SQuaRE: Sparsity-Quantization Resource-Efficient Optimization

Mehrshad Dehghani
Sharif University of Technology
Tehran, Iran
mehrshad.dehghani82@sharif.edu

Nezam Rohbani
Barcelona Supercomputing Center (BSC)
Barcelona, Spain
nrohbani@bsc.es

Hamid Sarbazi-Azad
Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran
azad@ipm.ir

Adrian Cristal
Barcelona Supercomputing Center (BSC)
Barcelona, Spain
adrian.cristal@bsc.es

Osman Sabri Unsal
Barcelona Supercomputing Center (BSC)
Barcelona, Spain
osman.unsal@bsc.es

Abstract

Quantization and structured $N:M$ sparsity reduce complementary inference costs, but their impact on quality and hardware efficiency varies across layers, architectures, and platforms. We present **SQuaRE** (Sparsity-Quantization Resource-Efficient Optimization), a hardware–software co-design framework that jointly selects sparsity and precision under an accuracy constraint. SQuaRE evaluates layer-wise sparsity–precision candidates using reconstruction error and platform-dependent resource costs, filters Pareto-dominated configurations, and derives a heterogeneous compression policy. We evaluate 16 combinations of four precisions and four sparsity patterns on DeiT-Tiny, Swin-Tiny, and ResNet-18. Under a 1.0-percentage-point accuracy-drop constraint applied to each layer in isolation, SQuaRE achieves mean energy reductions of 92.7%, 94.7%, and 84.3%, respectively. These results demonstrate that joint layer-wise optimization of sparsity and precision improves energy efficiency while respecting model-quality constraints.

Keywords

Sparsity, Quantization, DNN, LLM, Accuracy, Energy, Performance

1 Introduction

The rapid growth of deep neural networks (DNNs), particularly large language models (LLMs), has substantially increased deployment costs, with recent studies considering models from hundreds of millions to tens of billions of parameters [3, 6]. Reducing memory footprint and computation is therefore essential for efficient inference. Two widely adopted approaches are *sparsification*, which removes parameters and associated computations, and *quantization*, which represents parameters and activations using fewer bits [4, 17]. Their combination can substantially reduce storage, data movement, and computation. For example, OBC demonstrated up to 12× fewer theoretical operations with a 2% accuracy drop and 4× measured runtime speedup with a 1% drop [4]. JSQ reported a 7.96× computation reduction for LLaMA [5], while HCSQ achieved 14.96×–28.58× compression with less than 1% accuracy loss [15].

Among sparsity techniques, structured $N:M$ sparsity has attracted attention because its regular structure enables predictable execution on sparsity-aware hardware. In an $N:M$ model, every group of M weights contains exactly N non-zero values, providing fine-grained sparsity with hardware regularity. DominoSearch assigned different $N:M$ configurations to different layers to improve the accuracy–sparsity trade-off [14], while MaskLLM and ELSA explored learnable and layer-wise $N:M$ sparsity for LLMs and vision transformers, respectively [3, 9]. These studies establish sparsity configuration as an important optimization dimension.

In parallel, recent work has studied joint sparsity–quantization optimization. OBC introduced a unified post-training framework for pruning and quantization using Hessian-based weight selection and compensation, but its joint GPU evaluation considered only four sparsity–precision configurations [4]. JSQ explicitly coupled the two techniques for LLM compression [5], while Harma *et al.* showed that sparsity and quantization are non-orthogonal and that their ordering can affect model quality [6]. More recent methods, including SLOPE, Sherry, Sparse-BitNet, and SPARQ, explore aggressive joint compression and hardware support [1, 8, 10, 18]. HCSQ further incorporates hardware characteristics into joint $N:M$ sparsity and quantization optimization [15]. However, HCSQ uses Hessian-based layer ordering and iterative retraining to select hardware-supported configurations under a global accuracy budget. This makes the optimization dependent on layer order and does not directly evaluate the end-to-end impact and hardware cost of individual sparsity–precision choices. Overall, these works demonstrate the potential of joint compression while leaving substantial variation across layers and hardware platforms.

