HiFi-SAGE: High Fidelity GraphSAGE-based Latency Estimators for DNN Optimization

Shambhavi Balamuthu Sampath*[1,2], Leon Hecht[1], Moritz Thoma[1,2], Lukas Frickenstein[1,2], Pierpaolo Mori[2,3], Nael Fasfous[2], Manoj Rohit Vemparala[2], Alexander Frickenstein[2], Walter Stechele[1], Daniel Mueller-Gritschneder[1], Claudio Passerone[3]
[1]Technical University of Munich, Munich, Germany; [2]BMW Group, Munich, Germany; [3]Politecnico Di Torino, Turin, Italy

***Abstract*—As deep neural networks (DNNs) are increasingly deployed on resource-constrained edge devices, optimizing and compressing them for real-time performance becomes crucial. Traditional hardware-aware DNN search methods often rely on inaccurate proxy metrics, expensive latency lookup tables, or slow hardware-in-the-loop (HIL) evaluations. To address this, quasi-generalized latency estimators, typically meta-learning-based, were proposed to replace HIL evaluations and accelerate the search. These come with a one-time data collection and training cost and can adapt to new hardware with few measurements. However, they still have some drawbacks: (1) They increase complexity by trying to generalize across a range of diverse hardware types; (2) They depend on handcrafted hardware descriptors, which may fail to capture hardware characteristics; (3) They often perform poorly on new, unseen hardware that significantly differs from their initial training set. To overcome these challenges, this paper turns to the more straightforward platform-specific estimators that do not require hardware descriptors and can be easily trained on any hardware. We introduce HiFi-SAGE, a high fidelity GraphSAGE-based platform-specific latency estimator. When trained from scratch on only 100 latency measurements, our novel dual-head estimator design surpasses the state-of-the-art (SoTA) on the 10% error bound metric by up to 17.4 p.p. while achieving an impressive fidelity score of 99% on the diverse LatBench dataset. We demonstrate that applying HiFi-SAGE to a genetic algorithm-based DNN compression search, achieved a Pareto front comparable to real HIL feedback with a mean absolute percentage error (MAPE) of 2.54%, 2.48%, and 4.16%, for InceptionV3, DenseNet169, and ResNet50 respectively. Compared to existing platform-specific works, the lower number of latency measurements and higher fidelity scores positions HiFi-SAGE as an attractive alternative to replace expensive HIL setups. Code is available at: https://github.com/shamvbs/HiFi-SAGE** [*]

I. Introduction

Modern day deep neural network (DNN) edge deployments necessitate high task accuracies in resource-constrained environments with real-time latency budgets. To meet these demands, DNNs must be jointly optimized for both task and hardware metrics via hardware-aware (1) neural architecture search (NAS) [1] to find architectures tailored to specific hardware and/or (2) DNN compression search [2], [3] to reduce compute and memory overhead while preserving task performance. Both these approaches are iterative and necessitate numerous evaluations for task accuracy and latency throughout the search. This often demands procuring multiple deployment hardware or resorting to serverless inference [4] to maintain search speed. On-device measurements pose challenges in maintaining complex hardware-in-the-loop (HIL) setups and dealing with hardware unavailability. Often, the latency measurements are repeated multiple times to filter out the noisy ones, further adding to the overhead. Consequently, inference latency estimators have gained traction, as they accelerate search algorithms to prioritize optimal neural architectures for the target hardware. However, inaccurate latency estimations can severely misguide the search, leading to sub-optimal solutions. To address this, it is crucial that the estimated latencies reflect the compile-time and runtime DNN optimizations provided by hardware accelerators. The same network executed with different runtimes (e.g., TensorRT [5], ONNX Runtime [6]) can produce varying latencies due to optimizations like graph simplification, layer fusion, etc. This highlights the need for highly reliable end-to-end latency estimators that can replace HIL setups by accounting for inter-layer intricacies. Graph neural networks (GNNs) are well-known for capturing complex relationships in tasks like recommendation systems [7], and molecular chemistry [8]. They excel in handling inputs of diverse sizes, capturing structural patterns by learning shared representations of entire graphs, facilitating knowledge transfer even when labeled data is limited. Since DNNs are essentially directed graphs, GNNs can capture their inter-layer dependencies to enable end-to-end inference latency prediction. There are state-of-the-art (SoTA) platform-specific end-to-end latency estimators, such as nn-meter [9] and BRP-NAS [10]. Nn-meter uses a non-GNN approach to predict latency at the kernel level, demonstrating high accuracy but requiring expensive data collection and assuming only sequential kernel execution. In contrast, BRP-NAS employs Graph Convolutional Networks (GCNs) for graph-level latency estimation on the NASBench dataset [1], and addresses sample efficiency by training with only 900 measurements. Other GNN-based latency estimators are platform-agnostic. They often use meta-learning [11], [12], and are effective in NAS search spaces [1], [13] and on unseen hardware by training on a large pool of devices and fine-tuning for similar new hardware with minimal measurements. However, this method is only effective when the target hardware is similar to the pre-training devices [12]. HiFi-SAGE stands out by eliminating the costly pre-training phase and building a platform-specific latency estimator with a consistent core design and few measurements, similar to BRP-NAS but by supporting any arbitrary DNN. This makes HiFi-SAGE both efficient and highly tailored to specific devices, offering a more

