MAchinE Learning for Scalable meTeoROlogy and climate

D3.5 Performance model for hardware configurations

Karthick Panner Selvam & Mats Brorsson

www.maelstrom-eurohpc.eu

D3.5 Performance model for hardware configurations

Author(s):	Karthick Panner Selvam
Mats Brorsson	
Dissemination Level:	Public
Date:	31/03/2023
Version:	1.0
Contractual Delivery Date:	31/03/2023
Work Package/ Task:	WP3/ T3.5
Document Owner:	UL-SnT
Contributors:	UL-SnT
Status:	Final

MAELSTROM

Machine Learning for Scalable Meteorology and Climate

Research and Innovation Action (RIA)
H2020-JTI-EuroHPC-2019-1: Towards Extreme Scale Technologies and Applications

Project Coordinator:	Dr Peter Dueben (ECMWF)
Project Start Date:	01/04/2021
Project Duration:	36 months

Published by the MAELSTROM Consortium

Contact:
ECMWF, Shinfield Park, Reading, RG2 9AX, United Kingdom
Peter.Dueben@ecmwf.int

The MAELSTROM project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955513. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and United Kingdom, Germany, Italy, Luxembourg, Switzerland, Norway

Contents

Figures

Tables

1 Executive Summary

Deep Learning (DL) has become a cornerstone in many everyday applications that we rely on. However, ensuring the DL model uses the underlying hardware efficiently takes much effort. Knowledge about inference characteristics can help to find the right match so that enough resources are given to the model, but not too much.

We have developed a DL Inference Performance Predictive Model (DIPPM) that predicts a given input DL model's inference latency, energy, and memory usage on the NVIDIA A100 GPU. We also devised an algorithm to suggest the appropriate A100 Multi-Instance GPU profile from the output of DIPPM.

We developed a methodology to convert DL models expressed in multiple frameworks to a generalized graph structure used in DIPPM. It means DIPPM can parse input DL models from various frameworks. DIPPM not only helps to find suitable hardware configurations but also helps to perform rapid design-space exploration for the inference performance of a model.

We constructed a graph multi-regression dataset consisting of 10,508 different DL models to train and evaluate the performance of DIPPM. As a result, we reached a resulting Mean Absolute Percentage Error (MAPE) as low as 1.9%.

We have applied the current version of DIPPM to the four MAELSTROM applications (AP1, AP3, AP4, and AP5). In addition, we plan to increase the DIPPM dataset to improve the accuracy and capability to predict the performance of MAELSTROM applications.

DIPPM currently only predicts inference on the A100 Nvidia GPU. Therefore, the next step in the project will be to develop support for multiple architectures and training, including distributed training. This work will be reported in Deliverables 3.7 and 3.8.

2 Introduction

2.1 About MAELSTROM

To develop Europe's computer architecture of the future, MAELSTROM will co-design bespoke compute system designs for optimal application performance and energy efficiency, a software framework to optimise usability and training efficiency for machine learning at scale, and large-scale machine learning applications for the domain of weather and climate science.

The MAELSTROM compute system designs will benchmark the applications across a range of computing systems regarding energy consumption, time-to-solution, numerical precision and solution accuracy. Customised compute systems will be designed that are optimised for application needs to strengthen Europe's high-performance computing portfolio and to pull recent hardware developments, driven by general machine learning applications, toward needs of weather and climate applications.

The MAELSTROM software framework will enable scientists to apply and compare machine learning tools and libraries efficiently across a wide range of computer systems. A user interface will link application developers with compute system designers, and automated benchmarking and error detection of machine learning solutions will be performed during the development phase. Tools will be published as open source.

The MAELSTROM machine learning applications will cover all important components of the workflow of weather and climate predictions including the processing of observations, the assimilation of observations to generate initial and reference conditions, model simulations, as well as post-processing of model data and the development of forecast products. For each application, benchmark datasets with up to 10 terabytes of data will be published online for training and machine learning tool developments at the scale of the fastest supercomputers in the world. MAELSTROM machine learning solutions will serve as a blueprint for a wide range of machine learning applications on supercomputers in the future.

2.2 Scope of this deliverable

2.2.1 Objectives of this deliverable

Many essential tasks now rely on Deep learning models, for instance, in computer vision and natural language processing domains [3,13]. In recent years, researchers have focused on improving the efficiency of deep learning models to reduce the computation cost and energy consumption and increase their throughput without losing accuracy. At the same time, hardware manufacturers like NVIDIA have increased their computing power. For example, the NVIDIA A100[1] GPU half-precision Tensor Core can perform matrix operations at 312 TFLOPS. Deep learning models typically involve many matrix operations, which can be computationally intensive and time-consuming. Graphics Processing Units (GPUs) are commonly used to accelerate these computations because they can perform many parallel computations simultaneously. However, not all deep learning models will

[1] https://www.nvidia.com/en-us/data-center/a100/

fully utilize the GPU's processing power because the workload and number of matrix operations required for a given problem domain can vary significantly. For this reason, NVIDIA created the Multi-Instance GPU (MIG[2]) technology starting from the Ampere architecture; they split the single physical GPU into multi-isolated GPU instances, so multiple applications can simultaneously run on different partitions of the same GPU, which then can be used more efficiently.

However, determining the DL model's efficiency on a GPU is not straightforward. If we could predict parameters such as inference latency, energy consumption, and memory usage, we would not need to measure them on deployed models, which is tedious and costly. The predicted parameters could also support efficient Neural Architecture Search (NAS) [5], efficient DL model design during development, and avoid job scheduling failures in data centers. According to Gao et al. [6], most failed deep learning jobs in data centers are due to out-of-memory errors.

In order to meet this need, we have developed a novel Deep Learning Inference Performance Predictive Model (DIPPM) to support DL model developers in matching their models to the underlying hardware for inference. As shown in Figure 1, DIPPM takes a deep learning model expressed in any of the frameworks: PyTorch, PaddlePaddle, Tensorflow, or ONNX, and will predict the latency (ms), energy (J), memory requirement (MB), and MIG profile for inference on an Nvidia A100 GPU without running on it. Currently, the model is restricted to inference and the Nvidia A100 architecture, but we aim to relax these restrictions in future work. As far as we know, this is the first predictive model that can take input from any of the mentioned frameworks and predict all of the metrics above.