The joint sparsity–precision design space introduces a key systems challenge. Sparsity primarily reduces active computation, while lower bit-width reduces data storage, transfer, and processing. Their effects are not necessarily independent, and higher nominal compression does not always minimize latency or energy on a given platform [6, 15]. This raises the question: *given an accuracy target and hardware platform, how should sparsity and quantization be allocated across layers to maximize efficiency?*

We introduce SQuaRE: Sparsity-Quantization Resource-Efficient Optimization, a hardware–software co-design framework that jointly explores structured $N:M$ sparsity and quantization for neural network inference. Each layer can favor sparsity, lower precision, or their combination according to its sensitivity and platform. SQuaRE evaluates layer-wise sparsity–precision candidates based on quality impact and hardware cost, constructs Pareto-frontier choices, and derives a layer-wise allocation. We also outline an interaction-aware refinement procedure for enforcing a network-wide accuracy constraint, but do not evaluate this extension in the present study.

Experiments on DeiT-Tiny, Swin-Tiny, and ResNet-18 show substantial architecture-dependent differences in compression tolerance. Under a 1.0% accuracy-drop constraint, SQuaRE achieves mean energy reductions of 92.7%, 94.7%, and 84.3%, respectively, highlighting the benefits of hardware-aware, layer-wise joint optimization. These findings motivate further evaluation of joint network-wide assignments.

2 Proposed Method

SQuaRE is a hardware-aware compression framework that jointly explores structured $N:M$ sparsity and quantization to identify resource-efficient configurations for DNN inference. The key observation behind SQuaRE is that sparsity and quantization affect different components of inference cost. Structured sparsity reduces the number of weights involved in computation, while quantization reduces the number of bits required to represent and move weights and activations and can improve arithmetic throughput on low-precision hardware, enabling complementary efficiency benefits across different inference platforms.

As illustrated in Fig. 1, $N:M$ sparsity divides weights into groups of M consecutive elements and retains exactly N non-zero weights per group. For example, $2:4$ retains two of four weights, resulting in 50% sparsity, while $3:8$ and $4:8$ retain three of eight and four of eight weights, respectively. Different $N:M$ patterns can therefore provide the same sparsity ratio with different grouping and hardware requirements. Quantization complements sparsity by reducing the precision of the remaining weights, e.g., using 8-bit or 4-bit values instead of FP32. Thus, sparsity reduces the number of non-zero weights, while quantization reduces their precision.

SQuaRE jointly considers sparsity structure and numerical precision as interacting compression dimensions. Their effects on accuracy and hardware cost are not necessarily independent: sparsity can alter a layer's sensitivity to quantization, and vice versa. These effects also vary across layers due to differences in parameter distributions and computational roles. SQuaRE therefore performs layer-wise joint selection of sparsity and precision to capture these interactions and balance accuracy with hardware efficiency.

2.1 Joint Sparsity–Precision Design Space

For a network with L compressible layers, SQuaRE defines supported sparsity configurations $\mathcal{S}$ and quantization precisions $\mathcal{Q}$. In our initial implementation, we consider $\mathcal{S} = \{Dense, 2:4, 3:8, 4:8\}$ and $\mathcal{Q} = \{FP32, INT8, INT6, INT4\}$. The candidate set is platform-dependent and can be adapted to the capabilities of the target accelerator.

As shown in Fig. 1, each layer can move from the original dense representation to different $N:M$ structures. SQuaRE combines each supported sparsity structure with the available precisions to form a joint design space. Each layer is assigned one sparsity–precision pair:

$$c_\ell = (s_\ell, q_\ell), \qquad s_\ell \in \mathcal{S},\ q_\ell \in \mathcal{Q} \tag{1}$$

This formulation enables heterogeneous compression across the network. For example, one layer may use INT8+Dense, another INT8+2:4, and a less sensitive layer INT4+4:8. Thus, SQuaRE does not enforce a single sparsity level or precision globally; instead, it selects the configuration that best matches each layer's sensitivity and target hardware.

