Move the Roof: Model-driven Methodology for Designing Efficient Deep Learning Architectures

Move the Roof: Model-driven Methodology for Designing Efficient Deep Learning Architectures

Alexandre Rodrigues*
Aleksandar Ilic
Leonel Sousa
alexandre.d.rodrigues@inesc-id.pt
aleksandar.ilic@inesc-id.pt
las@inesc-id.pt
INESC-ID/IST, University of Lisbon
Lisbon, Portugal

Mateo Vázquez*
Mohammad Ali Maleki
Pedro Trancoso
mateo.vazquezmaceiras@chalmers.se
mohammad.ali.maleki@chalmers.se
ppedro@chalmers.se
Chalmers University of Technology
and University of Gothenburg
Göteborg, Sweden

Muhammad Waqar Azhar
waqar.azhar@zeropoint-tech.com
ZeroPoint Technologies AB
Göteborg, Sweden

Abstract

Designing efficient architectures for Deep Learning (DL) at the edge is challenging, as the growing demands of emerging models contrast with the tight resource constraints. This work proposes a model-driven methodology for the fast exploration of large design spaces, producing efficient architectures under performance constraints. Our methodology combines a strategy to aggressively prune the design space with a Cache-Aware Roofline Model (CARM)-based performance estimation to avoid costly simulations.

CCS Concepts

• **Computing methodologies** → **Modeling and simulation**; • **General and reference** → **Design**; • **Computer systems organization** → **Architectures**.

Keywords

Design Methodology, Design Space Exploration, Runtime, Efficiency, Edge, Deep Learning

ACM Reference Format:
Alexandre Rodrigues, Aleksandar Ilic, Leonel Sousa, Mateo Vázquez, Mohammad Ali Maleki, Pedro Trancoso, and Muhammad Waqar Azhar. 2026. Move the Roof: Model-driven Methodology for Designing Efficient Deep Learning Architectures. In *The 41st ACM/SIGAPP Symposium on Applied Computing (SAC '26), March 23–27, 2026, Thessaloniki, Greece.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3748522.3779938

1 Introduction

Balancing performance and resource utilization is becoming increasingly important when designing computing systems, especially in embedded systems with strict operational constraints. This challenge has been intensified with the recent surge of Deep Learning (DL) applications, whose computational demands grow exponentially [13]. Embedded systems running DL workloads on the edge are often expected to execute a fixed set of applications with clear Minimum Performance Requirements (MPRs). These MPRs may be defined by metrics such as Quality of Service (QoS) [1].

To achieve this balance, parametrizable architectures like Vector Processing Units (VPUs) [4] can be used. Such architectures can easily adapt to the system's requirements and constraints, but navigating their complex design space remains a challenge. Two key aspects must be considered when conducting Design Space Exploration (DSE), finding efficient configurations that: (1) satisfy the MPRs of all target applications; and (2) adapt to the requirements of the currently executing application.

Cycle-accurate simulators such as gem5 [3] are typically used for DSE, providing detailed and accurate execution metrics. However, their prohibitive simulation cost in large design spaces requires approaches to accelerate DSE, such as: (1) focusing the search on the relevant configurations by pruning the design space [5]; and (2) accelerating the evaluation of each visited configuration [10].

Costly simulations can be replaced with lightweight performance models, quickly estimating an architecture's ability to satisfy target MPRs. The Cache-Aware Roofline Model (CARM) [6] is a good candidate, providing accurate and detailed modeling of the architecture's maximum attainable performance. It captures the performance of the complete memory hierarchy and computational capacity in an intuitive plot, making it a powerful tool for optimization [9]. DL applications in particular are very accurately characterized by the CARM, as the underlying linear algebra kernels are typically memory or compute-bound. Recent works have done some preliminary exploration of its use in hardware design [2, 12], "moving the roof" to better align with the application requirements.

This work proposes a CARM-driven methodology for fast DSE, yielding efficient hardware configurations that meet MPRs for DL applications on the edge. The model-driven methodology predicts the performance of unseen configurations using the CARM, avoiding costly simulations. A sensitivity analysis assesses the impact of key architecture parameters on performance and the target metric, guiding the steepest descent process for fast and accurate DSE.