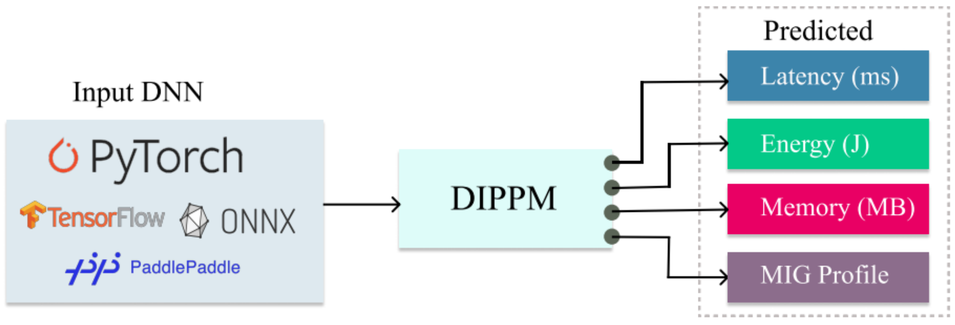


Figure 1: DIPPM can predict the Latency, Energy, Memory requirement, and MIG Profile for inference on an NVIDIA A100 GPU without running on it.

2.2.2 Work performed in this deliverable

- We have developed, trained and evaluated a performance predictive model which predicts inference latency, energy, memory, and MIG profile for A100 GPU with high accuracy.
- We have developed a methodology to convert deep learning models from various deep learning frameworks into generalized graph structures for graph learning tasks in our performance predictive model.
- We have devised an algorithm to suggest the MIG profile from predicted Memory for the given input DL model.

[2] https://docs.nvidia.com/datacenter/tesla/mig-user-guide/

- We have created an open-sourced performance predictive model dataset containing 10,508 graphs for graph-level multi-regression problems.
- The model has been applied on the MAELSTROM applications with an analysis on the model results.

2.2.3 Deviations and counter measures

The description of action (DOA) for the task where this deliverable D3.5 is written states:

> *In order to support the work in Task 3.4 and for being able to argue about the benefits of the proposed reference system design, performance models will be developed based on the benchmarking done in this task together with the benchmarking done in Task 2.4. This performance model will be parametrised for different architectural solutions with the purpose of guiding the design decisions of Task 3.4 and to project performance results of the test hardware configurations to large-scale compute systems. The performance model will be published as Deliverable D3.5.*

We have delivered a model but in order to reach its full potential, the model will in the coming months be extended with increased functionality as explained in section 7.

3 Related Work

Performance prediction of deep learning models on modern architecture is a rather new research field being attended to only since a couple of years back. Bouhali et al. [2] and Lu et al. [14] have carried out similar studies where a classical Multi-Layer Perceptron (MLP) is used to predict the inference latency of a given input DL model. Their approach was to collect high-level DL model features such as batch size, number of layers, and the total number of floating-point operations (FLOPS) needed. They then fed these features into an MLP regressor as input to predict the latency of the given model. Bai et al. [1] used the same MLP method but predicted both the latency and memory. However, the classical MLP approach did not work very well due to the inability to capture a detailed view of the given input DL model.

To solve the above problems, some researchers came up with a kernel additive method; they predict each kernel operation, such as convolution, dense, and LSTM, individually and sum up all kernel values to predict the overall performance of the DL model [8,15,18,20,22,24]. Yu et al. [23] used the wave-scaling technique to predict the inference latency of the DL model on GPU, but this technique requires access to a GPU in order to make the prediction.

Kaufman et al. and Dudziak et al. [9,4] used graph learning instead of MLP to predict each kernel value. Still, they used the kernel additive method for inference latency prediction. However, this kernel additive method did not capture the overall network topology of the model, and instead, it will affect the accuracy of the prediction. To solve the above problem, Liu et al. [12] used a Graph level task to generalize the entire DL model into node embeddings and predicted the inference latency of the given DL model. However, they did not predict other parameters, such as memory usage and energy consumption. Gao et al. [25] used the same graph-level task to predict the single iteration time and memory consumption for deep learning training but not for inference.

Li et al. [11] tried to predict the MIG profiles on A100 GPU for the DL models. However, their methodology is not straightforward; they used CUDA Multi-Process Service (MPS) values to predict the MIG, So the model must run at least on the target hardware once to predict the MIG Profile.

Most of the previous research work concentrated on parsing the input DL model from only one of the following frameworks (PyTorch, TensorFlow, PaddlePaddle, ONNX). As far as we are aware, none of the previous performance prediction models predicted Memory usage, Latency, Energy, and MIG profile simultaneously.

Our novel Deep Learning Inference Performance Predictive Model (DIPPM) fills a gap in previous work. DIPPM takes a deep learning model as input from various deep learning frameworks such as PyTorch, PaddlePaddle, TensorFlow, or ONNX and converts it to generalize graph with node features. We used a graph neural network and MIG predictor to predict the inference latency (ms), energy (J), memory (MB), and MIG profile for A100 GPU without running on it.

4 Methodology

The architecture of DIPPM consists of five main components: Relay Parser, Node Feature Generator, Static Feature Generator, Performance Model Graph Network Structure (PMGNS), and MIG Predictor, as shown in Figure 2. We will explain each component individually in this section.