*Correspondence to: shambhavi.balamuthu-sampath@tum.de

978-3-9826741-0-0/DATE25/© 2025 EDAA

practical and cost-effective solution for latency estimation.

Our main contributions are three-fold:

- We introduce HiFi-SAGE, a high fidelity GraphSage-based latency estimator that is trained from scratch using 100 measurements on any hardware **without** pre-training.
- We demonstrate that HiFi-SAGE outperforms the state-of-the-art latency estimators [10]–[12] with a high fidelity score of 0.99 and 0.94 across the diverse devices in LatBench [10] and HW-NAS-Bench [14] respectively.
- We showcase the integration of HiFi-SAGE in a genetic algorithm-based DNN compression search for edge deployment where the HiFi-SAGE guided search generated comparable Pareto fronts to the HIL-guided search.

II. Related Work

In this section, we discuss the literature on SoTA latency estimators that apply to the HW-aware NAS or compression methods, which we categorize as follows: (1) ***No pretraining***, i.e., if the latency estimator does not rely on extensive data-collection for pre-training, (2) ***GNN-based*** if its estimator architecture is a GNN and carries out graph-level latency predictions, (3) ***Arbitrary DNNs*** if it can capture architectures outside of NAS datasets with operator-level encoding and not cell-based encoding (4) ***Measurement-efficient*** if its usable with 150 measurements or less. This categorization of related works is shown in Table I.

A. Platform-specific latency estimators

nn-Meter [9] can predict the latency of *arbitrary DNNs* by summing up kernel-level latency predictions and deriving fusion rules from hardware measurements. However, it requires a significant data collection cost for pre-training, which can take up to 4.4 days depending on the runtime [15]. Additionally, nn-Meter assumes sequential kernel execution and does not model concurrent executions, which is a limitation for modern hardware. HiFi-SAGE addresses these issues. First, it eliminates the need for exhaustive pre-training and data collection; our latency estimator can be trained in a few minutes, and data collection takes less than 1.5 hours, assuming a 1-minute per inference setup. Second, by capturing runtime optimizations through measurements and predicting at a graph level, HiFi-SAGE inherently models all parallelism without hypothesizing fusion rules on blackbox edge hardware.

BRP-NAS [10], a popular work in graph-level latency prediction, uses a binary relation predictor for accuracy and a graph convolution-based latency predictor in a NAS framework. While it achieves *measurement-efficiency* by using 900 samples for training, it is limited to NASBench-201 architectures with a cell-based encoding scheme. HiFi-SAGE improves upon BRP-NAS by further enhancing measurement efficiency and encoding architectures at an operator level.

