques described in this paper, boosting on-the-fly deep learning inference in low-power devices.

Several references in this paper are to ArXiv preprints due to the rapidly evolving nature of memory-efficient deep learning architectures and techniques. Many significant innovations and recent advances appear initially in preprint repositories prior to peer-reviewed publication. Wherever possible, we have cited peer-reviewed versions; preprints are used primarily where peer-reviewed versions are not yet available, reflecting the cutting-edge and fast-moving developments within this research area.

2. Approaches and Techniques

To understand the trending research areas related to memory-efficient architectures, the approaches of some of the key works that were developed were reviewed. These studies put into action methods such as:

2.1. Model Compression Techniques

Methods to reduce redundant parameters, such as those introduced by Han et al. in “Deep Compression” [2], significantly decrease model size, enabling neural networks to efficiently deploy on resource-constrained devices. Other notable contributions include:

- Iandola et al. [3] contribute an approach to building compact deep learning models, claiming that simpler, smaller networks can be as accurate as their larger counterparts. This work achieves AlexNet level accuracy with greatly reduced parameters and a smaller model size.
- Zhang et al. [4] introduced a pipeline that combines tensor decomposition and pruning to optimize deep learning models for efficient inference, with an emphasis on model compression. Deng et al. [5] describe how model compression and hardware acceleration greatly increase the availability of deep learning models for constrained devices. This approach is a way to optimize both inference speed and memory consumption.

Model compression optimizes deep learning models by eliminating functional redundancies. In Han et al.’s work [2], “Deep Compression,” three strategies—pruning, quantization, and Huffman coding—are used to remove unnecessary connections, reducing parameters and overall memory footprint.

Han et al. [2] first pruned weights by removing insignificant connections, then applied quantization to reduce weight precision with lower-bit representations. Finally, they used Huffman coding to improve data storage efficiency.

The results were impressive: AlexNet’s size decreased by 35 times, VGG-16 by 49 times, all with minimal accuracy loss. These compressed models also showed better inference times and lower energy consumption, making them ideal for edge devices with limited resources.

2.2. Quantization Strategies

Quantization, the process of converting weights and activations into lower precision formats, is analyzed comprehensively by Gholami et al. [6]. To decrease the computational demands and the memory requirements of a model, quantization is used to reduce the precision of weights and activations in a model. In addition to the investigation by Gholami et al. [6], the works listed below provide supporting evidence:

- Wang et al.'s work [7] presents an automated framework for mixed precision quantization to optimize models for specific hardware constraints;
- Balaskas et al.'s study [8] delves into joint pruning and quantization techniques tailored to hardware configurations to improve efficiency.

Quantization enhances computational efficiency and reduces memory usage by lowering precision for model weights and activations. Gholami et al. [6] provide a detailed analysis of quantization strategies and their effects on model performance.

They classify algorithms by bit-width (e.g., 8-bit, 4-bit) and whether they use uniform or non-uniform quantization. The study highlights mixed-precision techniques, where critical layers operate at higher precision while less critical ones use lower precision, maintaining overall efficiency.

The survey shows quantization minimally impacts model accuracy for edge and embedded systems while achieving up to 4 times the memory reduction and faster inference. It emphasizes the importance of selecting methods suited to specific applications and hardware.

2.3. Hardware-Aware Optimization Methods

Incorporating hardware constraints into architecture design is vital, as demonstrated by Cai et al.'s *"ProxylessNAS"* [9]. Considering hardware constraints during neural network design ensures optimal performance on specific platforms. Complementing Cai et al.'s *"ProxylessNAS"* [9], the following studies contribute to this area:

- Cai et al. [10] presents a versatile network that can be easily adaptable to various hardware platforms with little or no change in the network architecture, i.e., without retraining.
- Krieger et al.'s research [11] offers an automated compression framework that considers hardware-specific components to tune and optimize neural networks.

Hardware-aware optimization involves designing deep learning models to align with specific hardware constraints. According to *"ProxylessNAS"* by Cai et al. [9], neural architecture search (NAS) frameworks optimize network architectures directly for target hardware without relying on proxy models.

"ProxylessNAS" uses a gradient-based search algorithm to tailor architectures based on hardware factors like latency, energy use, and memory. By evaluating architectures directly on target hardware, it ensures practical relevance.

The results demonstrate that ProxylessNAS-generated architectures outperform manually designed models in latency and efficiency while maintaining high accuracy. This approach is especially effective for resource-constrained edge devices.

2.4. Sparse Representations and Pruning Approaches

Techniques for reducing network complexity by removing less important parameters are effectively demonstrated by Han et al. [12]. There are techniques, including pruning, which focus on eliminating these parameters to enhance efficiency. Building on the framework by Han et al. [12], subsequent studies have also made contributions to this area:

- To reduce the size of a neural network while maintaining its accuracy, the paper by Tung et al. [13] combines pruning and quantization in a unified framework for efficient network compression.
- To reduce the size of models for deployment on edge devices, the study by Kim et al. [14] combines pruning, quantization, and knowledge distillation.