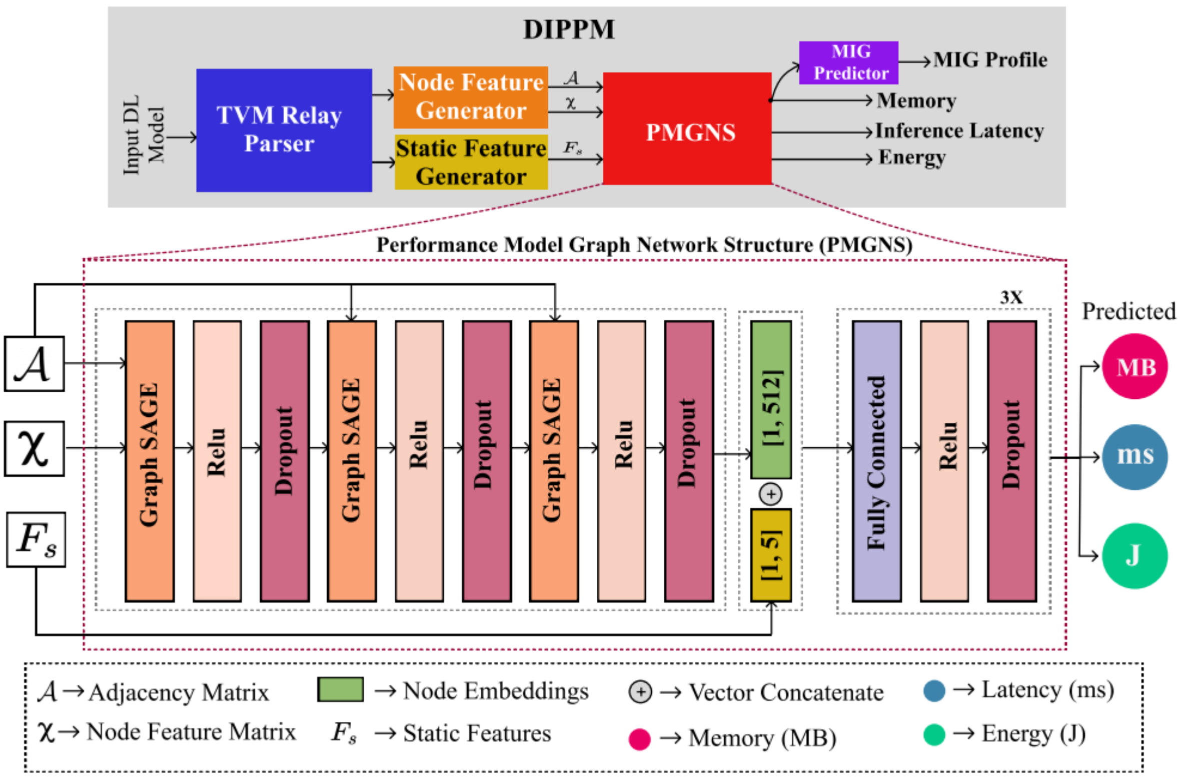


Figure 2: Overview of DIPPM Architecture

4.1 Deep Learning Model to TVM Relay IR

The Relay Parser takes as input a DL model expressed in one of several supported DL frameworks, converts it to an Intermediate Representation (IR), and passes this IR into the Node Feature Generator and the Static Feature Generator components.

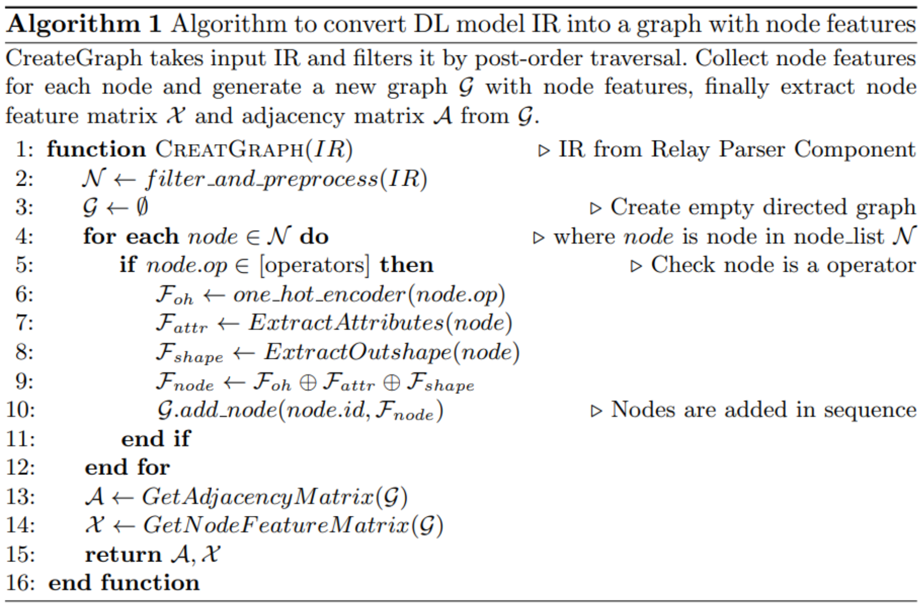
Most of the previously proposed performance models are able to parse the given input DL model from a single DL framework, not from several, as we already discussed in Section 3. To enable the use of multiple frameworks, we used a relay, which is a high-level IR for DL model. It has been used to compile DL models for inference in the TVM framework. We are inspired by the way they convert the DL model from various DL frameworks into a high-level IR format and therefore used their technique in our DIPPM architecture. It allows parsing given input DL models from various frameworks, but we have chosen to limit ourselves to PyTorch, TensorFlow, ONNX, and PaddlePaddle. We pass this DL IR to the subsequent components in our DIPPM architecture.

4.2 Node Feature Generator

The Node Feature Generator (NFG) converts the DL IR into an Adjacency Matrix (A) and a Node feature matrix (X) and passes this data to the PMGNS component.

The NFG takes the IR from the relay parser component. The IR is a computational data flow graph containing more information than is needed for our performance prediction. Therefore, we filter and pre-process the graph by post-order graph traversal to collect necessary node information. The nodes in the IR contain useful features such as operator name, attributes, and output shape of the operator, which after this first filtering step are converted into a suitable data format for our performance prediction. In the subsequent step, we loop through the nodes and, for each operator node, generate node features F_{node} with a fixed length of 32, as discussed in line number 9 on the Figure 3.