B. Quasi-generalized latency estimators

HELP [11] re-defines measurement-efficiency on a different paradigm. After a measurement-intensive pre-training phase over a large pool of 18 devices, it resorts to a measurement-efficient transfer phase to enable retargeting to unseen, but *correlated* devices. More recently, MultiPredict [12] improves upon HELP by emphasizing measurement-efficiency in both pre-training and transfer phases. Both methods focus on generating representative hardware descriptors for multi-hardware, multi-search space settings, rather than improving the estimator design itself, which is based largely based on [10]. In contrast, HiFi-SAGE focuses on the estimator design through design space exploration, with an additional ranking head that enhances *measurement-efficiency*. Moreover, it eliminates the need for designing representative hardware embeddings.

In summary, the SoTA end-to-end latency estimators focus on NAS and are measurement-inefficient with intensive pre-training and limited generalizability. In contrast, HiFi-SAGE aims for measurement-efficient training with a consistent estimator design that can encode any arbitrary DNN across a wide range of devices.

Related Work	No Pre-training	GNN-based	Arbitrary DNNs	Measurement-efficient
nn-meter [9]	✗	✗	✓	✗
BRP-NAS [10]	✓	✓	✗	✗
HELP [11]	✗	✓	✓	✓
MultiPredict [12]	✗	✓	✓	✓
HiFi-SAGE (Ours)	✓	✓	✓	✓

TABLE I: An overview of the related work. HiFi-GCN differs in its ability to train with limited measurements from the target device to outperform SoTA in the low-data regime.

III. Background: Graph Neural Networks

Graph neural networks (GNNs) are a class of architectures that operate on general graphs and are widely adopted for their ability to model interactions between the graph nodes [7]. A typical GNN for a graph regression task consists of a series of *message passing* layers to gather structural and feature information from its neighbourhood and a final aggregation function, also known as the readout layer, to obtain the graph embedding. Consider an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with a set of $\mathcal{V}$ nodes and $\mathcal{E}$ edges between the nodes, along with a set of node features $X \in \mathbb{R}^{d \times |\mathcal{V}|}$, where d is the dimension of the node features. Each node, $u \in \mathcal{V}$ goes through (1) aggregation of node features from its neighbourhood $\mathcal{N}(u)$ to generate a message, and (2) update at layer k of its node embedding $h_u^{(k)}$ with the message. The node features of the graph form the initial node embeddings. The general message passing function can be written as:

$$h_u^{(k)} = \sigma\left(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} h_v^{(k-1)} + b^{(k)}\right) \quad (1)$$

In Eq. 1 [16], for a given message-passing layer k, $W_{\text{self}}^{(k)}, W_{\text{neigh}}^{(k)} \in \mathbb{R}^{d(k) \times d(k-1)}$ are trainable parameter matrices, $b^{(k)}$ is the bias term and σ is a non-linearity function such as ReLU. After K iterations, the output of the final layer is used to define the node embeddings. Following this, a readout layer aggregates over all the final node embeddings to generate a graph embedding. With this graph embedding, which is a fixed-sized feature vector, a multi-layer perceptron can be used to regress for the target metric, i.e. latency in our case. The final readout layer aggregation can be done with simple functions

like sum, max and mean or using adaptive methods [17]–[19] that aim to learn complex patterns within the graph.

IV. Problem Definition

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represent an arbitrary DNN architecture as a graph. Let $\mathcal{P}$ denote the target hardware platform. The objective is to learn a function $f_\theta : \mathcal{G} \times \mathcal{P} \rightarrow \mathbb{R}$ that accurately estimates the latency L of executing the DNN architecture $\mathcal{G}$ on the platform $\mathcal{P}$, i.e.,