Sparse representations and pruning reduce the size and complexity of neural networks by removing less important parameters. Han et al. [12] in “Efficient Inference Engine (EIE)” showed how sparse representations optimize compressed networks for better output during inference with minimal memory and energy usage.

The EIE system used a sparsity-focused network and hardware accelerators to optimize matrix compression, supporting pruning. Dedicated inference engines processed the matrices for improved efficiency.

Experimental results showed a 13-fold increase in inference speed and a 3-fold reduction in energy usage compared to dense models. These improvements suggest the feasibility of using sparse representations for real-time deep learning on resource-limited devices.

This paper focuses on synthesizing these methods to identify key themes and potential future innovations in real-time DL systems.

3. Findings and Observations

This section summarizes empirical evidence and comparative outcomes from methods presented earlier, demonstrating their practical impact on memory efficiency, inference latency, and energy consumption in real-world scenarios.

Overall, these results illustrate how each technique addresses computational and memory challenges in resource-constrained DL deployments. By compromising between different resource usage and accuracy of the model, these techniques provide a way to design architectures appropriate for use in healthcare, autonomous systems, and predictive maintenance applications.

A complete picture of the possibilities of these approaches in augmenting system performance while still considering the heavy requirements of real-time applications is presented in this section.

3.1. Empirical Results of Model Compression

Compression techniques are commonly used to optimize DL models for specific tasks. Han et al. [2] reduced the size of a neural network by 35 times without sacrificing accuracy through pruning, quantization, and Huffman coding (Figure 1). Results show compression reduces memory requirements and improves inference speed. Future focus will likely be on automating the compression process to adjust on-the-fly based on workload needs.

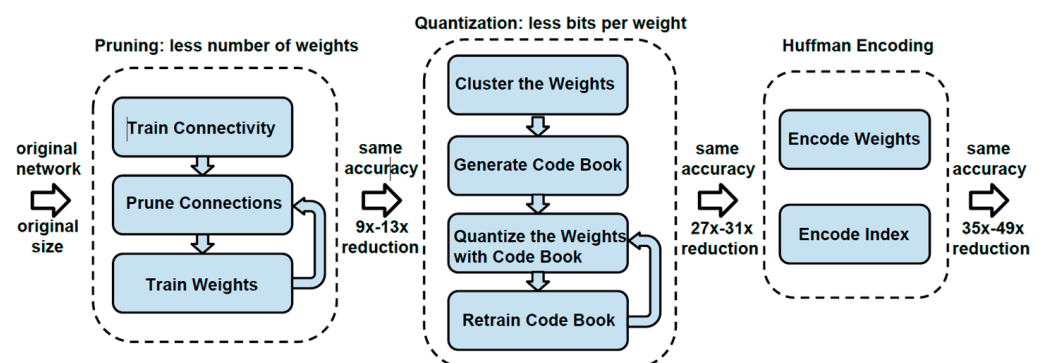


Figure 1. Neural network reduction without losing accuracy.

3.2. Quantization Outcomes on Edge Devices

The methods of quantization systematized by Gholami et al. [6] appear to be promising for low-power and edge devices. Quantization is the process of reducing the precision of parameters, for example from 32-bit to 8-bit, which greatly reduces the memory and computational requirements. Figure 2 illustrates a quantization policy

for MobileNet-V1 under latency constraints, highlighting how edge accelerators prioritize memory-bounded depth wise convolutions, while cloud accelerators optimize for computation-bounded tasks [7].