The central part of the NFG is to generate an **Adjacency Matrix** (A) and a **Node feature matrix** (X) as expressed in Figure 3. X has the shape of $[N_{op}, N_{features}]$, where N_{op} is the number of operator nodes in the IR and $N_{features}$ is the number of features. In order to create node features F_n for each *node*, first, we need to encode the node operator name into a one hot encoding as can be seen on Figure 3. Then extract the node attributes F_{attr} and output shape F_{shape} into vectors. Finally, perform vector concatenation to generate F_{node} for a node. We repeat this operation for each node and create the G. From the G, we extract A, X which are passed to the main part of our model, the Performance Model Graph Network Structure.



Algorithm 1 Algorithm to convert DL model IR into a graph with node features

CreateGraph takes input IR and filters it by post-order traversal. Collect node features for each node and generate a new graph $\mathcal{G}$ with node features, finally extract node feature matrix $\mathcal{X}$ and adjacency matrix $\mathcal{A}$ from $\mathcal{G}$.

```
1:  function CREATGRAPH($IR$)                          ▷ IR from Relay Parser Component
2:      $\mathcal{N} \leftarrow filter\_and\_preprocess(IR)$
3:      $\mathcal{G} \leftarrow \emptyset$                            ▷ Create empty directed graph
4:      for each $node \in \mathcal{N}$ do                ▷ where node is node in node_list $\mathcal{N}$
5:          if $node.op \in$ [operators] then                ▷ Check node is a operator
6:              $\mathcal{F}_{oh} \leftarrow one\_hot\_encoder(node.op)$
7:              $\mathcal{F}_{attr} \leftarrow ExtractAttributes(node)$
8:              $\mathcal{F}_{shape} \leftarrow ExtractOutshape(node)$
9:              $\mathcal{F}_{node} \leftarrow \mathcal{F}_{oh} \oplus \mathcal{F}_{attr} \oplus \mathcal{F}_{shape}$
10:             $\mathcal{G}.add\_node(node.id, \mathcal{F}_{node})$       ▷ Nodes are added in sequence
11:         end if
12:     end for
13:     $\mathcal{A} \leftarrow GetAdjacencyMatrix(\mathcal{G})$
14:     $\mathcal{X} \leftarrow GetNodeFeatureMatrix(\mathcal{G})$
15:     return $\mathcal{A}, \mathcal{X}$
16: end function
```

Figure 3: Node Feature Generator Algorithm

4.3 Static Feature Generator

The Static Feature Generator (SFG) takes the IR from the relay parser component and generates static features F_s for a given DL model and passes them into the graph network structure.

For this experiment, we limited ourselves to five static features. First, we calculate the F_{mac} total multiply-accumulate (MACs) of the given DL model. We used the TVM relay analysis API to calculate total MACs, but it is limited to calculating MACs for the following operators (in TVM notation): Conv2D, Conv2D transpose, dense, and batch matmul. Then we calculate the total number of convolutions F_{Tconv}, Dense F_{Tdense}, and Relu F_{Trelu} operators from the IR. We included batch size F_{batch} as one of the static features because it gives the ability to predict values for various batch sizes of a given model. Finally, we concatenate all the features into a vector F_s as expressed in equation 1. The feature set F_s is subsequently passed to the following graph network structure.

$$\mathcal{F}_s \leftarrow \mathcal{F}_{\text{mac}} \oplus \mathcal{F}_{\text{batch}} \oplus \mathcal{F}_{\text{Tconv}} \oplus \mathcal{F}_{\text{Tdense}} \oplus \mathcal{F}_{\text{Trelu}} \qquad \textit{Eq. 1}$$

4.4 Performance Model Graph Network Structure (PMGNS)

The PMGNS takes the node feature matrix (X), the adjacency matrix (A) from the Node Feature Generator component, and the feature set (F_s) from the Static feature generator and predicts the given input DL model's memory, latency, and energy, as shown in Figure 2.

The PMGNS must be trained before prediction, as explained in section 5. The core idea of the PMGNS is to generate the node embedding z from X and A and then to perform vector concatenation of z with F_s. Finally, we pass the concatenated vector into a Fully Connected layer for prediction, as shown in Figure 2. In order to generate z, we used the graphSAGE algorithm suggested by Hamilton et al. [7], because of its inductive node embedding, which means it can generate embedding for unseen nodes without pretraining.

GraphSAGE is a graph neural network framework that learns node embeddings in large-scale graphs by aggregating information from the nodes and their neighbours. GraphSAGE is designed to perform inductive learning, which means it can generalize to unseen nodes not present in the training set. This is achieved by learning a fixed size embedding for each node in the graph that captures the node's and its neighbours' features. During training, GraphSAGE uses a neighbourhood aggregation scheme to iteratively sample and aggregate the features of a node's neighbours, allowing it to capture local graph structure and generate node embeddings sensitive to each node's local neighbourhood. When new nodes are added to the graph, GraphSAGE can use the learned neighbourhood aggregation scheme to generate embeddings for these nodes based on their local neighbourhood. This allows GraphSAGE to perform inductive learning, as it can generalize its node embeddings to new nodes not present in the training set.

We already discussed that we generate node features of each node in the section 4.2. The graphSAGE algorithm will convert node features into a node embedding z which is more amenable for model training. The PMGNS contains three sequential graphSAGE blocks and three sequential Fully connected (FC) blocks as shown in Figure 2. At the end of the final graphSAGE block, we get the generalized node embedding of given X and A, which we concatenate with F_s. Then we pass the concatenated vector into FC to predict the memory (MB), latency (ms), and energy (J).