$$f_\theta(\mathcal{G}, \mathcal{P}) \approx L(\mathcal{G}, \mathcal{P})$$

Latency estimation for DNN architectures on hardware platforms is typically formulated as a regression problem, which has been addressed through both platform-specific and platform-agnostic SoTA solutions. For platform-specific solutions, the major challenge has been the need for massive measurement collection on each platform for which a latency estimator is to be built. For the latter, although transfer learning is employed for newer similar hardware, there still exists the one-time cost of measurement collection and training of the latency predictor using specific hardware descriptors. In contrast, we argue that a measurement-efficient solution can be explored to tackle the challenge directly by investigating the architecture of the learned function $f_\theta : \mathcal{G} \times \mathcal{P} \rightarrow \mathbb{R}$ that estimates the latency L of executing a DNN architecture $\mathcal{G}$ on a target hardware platform $\mathcal{P}$. The function f_θ should be: (1) ***Measurement-efficient***, i.e., it should require minimal latency measurements on the target platform to achieve accurate estimation. (2) ***Platform-specific***, i.e., it should capture the target hardware characteristics through measurements, eliminating the need for individual hardware embeddings or descriptors that add to the complexity and can be detrimental when poorly designed. (3) ***General***, i.e., it should model arbitrary DNNs of variable depths and branches, not limited to predefined NAS search spaces and architectures. The learned function f_θ should thereby enable fast evaluation of diverse workloads in hardware-aware NAS or DNN compression flows, where the bottleneck of hardware-in-the-loop (HIL) measurements is removed. With growing differences in the hardware design of domain-specific heterogenous inference engines, compilers and their data flows, quantization supports and CPU clock frequencies, the need for fast, simple and easily trainable, dedicated latency estimators is well-motivated.

V. Architecture

In this chapter, we present the architecture of HiFi-SAGE, designed for efficient training and consistent performance across diverse hardware. We describe the components of HiFi-SAGE from input preprocessing to the GNN backbone and deep predictor layers, along with their training parameters and evaluation metrics. The final design is illustrated in Fig. 1.

A. Input

HiFi-SAGE, based on the PyTorch Geometric library [20], accepts DNN models as input, either in the ONNX format or converted to ONNX representation. The ONNX model is transformed into a networkx graph [21], where nodes represent operations and edges capture data dependencies. The input graph undergoes preprocessing, such as, (1) Graph attributes are standardized with uniform node attributes, (2) Operation names are one-hot encoded, and weight shapes are reshaped for uniformity, (3) Node attributes are scaled for numerical stability and convergence, (4) Self-loops are added to input nodes for information propagation, and (5) Output nodes are removed to simplify the graph structure. To capture structural information, HiFi-SAGE employs random walk-based positional encoding [22]. Random walks originating from each node record self-loop values, incorporated as node attributes to understand structural relationships and potentially improve latency prediction. The preprocessed graph is converted into a PyTorch Geometric data structure, with the node feature matrix, edge index matrix, and batch data passed as inputs to the GNN backbone.

B. GNN Backbone

Our model employs the GraphSAGE convolutional architecture, known for handling large and dynamic graph structures [23]. The backbone comprises seven stacked GraphSAGE layers with residual connections for improved gradient flow and training stability. We leverage LCM aggregation [17] to capture the entire graph's structure by combining information from various nodes. The inputs are:

- Node feature matrix $\mathbf{x}$ of shape $[N, 59]$
- Edge index edge_idx of shape $[2, E]$
- Batch vector of shape $[N]$

Here, N and E denote the total nodes and edges across all batched graphs, respectively. The batch vector maps nodes to their respective graphs. The GraphSAGE layers transform inputs into 128-dimensional embeddings, maintained across layers. The final aggregation yields graph-level embeddings of shape $[B, 128]$, where B is the batch size. Dropout (rate=0.06) is applied after each GraphSAGE layer to mitigate overfitting.

C. Deep Predictor Layers

HiFi-SAGE employs two distinct Multi-Layer Perceptrons (MLPs): one for individual latency estimation and another for pairwise ranking. The individual latency estimation MLP is a four-layer network that regresses the generated graph embeddings to predict their respective latencies. To enhance the model's ability to learn from extensive pairwise interactions, we introduce a pairwise ranking mode. In this mode, all possible pairs of graph embeddings are considered. For each valid pair (i, j), the embeddings are concatenated to form a single combined embedding. These combined embeddings are passed through a series of two feed-forward linear layers of the pairwise MLP. Activation functions and dropout are applied between layers to introduce non-linearity and prevent overfitting. The final layer outputs the pairwise prediction scores, which are then processed through a sigmoid function as seen in Fig.1.

