Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

AMORL: A Training-Aware, Adaptive, Multi-Objective RL-CNN Framework for Dynamic DNN Mapping on Networks-on-Chip

Md Farhadur Reza

mreza2@eiu.edu

Eastern Illinois University

Research Article

Keywords: Network-on-Chip (NoC), Deep Neural Network (DNN), Many-Core Systems, Task Mapping, Reinforcement Learning, Convolutional Neural Network (CNN), Multi-Objective Optimization, Energy-Latency Trade-off, Dynamic Workload Adaptation

Posted Date: August 25th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10562561/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

AMORL: A Training-Aware, Adaptive, Multi-Objective RL-CNN Framework for Dynamic DNN Mapping on Networks-on-Chip

Md Farhadur Reza[1]

[1]Department of Mathematics and Computer Science, Eastern Illinois University, 600 Lincoln Ave., Charleston, 61920, IL, USA.

Contributing authors: mreza2@eiu.edu;

Abstract

As transistor technology advances and miniaturizes, single chips or multiple chiplets can integrate hundreds to thousands of cores. Networks-on-Chip (NoCs) have become the standard on-chip communication fabric for many-core systems, offering scalability, parallelism, and power efficiency over traditional buses. These properties enable efficient execution of computations and communications for deep neural network (DNN) layers on NoCs. However, traditional mapping strategies struggle with the diverse communication patterns of DNNs, such as inter-layer communication and one-to-many/many-to-one patterns in fully connected and convolutional layers, necessitating mapping techniques that mitigate NoC communication bottlenecks. This work proposes AMORL, a generalized reinforcement-learning and convolutional-neural-network (RL–CNN) framework for scalable, adaptive mapping of DNN workloads onto heterogeneous NoC architectures. RL learns temporal neuron-to-core mapping policies, while CNN captures spatial communication structure to guide locality-aware decisions. The framework jointly optimizes communication latency, load balancing, and hardware utilization under dynamic workloads. Evaluations across six DNN workloads, ResNet-50, BERT-tiny, MNIST, CIFAR-10, IRIS, and Face Recognition, on NoC sizes from $\mathbf{4 \times 4}$ to $\mathbf{32 \times 32}$ demonstrate that AMORL reduces communication cost by over $\mathbf{70\%}$, load imbalance by up to $\mathbf{86\%}$, and latency by more than $\mathbf{50\%}$ compared to ablated variants, while maintaining energy-efficient operation. Cycle-accurate gem5/Garnet simulations on $\mathbf{4 \times 4}$ and $\mathbf{8 \times 8}$ meshes further validate AMORL's mappings, showing sustained throughput up to $\mathbf{876\ Gbps}$ on the $\mathbf{4 \times 4}$ NoC and over $\mathbf{2.77\ Tbps}$ on the $\mathbf{8 \times 8}$ NoC, confirming the framework's robustness under realistic traffic conditions.

Keywords: Network-on-Chip (NoC), Deep Neural Network (DNN), Many-Core Systems, Task Mapping, Reinforcement Learning, Convolutional Neural Network (CNN), Multi-Objective Optimization, Energy-Latency Trade-off, Dynamic Workload Adaptation

1 Introduction

Deep learning has been revolutionizing applications as well as the systems that execute these applications, including supercomputers such as the the world's fastest supercomputer, El Capitan [1]. Deep neural networks (DNNs) are used to implement the wide range of deep learning applications, including image recognition, natural language processing, and speech synthesis. DNNs can contain thousands to millions of neurons, which create significant computational and communication demands. For example, AlexNet has 650K neurons and 60 million parameters [2], while GPT-3 has a total of 1.18 million neurons and 175 billion parameters [3, 4]. To meet the demands of deep learning, hundreds to thousands, even millions, of cores are now integrated into large-scale systems, either on a single chip or multiple chips, to run DNNs; for example, Cerebras WSE-3 integrated 900K cores in a single processor [5]. DNN workloads also incur significant amount of energy on large-scale system, for example, El Capitan requires approximately 30 megawatts (MW) of power, which is enough run a mid-size city. Their increasing complexity, characterized by multiple interconnected layers and a vast number of neurons, demands highly parallel computing architectures to efficiently process the computations. DNNs have various types of connectivity patterns among neurons, which need to be communicated when mapped to different processing units, called cores, in a system. A typical DNN accelerator executes DNNs composed of fully connected and convolution layers, as shown in Fig. 1. In fully connected layers, each hidden/output neuron uses all input neurons for computation. In convolution layers, each hidden/output neuron is connected to a smaller region of the input neurons through a kernel. Thus, neurons have either global or localized one-to-many and many-to-one communication in these layers. Therefore, communication can be diverse and intensive in DNNs.

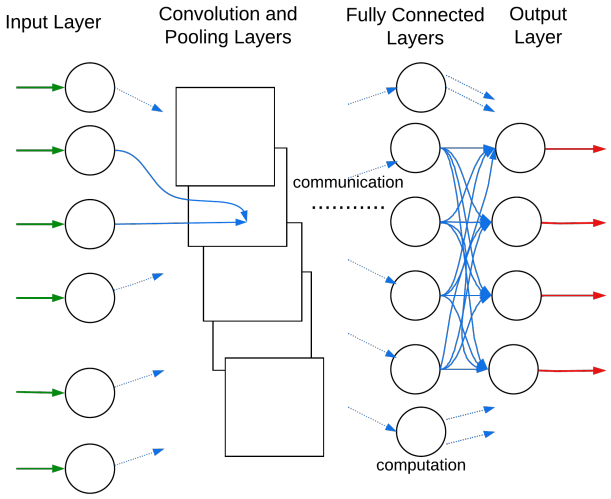


Fig. 1: Computation and communication in a typical DNN

In modern DNN accelerators, communication between distributed processing cores is managed by networks-on-chip (NoCs), which enable rapid data exchange with low latency and high throughput. NoCs carry all the communication, both between cores and between cores and memory, in large-scale high-performance computing systems, such as datacenters, cloud servers, and supercomputers [6–8].

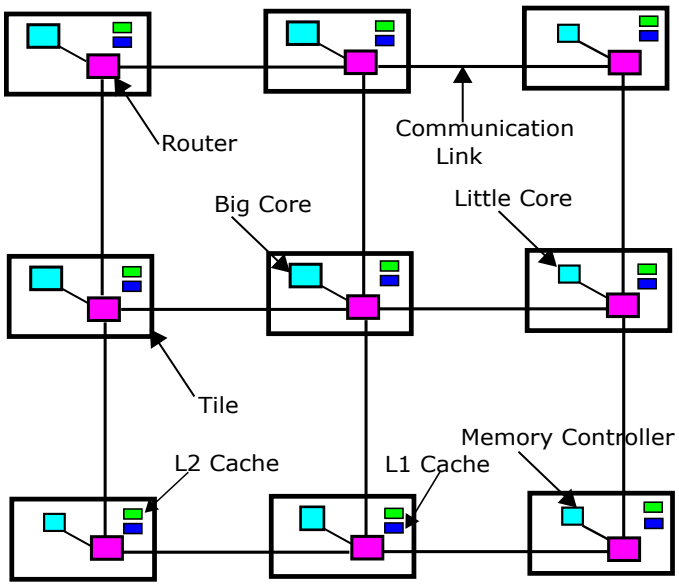


Fig. 2: 2D-Mesh-based heterogeneous NoC architecture

Fig. 2 depicts a multi-core architecture implemented on a 3×3 NoC, with tiles organized in a 2D grid. Each tile contains a processor with distinct computational power, its local L1 cache, a slice of the shared last-level L2 cache, and a router configured with varying communication capacities for data and control transmission between the tiles via links with varying bandwidths. Data moves between cores as it travels from the L1 and L2 caches of one core to another, and from a core to the memory controller to handle memory traffic, and so on. Billions of transistors can now be integrated on a single chip. However, from $90\,\mathrm{nm}$ to $7\,\mathrm{nm}$ transistor technology, the scaling down of interconnection energy efficiency has only reached $1.6x$, in contrast to the $6x$ scaling down of transistors [9]. Because of this disparity, energy consumption in the on-chip systems is increasingly being dominated by the interconnection delay [10]. As the scale and density of DNNs increase, more inter-core interconnections are required, and the impact of inter-core communication quickly becomes a significant design challenge for the NoC. Communication can consume 30% of the overall energy in a deep learning accelerator [11]. This increased communication energy translates directly into localized temperature rises and thermal hotspots on the NoC, which can cause frequency throttling, accelerate device wear, and therefore require thermal-aware strategies [12, 13]. Therefore, accelerating DNNs while ensuring energy and thermal efficiency is the major mapping challenge. Spatial dataflow and on-chip locality strongly affect communication cost and energy;

prior accelerator work (e.g., [14]) highlights how mapping decisions that preserve locality can reduce NoC traffic and system energy. Modern DNN accelerators such as Google's TPU [15] demonstrate how hardware structure (e.g., systolic arrays) changes the relative importance of mapping decisions for latency and energy. NoC architectures have become a widely adopted communication fabric for modern DNN accelerators due to their scalability, high on-chip bandwidth, parallel communication support, and ability to efficiently manage localized and global data movement across many processing elements. These properties enable flexible computation scheduling, improved memory-access regularity, and better utilization of distributed compute units in large neural networks [16–18].

Mapping DNNs onto NoCs poses significant challenges. First, scalable mapping techniques must accommodate the variable dimensions of both NoCs and DNNs, as fixed-size algorithms lack the flexibility required for modern systems. Second, simultaneous optimization of conflicting objectives, such as minimizing latency while balancing computational loads and reducing energy consumption, is a complex multi-objective problem. Finally, the dynamic nature of DNN workloads, where layer characteristics and traffic patterns may change in real time, requires an adaptive approach that traditional mapping strategies face challenges to provide.

Existing mapping strategies are often tailored to a specific DNN architecture and a fixed NoC configuration, achieving high performance for that singular use case. However, real-world accelerator workloads are dynamic and heterogeneous, often involving rapid context switches between different models. A mapper trained exclusively on one dataset (e.g., MNIST) may fail to generalize to another (e.g., a Face Recognition network), leading to suboptimal performance and requiring costly retraining. This work considers two complementary training paradigms: Single-Dataset Training, which specializes a mapper for peak performance on a known workload, and Multi-Dataset Training, which learns a robust, generalizable policy that can adapt to unseen DNNs and dynamic workload mixes without retraining. While we explored both approaches, we focus on the multi-dataset paradigm as it aligns with the practical demands of general-purpose AI accelerators, where workloads are heterogeneous and change over time. To rigorously evaluate our framework's scalability and generalizability, we select six DNN workloads spanning a wide range of sizes and communication patterns: from tiny IRIS (3 layers) and small MNIST (6 layers) to medium CIFAR-10 (8 layers) and Face Recognition (7 layers), and finally to large architectures such as ResNet-50 (53 layers), as well as a modern transformer-based workload (BERT-tiny, 8 layers). This diverse set allows us to assess performance across workloads with vastly different computational demands, communication intensities, and layer topologies—from fully connected networks to convolutional and transformer-based models, ensuring that the proposed approach is not tailored to any specific DNN family. To address the aforementioned challenges and achieve these performance improvements, we propose AMORL (Adaptive Multi-Objective RL-CNN), a generalized reinforcement-learning and convolutional-neural-network (RL–CNN) framework for scalable, adaptive mapping of DNN workloads onto heterogeneous NoC architectures. AMORL exploits the spatial awareness of CNNs and the adaptive learning capabilities of RL to optimize the allocation of DNN workloads onto NoCs, thereby delivering a unified solution for dynamic, multi-model environments. The major contributions of this work are as follows:

- *AMORL Hierarchical Learning Architecture*: We introduce AMORL, a hierarchical framework combining a meta-controller and an RL–CNN policy. The meta-controller dynamically weights five objectives, communication cost, load imbalance, energy efficiency, latency, and a thermal-risk metric, via a learned softmax, yielding an adaptive, context-aware scalar reward. Under the meta-controller, the CNN-based policy learns spatial mapping patterns on the NoC mesh, while the RL agent optimizes temporal sequences of cluster-to-core assignments.
- *Training-Aware, Phase-Specific Mapping*: We extend RL–CNN to be training-aware by encoding forward, backward, and weight-update phases in the state and applying phase-weighted communication penalties (e.g., doubling the cost during backprop). This design allows AMORL to place layers so as to accelerate every phase of DNN training, reducing forward-pass hops, clustering gradient flows, and co-locating update-heavy clusters, without changing the DNN's optimizer (e.g., Stochastic Gradient Descent (SGD)).
- *Fast Online Adaptation*: To handle sudden changes in workload (like a new batch size or a different model), we extract fixed-size workload features and trigger a quick online adaptation step. By performing a few gradient steps on a learned adaptation vector, using top-k similar past experiences, the mapper quickly specializes to new patterns without retraining from scratch.
- *Multi-Objective Reward with Thermal Modelling*: In addition to communication distance, imbalance, and energy, we incorporate per-hop latency and a simple thermal-risk model into a 5-dimensional reward vector. This lets AMORL balance speed (throughput), efficiency, and reliability (thermal safety), avoiding hot spots under heavy compute loads.
- *Scalability Across NoC Sizes and DNN Cluster Counts*: By training on random meshes from 4×4 up to 32×32 and DNNs with 3 to 53 cluster (from tiny IRIS to large ResNet-50), AMORL learns size-invariant, generalizable mapping policies. To promote generalization, we train the DNN mapper jointly on all six workloads, enabling the unified policy to produce mapping solutions for diverse configurations without per-workload retraining. At runtime (inference), it picks mappings in constant time, and the meta-controller adapts automatically to different NoC sizes and changing workloads.
- *Evaluation on Realistic Benchmarks and Platforms*: We conduct extensive evaluations using an in-house developed DNN mapper that generates neuron-to-core mappings for six DNN workloads, Resnet-50, BERT-tiny, MNIST, CIFAR-10, IRIS, and Face Recognition datasets, across NoC sizes ranging from 4×4 to 32×32 NoCs. The mapping traces are fed into the cycle-accurate Garnet NoC model [19] within the gem5 platform [20] for latency and throughput evaluation. The in-house mapper additionally computes communication cost, energy consumption, load imbalance, thermal risk, and related metrics, and supports component-wise comparison studies to analyze the impact of individual AMORL framework components.

Together, these advances yield a unified, adaptive, multi-objective mapping solution that accelerates both inference and training of diverse DNNs on many-core NoC platforms while keeping latency low, saving energy, balancing workloads, and preventing overheating.

Related work is presented in Sec. 2. Then we discuss our proposed RL-CNN framework for mapping DNNs onto NoCs in Sec. 3. The acceleration of DNN training on NoCs and the learning dynamics of the AMORL framework are discussed in Secs. 4 and 5, respectively. An

illustrative example of AMORL-based DNN-to-NoC mapping is presented in Sec. 6. Simulation results in Sec. 7 demonstrate the effectiveness of the proposed RL-CNN-based adaptive multi-objective mappings solution.