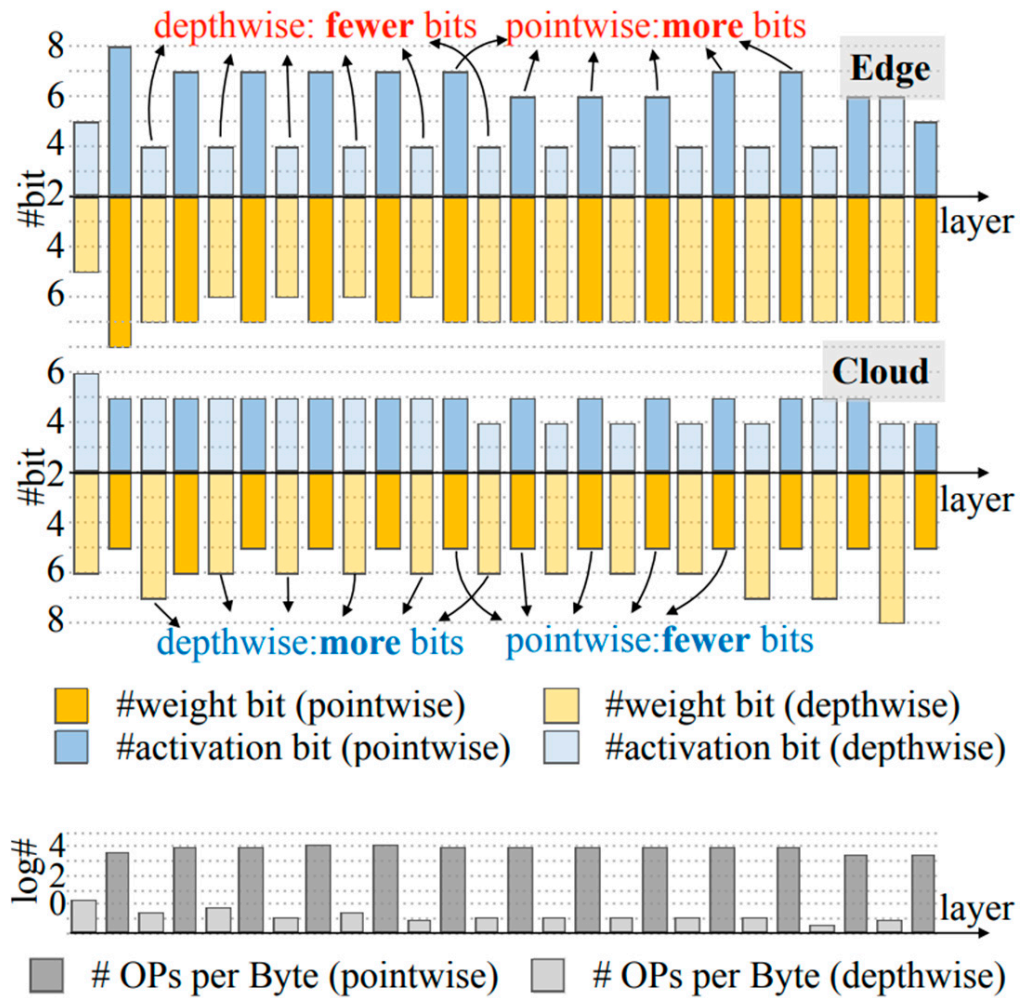


Figure 2. Quantization policy for MobileNet-V1.

Additionally, mixed precision quantization, where layers may adopt different precision levels, opens opportunities for further optimization; however, balancing accuracy and computational savings requires more research and innovation in the future.

3.3. Hardware-Aware Optimization Findings

Another approach is hardware-aware NAS as proposed in “ProxylessNAS” by Cai et al. [9], in which models can be designed to fit specific hardware limitations. As illustrated in Figure 3, ProxylessNAS consistently outperforms MobileNetV2 under various latency settings, showcasing its ability to optimize architectures based on hardware constraints [9]. The results demonstrate how significant improvements can be realized by aligning the architecture with the hardware.

For future research, broadening the capacity of this approach to enable heterogeneous hardware settings and real-time tasks is a promising area to pursue.

