ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

Assessing Graph Neural Networks for latency and power consumption prediction in application mappings on multicore architectures

O. Roussel[1,2], Z. Ghrayeb[2], S. Le Nours[2] and C. Sinoquet[1]

1 - Nantes University, LS2N, UMR CNRS 6004, Nantes, France
2 - Nantes University, IETR, UMR CNRS 6164, Nantes, France

Abstract. Accurately estimating the latency and power consumption of software applications deployed on multicore systems remains a major challenge for early-stage optimization, as existing methods typically rely on slow and resource-intensive simulations. This paper explores modeling application-to-architecture mappings as heterogeneous graphs and investigates Graph Neural Networks (GNNs) for predicting their performance. Four GNN models are evaluated across eleven datasets, considering five neural network-based software applications. The two best models achieve mean absolute percentage errors of about 2% for power prediction and 15% for latency, with prediction times of only a few tens of milliseconds. These results indicate the potential of GNN-based prediction as an efficient alternative to simulation-driven estimation, paving the way for early-stage AI-assisted mapping optimization.

1 Introduction

This work focuses on the automated and optimized deployment of software applications onto electronic chips. The rising complexity of applications, including embedded AI, graph processing, and IoT, complicates efficient deployment. Simulation-based and analytical methods are commonly used to optimize on-chip application deployment. Simulations are time- and cost-intensive, whereas analytical models, although fast, offer limited accuracy. Exact methods like mathematical programming are effective for small mapping problems, while heuristics have long been the dominant approach for efficiently exploring large solution spaces. However, for any heuristic, local search, or genetic algorithm, the performance of each generated solution (a mapping) must be evaluated quickly. These performance metrics may include power consumption, execution time (latency), memory bandwidth, and communication overhead.

The novelty of our work lies in exploring the potential of Graph Neural Networks (GNNs) to develop data-driven predictive models that take as input a heterogeneous graph, comprising a software application graph, a multiprocessor architecture graph, and a candidate mapping, and output performance metrics.

To date, only one GNN-based approach has been proposed to address this problem, using a custom-designed architecture [1]. To systematize this field, we introduce a unified framework with a generic graph-based mapping representation and evaluate four recent state-of-the-art GNNs within this framework for latency and power-consumption prediction.

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

2 Foundational principles and practical framework

2.1 Graph-based building blocks

An *application graph* abstracts a software application, with nodes as tasks and edges as dependencies. Different task decompositions yield distinct application graphs, as shown in Figure 1.

A *platform graph* provides a high-level abstraction for a multicore architecture where nodes represent hardware components and edges indicate interactions between them. In this work, we limit our focus to processing tiles (processing units and their local memories), a unique shared memory and a single communication bus. Figure 2(a) shows an example.

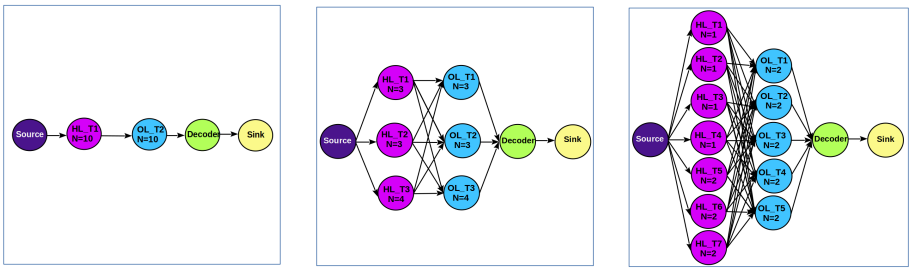
On-chip application deployment decomposes an application into parallel tasks executed on the tiles of a given multicore platform. A valid task-to-tile *mapping* assigns each task to a single tile. Figure 2(b) illustrates the concept of mapping.

2.2 Unified GNN-based task-to-tile mapping framework

Graph Neural Networks We aim to encapsulate an application graph, a platform graph, and a task-to-tile mapping into a unified graph-based representation suitable for GNNs. This representation is called the T2TM graph.