2 Related Works

There have been many works that use machine learning (ML), reinforcement learning(RL), and deep learning for various optimizations, such as power and thermal management, load balancing, adaptive routing, and predictive failure maintenance, in NoC-based multi-core architectures [21–31]. However, only a small number of studies have explored machine learning, deep learning, or reinforcement learning for mapping DNNs onto NoC architectures. Existing DNN mapping methods lack adaptability to dynamic NoC sizes. This gap motivates our AMORL approach. In [32], the mapper is designed with heuristics and iterative search strategies to rapidly locate optimal energy/performance spatial and temporal mapping points. [33] proposed a genetic algorithm-based technique for mapping DNNs. [34] proposed NN-aware mapping algorithm to reduce the distance of the data communication and used multicast data transmission to reduce the amount of packets with same data in each routing path. [35] proposed several rules for low latency mapping of NNs onto NoCs. Some of the rules include: neurons of different layers' can not be mapped in the same node, use least number of mapped or working nodes, and placing the neuron-mapped nodes as close as possible. Two mapping strategies for mapping spiking NNs onto NoC are presented in [36]. One strategy uses the Kernighan-Lin algorithm to put highly communicating tasks together. The second strategy performs a balanced distribution of neurons among the cores to reduce congestion. [37] proposed multi-objective evolutionary algorithm based approach to DNNs accelerator mapping space exploration. [38] explored different NoC architectures and mapping solutions for DNNs and also proposed a concentrated-mesh NoC and a load-balanced mapping solution for accelerating DNNs. [39] proposed hardware-based scheduling methods to efficiently schedule dependency-free sub-layer tasks. It maps compute- and memory-intensive tasks from multiple NNs onto accelerator's computation resources and memory bandwidth to execute them in parallel. [40] used layer scheduling to improve layer parallelism in DNN accelerator. Here, a scheduler checks layer dependence and memory constraints and optimize overall latency and energy of multiple DNN workloads running on an accelerator. [41] maps the CNN to the NoC using a two-step process: first, it maps the CNN's computations onto the PEs, and then it maps the PEs onto the NoC using a genetic algorithm. [16] present Integer Linear Programming (ILP) and simulated annealing (SA)-based neuron grouping solutions with the objective of minimizing the total volume of data communication among the neuron groups in both fully connected and partially connected DNN models and also minimizing the total communication cost of the system. [42] proposes a multilevel mapping approach for DNNs onto NoCs. First, it selects a specific region for mapping each layer, and in the second step, the neurons are mapped onto particular cores to improve performance in terms of latency and energy consumption. [43] propose a mapping strategy to unevenly distribute tasks of DNN onto PEs and reduce time consumption by balancing workload in NoCs. They also investigate the impact of different NN layers, communication protocols, and hardware platforms by analyzing the performance of a layer under various conditions, including packet size and NoC architecture.

Unlike most existing mapping works, our proposed RL-CNN-based framework, called AMORL, provide generalized mapping solutions to dynamically map any kind of DNNs (with any number of layers and neurons) to any kind of NoC architectures. AMORL also provides multi-objective mapping multi-objective mapping that jointly considers energy, latency, load, and thermal factors. Prior work like MAESTRO [44] has excelled at providing fast, accurate cost models for evaluating intra-layer dataflow mappings on specialized accelerators. In contrast, our work focuses on generating inter-layer spatial mappings for general-purpose NoCs, a problem that involves optimizing communication across an entire network of layers rather than data reuse within a single kernel.

3 Proposed RL-CNN Framework for Mapping DNNs on NoCs

The proposed RL-CNN framework follows multiple steps to learn optimal mapping policies for DNN workloads and adapt to dynamic changes in DNN workloads. Initially, the framework analyzes the DNN workload to determine both the computational and communication requirements of each neuron layer in DNN. Each DNN layer (convolutional, pooling, or fully connected) is modeled as a cluster, with computational demand quantified in floating-point operations per second (FLOPs) and communication requirements using the data dependencies between layers (in bytes exchanged with the next layer). Convolutional, pooling, and fully-connected layers each contribute a unique FLOP/communication pattern: convolutions drive local, overlap-friendly traffic; fully-connected layers generate costly all-to-all exchanges; and pooling or normalization layers incur minimal data movement. By converting these raw counts into a normalized compute vector and a communication intensity matrix, we obtain a fixed-size workload representation that can scale from small networks (e.g., IRIS with 3 layers) to very deep ones (e.g., ResNet-50 with 53 layers). This unified representation enables AMORL to learn a single policy that generalises across DNNs of vastly different sizes and structures.

After DNN workload analysis, next we feed this description into our RL–CNN policy. The framework uses RL to develop dynamic policies for neuron clusters-to-cores mapping. The clustering of multiple neurons into a single mapping decision offers a manageable and efficient approach to optimize core utilization and communication in large DNNs. RL is trained with various NoC configurations from 4×4 to 32×32. By training across diverse NoC sizes, the system learns generalizable mapping policies that balance communication, load distribution, and energy efficiency. The features of state of NoC resources and application demands for training RL include normalized cluster sizes (number of neurons), inter-cluster communication intensities, and core utilization. A CNN processes these state information to detect spatial patterns across the NoC, especially identifying regions experiencing high communication density and cores that are underutilized. The CNN outputs a probability map of shape $\text{clusters} \times \text{cores}$. Each entry (in the probability map) reflects the likelihood of mapping a particular DNN cluster to a given core, which guide the RL agent in its decision on mappings. We then sample or greedily select a one-to-one mapping from these probabilities and apply it to the environment, which computes the new core utilizations, total communication cost (the sum of communication intensities weighted by Euclidean distances), load-imbalance (standard deviation of utilizations), and energy consumption (compute + communication). These metrics are combined into a single, multi-objective reward to train the RL agent. Energy and

communication cost modeling follows established analytical tools (e.g., [44]), which compute communication bytes $\times$ hops and energy estimates from mapping decisions.

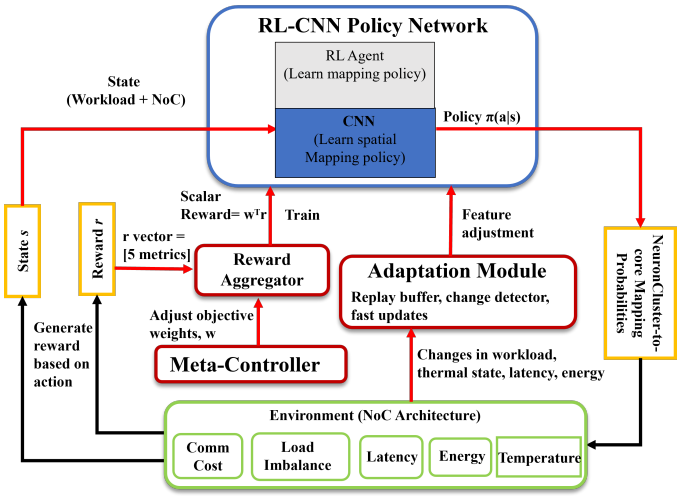


Fig. 3: RL-CNN based AMORL Framework for Mapping DNNs on NoCs.

Fig. 3 shows the AMORL framework, showing the RL-CNN policy network, meta-controller, adaptation module, and NoC environment. The details of the RL-CNN framework are presented in the following sections.

3.1 Performance Metrics

We have evaluated several metrics to assess the performance of our DNN mapping solutions. The load imbalance metric quantifies how evenly the computational load is distributed across the NoC cores. Computational load distribution per core is measured by computing the standard deviation of the core utilization. A high standard deviation indicates that the load is unevenly distributed, which is undesirable. The load imbalance is mathematically defined as:

$$\text{LoadImbalance} = \sum_{i=1}^{N} (u_i - \bar{u})^2 \tag{1}$$

where, u_i represents the utilization of core i, $\bar{u}$ denotes the mean utilization across all cores, and N is the total number of cores.

The total communication cost metric is designed to capture the cumulative latency incurred by inter-cluster communications. For each pair of clusters, the cost is determined by multiplying the Euclidean distance between the cores where they are mapped by the communication intensity between those clusters. The overall communication cost is the sum of these

products over all cluster pairs, as given by

$$\text{TotalCommCost} = \sum_{i=1}^{C} \sum_{\substack{j=1 \\ j \neq i}}^{C} \text{Comm}_{i,j} \cdot \text{Distance}_{i,j} \tag{2}$$

where, C is the total number of clusters, $\text{Comm}_{i,j}$ represents the communication intensity between clusters i and j, and $\text{Distance}_{i,j}$ is the Euclidean distance between mapped cores.

Energy efficiency is another metric for evaluating mapping performance. Energy efficiency is defined as the average energy consumed per unit of communication. The average energy per communication is computed by dividing the total energy consumption by the number of communication pairs, where the total energy is the sum of the products of communication volumes, corresponding distances, and energy consumption per byte $Energy_{byte}$. This is expressed mathematically as:

$$\text{Avg_energy/comm} = \frac{\sum_{i,j} \text{Comm}_{i,j} \cdot \text{Distance}_{i,j} \cdot \text{Energy}_{byte}}{\text{num_clusters} \cdot (\text{num_clusters} - 1)} \tag{3}$$

3.2 DNN Workload Integration

3.2.1 Cluster Modeling

Each DNN layer (convolutional, pooling, or fully connected layer) is modeled as a cluster characterized by two primary properties. The first is the computation demand, which is measured in FLOPs and is normalized to the range $[0, 1]$ based on the complexity of the layer. For a fully connected layer, the number of input and output neurons are used to calculate the FLOPs. For a convolutional layer, FLOPS are computed based on kernel size, channels, and spatial dimensions. For a pooling layer, the number of channels, kernel size, and height are used to compute the FLOPs. The second is the communication requirement, which is derived from the data dependencies between consecutive layers. For every layer except the last, the communication cost in terms of data size to the next layer is calculated. Since each output element is stored as a float32, multiplying by 4 gives the total number of bytes that must be transferred between layers. These values are scaled appropriately to prevent numerical instability. For example, a convolutional layer that requires 2 million FLOPs and has 3000 communication units might be mapped to a high-performance core that is located near the pooling layer that depends on its output.

3.2.2 Workload Selection for Evaluation

To comprehensively evaluate AMORL's adaptability across the spectrum of modern neural network architectures, we select six DNNs that cover a wide range of model sizes, communication patterns, and layer types: IRIS (3 layers, fully connected(FC)), MNIST (6 layers, convolution (conv)+FC), CIFAR-10 (8 layers, conv+FC), Face Recognition (7 layers, conv+FC), BERT-tiny (8 layers, transformer), and ResNet-50 (53 layers, deep convolutional with skip connections). This selection spans from tiny, fully connected networks to large convolutional and transformer-based models, ensuring that AMORL's learning-based mapping

policy is not tailored to any single architecture and remains effective across the heterogeneity of real-world DNN workloads.

3.2.3 Training Paradigms: Single Dataset vs. Multi-Dataset

The AMORL framework supports two distinct training modalities, each with its own advantages and target applications.

- *Single-Dataset Training*: In this paradigm, the RL agent and meta-controller are trained exclusively on workloads generated from a single DNN architecture (e.g., only CIFAR10 for thousands of episodes). This allows the model to deeply specialize, learning the precise computational and communication patterns of that specific network. The meta-controller converges to a static, optimal weighting of objectives (e.g., perhaps prioritizing load imbalance for this particular DNN's layers) and the policy learns a highly-tuned spatial mapping strategy. The result is often the highest possible performance for that specific workload but at the cost of brittleness; the model may perform poorly if the workload changes.
- *Multi-Dataset (Adaptive) Training*: This paradigm mirrors real-world conditions by randomly interleaving episodes from multiple, diverse DNN workloads (e.g., ResNet-50, BERT-tiny, MNIST, CIFAR-10, IRIS, and Face Recognition) throughout training. This forces the policy and meta-controller to become generalists. The meta-controller must learn to dynamically reinterpret its weighting strategy based on the system state, which now implicitly encodes the type of workload. It learns a more nuanced policy that balances objectives in a workload-aware manner. While convergence can be slower and more volatile due to conflicting gradients from different tasks, the resulting model is inherently robust and adaptive. It can handle unforeseen DNNs and shifting workload mixes at inference time, making it the superior choice for general-purpose AI accelerators.

For the experimental results presented in this paper, we adopt the multi-dataset paradigm, as it reflects the unpredictable, multi-model nature of real-world AI workloads. The single-dataset approach was also investigated but is not reported here.

3.2.4 Layer-to-Layer Variability in DNNs

DNNs are a heterogeneous assemblage of computational stages, each with its own characteristic mix of arithmetic intensity and data movement. Convolutional layers perform vast numbers of multiply-accumulate operations (MACs) on small $k \times k$ patches: every output pixel is computed by sliding a compact kernel over the input feature map. Because those input patches overlap heavily, convolution induces predominantly local traffic; each core needs only the few neighboring activations that fall under its kernel window. By contrast, fully-connected layers exhibit an all-to-all communication pattern: each output neuron must read every activation produced by the previous layer, producing global traffic that can traverse the entire NoC mesh. Between these extremes lie pooling and normalization operations, which incur relatively few FLOPs and minimal inter-core data exchange, and the attention mechanisms of transformer models, whose data-flow graphs vary at runtime with sequence length and token interdependencies, producing highly irregular and dynamic communication demands. Because each layer's ratio of compute to communication, its dependency graph, and its opportunities for data reuse differ so significantly, a static, one-size-fits-all mapping strategy will

either underutilize compute resources or create hot-spot congestion when faced with layers whose patterns do not match the assumption built into the mapping.

3.2.5 Batch Size and Data-Dependent Variations

In practice, the per-layer workload is also shaped by the size and nature of the input data. A larger batch multiplies both the number of MACs and the volume of activations that must be exchanged; doubling the batch size doubles the on-chip traffic for every layer, amplifying any existing communication bottlenecks. Moreover, many modern architectures, such as those for natural language processing (NLP), process inputs of variable length, and ReLU or other sparsity-inducing activations can zero out large fractions of intermediate tensors. Those zeroed activations needlessly occupy NoC bandwidth if not handled intelligently; conversely, a sparse layer can often be mapped with far fewer data exchanges, presenting an opportunity for on-the-fly remapping. In sum, the NoC must accommodate both predictable load scaling with batch size and unpredictable shifts in tensor shape and sparsity, or else suffer energy and latency penalties from misaligned resource allocation.