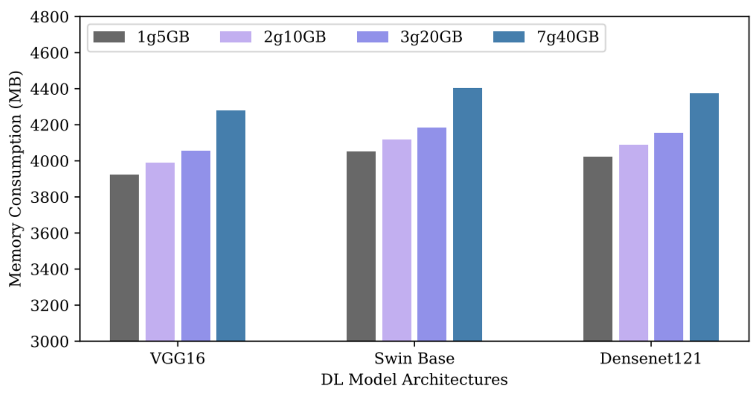


Figure 4: MIG Profile comparison of three different DL models memory consumption on A100 GPU. We used batch size 16 for VGG16 and Densenet121 model and batch size 8 for Swin base model.

4.5 MIG Predictor

The MIG predictor takes the memory prediction from PMGNS and predicts the appropriate MIG profile for a given DL model, as shown in Figure 2.

As mentioned in the introduction, the Multi-instance GPU (MIG) technology allows to split an A100 GPU into multiple instances so that multiple applications can use the GPU simultaneously. The different instances differ in their compute capability and, most importantly, in the maximum memory limit that is allowed to be used. The four MIG profiles of the A100 GPU that we consider here are: 1g.5gb, 2g.10gb, 3g.20gb, and 7g.40gb, where the number in front of ”gb” denotes the maximum amount of memory in GB that the application can use on that instance. For example, the maximum memory limit of 1g.5gb is 5GB, and 7g.40gb is 40GB.

For a given input DL model, PMGNS predicts memory for 7g.40gb MIG profile, which is the full GPU. We found that this prediction can be used as a pessimistic value to guide the choice of MIG profile. Figure 4 shows manual memory consumption measurements of the exact DL model inference on different profiles. The results show no significant difference in the memory allocation of DL in the different MIG profiles, even though the consumption slightly increases with the capacity of the MIG profile. Memory consumption is always the highest when running on the 7g.40gb MIG profile.

As mentioned, PMGNS predicts memory for 7g.40gb, so we claim that predicted memory will be an upper bound. Then we perform a rule-based prediction to predict the MIG profile for the given input DL model, as shown in equation 2. Where α has predicted memory from PMGNS.

$$\mathrm{MIG}\ (\alpha) = \begin{cases} 1\mathrm{g}.5\mathrm{gb}, \ \text{if } 0gb \ < \alpha < 5\mathrm{gb} \\ 2\mathrm{g}.10\mathrm{gb}, \text{if } 5\mathrm{gb} \ < \alpha < 10\mathrm{gb} \\ 3\mathrm{g}.20\mathrm{gb}, \text{if } 10\mathrm{gb} < \alpha < 20\mathrm{gb} \\ 7\mathrm{g}.40\mathrm{gb}, \text{if } 20\mathrm{gb} < \alpha < 40\mathrm{gb} \\ \text{None, otherwise} \end{cases} \qquad \textit{Eq. 2}$$

5 Experiments & Results

5.1 The DIPPM Dataset

We constructed a graph-level multi-regression dataset containing 10,508 DL models from different model families to train and evaluate our DIPPM. The dataset distribution is shown in **Fehler! Verweisquelle konnte nicht gefunden werden.**. To the best of our knowledge, the previous predictive performance model dataset doesn't capture memory consumption, inference latency, and energy consumption parameters for wide-range DL models on A100 GPU so we created our own dataset for performance prediction of DL models.

Our dataset consists of DL models represented in graph structure, as generated by the Relay parser described in section 4.1. We used the Nvidia Management Library[3] and the CUDA toolkit[4] to measure the energy, memory, and inference latency of each given model in the dataset. For each model, we ran the inference five times to warm up the architecture and then the inference 30 times, and then took the arithmetic mean of those 30 values to derive the Y, where Y consists of inference latency (ms), memory usage (MB), and energy (J) for a given DL on A100 GPU.

We used a full A100 40GB GPU, or it is equivalent to using 7g.40gb MIG profile to collect all the metrics.

Table 1: DIPPM Graph dataset distribution

Model Family	# of graphs	Percentage (%)
Efficientnet	1729	16.45
Mnasnet	1001	9.53
Mobilenet	1591	15.14
Resnet	1152	10.96
Vgg	1536	14.62
Swin	547	5.21
Vit	520	4.95
Densenet	768	7.31
Visformer	768	7.31
Poolformer	896	8.53
Total	**10508**	**100%**

[3] https://developer.nvidia.com/nvidia-management-library-nvml
[4] https://developer.nvidia.com/cuda-toolkit

5.2 Environment setup

We used an HPC cluster at the Jülich research centre in Germany called JUWELS Booster for our experiments[5]. It is equipped with 936 nodes, each with AMD EPYC 7402 processors, 2 sockets per node, 24 cores per socket, 512 GB DDR43200 RAM and 4 NVIDIA A100 Tensor Core GPUs with 40 GB HBM. The main software packages used in the experiments are Python 3.10, CUDA 11.7 torch 1.13.1, torch-geometric 2.2.0, torch-scatter 2.1.0, and torch-sparse 0.6.16.

Table 2: Settings in GNN comparison

Settings	Value
Dataset partition	Train (70%) / Validation (15%) / Test (15%)
Nr hidden layers	512
Dropout probability	0.05
Optimizer	Adam
Learning rate	$2.754 \cdot 10^{-5}$
Loss function	Huber

5.3 Evaluation

The Performance Model Graph Network Structure is the main component in DIPPM, and we used the PyTorch geometric library to create our model, as shown in **Fehler! Verweisquelle konnte nicht gefunden werden.**: Settings in GNN comparison. We split our constructed dataset into three parts randomly: a training set 70%, a validation set of 15%, and a test set of 15%.

In order to validate that graphSAGE performs better than other GNN algorithms and plain MLP, we compared graphSAGE with the following other algorithms: GAT [19], GCN [10], GIN [21], and finally, plain MLP without GNN. **Fehler! Verweisquelle konnte nicht gefunden werden.** summarizes the settings used. The learning rate was determined using a learning rate finder as suggested by Smith [17]. The Huber loss function achieved a higher accuracy than mean square error, which is why we chose that one.