Standard GNNs operate by learning node embeddings: fixed-dimensional vectors that encode the structural and feature information of each node in the graph. Edges guide information flow between nodes and, when present, their features contribute to this process.

A GNN layer is built upon the principle of Message Passing (MP), where each node receives information from its neighbors and updates its embedding accordingly. At each iteration of the MP process, neighboring nodes generate mes-



(a) (b) (c)

Fig. 1: Three application graphs for the same software application. The application implements prediction using a Multi-Layer Perceptron with a hidden and ouput layer of 10 neurons each. The application graphs differ in how each 10-neuron layer is distributed into tasks. Each node represents a task handling operations for N neurons, and the tasks in each layer are executed in parallel.

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

sages, an aggregation function combines them in a permutation-invariant way (e.g., sum, mean, or max), and an update function computes the new embedding. Stacking multiple MP layers enable GNNs to capture complex interactions within a graph.

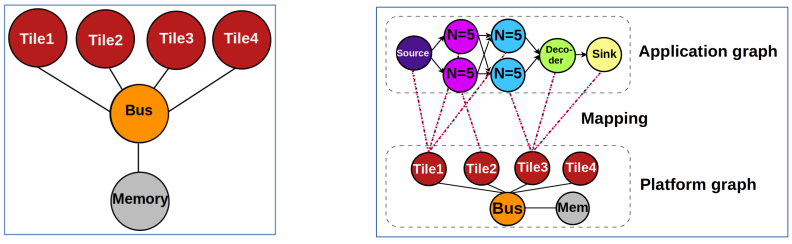
Handling heterogeneous graphs with GNNs T2TM graphs are heterogeneous, with multiple node and edge types, which standard GNNs cannot handle directly. We used the NetworkX library in PyTorch to create, parse, and manipulate T2TM graphs. While NetworkX does not natively support heterogeneous graphs, the `graphml` format preserves node and edge metadata during parsing. The resulting `type` attributes assigned to nodes and edges enable type-aware processing of T2TM graphs. The `to_hetero` function in PyTorch Geometric library enables Message Passing between different node and edge types.

2.3 Model training

A composite loss combines two weighted mean squared errors (MSEs) for latency and power consumption regressions: $\mathscr{L} = \lambda\ MSE_\ell + (1-\lambda)\ MSE_p$.

2.4 Investigated models

We considered four representative GNN Message-Passing layer designs and included the corresponding GNN types in our study: GraphSAGE, popularizing scalable neighborhood aggregation; Graph Attention Network (GAT), using attention to weight neighbor messages; Graph Isomorphism Network (GIN), enhancing expressiveness via stronger aggregation; and Meta General Layer (GeneralConv) [2], unifying these ideas into a highly configurable MP framework. Details on GraphSAGE, GAT, and GIN are provided in the recent survey in [3].



(a) (b)

Fig. 2: (a) Platform graph of a six-node multicore architecture with four processing tiles, a shared memory and a communication bus. (b) Mapping between the application graph of a Multi-Layer Perceptron and the platform graph shown in (a). This task-to-tile mapping is a heterogeneous representation containing multiple types of nodes and edges.

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.

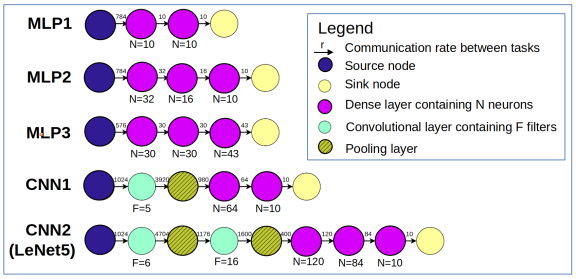


Fig. 3: Five neural network applications before task parallelization. Dense layers with N neurons and convolutional layers with F filters can be split into N or F tasks at most, generating multiple application graphs.