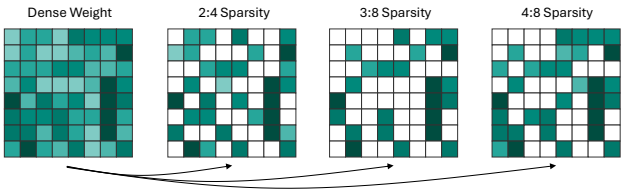


Figure 1: Visualization of different levels of $N:M$ pruning applied to neural network weights.

2.2 Candidate Generation

For every layer, SQuaRE generates candidates by combining the supported $N:M$ sparsity structures with available quantization precisions, Fig. 2-I. For a selected $N:M$ configuration, weights are grouped into sets of M consecutive elements, retaining the N largest-magnitude weights and pruning the rest. Different configurations may have different grouping granularities; for example, $2:4$ and $4:8$ both retain half the weights but may incur different metadata and execution overheads.

The surviving weights are then quantized using group-wise symmetric quantization with a configurable group size. Each candidate therefore defines a complete compressed representation; for example, INT4+$2:4$ combines the $2:4$ sparsity pattern with four-bit weights. Sparsity and quantization are evaluated jointly because their effects are not necessarily independent: quantization sensitivity depends on the retained weights, while hardware cost can vary with the sparsity structure. The final configuration is selected for each layer according to the target *Platform Configuration*, which specifies relevant hardware performance, energy, and memory characteristics.

2.3 Quality Impact Estimation

SQuaRE uses a small calibration set to estimate the quality impact of each candidate before constructing the final compressed model. For each layer, the calibration samples are propagated through the original network to obtain representative layer inputs. We then compare the output of the original layer with that produced by each sparsity–precision candidate.

For layer ℓ, the reconstruction error of candidate c is defined as

$$\mathcal{R}_\ell(c) = \left\| W_\ell X_\ell - \widehat{W}_{\ell,c} X_\ell \right\|_2^2, \tag{2}$$

where W_ℓ is the original weight matrix, X_ℓ denotes the calibration inputs to the layer, and $\widehat{W}_{\ell,c}$ is the corresponding sparsified and quantized weight matrix (Fig. 2-II).

Layer-output reconstruction is used instead of weight error because the same weight distortion can affect layers differently depending on their inputs. Thus, reconstruction error provides a common quality metric for comparing different $N:M$ structures and numerical precisions.

During final evaluation, the reconstruction estimates are validated against end-to-end model accuracy to assess how well the local metric predicts the global impact of different compression choices.