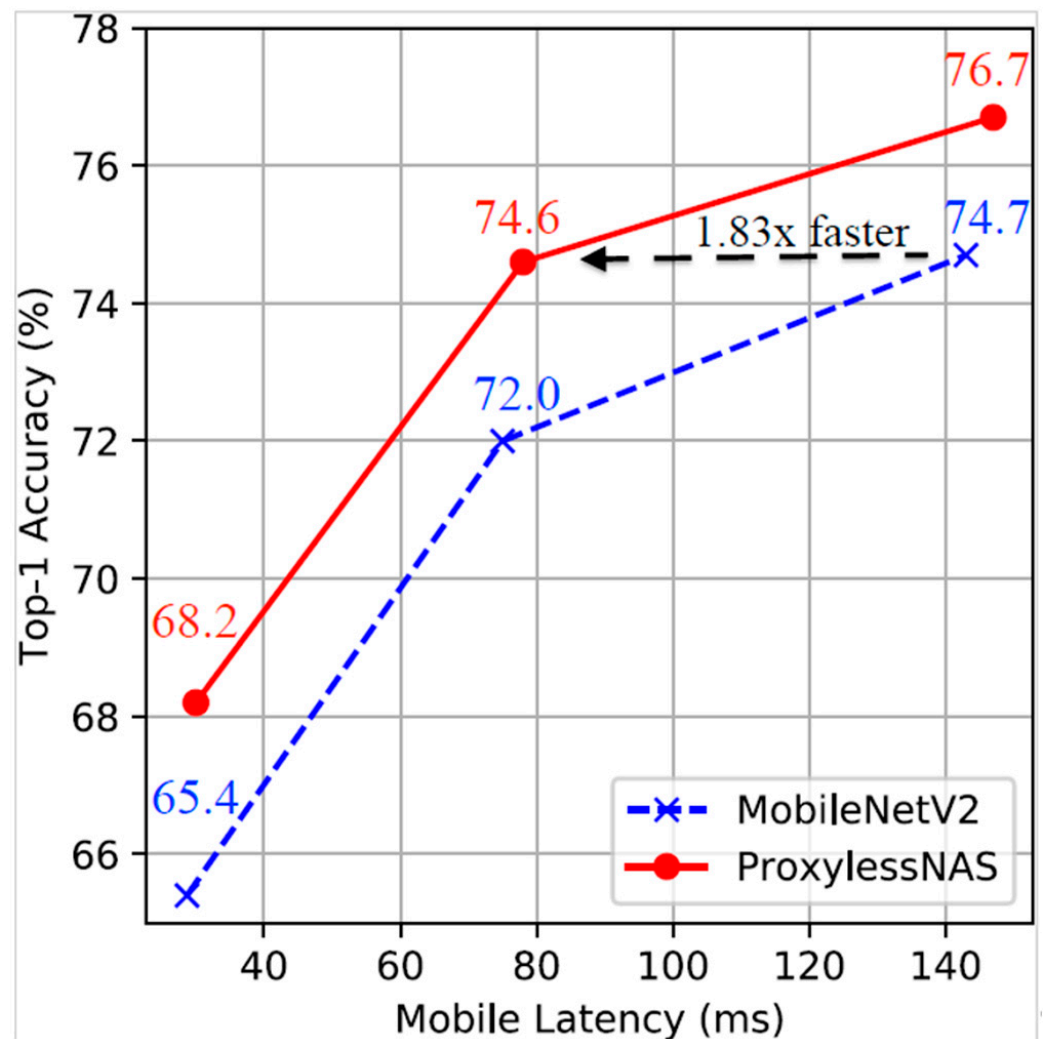


Figure 3. ProxylessNAS consistently outperforms MobileNetV2 under various latency settings.

3.4. Sparse Representations and Pruning: Efficiency and Limitations

EIE alum employs pruning to remove unhelpful connections, achieving sparse representations of DL models. As shown in Figure 4, part (a) shows the architecture of the leading non-zero detection node and part (b) shows the architecture of the processing element.

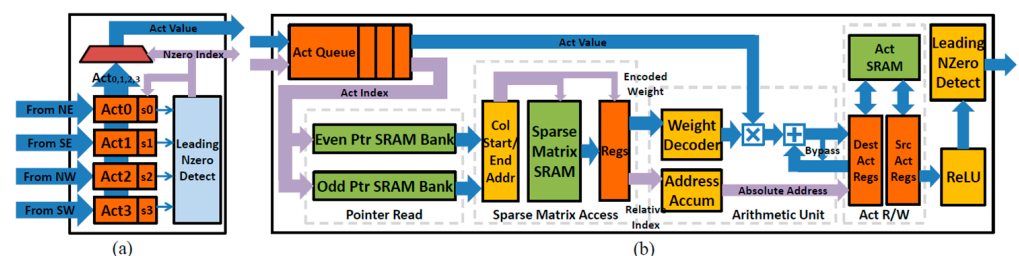


Figure 4. Architectures of the leading non-zero detection node and processing element: (a) architecture of leading non-zero detection node; (b) architecture of processing element.

A notable drawback of sparse networks is their increased inference time; however, sparse architecture is less memory-demanding and more efficient for inferencing when used with AI accelerators, increasing task speed. One constraint of this approach is the limited application of pruning for dynamic architectures and novel models like transformers.

3.5. System-Level Optimizations

Some aspects like cache usage, hierarchy of memory, and data movement might be included in the design of DNN and affect both latency and energy efficiency. Earlier studies, as illustrated in Figure 5, emphasized the necessity of data movement optimizations through techniques such as pruning, quantization, and Huffman encoding [2]. The input is the original model, and the output is the compressed model. These are three stage compression methods which describe how the original model is transformed into a compressed model to improve efficiency.

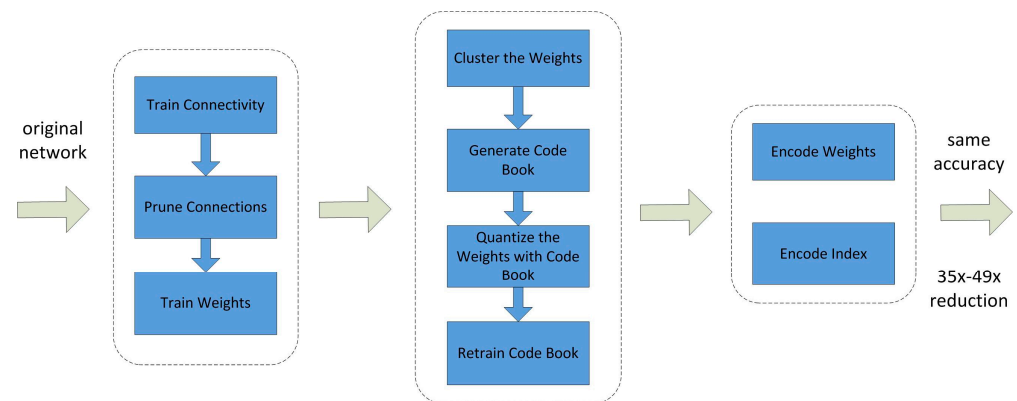


Figure 5. The three stage compression methods proposed include pruning, quantization, and Huffman encoding.

Later research could focus on capturing such optimizations within existing frameworks for easy adoption, especially in edge and embedded devices.

4. Implications and Future Directions

While Sections 2 and 3 outline established methodologies and demonstrate their practical efficacy, this section highlights emerging approaches and unresolved issues, identifying areas ripe for future exploration. We further elaborate on the context of the evolution of memory-efficient architectures by framing what we consider the most constructive future directions and what further gaps still exist for innovation.