3 Experiments

Datasets We used the 11 datasets from [4], where performance prediction relies on analytical models and simulations. Each dataset contains multiple T2TM graph instances of the same application deployed on a fixed multicore architecture. The instances differ in task decomposition and their assignment to tiles. All datasets involve neural network applications. Figure 3 shows the five applications considered: three based on Multi-Layer Perceptrons and two based on Convolutional Neural Networks.

Table 1 reports key statistics of all eleven datasets. The dataset `mlp1_expe` was created experimentally while the ten others were produced through analytical modeling combined with simulations. Suffixes CG (Clock Gating) and P (Polling) indicate different communication modes used to transfer data.

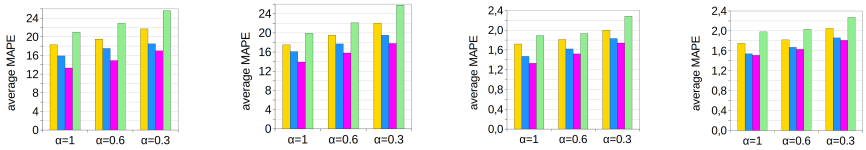
Implementation All GNN models were trained using PyTorch and ROCm (CUDA library for AMD GPUs) on an AMD Radeon RX 7900 XTX GPU, with an 8-core AMD Ryzen 7 5800X3D CPU (3.4 GHz) and 32 GB of DDR4 RAM.

Fixed hyperparameters There are no pre-processing layers. Each of the four classes of GNNs implemented has 8 identical layers (GraphSAGE, GAT, GIN, or GeneralConv) with PReLU activations. The MP aggregation function used was `mean`. Neither Batch Normalization nor Dropout were considered. A

Table 1: Statistics of the 11 datasets. n_m: number of task-to-tile mappings.

Dataset	n_m	Nodes		Edges		Dataset	n_m	Nodes		Edges	
		max	avg	max	avg			max	avg	max	avg
MLP1_expe	18	26	20.1	114	67.7	MLP3_CG	253	31	24.1	152	90.0
MLP1_CG	85	22	28.9	78	56.7	MLP3_P	247	31	23.4	152	83.6
MLP1_P	96	22	18.9	78	56.7	CNN1_CG	224	30	22.6	116	66.8
MLP2_CG	174	30	22.1	143	76.3	CNN1_P	276	30	23.0	116	69.2
MLP2_P	288	30	23.8	143	88.5	CNN2_CG	308	45	31.6	209	108.9
						CNN2_P	192	45	28.3	209	90.3

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.



(a) latency, $\lambda = 1$ (b) latency, $\lambda = 0.5$ (c) power, $\lambda = 0$ (d) power, $\lambda = 0.5$

Fig. 4: Prediction performance of the four GNN types across different λ and α settings. For each GNN type, the MAPE values are averaged over the 11 datasets for $\alpha = 1$, and over the 10 simulated datasets for the other two values of α. From left to right: GraphSage, GAT, GIN, GeneralConv.

post-processing module was added, comprising a linear layer, a ReLU activation, a 0.5-rate Dropout layer, and a final linear layer that produces a two-dimensional output. The optimizer was Adam. Preliminary experiments set the batch size, number of epochs, and learning rate to 8, 400, and 0.01, respectively.

Experimental setup Given the small dataset sizes, we used Repeated Random Sub-sampling to split the data randomly into training and test sets 10 times. The standard MAPE (Mean Absolute Percentage Error) metrics was computed on each test fold. The model's performance was averaged across the 10 splits. The split used for the `MLP1_expe` dataset was 80% (training) / 20% (test), while the split for other datasets was 90% / 10%. We varied λ hyperparameter in {0, 0.5, 1}. To study robustness to reduced training data, we introduced α, defined as the fraction of the training set actually used during training, and varied it in {0.3, 0.6, 1}.