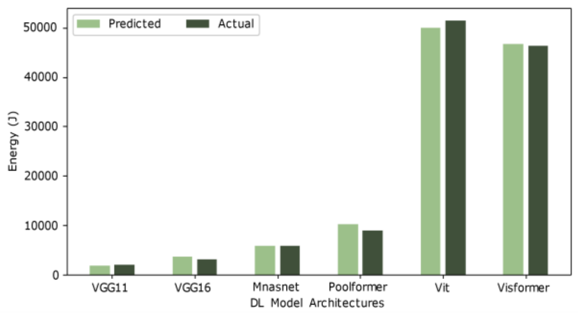
For the initial experiment, we trained for 10 epochs and used Mean Average Percentage Error (MAPE) as accuracy metric to validate DIPPM. A MAPE value close to zero indicates good performance on regression prediction. **Fehler! Verweisquelle konnte nicht gefunden werden.** shows that graphSAGE gives a lower MAPE value in all of training, validation, and test datasets. Without using a GNN, MLP gives 0.366 of MAPE. With graphSAGE, MAPE is 0.160 on the test dataset which is a significant improvement on a multi-regression problem. We conclude that graphSAGE outperforms other GNN algorithms, and MLP because of its inductive learning, as discussed in section 4.4.

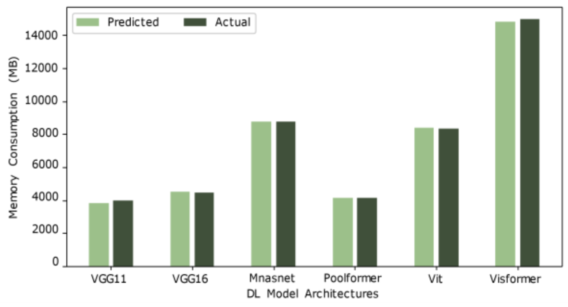
[5] https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

Table 3: Comparison with different GNN algorithms and MLP with graphSAGE, we trained all the models for 10 epochs and used Mean Average Percentage Error for validation. The results indicate that DIPPM with graphSAGE performs better than other variants.

Model	Training	Validation	Test
GAT	0.497	0.379	0.367
GCN	0.212	0.178	0.175
GIN	0.488	0.394	0.382
MLP	0.371	0.387	0.366
(Ours) GraphSAGE	**0.182**	**0.159**	**0.160**

After this encouraging result, we increased the number of epochs for training our DIPPM with graphSAGE to increase the prediction accuracy. After 500 epochs, we attained a MAPE of 0.041 on training and 0.023 on the validation dataset. In the end, we attained 1.9% MAPE on the test dataset. Some of the DIPPM predictions on the test dataset are shown in **Fehler! Verweisquelle konnte nicht gefunden werden.**.





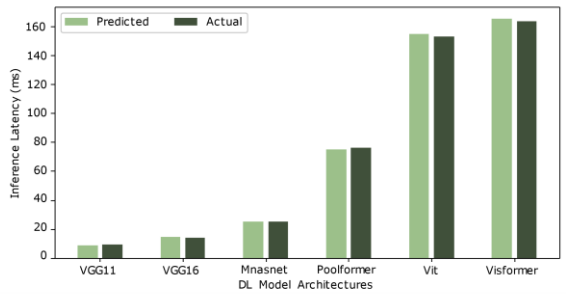


Figure 5: Comparison of actual value with DIPPM predicted values on the test dataset. Results show that DIPPM predictions are close to the actual predictions.

5.4 Prediction of MIG Profiles

In order to verify the MIG profile prediction for a given DL model, we compared the actual MIG profile value with the predicted MIG profile from the DIPPM, as shown in **Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.**. To calculate the actual suitable MIG profile, we divide actual memory consumption by the maximum memory limit of the MIG profiles. The higher the value is, the more appropriate profile for the given DL model.

For example, the predicted memory consumption for densenet121 at batch size 8 is 2865 MB. DIPPM predicted for 7g.40GB MIG profile. The actual memory consumption for the 7g.40gb MIG profile is 3272 MB. The actual memory consumption of 1g.5GB is 2918 MB, the percentage is 58%. Which is higher than other MIG profiles. Results show that DIPPM correctly predicted the MIG profile 1g.5gb for densenet121.

It is interesting to note that the densent121 models are from our test dataset, and the swin base patch4 model is not in our DIPPM dataset, but a similar swin base model family was used to train DIPPM. The convnext models are completely unseen to our DIPPM, but it's still predicting the MIG profile correctly.

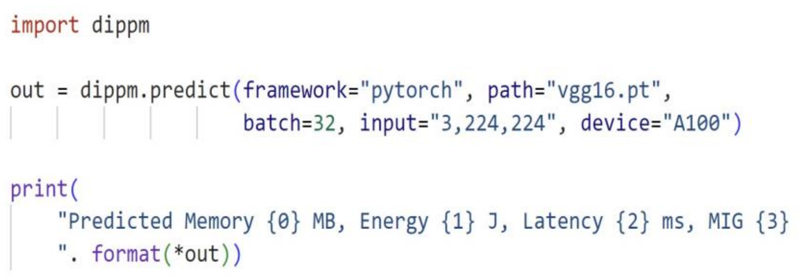
Table 4: DIPPM MIG profile prediction for seen and unseen DL model architectures. (densenet: seen, swin*: partially seen, convnext*: unseen).*

Model	Batch size	Predicted		Actual				
		MIG	Mem	Mem	1g. 5gb	2g.10gb	3g.20gb	7g.40gb
densenet121	8	1g.5gb	2865	3272	**58%**	30%	15%	8%
densenet121	32	2g.10gb	5952	6294		**60%**	30%	16%

swin base patch4	2	1g.5gb	2873	2944	**52%**	27%	14%	7%
swin base patch4	16	2g.10gb	6736	6156		**59%**	30%	15%
convnext base	4	1g.5gb	4771	1652	**61%**	31%	16%	4%
convnext base	128	7g.40gb	26439	30996				**77%**

5.5 DIPPM Usability aspects

DIPPM takes basic parameters like frameworks, model path, batch, and input size, and finally, device type. As of now, we only considered A100 GPU; we are working to extend DIPPM to various hardware platforms. With a simple python API call, DIPPM predicts memory, latency, energy, and MIG profile for the given model, as can be seen in Figure 6.



```
import dippm

out = dippm.predict(framework="pytorch", path="vgg16.pt",
                    batch=32, input="3,224,224", device="A100")

print(
    "Predicted Memory {0} MB, Energy {1} J, Latency {2} ms, MIG {3}
    ". format(*out))
```

Figure 6: A sample code to use DIPPM for performance prediction of VGG16 DL model developed by PyTorch framework.