4.1. Historical Context and Development Timeline

Understanding the evolution of memory-efficient architectures provides context for identifying future research opportunities. The timeline below summarizes key developments that have significantly influenced current deep learning (DL) systems:

- 2015–2016: Early breakthroughs in compression and pruning—Introduction of “*Deep Compression*” by Han et al. [2], pioneering methods like parameter pruning, quantization, and Huffman coding for memory reduction. Development of specialized inference engines optimizing sparse matrix representations and model compression [12].
- 2017–2018: Emergence of hardware-aware optimization—Introduction of Neural Architecture Search (NAS) frameworks such as ProxylessNAS, optimizing architectures specifically tailored to hardware constraints and real-time requirements [9].
- 2019–2022: Advancements in adaptive precision and hybrid methodologies—Extensive research in mixed-precision quantization, dynamically adjusting bit-width to optimize accuracy and resource use [6,7,15]. Growth of hybrid methods combining multiple strategies (e.g., simultaneous pruning and quantization) to achieve greater efficiencies [13,14].

- 2023–Present: Current trends and emerging research directions—Rising interest in Processing-in-Memory (PIM) architectures, addressing the von Neumann bottleneck by integrating computational units directly into memory [1]. Innovations in dynamic network structures, such as adaptive pruning and runtime quantization adjustments, for real-time adaptability [16,17]. Exploration of memory-efficient techniques in advanced architectures (e.g., transformers, large language models), targeting deployment in resource-constrained environments [17,18].

This historical perspective helps position our discussion of future-oriented methodologies and highlights how recent advancements open promising directions for subsequent research.

4.2. Future Research Directions

By combining dynamic quantization with hardware-aware optimization, models can adjust precision levels in real-time to match the workload and hardware limitations. This ensures critical layers use higher precision, while less important layers reduce precision to save space. Additionally, hardware-aware neural architecture search (NAS) enables fine-tuning on devices like GPUs or TPUs. Expected results:

- Enhanced Efficiency: Through the combination of pruning with real-time adaptive quantization, the program will not require as much memory.
- Real-Time Adaptability: Simply by switching the connections on and off, a trained model can respond to altering workloads without the need for rebuilding.
- Broader Applicability: These advanced technique choices can make machine learning work in edge and embedded devices that have tight resources; thus, they can tackle the real-time aspect of execution more efficiently.

Hybrid methodologies warrant further research to identify the most effective combinations of these techniques and achieve practical deployment, not only in a single but in a wide range of real-time applications.

4.2.1. Automated and Adaptive Compression

Automating this process dynamically adjusts model size according to real-time workloads, presenting significant performance gains. As shown in Figure 6, neural network parameter analysis reveals low entropy in exponent bits, highlighting potential for efficient compression. This insight underpins the NeuZip framework, which dynamically compresses these components for memory-efficient training and inference [19]. Future research should explore how frameworks like NeuZip can adapt to varying workloads while maintaining consistent performance across applications.

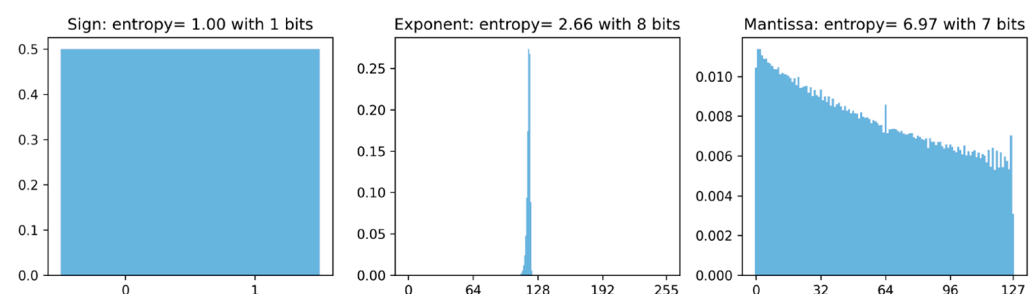


Figure 6. Histograms of sign bits, exponent bits, and mantissa bits in the Llama-3 8B model, highlighting low entropy in exponent bits for compression.

4.2.2. Quantization Beyond Mixed Precision

Mixed-precision quantization has proven effective for balancing accuracy and efficiency, as described by Gholami et al. [6]. By using varying bit-widths across different layers of a neural network, it allows for trade-offs between computational savings and model accuracy. However, recent advancements, such as adaptive bit-width quantization, have pushed the boundaries further.

As shown in Figure 7, adaptive bit-width quantization enables a neural network to dynamically adjust the precision of weights and activations. This approach allows for fine-grained adjustments, where the bit-width can vary not only across layers, but even within a single layer, depending on the computational budget or hardware constraints [15]. This flexibility leads to better optimization of accuracy and efficiency for diverse deployment scenarios, including resource-limited edge devices.