2 Design Methodology

The methodology's goal is to find a configuration that maximally optimizes the target metric (e.g. power consumption), while meeting the MPR of each target application. The MPR can be expressed

*Both authors contributed equally to this research.

SAC '26, Thessaloniki, Greece

ACM ISBN 979-8-4007-2294-3/2026/03
https://doi.org/10.1145/3748522.3779938

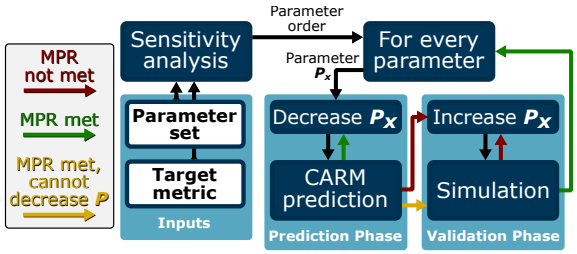

Figure 1: Parameter optimization process.

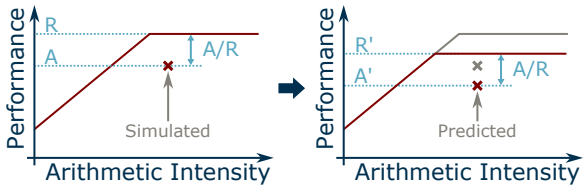

Figure 2: Performance prediction using CARM, in a compute-bound example. The initial configuration's application-to-roof performance ratio (A/R) is used to estimate the new configuration's performance (A') based on its new roof (R')

as any QoS metric, such as execution time or GFLOP/s. Given an initial configuration and its "performance budget" (i.e. the headroom above the MPR, or slack), the DSE must: (1) efficiently convert the "budget" into improvements to target metric; and (2) quickly evaluate the performance "cost" of a new configuration.

First, to know how to efficiently use the "budget", we perform a sensitivity analysis. This analysis begins by simulating the initial configuration, followed by successive simulations that individually lower each parameter to its next possible value. For every evaluated parameter, the improvement of the target metric is normalized to the associated performance cost. This yields each parameter's "cost-benefit", which determines the order they are optimized in, prioritizing those with higher results.

Following the cost-benefit order, each parameter is successively adjusted to the next value, lowering performance and improving the target metric. As shown in Figure 1, this continues until the prediction phase places the performance below the MPR, moving to the validation phase. The resulting configuration is then simulated to check whether the MPR is met. If not, the parameter is successively increased, repeating the simulation until a valid configuration is reached. Additionally, the observed application-to-roof performance ratios are used to tune future predictions.

Our fast CARM-based predictions are orders of magnitude faster than regular simulation of the application, allowing a configuration's "cost" to be evaluated quickly. Figure 2 describes this prediction mechanism. The CARM, which only needs to be constructed once per configuration, can be used to determine the maximum attainable performance instantaneously. While some applications do not reach the roof, their performance is still strongly correlated with the architecture's upper-bound. As such, the prediction accuracy is improved by applying a linear scaling factor, based on the previous ratios between simulated and predicted performance.

Table 1: Configurable parameters. Baseline underlined.

Parameter	Units / Comments	Configurations Tested
Memory connection†	VPU connected to	DRAM/<u>L2</u>
MVL†	Bits	2^{9/10/11/12/13/<u>14</u>}
Number of lanes†*	—	1/2/4/8/<u>16</u>
VPU Frequency*	GHz	1.0/1.25/1.5/1.75/<u>2.0</u>
L2 Frequency*	GHz	1.0/1.25/1.5/1.75/<u>2.0</u>

Configurable at: † Design time * Runtime

The described process can be performed independently for each target application. To obtain a design-time architecture capable of meeting the MPR of all applications, we take, for each parameter, the maximum value observed across the per-application optimized architectures. This can then be used as a starting point for determining runtime adaptations for each application, repeating the process with a set of parameters that can be modified at runtime.