3.2.6 Runtime Adaptation and Multi-Model Workloads

Finally, real accelerators operate in multi-model, multi-task environments, rather than executing one isolated DNN continuously. They handle streams of diverse models, such as image classification workloads in one moment, semantic segmentation in the next, followed by object detection, each network varying in depth, layer types, and inter-layer connectivity. Sudden spikes in demand for one model can overload cores that were previously idle or lightly loaded, while hardware conditions (such as thermal throttling or occasional core failures) can change the available compute and communication fabric in real time. An effective mapping solution must therefore go beyond a static assignment designed for a single network graph; it must continuously sense the current mix of active models, the prevailing batch sizes and data shapes, and the instantaneous health of the NoC resources, then adapt its placement policy to ensure balanced utilization, low communication latency, and energy-efficient operation under changing conditions.

3.3 RL-CNN Framework Details

3.3.1 State Representation

The state representation of the environment for RL includes three-channel feature data. The first channel holds the normalized compute cost (FLOPs) for each neuron cluster. The second channel represents the communication intensity between dependent clusters. The third channel shows the current utilization of the cores, which is updated dynamically as mappings are applied. The utilization is normalized to ensure numerical stability during training.

3.3.2 Policy Network Architecture

The policy network architecture is based on a CNN that uses spatial convolutions to learn NoC resource-aware mapping patterns. The CNN has two convolution layers and one fully connected layer. ReLU activation in convolution layers introduces nonlinearity. The fully connected network takes the flattened feature map from $feature_net$ and outputs a vector,

which is the product of the number of neuron clusters (grouped by layer) in DNNs and the total number of cores in NoCs.

The decoder component of the CNN produces an $n \times n$ probability map (for an 8×8 NoC, this map contains 64 elements), where each value in the range 0 to 1 indicates the likelihood of assigning a particular cluster to that core. The forward function in CNN defines how the network processes an input state that is passed through the $feature_net$ and then flattened. The softmax function is applied along the last dimension to convert logits into a probability distribution over the cores. A mapping function converts the probability outputs from the policy network selects the highest-probability core for neuron cluster mapping.

3.3.3 Reward/Loss Functions and Training

The RL-CNN training process involves designing a reward function r_t that combines multiple objectives, formulated as:

$$\begin{aligned} r_t = -\text{TotalCommCost} - 10 \cdot \sigma(\text{Utilization}) \\ \cdot\text{LoadImbalance} - 0.5 \cdot \text{TotalEnergy} \end{aligned} \tag{4}$$

where, TotalCommCost measures the sum of all cluster communication volumes weighted by the distances between their assigned cores, LoadImbalance indicates how evenly the computational load is distributed across cores, TotalEnergy is the total energy consumption in the NoC, and $\sigma(\text{Utilization})$ represents the standard deviation of core utilization.

In this framework, a positive reward indicates the agent's action improved performance metrics (reduce communication distance/energy/load imbalance), reinforces the agent to repeat similar actions. Similarly, a negative reward (penalty) indicates the agent's action led to a suboptimal results, such as increased energy consumption or latency, and explicitly tells the agent to avoid similar actions in the future.

The loss function implements a policy gradient. Policy gradient calculates log-likelihood of chosen actions and inverts sign to convert likelihood maximization, resulting in loss minimization problem. It also scales gradients by reward differences R_{diff} between the current reward and baseline reward, $R_{diff} = r_t - baseline$. A baseline reward is maintained and updated using exponential smoothing, $baseline = 0.9 \cdot baseline + 0.1 \cdot r_t$.

$$L = -\mathbb{E}\left[\log \pi(a \mid s) \cdot R_{diff}\right] \tag{5}$$

where, $\pi(a \mid s)$ is the policy action probability of taking action a given the state s, and R_{diff} is the reward differences from the baseline for the episode. An episode refers to a complete cycle of interaction between the RL agent and the NoC environment. if R_{diff} is large, the gradients can also become very large. Excessively large gradients can lead to unstable updates of the network parameters, resulting in the divergence of the training process, known as the gradient explosion problem. To address this, gradient scaling/clipping is applied during training to limit parameter updates within a stable range.

In our RL-CNN training process, each episode in training uses a random NoC size (e.g., 4×4 to 32×32). The focus is generalization that forces the model to adapt to any size immediately. The proposed mapping framework can map arbitrary NoC sizes, for example, it

works on 8×8 and 32×32 equally. The framework does not have any dependency on predefined NoC size sequence. The mapping framework can discover NoC size-invariant patterns for mapping. However, the challenge is that it may have slower convergence because it needs more training samples to avoid suboptimal local minima solutions. The mapping framework offers scalability through constant-time ($O(1)$) action selection during inference. Its dynamic adaptation capabilities allow the model to learn real-time adjustments.

3.3.4 Fixed-Size Pattern Extraction for Change Detection

To enable on-the-fly detection of significant workload shifts, we extract from each $N \times N$ state tensor a fixed-length workload pattern via adaptive pooling. Concretely, we take the compute channel of the state (channel 0), apply a 2D adaptive average pool to 16×16, and flatten to a 256-element vector. This workload pattern is both compact and resolution-invariant, so it can be compared across NoC sizes (4×4 to 32×32) for fast similarity or change-detection operations.

3.3.5 Multi-Objective Environment Metrics

Beyond communication cost, load imbalance, and energy, our updated environment now also tracks per-hop latency and a simple thermal risk model. At each mapping step, we accumulate:

- Latency as the maximum distance (in $\text{hops} \times \text{per-hop delay}$) seen across all communicated cluster pairs.
- Thermal risk by incrementally heating each core proportional to its compute energy and a thermal coefficient, and summing the on-chip temperature array.

These additional scalar metrics are returned as a 5-element reward vector r, enabling a richer adaptation stage. r is the reward vector NoC environment returns at each step.

$$r = \left[r_{\text{comm}},\ r_{\text{imbalance}},\ r_{\text{energy}},\ r_{\text{latency}},\ r_{\text{thermal}}\right]$$

3.3.6 Dynamic Weighting via Meta-Controller

To flexibly trade off these five objectives under varying system conditions, we introduce a small meta-controller network. The meta-controller is a small neural network that sits above the main RL-CNN policy and dynamically decides how much to emphasize each of our multiple objectives (communication cost, load balance, energy, latency, and thermal risk) at any given moment. It (meta-controller) takes as input a normalized 5-element system state, including current power budget, thermal headroom, achieved latency service-level, workload entropy (from change-detector, how surprising the new workload is), and time since last change (how long it's been since the last detected change), and outputs a 5-way softmax of objective weights, w. This dynamic weighting approach allows us to navigate the Pareto front of objectives, a core challenge in multi-objective sequential decision-making [45].

$$w = \left[w_{\text{comm}},\ w_{\text{imbalance}},\ w_{\text{energy}},\ w_{\text{latency}},\ w_{\text{thermal}}\right]^{\top}$$

During training, we combine the weighted reward vector into a single scalar r_{AMORL}, as shown in Eq. 6. It produces a normalized 5-element weight vector w, one weight per objective, as shown in Eq. 6. Instead of fixing a single reward formula once and for all, we use these weights to compute a weighted sum of our five raw metrics. This lets the system, for example, temporarily prioritize thermal safety if the chip is running hot, or favor latency when a real-time deadline approaches.

$$\begin{aligned} r_{\text{AMORL}} = \mathbf{w}^\top \mathbf{r} = w_{\text{comm}}\, r_{\text{comm}} + w_{\text{imbalance}}\, r_{\text{imbalance}} \\ + w_{\text{energy}}\, r_{\text{energy}} + w_{\text{latency}}\, r_{\text{latency}} + w_{\text{thermal}}\, r_{\text{thermal}} \end{aligned} \tag{6}$$

w is the weight vector produced by the meta-controller computes using a softmax operation, which ensures all its entries sum to one. We then apply w to our multi-objective reward during the policy-gradient update. At regular intervals, we also train the meta-controller itself by sampling similar past experiences via a cosine-similarity replay buffer and maximizing the alignment between its weights and the normalized multi-objective returns. During the main RL loop, we hold the meta-controller's outputs constant for each episode but periodically update its parameters by replaying past experiences: we measure how well its chosen weights would have performed on similar workloads, and back-propagate to improve its future weighting decisions. The meta-controller acts like a tactical manager: it monitors high-level signals about system health and workload demands, then adjusts the weights on the multi-objective reward so that the base RL-CNN policy always learns in the context most important to the current conditions.

3.3.7 Adaptation Module in the Policy Network

To react quickly when workloads shift, we augment the base CNN with an adaptation head. During a detected change, we retrieve the top-k most similar past experiences from the replay buffer and perform a few gradient steps to fine-tune a small 128-dim adaptation vector. This vector is added to the CNN's flattened features before the policy head, effectively steering the model toward mappings that worked well on similar patterns. At inference time, the same online adaptation entry point lets us recompute this vector and immediately adjust the policy without retraining all weights. This fast adaptation step is inspired by model-agnostic meta-learning (MAML) techniques [46] and is designed to mitigate catastrophic forgetting [47], allowing the policy to specialize to new workloads without degrading performance on previously learned ones.

4 Training-Aware Mapping

To accelerate the training of DNNs on a NoC, we extend our RL-CNN mapper to recognize and optimize for the distinct communication and compute patterns that occur during the forward pass, the backward pass (gradient propagation), and the weight-update phase (without modifying the underlying training algorithm). By modeling these phases explicitly, encoding them in the mapper's state, and adjusting the reward to emphasize the most critical bottlenecks, we enable the learned placement policy to reduce overall epoch time solely through hardware-aware placements (without altering the DNN's own training algorithm).

4.1 Phase-Specific Workload Characterization

During training the network does not only send forward activations from layers i to $i+1$; it must also propagate backward gradients from layers $i+1$ to i to compute weight updates. Those backward tensors are roughly the same size as the forward feature maps, so the NoC sees roughly one activation-sized transfer in the forward pass and another in the backward pass, effectively doubling activation traffic compared with inference. Reconfigurable dataflow accelerators (e.g., [48]) show that different dataflow choices (weight- vs. activation-stationary) change which data move at runtime, motivating a mapping approach that adapts to the dominant data movement pattern. Because our mapper assumes weights are resident on cores, namely weight-stationary, this doubled traffic is dominated by activations/gradients rather than by weights. Thus, in training, each layer i incurs three types of computation and communication:

- *Forward activations*: computing the layer output, moving feature maps from layer i to $i+1$.
- *Backward gradients*: propagating error gradients, moving data from layer $i+1$ back to i.
- *Weight updates*: synchronizing updated parameters (e.g. via an all-reduce) across cores or to a central memory controller.

We represent these demands in our workload generator by extending the communication matrix C to three matrices C^{fwd}, C^{bwd}, C^{upd}. Concretely, for a layer of size n_i:

$$C^{\text{fwd}}_{i,j} = \begin{cases} 4 \cdot \text{output_size}(i), & j = i+1, \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

$$C^{\text{bwd}} = \left(C^{\text{fwd}}\right)^T, \tag{8}$$

$$C^{\text{upd}}_{i,i} = 4\, n_i. \tag{9}$$

where the weight synchronization cost scales with the size of the layer. It is important to note that backward passes dominate training time (60–70%) due to gradient synchronization bottlenecks.

4.2 Encoding Training Phase in State

To inform the CNN about which phase is being mapped, we augment the state tensor with a one-hot phase indicator for forward, backward, and update phases:

$$S \in \mathbb{R}^{6\times N\times N} = \left[\, C^{\text{compute}},\ C^{\text{fwd}},\ U^{\text{util}},\ P^{\text{fwd}},\ P^{\text{bwd}},\ P^{\text{upd}} \,\right] \tag{10}$$

Here, $C^{compute}$ is the normalized FLOPs map, C^{fwd} is the forward-pass communication intensity, U^{util} is the current core utilization, and $P^{\text{fwd}},\ P^{\text{bwd}},\ P^{\text{upd}}$ is a one-hot encoding of the phase: During a forward pass,

$$P^{\text{fwd}},\ P^{\text{bwd}},\ P^{\text{upd}} = (1,0,0).$$

During backward, $P^{\text{fwd}},\ P^{\text{bwd}},\ P^{\text{upd}} = (0,1,0)$. During update, $P^{\text{fwd}},\ P^{\text{bwd}},\ P^{\text{upd}} = (0,0,1)$.

Thus the 6-channel state

$$S \in \mathbb{R}^{6\times N\times N}$$

consists of normalized compute, forward-pass communication, current utilization, and phase flags. Using a one-hot flag lets the convolutional filters learn phase-specific spatial patterns, for example, tightening placements around gradient-flow corridors during the backward phase, or clustering weight-update layers near memory during the update phase, without conflating those patterns with the base compute, communication, and utilization signals. This lets the CNN learn distinct spatial mapping strategies for each training phase in a single unified network.

4.3 Phase-Weighted Reward Function

Because the backward pass often dominates training time, we weight the communication penalty according to the current phase. Let d_{ij} be the Euclidean distance between cores assigned to clusters i and j. We compute the communication cost for each phase

$$\phi \in \{\text{fwd}, \text{bwd}, \text{upd}\}$$

as

$$\text{CommCost}^{\phi} = \sum_{i\neq j} C_{ij}^{\phi}\, d_{ij} \tag{11}$$

The scalar reward r_t at step t becomes

$$r_t = -\,w^{\phi}\,\text{CommCost}^{\phi} \;-\; 10\,\sigma(\text{util}) \;-\; 0.5\,E_{\text{total}} \tag{12}$$

where, w^{fwd}=1.0, w^{bwd}=2.0, and w^{upd}=1.5. By doubling the backward-pass penalty, the policy emphasizes minimizing gradient-propagation hops, which directly speeds up backpropagation.

4.4 Integration into the Training Loop

In practice, before each training epoch, we perform three mapping calls, one for each of the forward, backward, and weight-update phases, so that each pass runs with an optimized mapping generated by the current AMORL policy. This nested loop (epoch over phase) interleaves DNN forward, backward, and update computations with policy-gradient updates of the mapper, ensuring that placements continually adapt to evolving training dynamics (batch size changes, sparsity, phase shifts). The resulting mapping for each phase is generated by the current AMORL policy, as shown in Algorithm 1.

By extending our RL-CNN mapper to be training-aware, we leverage its spatial and temporal learning capabilities to accelerate every phase of DNN training without touching the network's SGD routines. Specifically, before each forward, backward, and weight-update phase, we regenerate the layer-to-core mapping so that the policy can place successive activation transfers, gradient flows, and parameter synchronizations on the most favorable cores. During the backward pass, when gradient propagation bottlenecks dominate, the one-hot

phase flags guide the CNN to compress hop distances along the critical error-flow corridors and evenly distribute backprop compute across underutilized tiles. In the update phase, the mapper clusters weight-update, heavy layers close to memory controllers or aggregate nodes, minimizing all-reduce latency. If the batch size grows or activation sparsity spikes mid-training, our wavelet-based change detector triggers a lightweight online adaptation step, smoothly adjusting the policy's feature offsets and remapping layers without interrupting the SGD loop. This clear separation of concerns, where the RL-CNN handles where to run each layer and the DNN framework handles how to update its parameters, yields tangible benefits: for example, on our NoC-based accelerator ResNet-50 epoch times fall from $86.5\,\mathrm{ms}$ down to $59.2\,\mathrm{ms}$ while maintaining identical accuracy.