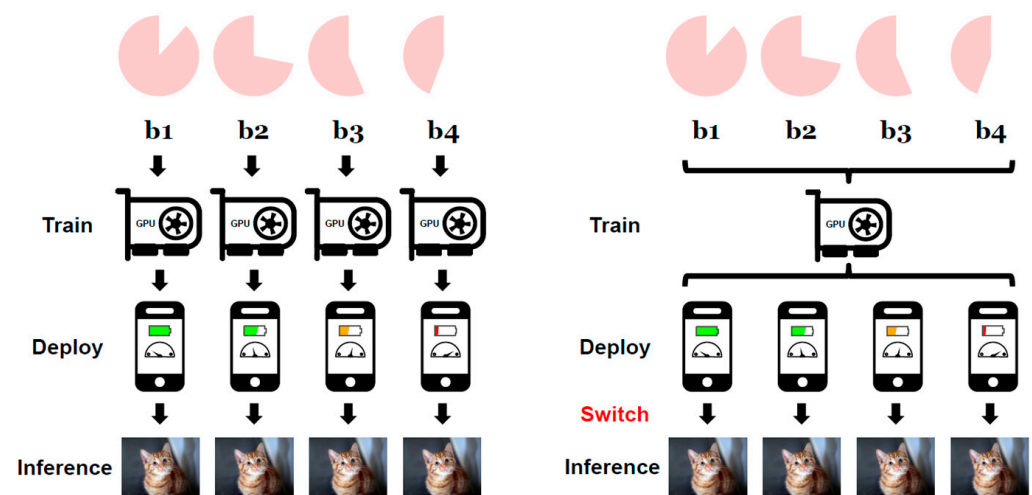


Figure 7. Adaptive bit-width quantization framework, showcasing dynamic precision adjustments for weights and activations to balance accuracy and efficiency.

Future research in this area could focus on incorporating stochastic quantization techniques to complement bit-width adaptation, enabling networks to learn optimal precision levels dynamically during training.

4.2.3. Heterogeneous Hardware Optimization

Heterogeneous hardware optimization uses NAS techniques like ProxylessNAS by Cai et al. [9] to align models with specific hardware configurations. ProxylessNAS directly optimizes architectures on target tasks and hardware, avoiding proxy tasks and meta-controllers. This improves efficiency by tailoring models to hardware types, such as CPUs for general tasks, GPUs for compute-intensive tasks, and TPUs for low-latency applications, as shown in Figure 8.



Figure 8. ProxylessNAS optimization directly aligns neural architectures to specific hardware, improving efficiency in GPU hours and memory use.

4.2.4. Dynamic Pruning for Real-Time Applications

Another promising avenue of work is Dynamic DL where the structure of the network is no longer considered static. They could work in scenarios when connections are made and broken in real-time based on what is fed in or due to the constraints of the system. Such techniques can support adaptive inference ensuring dependable performance in dynamic and unpredictable environments.

A practical example of this approach is the Learning Kernel Activation Module (LKAM), as shown in Figure 9. The LKAM module dynamically adjusts kernel activation during the inference phase [16]. This real-time switching process allows the model to optimize computational efficiency without compromising accuracy, supporting adaptive inference in dynamic environments. The module leverages a thresholding mechanism to selectively activate kernels based on input characteristics.

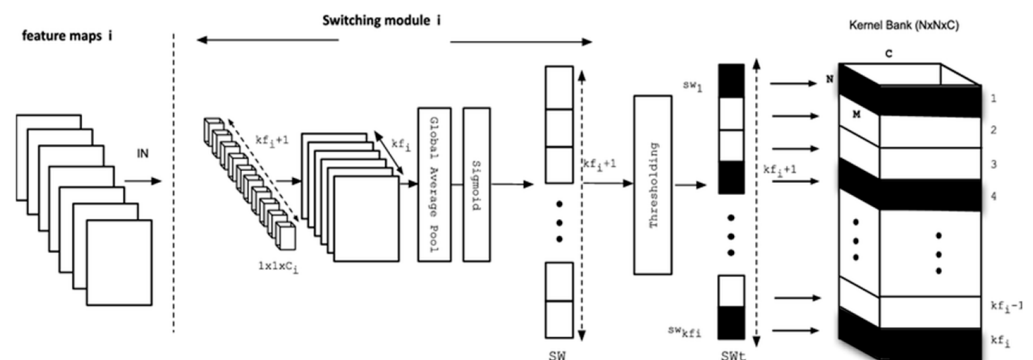


Figure 9. The switching process of the LKAM module during inference, dynamically selecting kernels based on real-time inputs to optimize computational efficiency.

4.2.5. Data Movement and Cache Optimization

Reducing data motion and accessing data already in the hardware could greatly assist in reducing latency and improving the power efficiency of the system. One avenue of understanding may be algorithm development where data paths themselves dynamically change in active systems, allowing for the avoidance of chokepoints and enhancing aggregate throughput.

As Figure 10 shows, system architectures that split the computational and memory parts create problems in moving data across the network that connects them [19]. This arrangement can reduce system performance and use more power. To fix these issues, we can adjust data paths on-the-fly to maintain system efficiency and save energy while the system is working.