D. Training Parameters and Evaluation Metrics

For all the experiments of HiFi-SAGE, we use the training parameters as described in Table II. We consistently saw improved prediction performance at lower batch sizes and stable

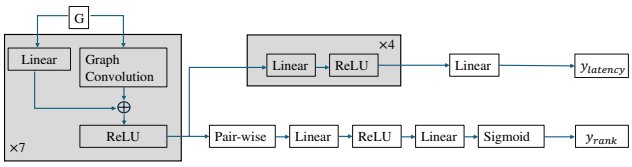


Fig. 1: HiFi-SAGE design, with a stack of 7 GraphSage layers producing the graph embeddings, that is then passed to the latency estimator head and a ranking predictor head.

training runs with the Adam optimizer. The learning rate was lowered by $\times 0.5$ if the validation error plateaued for 10 epochs. The maximum training epoch was 300 with early stopping if the validation error did not decrease for 50 epochs. Furthermore, we employ label scaling in powers of 10 to ensure stability in training and to maintain similar units for latency prediction in nanoseconds.

Parameter	Description
Learning Rate	0.001
Learning Rate Schedule	ReduceLROnPlateau
Batch Size	8
Optimizer	Adam
Training Epochs	300 (early stopping patience of 50 epochs)
Dropout Rate GNN	0.06
Dropout Rate Pairwise	0.01

TABLE II: Hyperparameters used for training HiFi-SAGE.

Following the works of [10], [11], we consider the following evaluation metrics: ***Mean Absolute Percentage Error (MAPE)*** measures the average absolute deviation between the predicted and actual values, expressed as a percentage of the actual values, providing a scale-invariant metric for intuitive understanding of prediction accuracy; ***Error Bounds*** quantify the proportion of predictions within specified error thresholds (e.g., 10%, 5%, or 1%) of the actual values, assessing the closeness of the predicted distribution to the original measurement distribution; and ***Spearman's Rank Correlation Coefficient or the fidelity score***, evaluates the monotonic relationship between the predicted and actual rankings of the latencies, measuring the degree of agreement between the two rankings, with a value of 1.0 indicating a perfect rank correlation.

VI. Architecture Ablation

This section presents the most impactful design choices that lowered the mean MAPE and improved the mean fidelity score (ρ) of HiFi-SAGE across the diverse devices in LatBench. For our architecture exploration, we draw inspiration from two GCN baselines, *(1) BRP-NAS* [10] and *(2) DGL Lifescience* [24]. The former is a popular GCN-based latency estimator and the only comparable work for measurement-efficient training from scratch of latency estimators, while the latter works with heterogeneous graph structures of varying sizes. We start off with 4 GCN layers which a common choice in SoTA latency estimators [10], [11]. We re-implement a modified version of BRP-NAS using Pytorch Geometric [25] and with the flexibility of encoding individual operators of the input DNN as opposed to the cell-based encoding in the original paper. The design starts with a GCN baseline with a depth of 4, a global mean pooling layer to aggregate information from all nodes and one dense layer for predicting the latency. The design space is explored across the macro level of the GNN architecture itself making important choices with (1) Convolution Operator (2) Aggregation Function (3) Depth and Width Scaling and (4) Pairwise Loss Function.

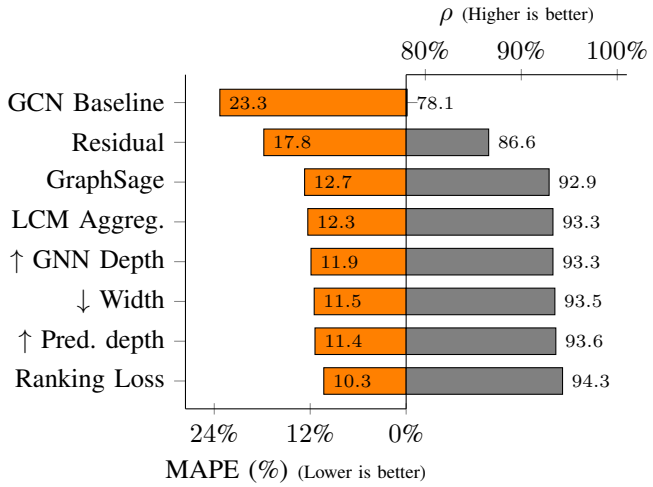


Fig. 2: Ablation Study of design decisions. Metrics reported are a mean of all 6 devices in LatBench, averaged over 10 random seeds.