Algorithm 1 AMORL Framework-Based DNN Mapping with Online Adaptation

Input: Maximum epochs M
Output: Trained AMORL mapping policy π^* (including meta-controller and adaptation)

for epoch $= 1$ to M **do**
  for all $\phi \in \{\text{forward}, \text{backward}, \text{update}\}$ **do**
    state $\leftarrow$ env.get_state(ϕ)
    mapping $\leftarrow \pi(a \mid s)$ ▷ RL–CNN policy
    EXECUTEPHASE(ϕ, mapping)
    $\mathbf{r} \leftarrow$ env.compute_metrics(mapping)
    $R \leftarrow \mathbf{w}^\top \mathbf{r}$ ▷ Reward aggregation
    POLICYUPDATE(R)
    if ADAPTATIONMODULE.DETECTSHIFT(state) **then**
      policy $\leftarrow$ ADAPTATIONMODULE.FASTUPDATE($\cdots$)
    end if
  end for
end for

5 Learning Dynamics in AMORL Framework

5.1 Temporal Learning in AMORL Framework

In the RL-CNN-based AMORL framework, the agent makes a series of mapping decisions: layer after layer, clusters of neurons are assigned to cores in turn, and each choice both shapes and is shaped by those that came before. At step t, the agent observes the current NoC state, compute demands, communication intensities, and accumulated utilization, and selects an action a_t, namely map layer t to core (x, y). Once that assignment is made, the environment updates core utilizations and recalculates communication costs, yielding a new state s_{t+1}. The very same policy then chooses the next action a_{t+1}, and so on until all NN layers are placed. The final reward reflects the whole chain of choices: a bad early decision can force long data paths later, while a good early placement (e.g., lower-latency and energy-efficient) makes all remaining layer mappings provide higher performance and energy-efficiency.

Because every placement changes both load balance and distances between cores, the agent is learning in a changing situation where each decision affects what comes next. In

the beginning, the policy explores randomly and often gets very poor rewards. Over time, we keep a running average of past rewards as a baseline. By subtracting that baseline from each new reward, we get an advantage signal that tells us whether a mapping was better or worse than usual. In the learning phase, the policy-gradient update increases the probability of action sequences that yield less-negative cumulative penalties, while suppressing those that incur heavy latency or energy costs. Random NoC sizes, different DNNs, and shifting load patterns force the policy to discover strategies that place high-traffic layers close together, spread heavy compute across the mesh to avoid hotspots, and cut overall communication energy through careful temporal ordering.

5.2 Spatial Learning in AMORL Framework

Complementing temporal learning in AMORL Framework, the CNN component gives the agent spatial intelligence by treating the NoC as a three-dimensional grid: two spatial dimensions of cores and one channel dimension encoding compute, communication, and utilization features. This image of the NoC is fed through successive convolutional layers. The first convolution identifies local compute islands, detecting small clusters of high FLOP demand much like edge detectors in vision; a pooling layer then abstracts these local patches into broader communication regions, coalescing information about where data streams concentrate. A deeper convolutional layer integrates these neighborhoods into global dependencies, recognizing distant but heavily communicating core pairs and the corridors those data flows will traverse.

Through back-propagation driven by the multi-objective reward, the CNN learns to spot three critical spatial patterns. Hotspot detection flags over-utilized core clusters, biasing the policy against further placement there. Communication corridor recognition uncovers the frequent paths linking producer-consumer pairs, guiding the policy to map new layers adjacent to existing ones and thus curb hop counts. Finally, dead-zone identification locates under-utilized regions that can absorb new compute load without congestion. By fusing these spatial features with the RL agent's temporal sequencing, the framework produces mapping policies that respect the mesh topology, for example, that a core at (0,0) communicates far more cheaply with (0,1) than with (3,3), while flexibly adapting to different mesh layouts and even link failures.

Together, the temporal and spatial learning modules enable a mapping solution that balances communication latency, load distribution, and energy efficiency: the RL agent sequences layer placements optimally over time, while the CNN ensures those placements align with the geometric and traffic constraints of the NoC fabric.

5.3 Online Adaptation and AMORL Workflow

When the fixed-size pattern extractor and wavelet-based monitor detect a significant shift in the incoming DNN workload signature, we trigger fast adaptation. First, we reset a time since change counter and raise the workload entropy input to the meta-controller. Next, we call our fast adaptation routine: it samples up to 64 past state–action episodes whose signatures best match the new pattern, then runs 3–5 gradient steps on the adaptation vector only, guided by their log-probabilities. This produces a custom adjustment that, when applied in

the AMORL-based mapper's forward pass, quickly shifts the policy toward previously successful mapping configurations for that workload. By combining this quick inner-loop with our slower, baseline policy-gradient outer-loop, AMORL can both generalize across NoCs and rapidly specialize to sudden changes in layer patterns, batch sizes, or multi-model mixes.

6 AMORL Framework-based DNN-to-NoC Mapping Example

To demonstrate the full operation of the RL–CNN-based AMORL framework, including all components shown in Fig. 3, consider a fully connected DNN layer composed of sixteen clusters (CL0–CL15), each containing between 32 and 64 neurons. The mapping process begins with the construction of the state s, which includes the normalized cluster sizes (e.g., a 64-neuron cluster mapped to 1.0), the 16×16 communication matrix describing pairwise communication intensities, and the current NoC state, such as temperature, link utilization, and congestion levels. In this example, clusters CL0 and CL1 communicate with intensity 0.9, CL1 and CL2 with intensity 1.0, and CL5 and CL10 with intensity 0.6, while all other pairs exhibit low interaction (< 0.3). This full state tensor is fed into the RL–CNN policy network, where the CNN first extracts spatial features that capture locality structure in the NoC, identify lightly utilized regions such as corner and edge tiles, and highlight regions susceptible to congestion. These features are then passed to the RL agent, which maintains and updates the mapping policy that evaluates how well different placements are expected to perform.

From these spatial and policy features, the RL–CNN model produces the distribution $\pi(a \mid s)$, representing the probability of assigning each cluster to each of the 64 cores. The RL agent then samples or selects the final mapping action a. For example, because clusters CL1 and CL2 exhibit the strongest interaction, the agent assigns them to adjacent cores such as (3,3) and (3,4) (Manhattan distance 1), while moderately communicating clusters like CL5 and CL10 may be mapped to (2,2) and (2,3). Low-communication clusters such as CL12 are positioned at low-traffic tiles, for instance at (0,0). The policy network might assign high placement probabilities, for instance, 0.95 for core (3,3), 0.92 for core (3,4), and 0.85 for core (4,3), indicating confident spatial preferences derived from the CNN features and the RL mapping policy.

Once the action is executed, the environment (NoC architecture) computes a set of five performance metrics: communication cost, load imbalance, latency, energy, and temperature. These metrics form the reward vector r, which captures multi-objective performance for the current mapping. The vector is then passed to the reward aggregator, which combines the metrics into a scalar reward $R = \mathbf{w}^\top \mathbf{r}$. The weights w are dynamically determined by the meta-controller, which continually adjusts the importance of each objective. For example, if the temperature increases, the meta-controller raises the thermal weight and prioritizes thermally efficient mappings. This allows the framework to adapt reward emphasis based on runtime conditions.

In parallel, the environment also sends operational feedback, such as changes in workload phase, thermal state, latency spikes, or energy fluctuations, to the adaptation module. This module contains a replay buffer that stores recent transitions, a change detector that identifies distribution shifts or abrupt performance degradations, and a fast weight-update engine that

applies lightweight gradient updates to selected layers of the RL–CNN network. When the change detector signals a shift, the adaptation module performs rapid specialization by updating CNN features or policy parameters using only a few samples from the replay buffer. This mechanism prevents performance degradation under dynamic conditions without requiring full retraining.

Finally, the updated features and policy weights feed back into the RL–CNN policy network, completing the learning loop. The combined effect of spatial feature extraction, reward-driven policy optimization, dynamically weighted objectives, and rapid adaptation enables the system to consistently place highly communicating clusters adjacently, distribute low-traffic clusters to peripheral tiles, avoid thermal hotspots, and maintain load balance across the mesh. As a result, the RL–CNN framework-based AMORL produces communication-aware and thermally efficient mappings, reducing hop distances, minimizing congestion, and improving both performance and energy efficiency during DNN execution on NoC-based accelerators.

7 Simulations

We have developed a mapping simulator implemented in Python that includes training DNNs using multiple datasets, modeling computation and communication of NoC resources and DNNs, and training and testing RL-CNN-based mapping solutions. The in-house-developed mapper is configured with a maximum NoC size of 32×32, but during training and testing, the actual NoC size is dynamically chosen between 4×4 and 32×32. The goal is to learn a mapping policy that works well across different NoC configurations. This also allows the mapper to evaluate the adaptability of the mapping solution to various NoC configurations. The mapper uses ResNet-50, BERT-tiny, MNIST, CIFAR-10, IRIS, and Face Recognition datasets to model DNNs. DNNs for ResNet-50, BERT-tiny, MNIST, CIFAR-10, IRIS, and Face Recognition have 53, 8, 6, 8, 3, and 7 layers, respectively. Each DNN layer of neurons is used as a cluster for mapping. Training episodes are configurable number of episodes with random selection of NoC size and workload variations. Every 100 episodes, training statistics (e.g., reward, loss, metrics) are printed for monitoring. The environment's step function returns the reward, next state, and performance metrics. The mapping performance was evaluated on various NoC sizes and across different training episodes using reward, loss, load imbalance, communication cost, latency, and total energy consumption.

The mapping solution results from the in-house-developed mapper are fed to the cycle-accurate NoC model, Garnet [19], on the gem5 [20] platform for evaluating performance. Number of cores and routers are modified to simulate different NoC sizes. 64-core and 16-core NoCs are simulated in gem5/Garnet for latency and throughput evaluations, as the gem5 only allows system with exponential of two for square mesh NoCs. gem5/Garnet configurations for 64-core 2D Mesh NoC are shown in Table. 1. The simulation configurations for gem5/Garnet include the following settings: instruction set architecture = ALPHA, interconnect frequency = $1\,\mathrm{GHz}$, number of virtual networks (VN) = 3, number of virtual channels (VC) per VN = 4, number of buffers per VC = 4, flit width = 128-bit, and XY-routing. Both flit width and link width are set to 128 bits. Each core has a router associated with it for communication with other cores and memory in Garnet, as assumed in the proposed mathematical formulation and mapping algorithms. XY-routing algorithm [49] is used for routing

flits between neurons mapped to nodes in 2D-Mesh NoCs. Data injection rate for the input neurons of the first layer is defined as the number of packets input into each node per cycle. The injection rate between DNN layers is triggered once the neuron has collected all inputs from the previous layer. Latency and throughput of all the solutions are evaluated at different injection rates, ranging from 0.05 to 0.1, to demonstrate the robustness of the algorithms.

Table 1: gem5 and Garnet configurations for the 64-core NoC

Instruction Set Architecture (ISA)	ALPHA
CPU Frequency	2GHz
Interconnect Frequency	1GHz
Interconnection Networks	GARNET
No. of Virtual Networks(VN)	3
No. of Virtual channels(VC) per VN	4
No. of Buffers per VC	4
Network Size	8×8 Mesh
No. of Cores	64
No. of Routers	64
No. of Cores per Router	1
Routing	XY-Routing
Flit Width	128-bit
Max. Packets per Node	50000
Simulation Cycle	1000

7.1 Training Results

We evaluated AMORL, a two-level learning architecture combining a meta-controller and a policy-gradient CNN, on the mapping of six DNN workloads derived from ResNet-50, BERT-tiny, MNIST, CIFAR-10, IRIS, and Face Recognition over 2500 training episodes. Every 100 episodes, we recorded the raw episodic reward (the negative sum of communication distance, load imbalance, and energy cost), the AMORL reward (obtained by combining the reward vector using the meta-controller's learned weights), and the policy-gradient loss.

A single AMORL model was trained jointly on all six workloads by randomly selecting one dataset during each training episode. Consequently, the training log reflects both workload-specific characteristics and the additional variance introduced by interleaved multi-task learning. Nevertheless, two consistent trends emerge throughout training. First, the meta-controller rapidly adjusts the objective weights during the initial few hundred episodes. Starting from its initial distribution, the learned weight vector progressively reduces the emphasis placed on the numerically dominant energy objective while assigning greater importance to latency. Second, despite the stochastic alternation among datasets, the proposed two-level architecture consistently exhibits the three-stage learning behavior observed previously: rapid performance improvement during early training, moderate oscillations during intermediate training, and convergence to a stable plateau during later episodes. This behavior is consistently observed across all six workloads, including the newly introduced ResNet-50 and BERT-tiny benchmarks, demonstrating the generalization capability of the proposed framework.

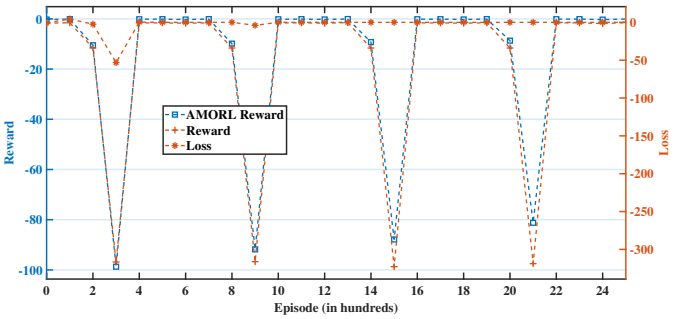


Fig. 4: Reward and Loss in various NoCs for Training DNNs for All the datasets

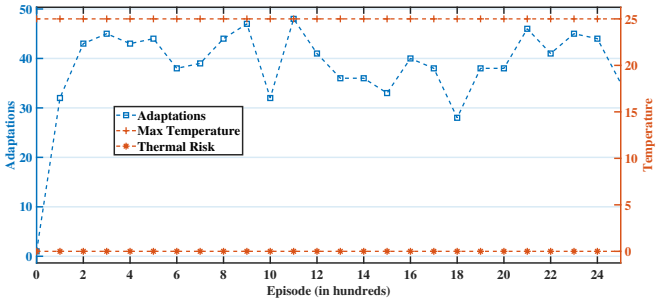


Fig. 5: Adaptations and Thermal Risks in various NoCs for Training DNNs for All the datasets

Representative training episodes further illustrate the importance of objective scaling, as shown in Figs. 4 and 5. At episode 0 (CIFAR-10 on a 19×19 NoC), the raw reward is highly negative ($-441{,}230$), while the corresponding AMORL reward remains comparatively large ($-102{,}914$). These large values are dominated by the energy objective (approximately 4.4×10^5), which is several orders of magnitude larger than the communication, load-imbalance, latency, and thermal objectives. During training, however, the meta-controller gradually reduces the energy weight from approximately 0.233 to 0.144–0.167 (a 28–38% reduction) while simultaneously increasing the latency weight from approximately 0.215 to 0.290 by episode 2500 (approximately 35%). This adaptive reweighting substantially reduces the magnitude of the AMORL reward, even though the raw reward remains dominated by the energy objective, allowing the policy to learn meaningful trade-offs among all optimization objectives.