3 Evaluation

Experimental Setup: For evaluation, we use the flexible gem5-based RISC-V VPU simulator presented in [11]. This is a highly parametrizable architecture that is a good match for DL applications, operating on large vector registers of a given Maximum Vector Length (MVL). Instructions operate in parallel across multiple vector lanes, each having various functional units. Table 1 lists the parameters we explore, as well as the initial, baseline configuration. We distinguish between parameters that can be modified at design-time and at runtime, optimizing them for area and power respectively. Power and area estimations are obtained by modeling the VPU in McPAT [8] for the 22nm CMOS node. Finally, the effectiveness of the DSE was validated against a brute-force search.

The target applications are sourced from XRBench [7], a DL benchmark suite for Virtual Reality (VR) edge devices. Each application is composed of a set of tasks (i.e. DL models) and defines a Frames per Second (FPS) requirement, which we use as the MPR. We scale down MPRs uniformly, allowing them to be met by the initial VPU configuration and its modest performance.

Architecture Optimization: The sensitivity analysis described in Section 2 was conducted for the targeted VPU and applications, targeting die area and power consumption for design time and runtime parameters respectively. Figure 3 shows its results, with a higher value indicating the parameter is more effective at lowering the target metric for an equal performance impact. These values are used to select the order in which the design space is traversed, starting with the parameter with the higher value in each stage.

Following the established parameter order, the proposed methodology is used to traverse the design space. We first apply our methodology at the design stage, determining the values for static parameters. As the MPRs of all applications need to be met, the most demanding one limits how far the parameters can be reduced at this stage. This leads to a die area of 19.02mm^2, a modest decrease of 4.1% compared to the initial configuration. However, this means a large performance budget is left for the runtime stage, which produces power-optimized runtime configurations for each application. These optimizations reduce power consumption by up to 42.9% compared to the baseline configuration, with an average of

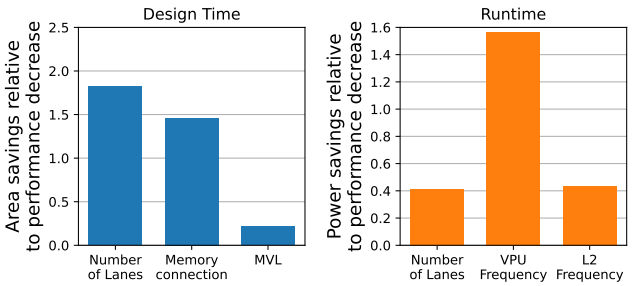

Figure 3: Results from the sensitivity analysis. Impact of each parameter on power and area, normalized to the impact on performance, averaged across all applications.

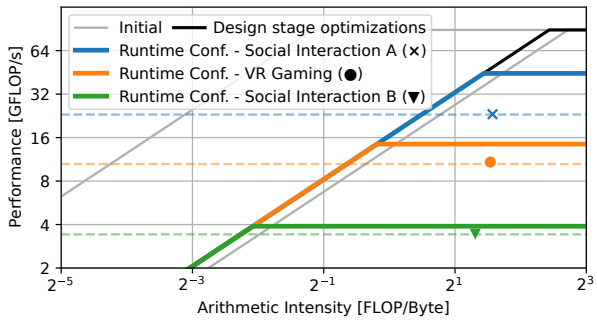

Figure 4: CARM plot showing the rooflines of the initial configuration, the design time optimizations, and the runtime configurations for three out of the seven evaluated applications. The dashed lines represent the respective MPRs.

28.6% across all applications. Despite the steepest descent not providing guarantees, the best configuration is found for all but two applications, with the highest error being only 2.0% over the ideal case. Figure 4 depicts the CARM of the initial configuration, and after the design and runtime optimizations have been applied.

Design Space Exploration Time: Figure 5 shows the traversal of the design space for the *VR Gaming* application, plotted according to the performance and power consumption for the runtime optimization stage. The highlighted path demonstrates the methodology's efficient traversal of the design space, remaining near the Pareto front and finding the configuration with the smallest power consumption that meets the MPR.

Our methodology simulates only 0.09% to 0.23% of the total points in the design space to find the hardware configuration. The proposed CARM-based predictions reduce the number of required simulations by an average of 61.8%, evaluating a configuration multiple orders of magnitude faster than a simulation.