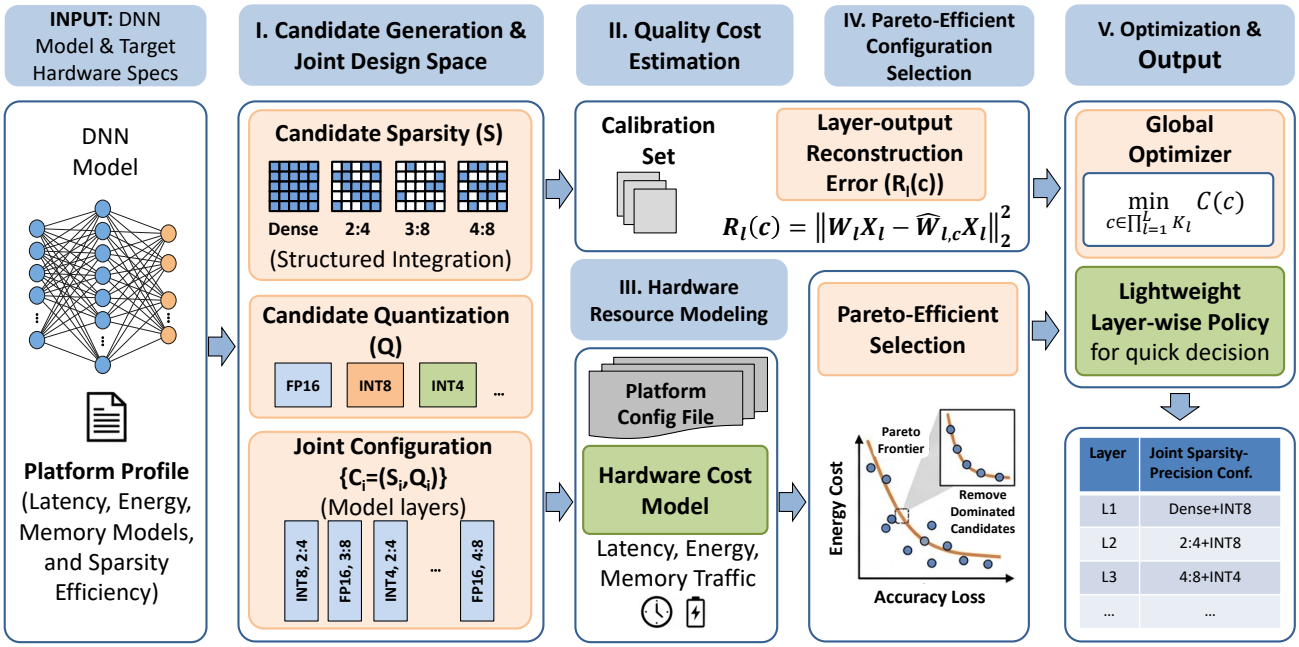


Figure 2: Given a DNN and target platform constraints, SQuaRE: (I) builds a joint $N:M$ sparsity and precision space; (II) evaluates layer reconstruction error; (III) models hardware costs (latency, energy, memory); (IV) filters for Pareto-optimal choices; and (V) allocates layer-wise configurations under quality budget ε.

2.4 Hardware Resource Modeling

SQuaRE associates each sparsity–precision candidate with a resource cost for layer-wise selection. In this initial evaluation, we use a simplified normalized energy proxy:

$$\widetilde{E}_\ell(c) = \frac{b_c}{32}\,\rho_c, \qquad \rho_c = \begin{cases} 1, & \text{dense}, \\ N/M, & N:M \text{ sparse}, \end{cases} \tag{3}$$

where b_c is the precision in bits and ρ_c is the retained-weight fraction. Each layer's FP32+dense configuration therefore has normalized cost 1.

This proxy assumes linear scaling with bit-width and retained-weight density. It does not explicitly model sparsity metadata, activation traffic, memory hierarchy, hardware utilization, or execution overheads. The resulting values are comparative cost estimates rather than measured hardware energy.

For L selected layers, we report the unweighted mean proxy reduction:

$$S_{\text{mean}} = 100\left(1 - \frac{1}{L}\sum_{\ell=1}^{L}\widetilde{E}_\ell(c_\ell)\right)\%. \tag{4}$$

This statistic gives each selected layer equal weight and should not be interpreted as total network energy reduction. Platform-calibrated costs can replace this proxy in future work.

2.5 Pareto-Efficient Configuration Selection

Each layer's candidates form a multi-objective design space in which lower quality degradation, latency, energy, and memory traffic are desirable. SQuaRE discards dominated candidates and retains only Pareto-efficient ones. A candidate is Pareto-efficient if no alternative is no worse in every objective and strictly better in at least one. For example, if INT8+2:4 has lower latency and energy than INT8+Dense without higher reconstruction error or memory traffic, the latter is dominated. The resulting Pareto frontier reduces the search space while preserving the relevant quality–resource trade-offs.