The policy-gradient loss follows a similar trend. Large-magnitude losses appear during the early stages of training for communication-intensive workloads such as CIFAR-10 and Face Recognition (e.g., $-342{,}489$ at episode 0 and $-602{,}785$ at episode 200), reflecting poor initial mappings and the large gradients induced by the energy objective. The inclusion of ResNet-50 introduces even larger optimization challenges, producing representative losses such as -53.17 at episode 300 because of the substantially higher computational and communication

requirements of the 53-layer network. In contrast, certain episodes, such as episode 100 for the IRIS workload, produce very large positive losses (approximately 1.4×10^6), corresponding to strong corrective updates generated by the policy-gradient algorithm. As training progresses, the loss values gradually converge toward zero across all workloads, indicating that both the policy network and the meta-controller have reached a stable operating point requiring only minor policy adjustments.

The multi-workload training experiments reveal two important observations. First, the learned meta-weights converge to a stable distribution that gradually assigns greater importance to latency while maintaining meaningful weights for communication and load imbalance. Rather than permanently favoring a single objective, the meta-controller learns to balance multiple competing optimization goals dynamically. Second, adaptation events remain non-zero throughout training and vary according to both workload and NoC configuration, demonstrating that the agent continues to adapt its mapping strategy without destabilizing the learned policy. In particular, ResNet-50 consistently requires more policy adaptations (up to 47) than smaller workloads such as IRIS (as few as 28), reflecting the substantially greater computational and communication complexity of the 53-layer network. Meanwhile, the thermal-change flag is never activated during the reported experiments, indicating that all evaluated mappings remain within safe thermal operating conditions. Overall, these results demonstrate that a single AMORL model can successfully learn a unified adaptive mapping strategy capable of efficiently handling highly diverse workloads, ranging from the lightweight IRIS network to the large-scale ResNet-50 model.

7.2 Testing Results

To evaluate AMORL's ability to generalize across different DNN workloads and NoC sizes, we trained the RL-CNN policy using the combined datasets (ResNet-50, BERT-tiny, MNIST, CIFAR-10, IRIS, and Face Recognition) and tested each individual DNN on NoC sizes ranging from 4×4 to 32×32. We measured four metrics relevant to NoC design: load imbalance, total energy, communication cost, and max latency. These metrics cover both structural mapping quality (imbalance, communication distance) and runtime network behavior (energy, latency), allowing us to verify whether AMORL can maintain high-quality mappings even in regimes not seen during training. The results demonstrate clear patterns across datasets and scaling trends, revealing the strengths and limits of AMORL when applied to larger network fabrics.

7.2.1 Load Imbalance

AMORL demonstrates effective load balancing across all six DNN workloads, with load imbalance consistently decreasing as NoC size increases, as shown in Fig. 6. For ResNet-50, load imbalance improves from 0.264 at 8×8 to 0.084 at 32×32, representing a 68% reduction as additional cores provide greater flexibility for distributing the 53 layers. BERT-tiny shows a similar trend, with load imbalance decreasing from 0.241 at 4×4 to 0.031 at 32×32, an 87% reduction. MNIST achieves load imbalance reduction from 0.247 at 4×4 to 0.033 at 32×32 (87% reduction), while CIFAR-10 improves from 0.314 to 0.044 (86% reduction). IRIS shows load imbalance decreasing from 0.263 to 0.035 (87% reduction), and Face Recognition from 0.318 to 0.044 (86% reduction). These improvements are a direct consequence of

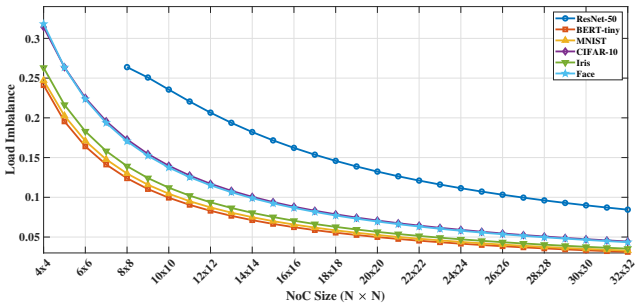


Fig. 6: Load Imbalance in various NoCs for Testing DNNs for six DNN workloads

AMORL's spatial learning capabilities, the CNN component identifies underutilized regions and distributes computational clusters accordingly, while the RL agent sequences placements to avoid overloading specific cores. The consistent reduction across all workloads, from tiny IRIS (3 clusters) to large ResNet-50 (53 clusters), confirms AMORL's scalability and adaptability. Notably, load imbalance values approach near-zero levels at larger meshes (0.031–0.035 for 32×32), indicating near-perfect distribution of computational load across cores. This balanced utilization is critical for preventing hotspots, maximizing throughput, and ensuring efficient resource utilization in NoC-based DNN accelerators.

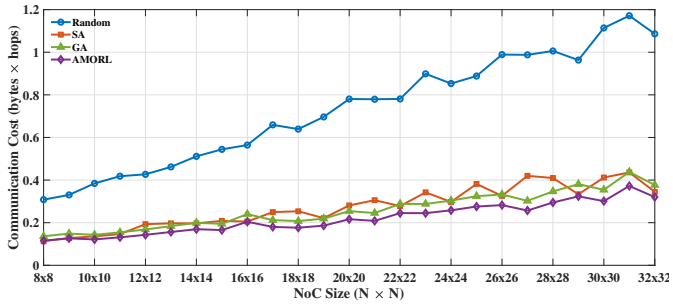


Fig. 7: Communication Cost Comparison among different solutions in various NoCs for Testing DNNs for ResNet-50 dataset

7.2.2 Communication Cost

AMORL consistently achieves significant communication cost reductions across all six DNN workloads and NoC sizes, substantially outperforming Random, Simulated Annealing (SA), and Genetic Algorithm (GA) baselines, as shown in Figs. 7, 8, 9, 10, 11, and 12. The communication benefits are most pronounced for large-scale workloads such as ResNet-50, where AMORL achieves an average communication cost reduction of 66% compared to Random, 32% compared to SA, and 24% compared to GA, with peak savings of up to 68% over Random and 40% over SA at larger NoC sizes. For Face Recognition, AMORL reduces

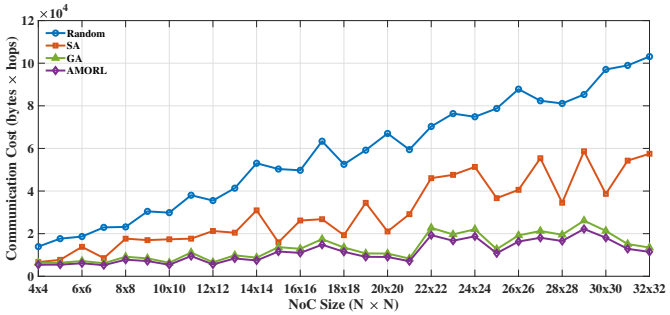


Fig. 8: Communication cost comparison among different solutions in various NoCs for Testing DNNs for BERT-tiny dataset

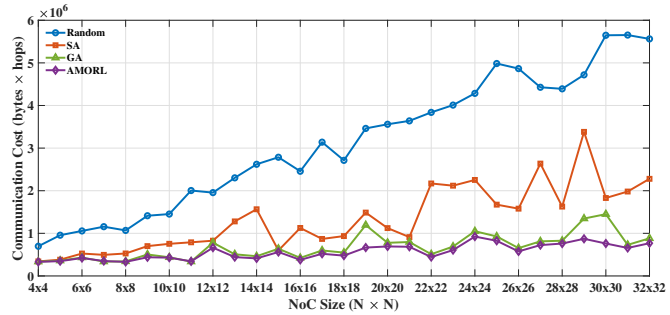


Fig. 9: Communication cost comparison among different solutions in various NoCs for Testing DNNs for CIFAR-10 dataset

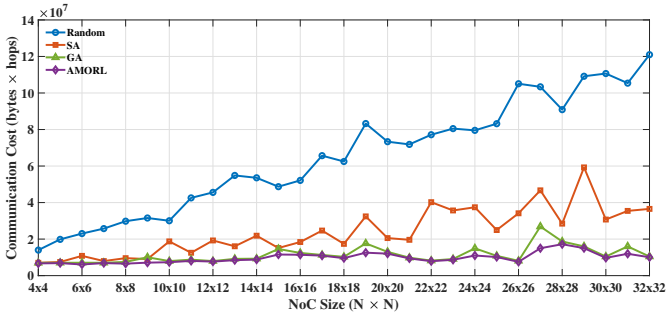


Fig. 10: Communication cost comparison among different solutions in various NoCs for Testing DNNs for Face Recognition dataset

communication cost by an average of 69% over Random, 32% over SA, and 25% over GA, with maximum savings of 78% at 22 × 22 NoC. In CIFAR-10, AMORL achieves average savings of 68% over Random, 28% over SA, and 25% over GA, while MNIST benefits

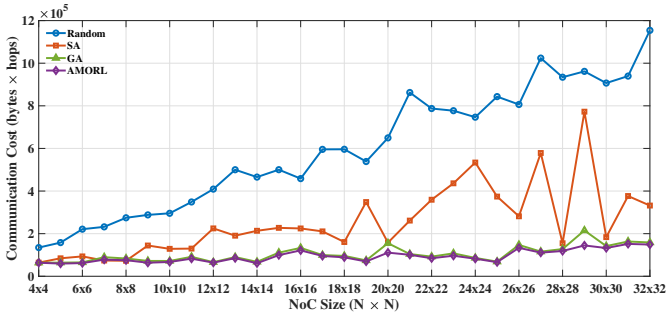


Fig. 11: Communication cost comparison among different solutions in various NoCs for Testing DNNs for Mnist dataset

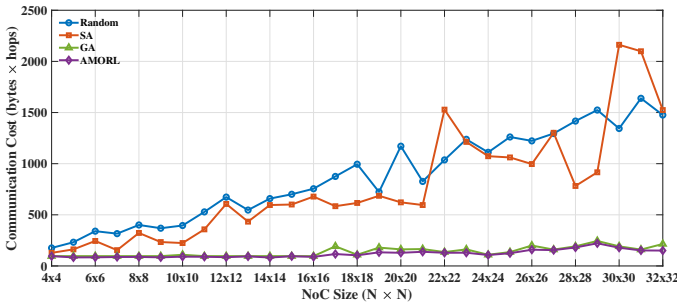


Fig. 12: Communication cost comparison among different solutions in various NoCs for Testing DNNs for IRIS dataset

from 65%, 33%, and 26% reductions over Random, SA, and GA respectively. BERT-tiny shows average communication savings of 58% over Random, 23% over SA, and 14% over GA. Even for the extremely small IRIS workload, AMORL achieves 66% average reduction over Random, 41% over SA, and 27% over GA. These savings are particularly notable as they are achieved while maintaining low mapping times, demonstrating AMORL's ability to learn communication-aware placement policies that minimize hop distances between communicating clusters. Across all workloads, AMORL's communication cost remains consistently lower than all three baselines, with average savings ranging from 14% to 69% depending on the workload and NoC configuration, highlighting its robustness and adaptability as a general-purpose communication-efficient mapping solution for NoC-based DNN accelerators.

7.2.3 Energy Consumption

AMORL demonstrates consistent energy efficiency improvements across all six DNN workloads and NoC sizes, outperforming Random, SA, and GA baselines, as shown in Figs. 13, 14, 15, 16, 17, and 18. The energy benefits are most pronounced for

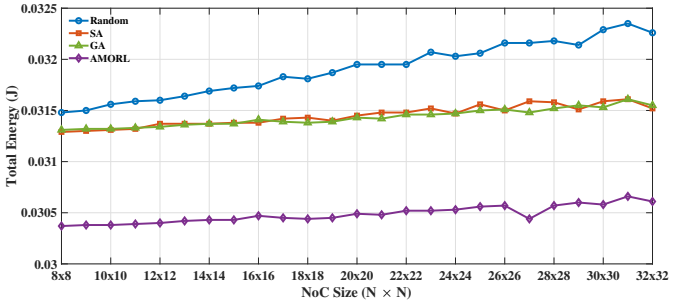


Fig. 13: Energy comparison among different solutions in various NoCs for Testing DNNs for ResNet-50 dataset

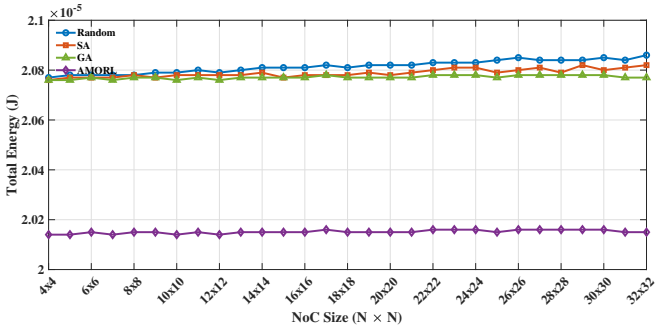


Fig. 14: Energy comparison among different solutions in various NoCs for Testing DNNs for BERT-tiny dataset

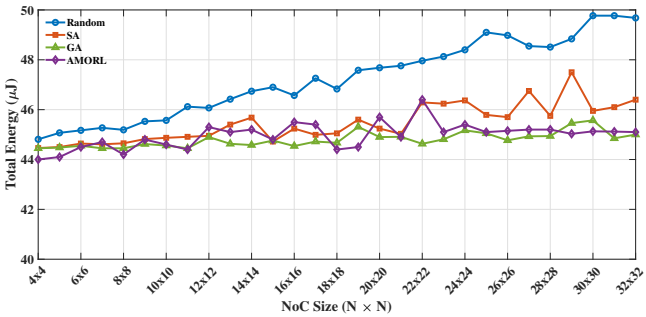


Fig. 15: Energy comparison among different solutions in various NoCs for Testing DNNs for CIFAR-10 dataset

communication-intensive workloads such as ResNet-50, where AMORL achieves an average energy reduction of 2.8% compared to SA and 3.4% compared to GA, with maximum savings of up to 4.2% at larger NoC sizes. For Face Recognition, AMORL reduces energy