A. Dataset

To prove competitive performance and reproducibility, we leverage the open source latency benchmark on NASBench-201, *LatBench*, released by *BRP-NAS* [10]. The dataset consists of latency measurements on six different devices of varying scales namely, *Intel Core i7-7820X (D.CPU)*, *NVIDIA GTX 1080 Ti (D.GPU)*, *NVIDIA Jetson Nano (E.GPU)*, *Google EdgeTPU (E.TPU)*, *Qualcomm Adreno 612 GPU (M.GPU)*, and *Qualcomm Hexagon 690 DSP (M.DSP)*. The dataset helps to validate HiFi-SAGE's ability to effectively model diverse hardware that are not strongly correlated to each other as presented in [10] without explicit hardware embeddings. *LatBench* has 15,625 latency measurements for the 32×32 input size images of which we randomly sample 100 measurements for training, 100 for validation and the remaining for test.

B. GNN block Design

We investigate the fundamental block design responsible for message passing. To a baseline closely resembling the BRP-NAS backbone, the first addition is a dense residual layer to every GCN layer. This significantly impacted both MAPE and ρ as presented in Fig. 2, with a 5.5 percentage points (p.p.) reduction and 8.5 p.p. improvement respectively. As for the convolution operation of the block, a simple switch from GCN to GraphSage [23] known for learning large and dynamic graphs well, caused an increase in the fidelity score by 6.3 p.p. and decreased the mean error by 5.1 p.p. It is important to note that the mean metrics encapsulate the improvements across all the 6 platform-specific latency estimators trained for the 6 devices in LatBench dataset. The design changes reflected differently for each target providing reassurance for the need

of platform-specific estimators. *GraphSAGE layer with a dense residual connection is the core repeating block of HiFi-SAGE.*

C. Aggregation Function

The aggregation or readout function influences the final graph embedding that is regressed upon for latency estimation. BRP-NAS uses a global node to aggregate information from all nodes that can get computationally expensive and inefficient for larger graphs and has a fixed aggregation mechanism potentially diluting node-level information when mixing it with data from distant nodes. This is replaced with a learnable cluster-based readout function, LCM aggregation [17] that scales well with larger graphs by employing an adaptive aggregation mechanism learning the useful patterns. This change brought HiFi-SAGE an additional 0.4 p.p. gain in ρ and a 0.4 p.p. reduction in MAPE. *HiFi-SAGE employs LCM aggregation for readout.*

D. Depth and Width Scaling

A wide GNN layer allows the model to capture more complex and diverse features from the input graph, enabling it to learn more expressive representations while a deep GNN backbone allows the model to capture information from a larger neighborhood, enabling it to learn more global and contextual features from the graph. For a cell-based NAS search space in *LatBench*, there exists a saturation point for deep GNNs, as the deeper they get, they tend to remove the node-specific information with excessive neighbourhood data accumulation. BRP-NAS employed 4 GCN layers with 600 hidden units per layer and finally a dense layer for predictions. The performance variations upon increasing HiFI-SAGE's backbone depth to 7, predictor depth to 4 and decreasing the backbone and predictor hidden features to 128 and 256 repectively are shown in Fig. 2. The cumulative effect has been a 0.3 p.p. improvement in ρ and 0.9 p.p. reduction in MAPE. *HiFi-SAGE has a larger depth and lower width compared to SoTA.*

E. Pairwise Loss Function

Inspired by [10] that introduced a binary relation predictor for ranking the performance of models which indirectly improved sample efficiency, we evaluate an additional prediction head that computes pair-wise latency ranking of all possible pairs in a batch. An additional binary cross entropy loss is applied on the predicted and actual latency ranks and is combined with the absolute prediction loss, i.e., Huber loss [26]. We scale the pairwise ranking loss term with a weighting factor of 1.5, that helped in further lowering the MAPE by 1.1 p.p. and improving the fidelity score by 0.7 p.p.
The pairwise ranking loss component in HiFi-SAGE lowered the MAPE especially in measurement-efficient settings, as it regularizes the training by augmenting all combinations of pairs in a given batch of samples.