Let $\mathbf{c} = (c_1, \ldots, c_L)$ denote a network configuration, where $c_\ell \in \mathcal{K}_\ell$ is selected from layer ℓ's candidate set. The network-level optimization problem is

$$\min_{\mathbf{c}\in\prod_{\ell=1}^{L}\mathcal{K}_\ell} C(\mathbf{c}) \quad \text{s.t.} \quad A_0 - A(\mathbf{c}) \le \epsilon, \tag{5}$$

where $C(\mathbf{c})$ is the total network resource cost, $A(\mathbf{c})$ is its measured end-to-end accuracy, A_0 is the FP32+dense baseline accuracy, and ϵ is the permitted drop. Depending on the target platform, C may represent latency, energy, memory traffic, or their weighted combination. Because cross-layer interactions make $A(\mathbf{c})$ non-additive, Eq. 5 cannot generally be solved by summing isolated layer-wise penalties. The global constraint permits sensitive layers to retain conservative configurations while less sensitive layers use more aggressive sparsity–precision combinations.

2.6 Lightweight Layer-Wise Policy

Although the global formulation provides the optimal configuration under the specified model, searching all network-wide assignments becomes expensive as the number of layers and candidates grows. SQuaRE therefore also provides a lightweight layer-wise policy that avoids exhaustive global search while retaining hardware-aware candidate selection.

The policy evaluates the incremental hardware benefit of moving to a more efficient candidate against its additional quality loss.

Candidates offering large reductions in latency, energy, or memory traffic for small quality penalties are prioritized, while Pareto-dominated candidates are discarded. The resulting policy produces a heterogeneous compression map, assigning each layer a joint $N:M$ sparsity structure and numerical precision based on these trade-offs. We compare this lightweight policy with the global optimizer to evaluate how closely a simple layer-wise strategy approaches the best resource allocation.

Overall, SQuaRE formulates sparsity and quantization as a joint, hardware-aware resource allocation problem, producing a layer-wise mapping between $N:M$ patterns and numerical precision based on the trade-off between quality degradation and target-platform resource costs (Fig. 2-V). In other words, SQuaRE first constructs a complete, order-independent sensitivity map by testing every candidate on every selected layer in isolation, while leaving the rest of the network dense and full precision. Evaluating the selected configurations jointly and accounting for their end-to-end interactions remain necessary next steps. This post-training procedure applies uniformly to both CNN and Transformer architectures.

2.7 Interaction-Aware Global Configuration Refinement

The preceding analysis selects candidates independently for each layer based on local quality–resource trade-offs. However, independently efficient choices may not form a feasible network-wide assignment because compression errors can interact across layers. Let $\mathbf{c} = (c_1, \ldots, c_L)$ denote the current assignment, $A(\mathbf{c})$ its end-to-end accuracy on a held-out refinement set, and A_0 the FP32+dense accuracy. The refinement seeks a low-energy assignment satisfying

$$A_0 - A(\mathbf{c}) \leq \epsilon. \tag{6}$$

SQuaRE initializes $\mathbf{c}$ from the layer-wise Pareto frontiers and, if the accuracy constraint is violated, probes each layer's next safer frontier point c_ℓ^+. It selects the move providing the largest accuracy recovery per unit energy penalty:

$$\frac{A(\mathbf{c}_{\ell \to c_\ell^+}) - A(\mathbf{c})}{E_\ell(c_\ell^+) - E_\ell(c_\ell)}. \tag{7}$$

After each move, the joint model is reevaluated and the contextual scores are updated. Once feasible, the procedure can similarly probe more aggressive frontier points to reduce energy while preserving the global accuracy constraint. Pairwise or block-wise moves can be used when individual moves cannot improve the assignment.

This interaction-aware refinement is described as a future extension and is not evaluated in the current version of the paper. Unlike Hessian-ordered or reconstruction-additive scheduling [4, 15], it would base decisions on end-to-end marginal effects measured in the currently compressed network.

3 Experimental Results

We evaluate SQuaRE on three image-classification models: DeiT-Tiny and Swin-Tiny, Transformer-based vision networks with approximately 5M and 28M parameters, respectively, and ResNet-18, a convolutional neural network (CNN) with approximately 11.7M parameters [7, 11, 16]. All models are evaluated on the ImageNetV2 Matched Frequency dataset [13]. For each model, we explore all 16 combinations of four sparsity configurations Dense, 2:4, 3:8, 4:8 and four precisions FP32, INT8, INT6, INT4, covering a broad range of compression levels and hardware requirements.