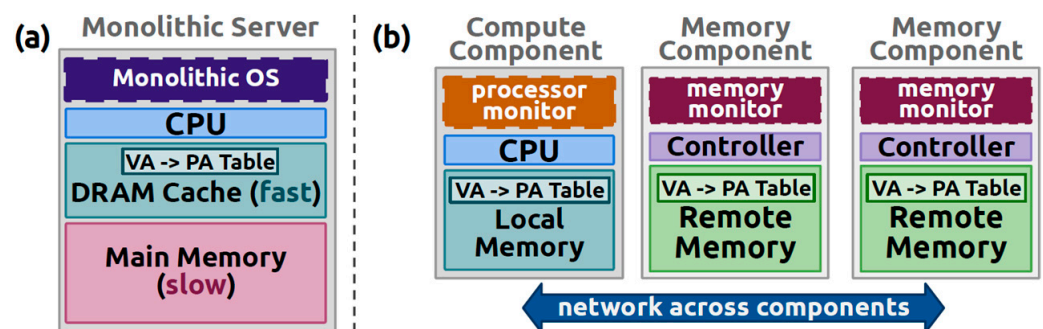


Figure 10. Comparison of monolithic and disaggregated system architectures, showcasing the separation of components and the need for efficient data movement strategies: (a) Monolithic system. (b) Disaggregated system.

4.2.6. Integrating Memory-Efficient Techniques in Emerging Architectures

The popularity of transformer models and large language models is on the rise, but they are still prohibitively expensive. Incorporating memory-efficient techniques like quantization and pruning in such architectures may help mitigate the imbalance between their capabilities and the constraints of edge devices. Among these techniques, pruning is particularly effective for transformers, as it directly addresses computational complexity and memory usage.

Figure 11 shows how attention map guided pruning, combining token and head pruning, reduces the transformer models' complexity and memory usage. This method optimizes model size for edge devices while preserving accuracy by selectively removing less critical components.

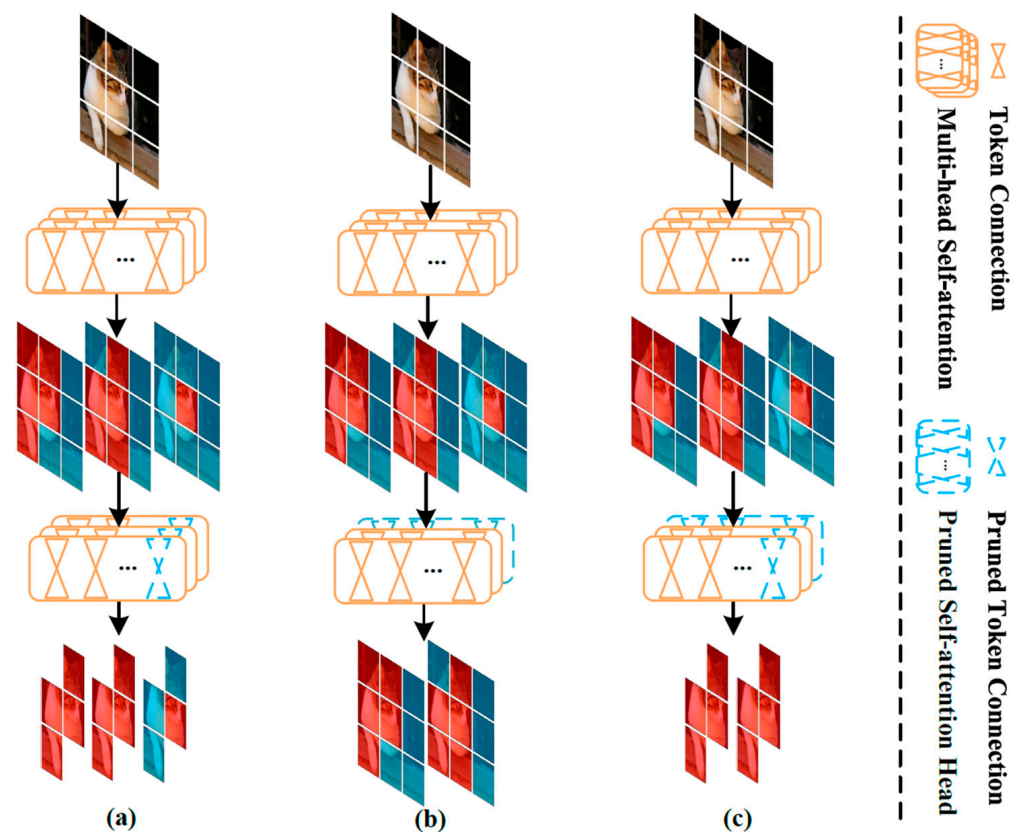


Figure 11. Pruning strategies for transformers: (a) token pruning; (b) head pruning; (c) attention map guided pruning for memory and computational efficiency.

4.3. Challenges in Real-Time Deployment

Despite these encouraging directions, several challenges need to be addressed:

- **Efficient and accurate practice:** Achieving these optimal tradeoffs remains a key concern in engineering practice, particularly in safety-critical applications. For instance, the development of autonomous driving vehicles demands the utmost consideration for the effectiveness of such trade-offs/sacrifices.
- **Task-specific scalability:** Model or task-specific interventions and improvements may not be scaled easily and would need new scopes of evaluation.
- **Set of standards along with benchmarks:** The adoption of cross-comparison techniques for memory-efficient techniques suffers from a lack of standard benchmarks and this curtails any attempts at uniform evaluation.

All these challenges should be properly analyzed, and certain studies may be executed which would produce optimal solutions.