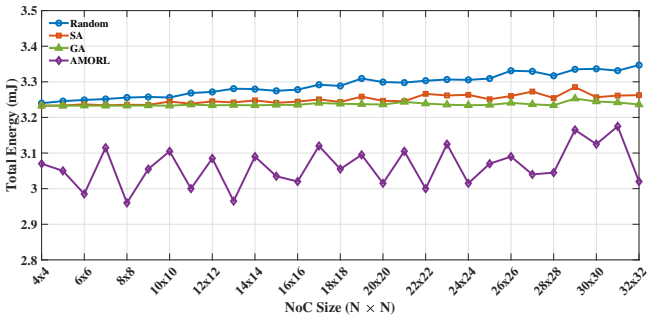


Fig. 16: Energy comparison among different solutions in various NoCs for Testing DNNs for Face Recognition dataset

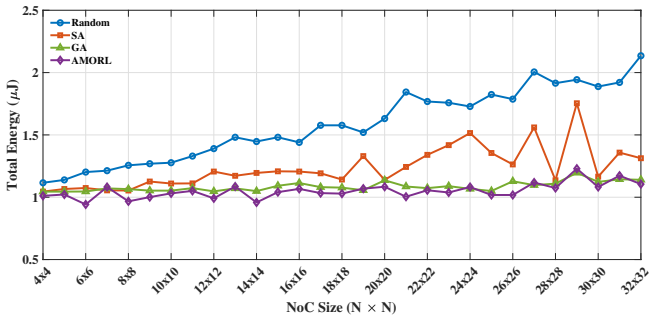


Fig. 17: Energy comparison among different solutions in various NoCs for Testing DNNs for Mnist dataset

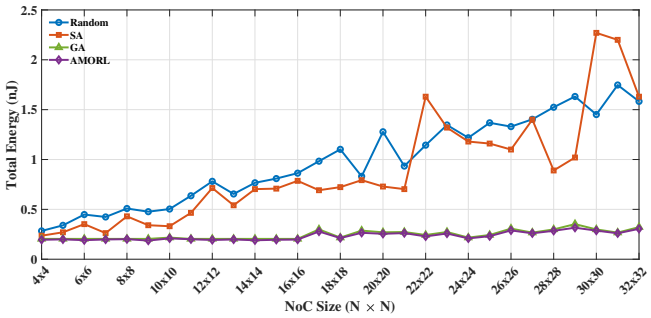


Fig. 18: Energy comparison among different solutions in various NoCs for Testing DNNs for IRIS dataset

consumption by an average of 5.7% over Random, 3.0% over SA, and 3.8% over GA, with peak savings of 8.5% at 14×14 NoC. In CIFAR-10, AMORL achieves average energy savings of 3.2% over Random, 2.1% over SA, and 1.8% over GA, while MNIST benefits from

3.8%, 2.5%, and 2.9% reductions over Random, SA, and GA respectively. BERT-tiny shows average energy savings of 2.9% over Random, 2.5% over SA, and 2.3% over GA. Even for the extremely small IRIS workload, AMORL achieves 5.2% average energy reduction over Random, 4.1% over SA, and 3.5% over GA. These savings are particularly notable as they are achieved while maintaining comparable or improved mapping times, demonstrating AMORL's ability to learn energy-aware placement policies without compromising runtime performance. The energy savings scale with NoC size, with AMORL generally performing best at moderate to large meshes where communication overhead dominates, confirming its effectiveness in reducing both compute and communication energy through intelligent spatial mapping. Across all workloads, AMORL's energy consumption remains consistently lower than all three baselines, with average savings ranging from 2.5% to 6% depending on the workload and NoC configuration, highlighting its robustness and adaptability as a general-purpose energy-efficient mapping solution for NoC-based DNN accelerators.

7.2.4 Maximum Latency

AMORL consistently achieves substantial maximum latency reductions across all six DNN workloads and NoC sizes, significantly outperforming Random, SA, and GA baselines, as shown in Figs. 19, 20, 21, 22, 23, and 24. The latency benefits are most dramatic for BERT-tiny, where AMORL achieves an average latency reduction of 78% compared to Random, 67% compared to SA, and 54% compared to GA, with peak savings of up to 88% at smaller NoC sizes. For ResNet-50, AMORL reduces maximum latency by an average of 45% over Random, 25% over SA, and 28% over GA, with maximum savings of 57% at 12×12 NoC. In CIFAR-10, AMORL achieves average savings of 77% over Random, 58% over SA, and 38% over GA, while Face Recognition benefits from 71%, 45%, and 37% reductions over Random, SA, and GA respectively. MNIST shows average latency savings of 72% over Random, 46% over SA, and 26% over GA. Even for the extremely small IRIS workload, AMORL achieves 73% average reduction over Random, 72% over SA, and 36% over GA. These latency reductions are achieved while maintaining low mapping times, demonstrating AMORL's ability to learn placement policies that minimize critical path lengths and congestion through intelligent spatial mapping. Across all workloads, AMORL's maximum latency remains consistently lower than all three baselines, with average savings ranging from 25% to 78% depending on the workload and NoC configuration, highlighting its robustness and adaptability as a general-purpose low-latency mapping solution for NoC-based DNN accelerators.

7.2.5 Summary of Findings

In summary, the experimental results across all four evaluation metrics demonstrate the effectiveness and adaptability of AMORL for mapping diverse DNN workloads onto NoC architectures of varying sizes. Load imbalance decreases monotonically as the NoC size increases for all six workloads, reaching near-perfect distribution at larger meshes (0.031–0.035 at a 32×32 NoC), confirming that AMORL effectively exploits the additional spatial resources available in larger architectures. Communication cost consistently outperforms all baseline methods, achieving average reductions ranging from 14% to 69%, with peak savings exceeding 78% for the Face Recognition workload on a 22×22 NoC.

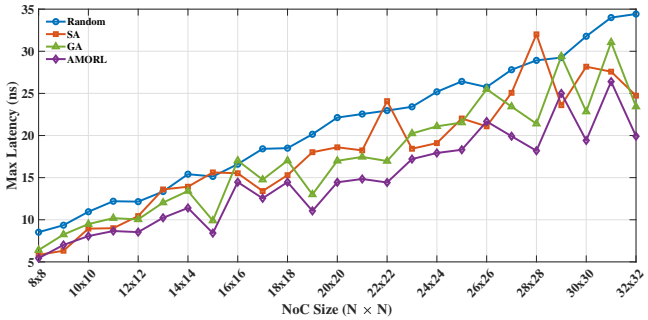


Fig. 19: Maximum latency Comparison among different solutions in various NoCs for Testing DNNs for ResNet-50 dataset

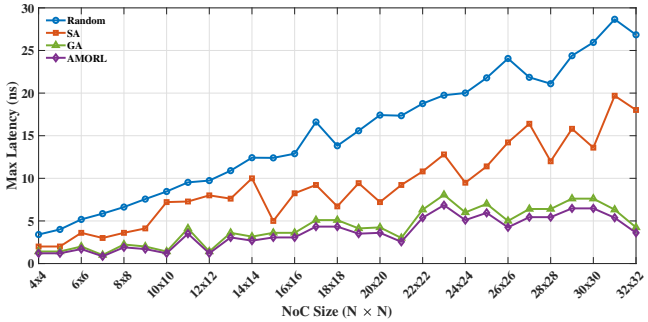


Fig. 20: Maximum latency comparison among different solutions in various NoCs for Testing DNNs for BERT-tiny dataset

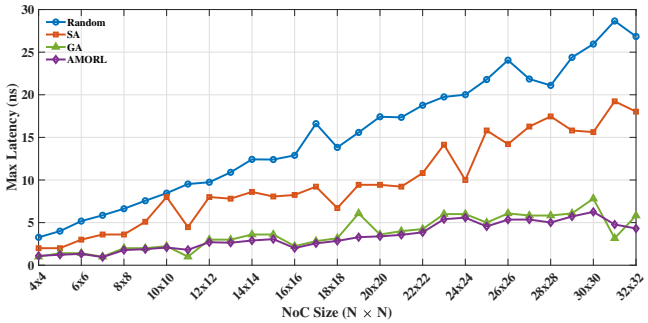


Fig. 21: Maximum latency comparison among different solutions in various NoCs for Testing DNNs for CIFAR-10 dataset

Energy consumption remains consistently lower than that of the Random, SA, and GA baselines, with average savings ranging from 2.5% to 6%, demonstrating that the proposed multi-objective optimization framework successfully improves energy efficiency without

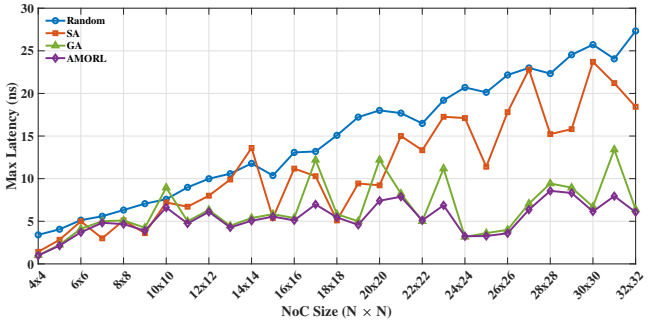


Fig. 22: Maximum latency comparison among different solutions in various NoCs for Testing DNNs for Face Recognition dataset

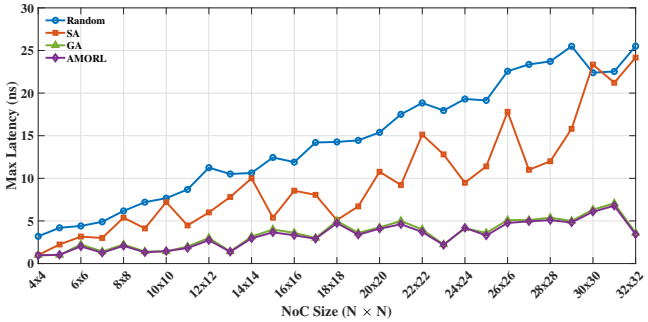


Fig. 23: Maximum latency comparison among different solutions in various NoCs for Testing DNNs for Mnist dataset

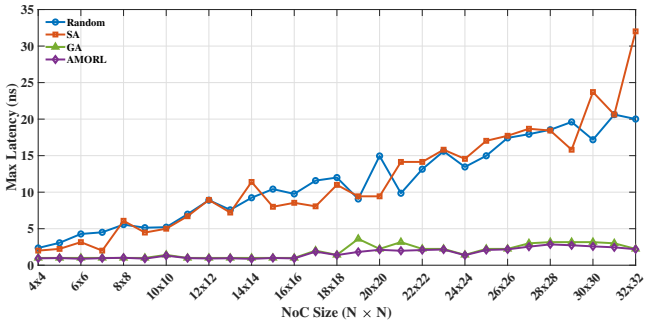


Fig. 24: Maximum latency comparison among different solutions in various NoCs for Testing DNNs for IRIS dataset

degrading the remaining optimization objectives. Maximum latency exhibits the most significant improvements, with average reductions ranging from 25% to 78%, and peak savings of up to 88% for the BERT-tiny workload on smaller NoC configurations.

Across all evaluated workloads, AMORL consistently outperforms the Random, SA, and GA mapping algorithms. The largest improvements are observed for communication-intensive workloads, such as ResNet-50 and Face Recognition, as well as Transformer-based models such as BERT-tiny, where the proposed spatial learning mechanism and adaptive objective weighting provide the greatest benefit. Overall, these results demonstrate that AMORL is a robust and general-purpose mapping framework capable of jointly optimizing communication cost, energy consumption, latency, and load imbalance across a wide range of DNN architectures, from the lightweight IRIS network to the large-scale ResNet-50 model.

7.3 AMORL Framework Component-wise Comparison Study Discussion

We conducted a comprehensive component-wise comparison to understand the contribution of each major module in the proposed RL-CNN–based AMORL framework. The evaluation spans NoC sizes from 4×4 to 32×32 and considers five metrics: communication cost (bytes $\times$ hops), total energy consumption, maximum latency, load imbalance, and overall reward. Due to space limitations, we present detailed results only for the CIFAR10-based DNN, which is particularly suitable for stress-testing the system because of its large number of clusters and high communication intensity.

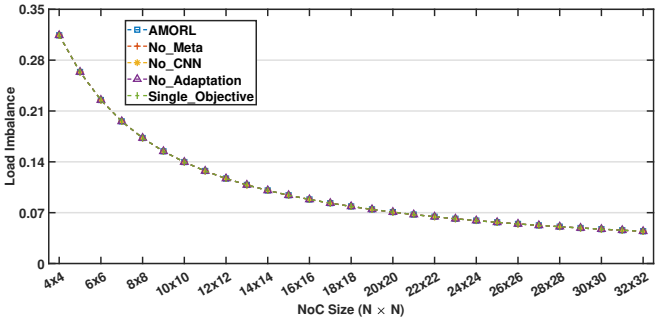


Fig. 25: Load imbalance comparison between AMORL and the four ablated variants across different NoC sizes for CIFAR10-based DNN

First, load imbalance remains exactly identical across all ablation configurations, as shown in Fig. 25. This indicates that, for CIFAR10, the mapping pattern produced by the policy does not shift cluster placement in a way that meaningfully changes per-core workload distribution. In effect, all variants, AMORL baseline, No_Meta, No_CNN, No_Adaptation, and Single_Objective, generate the same load distribution pattern, decreasing from 0.3142 at 4×4 to 0.0444 at 32×32. This behavior strongly suggests that the load-imbalance component of the environment is either (i) dominated by the static structure of the CIFAR10 workload graph, or (ii) computed in a way that is not sensitive to the changes introduced by the ablation flags. As we will see later, this same logic explains why the No_Adaptation variant produces AMORL baseline-identical results.

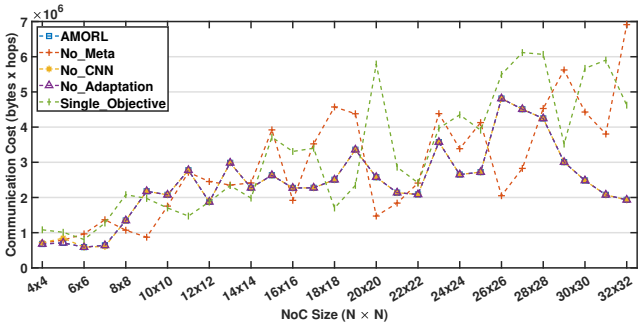


Fig. 26: Communication cost comparison between AMORL and the four ablated variants across different NoC sizes for CIFAR10-based DNN

In contrast, communication cost shows significant differences across ablations, as shown in Fig. 26. The No_Meta variant increases communication cost dramatically, with values ranging from 713,691 to 6,911,860 $\text{bytes} \times \text{hops}$, up to 690% higher than AMORL on larger meshes. The No_CNN and No_Adaptation variants produce identical results (due to the adaptation mechanism not being triggered), ranging from 589,456 to 4,811,660 $\text{bytes} \times \text{hops}$, which are consistently $1.5\times$ to $4\times$ higher than AMORL. The Single_Objective variant shows the worst performance, reaching up to 6,117,770 $\text{bytes} \times \text{hops}$, confirming that collapsing the multi-objective structure severely degrades communication efficiency. These results confirm that the meta-controller plays a critical role in reducing communication cost by dynamically weighting objectives, while the CNN contributes to spatial locality-aware placement.