6 Application on MAELSTROM Applications

MAELSTROM have six different applications to solve various weather and climate related problems. First, we will explain each application and later we will discuss about DIPPM limitations and prediction on each MASELSTOM application. More details of each application can be found in D1.2[6].

The objective of AP1 is to train deep learning models on TBs of training data with input grid shape of [1, 512,512,17] to improve MET Norway's operational short-range forecasts of temperature and precipitation for the Nordic countries. AP1 used TensorFlow framework to train their model. They used CNN architecture for training and prediction.

The objective of AP2 to enhance weather prediction by incorporating the Twitter information. They used Hugging face NLP based transformer model call Deberta to train their transformer model.

The objective of AP3 to build neural network emulators to speed-up weather forecast models and data assimilation. AP3 used TensorFlow framework to build their deep learning model.

The objective of AP4 to improve the forecast skill for global ensemble predictions as a post-processing step by using deep neural network. AP4 used PyTorch framework and CNN based architecture to train their model. AP4 accepts input grid size of [14, 361, 720].

The objective of AP5 to improve the spatial resolution of input atmospheric temperature by using deep learning-based downscaling technique. AP5 used TensorFlow framework and CNN based architecture to train their model. AP5 accepts input grid size of [96, 120, 10].

The objective of AP6 to improve predictions of power production from renewable energy sources in order to optimise the production of renewable energy. AP6 used classical random forest regressor model to train their model to predict the power production.

We have applied DIPPM on four of the MAELSTROM applications (AP1, AP3, AP4 and AP5). DIPPM is limited to predict only deep learning models, so we excluded AP6 for prediction as it is not deep learning. AP2 uses an NLP transformer model architecture and the current version of DIPPM is not yet adapted for NLP based transformer models which is why it is excluded here. We plan to extend DIPPM capability on predicting AP2 performance for the future.

Table 5:DIPPM prediction results for AP1

	Inference latency (ms)	Memory Consumption (MB)	Energy (J)
Actual values	176.6	40227	55502
DIPPM prediction	3.6	219	378
MAPE %	97.9	99.9	99.3

[6] https://www.maelstrom-eurohpc.eu/content/docs/uploads/doc8.pdf

Table 6: DIPPM prediction results for AP3

	Inference latency (ms)	Memory Consumption (MB)	Energy (J)
Actual values	130	2135	13296
DIPPM prediction	67	3004	19564
MAPE %	48.4	40.7	47.14

Table 7: DIPPM prediction results for AP4

	Inference latency (ms)	Memory Consumption (MB)	Energy (J)
Actual values	4.9	2823	470
DIPPM prediction	2.7	2166	283
MAPE %	44.8	23.2	23.7

Table 8: DIPPM prediction results for AP5

	Inference latency (ms)	Memory Consumption (MB)	Energy (J)
Actual values	8.1	2733	1278
DIPPM prediction	18.6	2758	4107
MAPE %	129.6	0.9	221.3

Discussion:

AP1, AP4, and AP5 are based on CNN, and AP3 is based on RNN. The DIPPM dataset contains popular CNN and vision-based transformer architectures, as discussed in section 5.1. We have not yet included all MAELSTROM applications in the dataset because we want to avoid biased performance predictions towards MAELSTROM applications. We, therefore, did not include the model architectures similar to MAELSTROM application in the training dataset. We used MAPE% as evaluating metrics. When the MAPE% are closer to zero indicates good prediction accuracy. DIPPM predicts good accuracy on AP5 memory consumption but low accuracy on inference latency and energy metrics, as shown in Table 8. DIPPM prediction results on AP1, AP3, and AP4 MAPE% values are higher on latency, memory and energy metrics, which indicates DIPPM is not predicting with reasonable accuracy, as shown in Table 5 - 7.

We assumed DIPPM could generalize four MAELSTROM applications with the current dataset. This study on the MAELSTROM application shows that our assumption was wrong. DIPPM could not efficiently generalize the given MAELSTROM applications with the current dataset. We plan to include model architectures similar to MAELSTROM applications in the DIPPM dataset to improve prediction accuracy.

Planned Action

To increase the DIPPM prediction accuracy on (AP1, AP3, AP4 and AP5) and capability to predict AP2 performance.

- We plan to include the following models in the DIPPM dataset: 3D CNN models, NLP-based transformers models, and weather and climate prediction models and models like MAELSTROM applications.
- Increase the static feature attributes of DIPPM to incorporate more model characteristics of the given DL model.

These changes will increase the DIPPM prediction accuracy not only for all five MALESTROM but also improve the accuracy of generic DL models.

7 Roadmap of future development

The current version of DIPPM is limited to predicting only Inference characteristics of a given deep learning model. We plan to extend the capability for predicting Training and Distributed training performance characteristics. In order to help Task 3.4, develop a “solution design and architecture blueprint” for MAELSTROM applications. We plan to extend our predictive performance model on seen and unseen hardware architectures by feeding hardware configurations as inputs to our predictive performance model.

8 Conclusion

We have developed a novel Deep Learning (DL) Inference Performance Predictive Model (DIPPM) to predict the inference latency, energy, and memory consumption of a given input DL model on an A100 GPU without running on it. Furthermore, we devised an algorithm to select the appropriate MIG profile from the memory consumption predicted by DIPPM.