4 Results and discussion

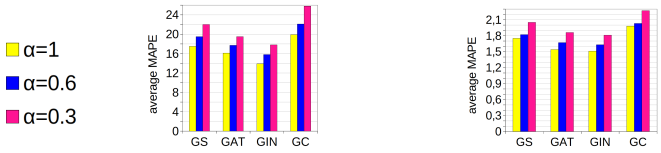
It was observed that each prediction required only a few tens of milliseconds.

Figure 4 shows that, regardless of the values of λ and α, and for all GNN types, predicting power consumption is much easier than predicting latency.

The most favorable situation for latency prediction occurs when $\lambda = 1$. Symmetrically, power-consumption prediction is optimal when $\lambda = 0$. The average latency MAPE increases (with a relative increase of up to 6.0%) in 10 out of 12 cases (4 GNN types $\times$ α values) when decreasing λ from 1 to 0.5; increasing λ from 0 to 0.5 leads to a relative increase in average power-consumption MAPE in 11 cases, reaching up to 13.5%. For latency, the already poor prediction quality at $\lambda = 1$ is only slightly worsened when using a weighted loss. In contrast, power-consumption prediction quality is relatively more affected, but this is not detrimental, since the quality is already very high under optimal conditions ($\lambda = 0$).

Figure 5 shows how decreasing the size of training data impacts the prediction performance. Over the 4 GNNs, the mean relative decrease of the latency average MAPE amounts to 11.5% when decreasing α from 1 to 0.6, and it amounts to 13.1% when decreasing α from 0.6 to 0.3. These relative decrease

ESANN 2026 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 22-24 April 2026, i6doc.com publ., ISBN 9782875870964. Available from http://www.i6doc.com/en/.



(a) latency, $\lambda = 0.5$ (b) power consumption, $\lambda = 0.5$

Fig. 5: Impact of α on prediction performance under the $\lambda = 0.5$ setting. GS: GraphSAGE, GC: GeneralConv. From left to right: α values are 1, 0.6 and 0.3.

amounts to 5.7% and 11.7% for the power consumption average MAPE. Thus, on the datasets considered, decreasing α from 1 to 0.6 remains acceptable for power consumption prediction. Gaining this robustness insights is nontrivial in machine-learning-assisted on-chip application deployment, where creating training datasets is time-consuming.

Figure 5 also highlights that GIN performs best for both latency and power consumption, followed by GAT. GraphSAGE and GeneralConv lag behind, with GeneralConv being the least accurate. This ranking is consistent across all datasets (not shown).

Neither code nor datasets used in [1] are available. In [4], the authors report MAPE values of at most 3% for both latency and power-consumption predictions and prediction times around tenths of seconds. Our GNN-based framework achieves comparable power-consumption performance 10 times faster.

5 Conclusion and future work

We presented a GNN-based framework to predict latency and power consumption in task-to-tile mappings. Experiments on datasets ranging from a few dozen to a few hundred instances show that GNN models can achieve low MAPE for power consumption, even with limited training data, producing predictions in real time. Although latency prediction errors remain relatively high, performance could be improved either by increasing the amount of training data or integrating additional knowledge beyond structural information. Achieving accurate latency prediction would enable very fast, on-the-fly bi-criteria evaluation of mappings, opening the way for integration into metaheuristic optimization.

References

[1] L. Ferretti, A. Cini, G. Zacharopoulos, C. Alippi and L. Pozzi, Graph Neural Networks for High-Level Synthesis Design Space Exploration, *ACM Transactions on Design Automation of Electronic Systems*, 28(2):1-20, 2022.

[2] J. You, Z. Ying, and J. Leskovec, Design space for graph neural networks. In *proceedings of Advances in Neural Information Processing Systems* (NeurIPS), pages 10727-10738, 2020.

[3] W. Ju, Z. Fang, Y. Gu, Z. Liu, Z. et al, A comprehensive survey on deep graph representation learning, *Neural Networks*, 169: 106207, 2024.

[4] Q. Dariol, Early timing and energy prediction and optimization of artificial neural networks on multi-core platforms. PhD Thesis, Nantes University, 2023.