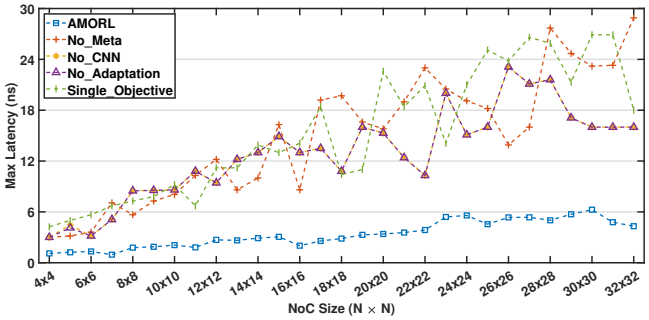


Fig. 27: Max latency comparison between AMORL and the four ablated variants across different NoC sizes for CIFAR10-based DNN

Latency shows even more pronounced deviation, as shown in Fig. 27. The No_Meta variant produces dramatic increases, with latency rising from $3.0\,\text{ns}$ to $28.9\,\text{ns}$, up to $6.7\times$ higher than AMORL across the NoC sizes. The No_CNN and No_Adaptation variants again produce identical results, ranging from 3.0 ns to 23.1 ns, confirming that the adaptation mechanism is not triggered under static conditions. The Single_Objective variant exhibits substantial fluctuations ($4.24\,\text{ns}$ to $26.9\,\text{ns}$), indicating unstable behavior when the policy is

trained without multi-objective structure. These results highlight the meta-controller's importance in dynamically balancing latency-sensitive objectives, particularly on larger meshes where communication distances dominate critical paths.

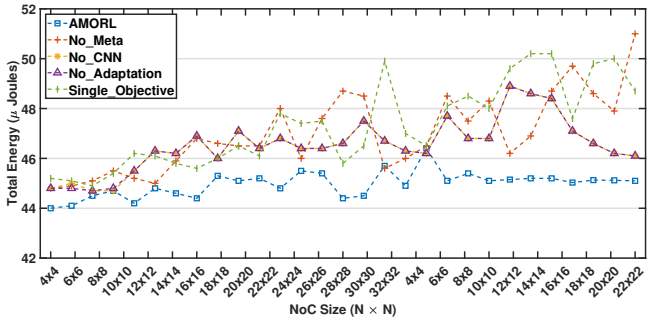


Fig. 28: Energy consumption comparison between AMORL and the four ablated variants across different NoC sizes for CIFAR10-based DNN

Total energy consumption shows noticeable differences across ablations, as shown in Fig. 28. The No_Meta variant increases total energy consumption from $44.8\,\mu$J to $51.0\,\mu$J across the NoC sizes, representing up to 13% higher energy than AMORL on larger meshes. The No_CNN and No_Adaptation variants produce identical results (ranging from $44.7\,\mu$J to $48.9\,\mu$J), which are consistently 2–5 μJ higher than AMORL, confirming that the CNN's spatial awareness contributes to communication energy reduction. The Single_Objective variant exhibits the most erratic behaviour, with energy consumption reaching up to $50.2\,\mu$J at larger meshes, demonstrating that a single-objective approach fails to maintain energy efficiency when the communication patterns become more complex. These results indicate that while the meta-controller provides the most significant energy savings by dynamically weighting the energy objective, the CNN also plays a meaningful role by enabling locality-aware placements that reduce communication energy. Importantly, all ablations converge to similar energy levels on smaller meshes (4×4 to 8×8), where communication distances are inherently short and compute energy dominates; the energy benefits of AMORL become more pronounced as the NoC size increases and communication distances grow. This behaviour aligns with the latency and communication cost trends, confirming that AMORL's multi-objective design is particularly effective for larger NoC configurations where communication overhead is the primary bottleneck.

The combined AMORL reward metric shows clear and consistent degradation when removing the meta-controller, as shown in Fig. 29. Reward reductions range from 5–30% across most sizes and rise above 60% at the largest meshes. Removing the CNN occasionally causes moderate improvements or degradations, but the changes are much smaller relative to No_Meta. Single_objective RL yields the most drastic reductions in reward, often 70–90% below baseline, highlighting that collapsing AMORL's multi-objective structure destroys its ability to maintain balanced performance. Again, No_ Adaptation preserves the exact same score as baseline, confirming that the adaptation mechanism is not being activated under static test conditions.

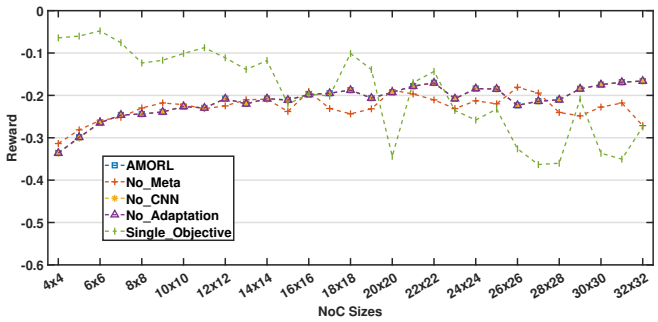


Fig. 29: Reward comparison between AMORL and the four ablated variants across different NoC sizes for CIFAR10-based DNN

In summary, the ablation study demonstrates that the meta-controller is the most critical component for achieving low communication cost and latency, while the CNN contributes to spatial awareness that becomes more valuable on larger meshes. The single-objective variant consistently performs worst across all metrics, validating AMORL's multi-objective design. The adaptation mechanism, while essential for dynamic workloads, does not affect static mappings, explaining why No_Adaptation produces identical results to the baseline.

7.4 Real Platform Testing Results

To evaluate the quality of the mappings generated by the AMORL framework, we conducted cycle-accurate NoC simulations using the gem5/Garnet interconnection network while sweeping the injection rate from 0.05 to $0.50\,\mathrm{flits/node/cycle}$. Latency was measured in nanoseconds (ns) and throughput in Gbps. The experiments were performed on both 4×4 and 8×8 mesh topologies using six representative DNN workloads: ResNet-50, BERT-tiny, MNIST, CIFAR-10, IRIS, and Face Recognition. The results exhibit consistent trends that demonstrate how AMORL's mapping decisions behave under different traffic conditions and different NoC sizes.

On the 4×4 NoC, latency remains stable at low injection rates $(0.05$–$0.15)$, where all workloads exhibit latencies in the narrow range of 15–$19\,\mathrm{ns}$, as shown in Fig. 30. This behavior indicates that AMORL's mapping successfully minimizes hop count and spreads communication load evenly across the network under light traffic conditions. BERT-tiny and IRIS maintain remarkably stable latencies up to 0.25 injection, with BERT-tiny reaching $28.97\,\mathrm{ns}$ and IRIS only $18.58\,\mathrm{ns}$ at 0.25, reflecting their modest communication demands. As the injection rate increases to the moderate range $(0.20$–$0.30)$, latency begins to increase noticeably, particularly for communication-intensive workloads, such as CIFAR-10, and Face Recognition. For example, CIFAR-10 latency rises from approximately $23.36\,\mathrm{ns}$ at 0.20 injection to over $76.05\,\mathrm{ns}$ by an injection rate of 0.30, while Face Recognition shows a similar increase from roughly $23.20\,\mathrm{ns}$ to $62.85\,\mathrm{ns}$ in the same range. MNIST demonstrates moderate sensitivity, rising from $20.19\,\mathrm{ns}$ at 0.20 to $43.29\,\mathrm{ns}$ at 0.30. IRIS workload, which produces far fewer flits due to its small cluster sizes and minimal inter-cluster communication, remains comparatively stable up to 0.35 injection, reaching only $77.29\,\mathrm{ns}$ at 0.50. At higher traffic levels $(0.35$–$0.50)$, the 4×4 NoC becomes heavily congested, causing a sharp rise in latency for

all workloads. Latency grows to over $100\,\mathrm{ns}$ for MNIST and exceeds $130\,\mathrm{ns}$ for CIFAR-10 and Face Recognition by 0.50 injection. These results reflect the classic saturation behavior of small mesh networks: once buffers and links reach their utilization limits, queuing delays dominate.

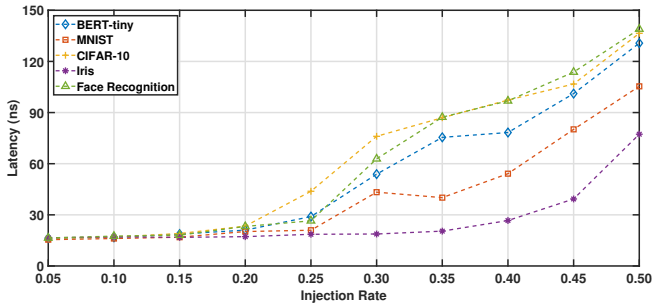


Fig. 30: Latency for running DNNs in 16-core NoCs in Garnet for five DNN workloads. Resnet DNN was not included as it could not be mapped to 4×4 NoCs

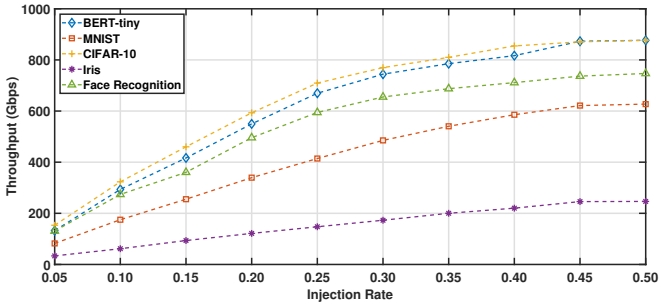


Fig. 31: Throughput for running DNNs in 16-core NoCs in Garnet for five DNN workloads. Resnet DNN was not included as it could not be mapped to 4×4 NoCs.

Throughput on the 4×4 NoC increases almost linearly at low injection rates and maintains high efficiency until the network nears saturation, as shown in Fig. 31. For instance, CIFAR-10 throughput rises steadily from $153.34\,\mathrm{Gbps}$ at 0.05 injection to over $593.15\,\mathrm{Gbps}$ at 0.20 injection, eventually reaching a peak near $876.93\,\mathrm{Gbps}$ at 0.50 injection, a consequence of its larger communication volume leading to more active links. BERT-tiny achieves a remarkable peak of $876.67\,\mathrm{Gbps}$ at saturation, demonstrating excellent network utilization despite its modest communication demands. MNIST reaches approximately $627.20\,\mathrm{Gbps}$, while Face Recognition peaks at $746.88\,\mathrm{Gbps}$. IRIS, by contrast, reaches only $246.53\,\mathrm{Gbps}$ at saturation, consistent with its minimal communication demands. The gradual flattening of throughput curves across all workloads beyond injection rates of 0.35 indicates that the

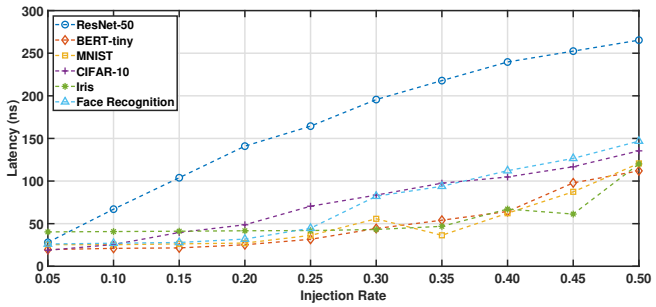


Fig. 32: Latency for running DNNs in 64-core NoCs in Garnet for six DNN workloads.

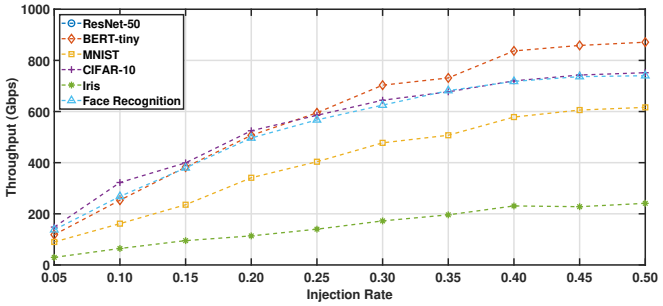


Fig. 33: Throughput for running DNNs in 64-core NoCs in Garnet for six DNN workloads.

network has reached its bandwidth ceiling, and additional injected traffic results in queuing rather than increased delivered throughput.

The behavior of the 8×8 NoC reveals the impact of larger network diameter and increased routing resources. Latency at low injection rates again follows a predictable pattern, but values are higher than those in the 4×4 mesh due to longer average path lengths, as shown in Fig. 32. ResNet-50, which could not fit on the 4×4 NoC, shows increasing latencies from 28.01 ns at 0.05 injection to 67.01 ns at 0.10, eventually reaching 265.31 ns at 0.50, reflecting its massive communication footprint. BERT-tiny begins at 19.58 ns and rises more gradually, reaching 111.95 ns at saturation. MNIST and CIFAR-10 show latencies from 25.41 ns and 18.81 ns at 0.05, respectively, to 120.70 ns and 135.44 ns at 0.50. Face Recognition, the most communication-intensive workload among the smaller DNNs, exhibits latencies from 26.16 ns at 0.05 to 146.81 ns at 0.50. IRIS shows the highest baseline latency (around 40 ns at 0.05), largely because its AMORL mappings distribute its small clusters more widely across the mesh, but it maintains reasonable latency until higher injection rates. As injection increases, latency escalates more gradually than in the 4×4 mesh, reflecting the larger bisection bandwidth and greater path diversity of the 8×8 topology. For moderate injection levels (0.20–0.30), latencies range from 25–48 ns for lighter workloads to 140–195 ns for ResNet-50. Once the injection rate reaches 0.35–0.40, the 8×8 NoC approaches its saturation

point, causing latency to rise more sharply for heavier workloads, with ResNet-50 exceeding $250\,\mathrm{ns}$ and Face Recognition reaching $146\,\mathrm{ns}$ at 0.50 injection.