The model includes a methodology to convert the DL model represented in various frameworks to a generalized graph structure for performance prediction. To our knowledge, DIPPM can help develop an efficient DL model to utilize the underlying GPU effectively. Furthermore, we constructed a multi-regression graph dataset containing 10,508 DL models for performance prediction. It can even be used to evaluate other graph-based multi-regression GNN algorithms. We achieved 1.89% MAPE on our dataset.

Finally, we have applied the current version of DIPPM to the four MAELSTROM applications (AP1, AP3, AP4, and AP5) and analysed the prediction results. We plan to increase the DIPPM dataset to improve accuracy and capability to predict the performance of MAELSTROM applications.

9 References

1) Bai, L., Ji, W., Li, Q., et al.: Dnnabacus: Toward accurate computational cost prediction for deep neural networks. CoRR **abs/2205.12095** (2022)

2) Bouhali, N., Ouarnoughi, H., Niar, S., et al.: Execution time modeling for cnn inference on embedded gpus. In: Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings. p. 59–65. DroneSE and RAPIDO '21, Association for Computing Machinery, New York, NY, USA (2021)

3) Brown, T.B., Mann, B., Ryder, N., et al.: Language models are few-shot learners. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS'20, Curran Associates Inc., Red Hook, NY, USA (2020)

4) Dudziak, L., Chau, T., Abdelfattah, M.S., et al.: Brp-nas: Prediction-based nas using gcns. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS'20, Curran Associates Inc., Red Hook, NY, USA (2020)

5) Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. J. Mach. Learn. Res. **20**(1), 1997–2017 (mar 2021)

6) Gao, Y., Liu, Y., Zhang, H., et al.: Estimating GPU memory consumption of deep learning models. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 1342–1352. ESEC/FSE 2020, Association for Computing Machinery, New York, NY, USA (Nov 2020)

7) Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 1025–1035. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)

8) Justus, D., Brennan, J., Bonner, S., et al.: Predicting the computational cost of deep learning models. In: 2018 IEEE International Conference on Big Data (BigData). pp. 3873–3882. IEEE Computer Society, Los Alamitos, CA, USA (dec 2018)

9) Kaufman, S., Phothilimthana, P., Zhou, Y., et al.: A learned performance model for tensor processing units. In: Smola, A., Dimakis, A., Stoica, I. (eds.) Proceedings of Machine Learning and Systems. vol. 3, pp. 387–400 (2021)

10) Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)

11) Li, B., Patel, T., Samsi, S., et al.: Miso: Exploiting multi-instance gpu capability on multi-tenant gpu clusters. In: Proceedings of the 13th Symposium on Cloud Computing. p. 173–189. SoCC '22, Association for Computing Machinery, New York, NY, USA (2022)

12) Liu, L., Shen, M., Gong, R., et al.: Nnlqp: A multi-platform neural network latency query and prediction system with an evolving database. In: Proceedings of the 51st International Conference on Parallel Processing. ICPP ’22, Association for Computing Machinery, New York, NY, USA (2023)

13) Liu, Z., Lin, Y., Cao, Y., et al.: Swin transformer: Hierarchical vision transformer using shifted windows. In: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 9992–10002. IEEE Computer Society, Los Alamitos, CA, USA (oct 2021)

14) Lu, Z., Rallapalli, S., Chan, K., et al.: Augur: Modeling the resource requirements of convnets on mobile devices. IEEE Transactions on Mobile Computing **20**(2), 352–365 (2021)

15) Qi, H., Sparks, E.R., Talwalkar, A.: Paleo: A performance model for deep neural networks. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net (2017)

16) Roesch, J., Lyubomirsky, S., Weber, L., et al.: Relay: A new ir for machine learning frameworks. In: Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. p. 58–68. MAPL 2018, Association for Computing Machinery, New York, NY, USA (2018)

17) Smith, L.N.: Cyclical learning rates for training neural networks. In: 2017 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 464–472 (2017)

18) Sponner, M., Waschneck, B., Kumar, A.: Ai-driven performance modeling for ai inference workloads. Electronics **11**(15) (2022)

19) Veliˇckovi´c, P., Cucurull, G., Casanova, A., et al.: Graph Attention Networks. International Conference on Learning Representations (2018), accepted as poster

20) Wang, C.C., Liao, Y.C., Kao, M.C., et al.: Toward accurate platform-aware performance modeling for deep neural networks. SIGAPP Appl. Comput. Rev. **21**(1), 50–61 (jul 2021)

21) Xu, K., Hu, W., Leskovec, J., et al.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019)

22) Yang, C., Li, Z., Ruan, C., et al.: PerfEstimator: A Generic and Extensible Performance Estimator for Data Parallel DNN Training. In: 2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence). pp. 13–18 (May 2021)

23) Yu, G.X., Gao, Y., Golikov, P., et al.: Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training. In: Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC’21) (2021)

24) Zhang, L.L., Han, S., Wei, J., et al.: Nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services. p. 81–93. MobiSys '21, Association for Computing Machinery, New York.

25) Yanjie Gao, Xianyu Gu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. 2021. Runtime Performance Prediction for Deep Learning Models with Graph Neural Network. Technical Report MSR-TR-2021-3. Microsoft. https://www.microsoft.com/en-us/research/publication/runtime- perf

Document History

Version	Author(s)	Date	Changes
0.1	Karthick Panner Selvam and Mats Brorsson (UL-SnT)	20/03/2023	Initial draft
1.0	Karthick Panner Selvam and Mats Brorsson (UL-SnT)	31/03/2023	Final version

Internal Review History

Internal Reviewers	Date	Comments
Thomas Nipen (MetNor)	29/03/2023	Minor comments and suggestions provided
Saleh Ashkboos (ETH Zurich)	29/03/2023	Minor comments and suggestions provided

Estimated Effort Contribution per Partner

Partner	Effort
UL-SnT	6 PM
Total	**06 PM**