4 Conclusions

This work proposes an effective, yet lightweight and intuitive methodology to efficiently perform architectural DSE for DL applications in edge systems, meeting MPRs while maximizing efficiency. Evaluated on the different applications of the XRBench benchmark suite running on a existing RISC-V VPU, our proposal reaches optimized designs by simulating only 0.09-0.23% of the design space.

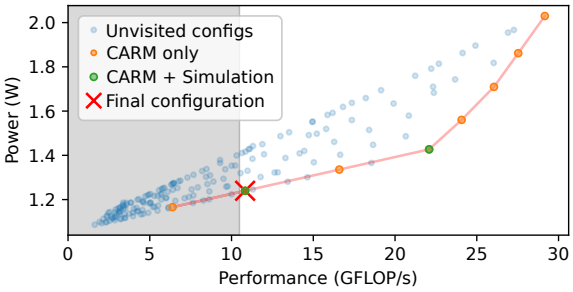

Figure 5: Traversal of the design space with the proposed methodology, displaying performance and power for the runtime configuration. Gray area marks the subspace where the MPR is not met. Starting point is in the top right.

Acknowledgments

This work was supported in part by the National funds through the Fundação para a Ciência e a Tecnologia, I.P. (FCT) under Project UID/50021/2025 and Project UID/PRR/50021/2025; in part by the Swedish Foundation for Strategic Research (contract number CHI19-0048) under the PRIDE project; and in part by the European High-Performance Computing Joint Undertaking (JU) through the eProcessor project (grant agreement No 956702), Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 (EPI SGA2), and the Digital Autonomy with RISC-V in Europe (DARE) Specific Grant Agreement 1 (SGA1) (Agreement 101202459). The JU receives support from the through European Union's Horizon Europe Research and Innovation Program and Spain, Germany, Czechia, Italy, The Netherlands, Belgium, Finland, Greece, Croatia, Portugal, Poland, Sweden, France, and Austria.

References

[1] M Waqar Azhar, M Pericàs, and P Stenström. 2019. SaC: exploiting execution-time slack to save energy in heterogeneous multicore systems. In *Proceedings of the 48th ICPP*, 1–12.
[2] M Waqar Azhar, S Zouzoula, and P Trancoso. 2023. ARADA: adaptive resource allocation for improving energy efficiency in deep learning accelerators. In *Proceedings of the 20th ACM CF*, 63–72.
[3] N Binkert et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39, 2, 1–7.
[4] D Dabbelt et al. 2016. Vector processors for energy-efficient embedded systems. In *Proceedings of the Third ACM International Workshop on Many-Core Embedded Systems*, 10–16.
[5] V Fu, L Zaourar, Alix Munier-Kordon, and M Duranton. 2024. Design space exploration of hpc systems with random forest-based bayesian optimization. In *Proceedings of the 16th RAPIDO*, 9–15.
[6] A Ilic, F Pratas, and L Sousa. 2013. Cache-aware roofline model: upgrading the loft. *IEEE CAL*, 13, 1, 21–24.
[7] H Kwon et al. 2023. XRBench: an extended reality (xr) machine learning benchmark suite for the metaverse. *Proceedings of MLSYS*, 5.
[8] S Li et al. 2009. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd IEEE/ACM MICRO*, 469–480.
[9] D Marques et al. 2017. Performance analysis with cache-aware roofline model in intel advisor. In *2017 HPCS*. IEEE, 898–907.
[10] S Pandey, A Yazdanbakhsh, and H Liu. 2024. Tao: re-thinking dl-based microarchitecture simulation. *Proceedings of the ACM POMACS*, 8, 2, 1–25.
[11] C Ramírez et al. 2020. A RISC-V simulator and benchmark suite for designing and evaluating vector architectures. *ACM TACO*, 17, 4.
[12] A Rodrigues, L Sousa, and A Ilic. 2023. Performance modelling-driven optimization of RISC-V hardware for efficient SpMV. In *ICS*. Springer, 486–499.
[13] N C Thompson, K Greenewald, K Lee, and G F Manso. 2020. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*.