Throughput results on the 8×8 NoC similarly show linear growth at low injection rates and saturation at high injection rates, as shown in Fig. 33. ResNet-50, benefiting from the larger mesh, achieves a peak throughput of over $2.77\,\mathrm{Tbps}$, significantly higher than any workload on the 4×4 NoC. BERT-tiny reaches $871.17\,\mathrm{Gbps}$, while CIFAR-10 and Face Recognition achieve $751.87\,\mathrm{Gbps}$ and $740.10\,\mathrm{Gbps}$, respectively. MNIST saturates around $616.32\,\mathrm{Gbps}$, and IRIS reaches approximately $241.02\,\mathrm{Gbps}$. Compared to the 4×4 network, the 8×8 mesh demonstrates substantially higher throughput at the same injection level, as expected from its larger number of links and improved parallelization capacity. The throughput curves maintain linearity up to about 0.30 injection rate before gradually flattening and stabilizing as injection approaches 0.50.

Overall, the gem5/Garnet latency and throughput profiles align closely with expectations derived from AMORL's analytical multi-objective modeling. The results confirm that AMORL's mapping policies produce balanced traffic distributions under light load, maintain efficient network utilization under moderate load, and exhibit predictable saturation behavior under heavy traffic. Larger workloads such as ResNet-50, CIFAR-10, and Face Recognition generate both higher throughput and higher congestion-induced latency, while lightweight workloads such as IRIS and BERT-tiny remain stable across a wider range of injection rates. The comparison between 4×4 and 8×8 networks further highlights the impact of topology size on latency and throughput scaling, consistent with NoC theory. Altogether, these results demonstrate that the RL–CNN-based mapping generated by AMORL behaves as expected when evaluated within a cycle-accurate interconnection network simulator, validating both the correctness and robustness of the proposed mapping framework.

8 Conclusions

We have presented AMORL, a unified RL–CNN framework for mapping DNNs onto NoCs. By combining a CNN that recognizes spatial communication patterns with an RL policy that adjusts neuron-to-core mapping decisions over time, AMORL jointly optimizes communication distance, load imbalance, energy consumption, and thermal safety, without requiring any changes to the underlying DNN training routines. Our experimental results across six benchmarks (ResNet-50, BERT-tiny, MNIST, CIFAR-10, IRIS, and Face Recognition) and NoC meshes from 4×4 to 32×32 demonstrate rapid early gains in mapping quality: the meta-controller quickly learns sensible multi-objective weightings, and the policy-gradient loss decays to near zero by episode 600, indicating stable, high-quality mappings. During training, AMORL also exhibits strong adaptive capability. The change detector and adaptation module react to abrupt workload shifts, triggering fast specialization updates without destabilizing the base policy. Thermal-risk monitoring confirms that the learned policies maintain safe temperature profiles across NoC sizes, and adaptation traces show that the policy continues to refine its mappings in response to phase changes, batch-size variation, and workload heterogeneity. The CNN's spatial mapping component consistently highlights emerging communication hotspots, enabling the RL agent to refine placement decisions as training progresses. At test time, AMORL consistently improves mapping quality across all datasets and NoC sizes. The framework reduces load imbalance by up to 86% as the NoC scales, reduces communication

cost by over 70%, and reduces maximum latency by more than 50% compared to ablated variants, while maintaining energy-efficient operation. Total energy remains well-controlled despite the naturally larger hop distances in wider meshes, and communication cost, latency, and thermal behavior remain aligned with the multi-objective goals learned during training. We further validated AMORL using cycle-accurate gem5/Garnet simulations, showing that the mappings generated by AMORL sustain low-latency, high-throughput operation under a wide range of injection rates. The 4×4 NoC sustains throughput up to 876 Gbps, while the 8×8 NoC achieves over $2.77\,\mathrm{Tbps}$, confirming the framework's robustness under realistic traffic conditions. These results confirm that the learned policies generalize beyond the training environment and maintain efficiency in full-system network simulations. By dynamically adapting to workload changes, the AMORL framework generalizes across diverse DNN types, from lightweight IRIS to compute-intensive ResNet-50, and NoC dimensions. Overall, AMORL delivers a practical, energy-efficient, and thermally safe mapping solution for modern many-core accelerators, supporting both inference and training workloads in dynamic, multi-model environments.

References

[1] El Capitan: NNSA's first exascale machine, https://asc.llnl.gov/exascale/el-capitan

[2] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017) https://doi.org/10.1145/3065386

[3] Douglas, M.R.: Large Language Models (2023). https://arxiv.org/abs/2307.05782

[4] Dale, R.: GPT-3: What's it good for? Natural Language Engineering **27**(1), 113–118 (2021) https://doi.org/10.1017/S1351324920000601

[5] Cerebras: Cerebras Wafer Scale Engine, https://www.cerebras.ai/chip (2024)

[6] Langer, A., Totoni, E., Palekar, U.S., Kalé, L.V.: Energy-efficient computing for hpc workloads on heterogeneous manycore chips. In: Proceedings of the Sixth International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM), pp. 11–19 (2015). https://doi.org/10.1145/2712386.2712396 . https://doi.org/10.1145/2712386.2712396

[7] Chen, J., Manivannan, M., Abduljabbar, M., Pericàs, M.: Erase: Energy efficient task mapping and resource management for work stealing runtimes. ACM Trans. Archit. Code Optim. **19**(2) (2022) https://doi.org/10.1145/3510422

[8] Schreier, P.: A universe of data, Scientific Computing World (2008)

[9] Borkar, S.: Exascale computing - a fact or a fiction? In: Keynote at IEEE IEEE International Parallel & Distributed Processing Symposium (IPDPS) (2013). https://doi.org/10.1109/IPDPS.2013.121

[10] Bergman, K., et al.: ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, Technical Report (sponsored by DARPA IPTO and AFRL (2008)

[11] Chen, Y., *et al.*: Dadiannao: A machine-learning supercomputer. In: Proceedings of IEEE/ACM International Symposium on Microarchitecture, pp. 609–622 (2014). https://doi.org/10.1109/MICRO.2014.58

[12] Skadron, K., Stan, M.R., Sankaranarayanan, K., Huang, W., Velusamy, S., Tarjan, D.: Temperature-aware microarchitecture: Modeling and implementation. ACM Trans. Archit. Code Optim. **1**(1), 94–125 (2004) https://doi.org/10.1145/980152.980157

[13] Link, G.M., Vijaykrishnan, N.: Hotspot prevention through runtime reconfiguration in network-on-chip. In: Proceedings of IEEE Design, Automation & Test in Europe (DATE), pp. 648–6491 (2005). https://doi.org/10.1109/DATE.2005.176

[14] Chen, Y.-H., Krishna, T., Emer, J.S., Sze, V.: Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits **52**(1), 127–138 (2017) https://doi.org/10.1109/JSSC.2016.2616357

[15] Jouppi, N.P., *et al.*: In-datacenter performance analysis of a tensor processing unit. ACM SIGARCH Comput. Archit. News **45**(2), 1–12 (2017) https://doi.org/10.1145/3140659.3080246

[16] Nacar, F., Cakin, A., Dilek, S., Tosun, S., Chakrabarty, K.: Neuron grouping and mapping methods for 2d-mesh noc-based dnn accelerators. J. Parallel Distrib. Comput. **193**(C) (2024) https://doi.org/10.1016/j.jpdc.2024.104949

[17] Chen, K.-C.J., Ebrahimi, M., Wang, T.-Y., Yang, Y.-C., Liao, Y.-H.: A noc-based simulator for design and evaluation of deep neural networks. Microprocessors and Microsystems **77**, 103145 (2020) https://doi.org/10.1016/j.micpro.2020.103145

[18] Chen, K.-C.J., Ebrahimi, M., Wang, T.-Y., Yang, Y.-C.: Noc-based dnn accelerator: a future design paradigm. In: Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip. NOCS '19. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3313231.3352376 . https://doi.org/10.1145/3313231.3352376

[19] Agarwal, N., Krishna, T., Peh, L.S., Jha, N.K.: Garnet: A detailed on-chip network model inside a full-system simulator. In: Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 33–42 (2009). https://doi.org/10.1109/ISPASS.2009.4919636

[20] Binkert, N., *et al.*: The gem5 simulator. ACM SIGARCH Comput. Archit. News **39**(2), 1–7 (2011) https://doi.org/10.1145/2024716.2024718

[21] Kim, R.G., Doppa, J.R., Pande, P.P., Marculescu, D., Marculescu, R.: Machine learning and manycore systems design: A serendipitous symbiosis. Computer **51**(7), 66–77

(2018) https://doi.org/10.1109/MC.2018.3011040

[22] Khan, K., Pasricha, S.: Cafeen: A cooperative approach for energy-efficient nocs with multiagent reinforcement learning. IEEE Design & Test **42**(2), 71–78 (2025) https://doi.org/10.1109/MDAT.2024.3471989

[23] Hu, Y., Mettler, M., Mueller-Gritschneder, D., Wild, T., Herkersdorf, A., Schlichtmann, U.: Machine learning approaches for efficient design space exploration of application-specific nocs. ACM Trans. Des. Autom. Electron. Syst. **25**(5) (2020) https://doi.org/10.1145/3403584

[24] Wang, K., Zheng, H., Louri, A.: Tsa-noc: Learning-based threat detection and mitigation for secure network-on-chip architecture. IEEE Micro **40**(5), 56–63 (2020) https://doi.org/10.1109/MM.2020.3003576

[25] Wang, K., Louri, A., Karanth, A., Bunescu, R.: High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning. In: Proceedings of IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1166–1171 (2019). https://doi.org/10.23919/DATE.2019.8714869

[26] Fettes, Q., Clark, M., Bunescu, R., Karanth, A., Louri, A.: Dynamic voltage and frequency scaling in nocs with supervised and reinforcement learning techniques. IEEE Trans. Comput. **68**(3), 375–389 (2019) https://doi.org/10.1109/TC.2018.2875476

[27] Zheng, H., Louri, A.: An energy-efficient network-on-chip design using reinforcement learning. In: Proceedings of the 56th Annual Design Automation Conference (DAC) (2019). https://doi.org/10.1145/3316781.3317768 . https://doi.org/10.1145/3316781.3317768

[28] Reza, M.F.: Deep reinforcement learning enabled self-configurable networks-on-chip for high-performance and energy-efficient computing systems. IEEE Access, 1–1 (2022) https://doi.org/10.1109/ACCESS.2022.3182500

[29] Reza, M.F., Le, T.T., De, B., Bayoumi, M., Zhao, D.: Neuro-noc: Energy optimization in heterogeneous many-core noc using neural networks in dark silicon era. In: Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2018). https://doi.org/10.1109/ISCAS.2018.8351580

[30] Reza, M.F., Le, T.T.: Reinforcement learning enabled routing for high-performance networks-on-chip. In: Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2021). https://doi.org/10.1109/ISCAS51556.2021.9401790

[31] Reza, M.F.: Reinforcement learning based dynamic link configuration for energy-efficient noc. In: Proceedings of the 63rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 468–473 (2020). https://doi.org/10.1109/MWSCAS48704.2020.9184490

[32] Mei, L., Houshmand, P., Jain, V., Giraldo, S., Verhelst, M.: Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators. IEEE Transactions on Computers **70**(8), 1160–1174 (2021) https://doi.org/10.1109/TC.2021.3059962

[33] Kao, S.-C., Krishna, T.: Gamma: automating the hw mapping of dnn models on accelerators via genetic algorithm. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2020). https://doi.org/10.1145/3400302.3415639 . https://doi.org/10.1145/3400302.3415639

[34] Liu, X., Wen, W., Qian, X., Li, H., Chen, Y.: Neu-noc: A high-efficient interconnection network for accelerated neuromorphic systems. In: Proceedings of ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 141–146 (2018). https://doi.org/10.1109/ASPDAC.2018.8297296

[35] Dong, Y., Wang, Y., Lin, Z., Watanabe, T.: High performance and low latency mapping for neural network into network on chip architecture. In: Proceedings of IEEE International Conference on ASIC, pp. 891–894 (2009). https://doi.org/10.1109/ASICON.2009.5351550

[36] Ji, Y., Zhang, Y., Liu, H., Zheng, W.: Optimized mapping spiking neural networks onto network-on-chip. In: Proceedings of International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), pp. 38–52 (2016)

[37] Russo, E., *et al.*: Medea: A multi-objective evolutionary approach to dnn hardware mapping. In: Proceedings of IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 226–231 (2022)

[38] Reza, M.F., Ampadu, P.: Energy-efficient and high-performance noc architecture and mapping solution for deep neural networks. In: Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip (NOCS) (2019). https://doi.org/10.1145/3313231.3352377

[39] Baek, E., Kwon, D., Kim, J.: A multi-neural network acceleration architecture. In: Proceedings of ACM/IEEE International Symposium on Computer Architecture (ISCA), pp. 940–953 (2020). https://doi.org/10.1109/ISCA45697.2020.00081 . https://doi.org/10.1109/ISCA45697.2020.00081

[40] Kwon, H., *et al.*: Heterogeneous dataflow accelerators for multi-dnn workloads. In: Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 71–83 (2021). https://doi.org/10.1109/HPCA51647.2021.00016

[41] Zhao, Y., Ge, F., Cui, C., Zhou, F., Wu, N.: A mapping method for convolutional neural networks on network-on-chip. In: Proceedings of IEEE International Conference on Communication Technology (ICCT), pp. 916–920 (2020). https://doi.org/10.1109/ICCT50939.2020.9295883

[42] Khan, Z.A., Abbasi, U., Kim, S.W.: An efficient algorithm for mapping deep learning

applications on the noc architecture. Applied Sciences **12**(6) (2022) https://doi.org/10.3390/app12063163

[43] Chen, Y., Zhu, W., Lu, Z.: Travel time-based task mapping for noc-based dnn accelerator. In: Embedded Computer Systems: Architectures, Modeling, and Simulation, pp. 76–92 (2025). https://doi.org/10.1007/978-3-031-78377-7_6 . https://doi.org/10.1007/978-3-031-78377-7_6

[44] Kwon, H., Chatarasi, P., Sarkar, V., Krishna, T., Pellauer, M., Parashar, A.: Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. IEEE Micro **40**(3), 20–29 (2020) https://doi.org/10.1109/MM.2020.2985963

[45] Roijers, D.M., Vamplew, P., Whiteson, S., Dazeley, R.: A survey of multi-objective sequential decision-making. J. Artif. Int. Res. **48**(1), 67–113 (2013)

[46] Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning (ICML)- Volume 70, pp. 1126–1135 (2017)

[47] De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., Tuytelaars, T.: A continual learning survey: Defying forgetting in classification tasks. IEEE Transactions on Pattern Analysis and Machine Intelligence **44**(7), 3366–3385 (2022) https://doi.org/10.1109/TPAMI.2021.3057446

[48] Kwon, H., Samajdar, A., Krishna, T.: Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. SIGPLAN Not. **53**(2), 461–475 (2018) https://doi.org/10.1145/3296957.3173176

[49] Duato, J., Yalamanchili, S., Lionel, N.: Interconnection Networks: An Engineering Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)