4.4. Opportunities for Edge and Embedded Systems

The results from such studies strengthen the case for integrating efficient DL architecture into edge and embedded systems.

Edge and embedded systems are key areas where deep learning (DL) can have significant impact. Edge devices, like smart objects and autonomous cars, process data near its source, enabling real-time decisions with minimal reliance on cloud servers. Embedded devices are specialized systems within devices, designed for specific tasks with constraints on memory, energy, and operations.

Addressing these challenges and pursuing recommended research directions paves the way for innovations that enhance AI deployment in real-time applications across industries such as healthcare, transportation, and manufacturing.

5. Conclusions

Memory-efficient architecture for deep learning is a promising area for development. By overcoming current challenges, researchers can unlock the full potential of real-time AI systems for resource-constrained environments without sacrificing precision or efficiency.

This paper reviewed optimization techniques essential for enabling deep learning deployment in resource-constrained, real-time environments. Methods such as model compression, quantization, pruning, and hardware-specific design enable efficient edge and embedded systems while maintaining accuracy.

The findings are valuable for applications in healthcare, autonomous systems, and drones. Future work should focus on simplifying optimization and developing benchmarks to assess these techniques across diverse use cases.

Author Contributions: Conceptualization, B.D., E.D. and D.M.; methodology, B.D., E.D. and D.M.; software, B.D., E.D. and D.M.; validation, B.D., E.D. and D.M.; formal analysis, B.D., E.D. and D.M.; investigation, B.D., E.D. and D.M.; resources, B.D., E.D. and D.M.; data curation, B.D., E.D. and D.M.; writing—original draft preparation, B.D., E.D. and D.M.; writing—review and editing, B.D., E.D. and D.M.; visualization, B.D., E.D. and D.M.; supervision, B.D., E.D. and D.M.; project administration, B.D., E.D. and D.M.; funding acquisition, B.D., E.D. and D.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Dataset available on request from the author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kaur, R.; Asad, A.; Mohammadi, F. A comprehensive review of processing-in-memory architectures for deep neural networks. *Computers* **2024**, *13*, 174. [[CrossRef](#)]
2. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv* **2015**, arXiv:1510.00149.
3. Iandola, F.N. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
4. Zhang, Q.; Zhang, M.; Wang, M.; Sui, W.; Meng, C.; Yang, J.; Kong, W.; Cui, X.; Lin, W. Efficient deep learning inference based on model compression. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Salt Lake City, UT, USA, 18–22 June 2018; pp. 1695–1702.
5. Deng, L.; Li, G.; Han, S.; Shi, L.; Xie, Y. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* **2020**, *108*, 485–532. [[CrossRef](#)]

6. Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Mahoney, M.W.; Keutzer, K. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2022; pp. 291–326.
7. Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; Han, S. Haq: Hardware-aware automated quantization with mixed precision. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 16–20 June 2019; pp. 8612–8620.
8. Balaskas, K.; Karatzas, A.; Sad, C.; Siozios, K.; Anagnostopoulos, I.; Zervakis, G.; Henkel, J. Hardware-aware DNN compression via diverse pruning and mixed-precision quantization. *IEEE Trans. Emerg. Top. Comput.* **2024**, *12*, 1079–1092. [[CrossRef](#)]
9. Cai, H.; Zhu, L.; Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv* **2018**, arXiv:1812.00332.
10. Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; Han, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv* **2019**, arXiv:1908.09791.
11. Krieger, T.; Klein, B.; Fröning, H. Towards hardware-specific automatic compression of neural networks. *arXiv* **2022**, arXiv:2212.07818. [[CrossRef](#)]
12. Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M.A.; Dally, W.J. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Comput. Archit. News* **2016**, *44*, 243–254. [[CrossRef](#)]
13. Tung, F.; Mori, G. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 7873–7882.
14. Kim, J.; Chang, S.; Kwak, N. PQK: Model compression via pruning, quantization, and knowledge distillation. *arXiv* **2021**, arXiv:2106.14681. [[CrossRef](#)]
15. Jin, Q.; Yang, L.; Liao, Z. Adabits: Neural network quantization with adaptive bit-widths. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 14–19 June 2020; pp. 2146–2156.
16. Nikolaos, F.; Theodorakopoulos, I.; Pothos, V.; Vassalos, E. Dynamic pruning of CNN networks. In Proceedings of the 10th International Conference on Information, Intelligence, Systems and Applications (IISA), Patras, Greece, 15–17 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–5.
17. Mao, J.; Yao, Y.; Sun, Z.; Huang, X.; Shen, F.; Shen, H.T. Attention map guided transformer pruning for edge device. *arXiv* **2023**, arXiv:2304.01452. [[CrossRef](#)]
18. Hao, Y.; Cao, Y.; Mou, L. NeuZip: Memory-efficient training and inference with dynamic compression of neural networks. *arXiv* **2024**, arXiv:2410.20650.
19. Giannoula, C.; Huang, K.; Tang, J.; Koziris, N.; Goumas, G.; Chishti, Z.; Vijaykumar, N. DaeMon: Architectural support for efficient data movement in disaggregated systems. *arXiv* **2023**, arXiv:2301.00414. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.