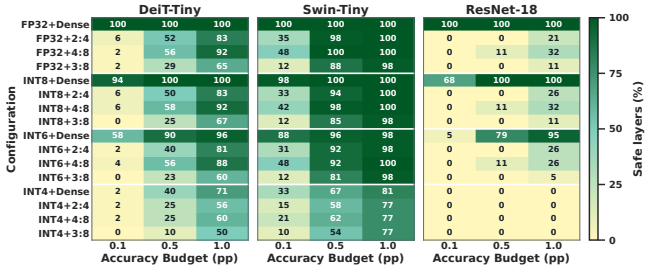


Figure 3: Percentage of layers that tolerate each sparsity–precision configuration under top-1 accuracy-drop budgets.

SQuaRE follows a hardware–software co-design approach. For a target platform, it reads a configuration file containing latency, energy, and memory-traffic characteristics. The energy models for different precisions and sparsity levels are derived from prior hardware-oriented formulations for quantized computation [12] and $N:M$ sparse computation [2]. SQuaRE uses these models to estimate the energy cost and reconstruction-induced accuracy impact of each sparsity–precision configuration for every layer. A Pareto-efficient selection then identifies layer-wise configurations that minimize energy under the target accuracy constraint, which are applied to construct the compressed network.

3.1 Layer-Wise Compression Tolerance

We first evaluate how safely each sparsity–precision configuration can be applied to individual layers. For every layer, we test all 16 possible combinations and retain a configuration only when the resulting top-1 accuracy drop remains within the predefined budget. We consider accuracy-drop budgets of 0.1%, 0.5%, and 1.0%. Fig. 3 shows the percentage of layers for which each configuration is safe to apply under each accuracy constraint.

The results demonstrate substantial differences across architectures. For DeiT-Tiny, with a strict 0.1% budget, INT8+Dense is safe for 92% of the layers, while only 54% are safe for INT6+Dense and none for the aggressive INT4+3:8 configuration. With a 1.0% budget, however, INT4+3:8 becomes safe for 50% of the layers. Swin-Tiny shows substantially higher tolerance; for example, INT6+4:8 is safe for all layers at a 1.0% accuracy budget. In contrast, ResNet-18 is considerably more sensitive to structured $N:M$ sparsity, with most sparse configurations becoming unsafe even at the 1.0% budget, whereas dense quantization remains more tolerable. These results indicate that the sensitivity to joint sparsity and quantization is strongly architecture dependent, motivating layer-wise rather than uniform compression.

3.2 Layer-Wise Accuracy–Energy Trade-off

Fig. 4 shows the layer-wise top-1 accuracy impact of the sparsity–precision configurations selected under three accuracy budgets (0.1% , 0.5% , and 1.0%). For each layer, the compression is applied based on the minimum-energy selection: SQuaRE identifies the most energy-efficient configuration that satisfies the accuracy constraint. Each individual dot represents the top-1 accuracy drop

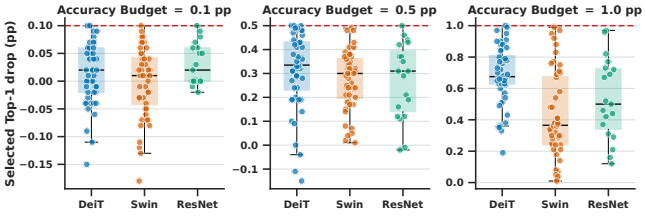


Figure 4: Accuracy drop per layer for configurations selected by SQuaRE under 0.1%, 0.5%, and 1.0% constraints (red dashed lines). Each dot represents a single layer's top-1 accuracy drop.

(in percentage points) associated with a single evaluated layer (48 layers each for DeiT-Tiny and Swin-Tiny, and 19 layers for ResNet-18). Negative values indicate an accuracy improvement, potentially arising from finite-sample variation or a regularizing effect of compression, whereas positive values represent an accuracy drop, with the isolated layer-wise accuracy-drop budget indicated by the red dashed line. The box plots summarize the layer-wise distribution: the center line denotes the median drop, the box spans the interquartile range (IQR), and the whiskers extend to the typical range, highlighting extreme outliers (e.g., drops reaching down to −0.18% for Swin-Tiny under the 0.1% constraint). As the allowed accuracy budget expands from ≤ 0.1% to ≤ 1.0%, SQuaRE systematically selects more aggressive, energy-saving configurations, shifting the median layer-wise drops upwards (e.g., for DeiT-Tiny, rising from approximately 0.02% up to nearly 0.68%).

3.3 Energy Consumption Optimization

We report the unweighted mean of the selected layers' modeled energy reductions relative to their FP32+dense counterparts. The energy consumption of each layer is optimized by selecting the configuration that maximizes energy savings under a given top-1 accuracy drop constraint (Fig. 5). For Vision Transformers, DeiT-Tiny significantly reduces energy by aggressively adopting combined sparsity and quantization; under a ≤ 1.0% drop limit, 24 out of 48 layers select the extreme INT4+3:8 configuration, achieving a mean energy saving of 92.7%. Swin-Tiny exhibits a similar trend, shifting heavily to INT4+3:8 as constraints relax and achieves a mean energy saving of 94.7% at the same limit. In contrast, ResNet-18 is highly sensitive to $N:M$ sparsity because structural pruning damages CNN convolutional filters, forcing a majority of layers to remain in a dense format. Nevertheless, quantization remains effective for CNNs, with ResNet-18 optimizing energy primarily through precision scaling from FP32+Dense down to INT6+Dense. Its selected configurations achieve an average energy saving of 84.3% under the 1.0% constraint.

4 Conclusions

We introduced SQuaRE, a framework for joint layer-wise optimization of $N:M$ sparsity and quantization under hardware constraints. Experiments on DeiT-Tiny, Swin-Tiny, and ResNet-18 show architecture-dependent compression sensitivity: transformers tolerate aggressive configurations such as INT4+4:8, while ResNet-18 is more sensitive to structured sparsity. Under a 1.0% accuracy budget, SQuaRE selects aggressive configurations for 24 of 48 DeiT-Tiny

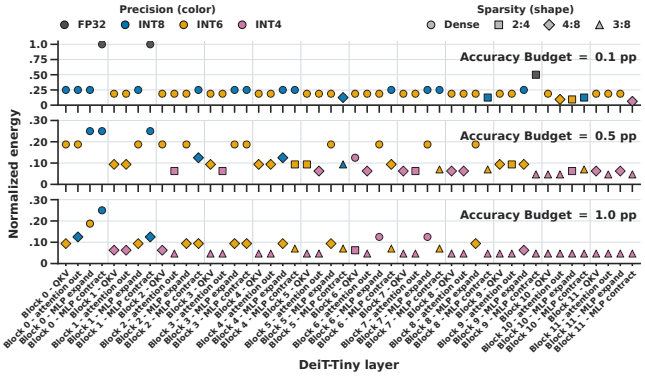


Figure 5: Energy-optimal layer configurations under three accuracy drop budgets.

layers, achieving ∼92.7% average energy reduction. These results highlight the value of treating sparsity and quantization as coupled, hardware-aware design choices. Future work will evaluate fully compressed networks under a network-wide accuracy budget, compare against uniform compression and established allocation methods, and assess larger and more recent architectures. Hardware validation will also be needed to establish realized energy, latency, and memory-traffic benefits.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments and suggestions, which helped improve the quality and clarity of this work. This work has been supported by the European High-Performance Computing Joint Undertaking (JU) under Grant Agreement No. 10120245 (DARE SGA1), by the European HiPEAC Network of Excellence, and by the Generalitat de Catalunya (contract 2021-SGR-00763).

References

[1] Seonggyu Choi and Hyungmin Cho. 2025. SPARQ: An Accelerator Architecture for Large Language Models with Joint Sparsity and Quantization Techniques. In *Proceedings of the 26th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. 3–15. https://doi.org/10.1145/3735452.3735523

[2] Chao Fang, Wei Sun, Aojun Zhou, and Zhongfeng Wang. 2023. CEST: Computation-efficient N: M sparse training for deep neural networks. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–2.

[3] Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. 2024. MaskLLM: Learnable Semi-Structured Sparsity for Large Language Models. In *Advances in Neural Information Processing Systems*, Vol. 37.

[4] Elias Frantar, Sidak Pal Singh, and Dan Alistarh. 2022. Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning. In *Advances in Neural Information Processing Systems*, Vol. 35. 4475–4488.

[5] Jinyang Guo, Jianyu Wu, Zining Wang, Jiaheng Liu, Ge Yang, Yifu Ding, Ruihao Gong, Haotong Qin, and Xianglong Liu. 2024. Compressing Large Language Models by Joint Sparsification and Quantization. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. 16945–16957.

[6] Simla Harma, Ayan Chakraborty, Elizaveta Kostenok, Danila Mishin, Dongho Ha, Babak Falsafi, Martin Jaggi, Ming Liu, Yunho Oh, Suvinay Subramanian, and Amir Yazdanbakhsh. 2025. Effective Interplay Between Sparsity and Quantization: From Theory to Practice. In *International Conference on Learning Representations*.

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

[8] Hong Huang, Decheng Wu, Qiangqiang Hu, Guanghua Yu, Jinhai Yang, Jianchen Zhu, Xue Liu, and Dapeng Wu. 2026. Sherry: Hardware-Efficient 1.25-Bit Ternary Quantization via Fine-grained Sparsification. In *Proceedings of the 64th Annual*

Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 11186–11204. https://doi.org/10.18653/v1/2026.acl-long.513

[9] Ning-Chi Huang, Chi-Chih Chang, Wei-Cheng Lin, Endri Taka, Diana Marculescu, and Kai-Chiang Wu. 2024. ELSA: Exploiting Layer-wise N:M Sparsity for Vision Transformer Acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 8006–8015.

[10] Dan Liu, Nikita Dvornik, and Xue Liu. 2025. Extreme Model Compression with Structured Sparsity at Low Precision. In *The British Machine Vision Conference*.

[11] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF international conference on computer vision (ICCV)*. Ieee, 9992–10002.

[12] Bert Moons, Koen Goetschalckx, Nick Van Berckelaer, and Marian Verhelst. 2017. Minimum energy quantized neural networks. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 1921–1925.

[13] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. 2019. Do ImageNet Classifiers Generalize to ImageNet?. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 5389–5400.

[14] Wei Sun, Aojun Zhou, Sander Stuijk, Robert Wijnhoven, et al. 2021. DominoSearch: Find Layer-wise Fine-grained N:M Sparse Schemes from Dense Neural Networks. In *Advances in Neural Information Processing Systems*, Vol. 34.

[15] Minghao Tang, Ming Ling, Minhua Ren, Zhihua Cai, Zhen Liu, Shidi Tang, and Jianjun Li. 2026. Hessian-driven N:M Sparsity and Quantization Co-optimization for Edge Device Deployment. *Integration* 107 (2026), 102629. https://doi.org/10.1016/j.vlsi.2025.102629

[16] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*. PMLR, 10347–10357.

[17] Chengxi Ye, Grace Chu, Yanfeng Liu, Yichi Zhang, Lukasz Lew, Li Zhang, Mark Sandler, and Andrew Howard. 2026. Robust Training of Neural Networks at Arbitrary Precision and Sparsity. In *International Conference on Learning Representations*.

[18] Di Zhang, Xun Wu, Shaohan Huang, Yudong Wang, Hanyong Shao, Yingbo Hao, Zewen Chi, Li Dong, Ting Song, Yan Xia, Zhifang Sui, and Furu Wei. 2026. Sparse-BitNet: 1.58-bit LLMs are Naturally Friendly to Semi-Structured Sparsity. arXiv:2603.05168 [cs.CL]