F. HiFi-SAGE: Final Design

As shown in Fig. 1, the deep backbone and the predictor layers end with two prediction heads, one to estimate the latency and the other to compute the pairwise ranking, encouraging the model to account for the right ordering of the estimated latencies. This improved design of an end-to-end latency estimator that can be trained on any arbitrary DNN search space without actual knowledge of the hardware with 100 measurements alone, is evaluated for its prediction accuracy and fidelity in the following section.

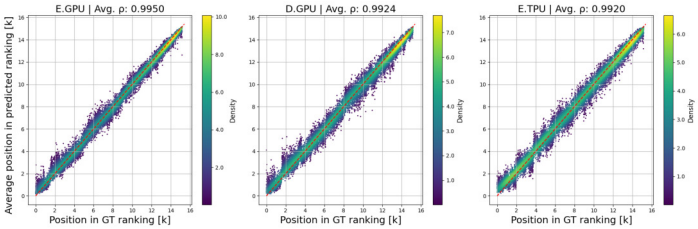


Fig. 3: Density of correct latency rankings by HiFi-SAGE trained with 100 samples on 10 random seeds. Mean ranking of each sample in LatBench dataset, compared to the ground truth ranking. Units of the axes are in k, i.e thousands.

VII. Evaluation

We first prove HiFi-SAGE's competence in measurement-efficiency on the *LatBench* dataset using the training parameters in Table II. BRP-NAS [10] is a popular work that is platform-specific and employs sample efficient training with 900 random latency measurements. To challenge this, we train on 100 randomly sampled measurements from LatBench. As seen in Table III, we report the Error bounds metric averaged over 100 different training runs with 100 random samples each.

Error bounds	Accuracy [%]					
	D. CPU	**D. GPU**	**E. GPU**	**E. TPU**	**M. GPU**	**M. DSP**
BRP-NAS *(100 pts.)*						
±1%	6.1±1.7	5.9±1.3	9.9±1.3	6.2±1.0	5.2±0.9	10.3±1.1
±5%	27.9±5.5	28.7±3.6	44.6±4.0	30.0±3.6	24.9±3.4	48.0±3.8
±10%	51.5±8.3	52.9±5.0	71.8±3.5	54.6±5.7	46.3±4.1	78.4±3.6
Ours *(100 pts.)*						
±1%	**8.3**±0.8	**8.3**±0.8	**10.4**±1.0	**9.5**±0.6	**8.4**±1.7	**12.5**±1.0
±5%	**38.4**±3.2	**38.6**±2.9	**46.8**±3.2	**43.7**±2.0	**38.4**±6.4	**54.5**±3.3
±10%	**65.0**±4.0	**65.0**±3.5	**73.2**±2.9	**72.0**±2.1	**62.2**±7.5	**80.4**±2.6

TABLE III: Performance comparison of latency prediction performance with BRP-NAS, whose numbers are taken from the paper [10]. The values of HiFi-SAGE are a mean over 100 training runs.

Clearly, HiFi-SAGE surpasses BRP-NAS in all the datasets. The highest improvements in the 10% Error bounds metric are 17.4 percentage points (p.p.), 15.9 p.p. and 13.5 p.p. in the E.TPU, M.GPU and D.CPU datasets respectively. Interestingly, these three devices have uncorrelated latency measurements, with 0.54 and 0.84 correlation factors for M.GPU and D.CPU respectively compared to E.TPU [10]. Despite this, HiFi-SAGE showed top improvements in prediction accuracy with these uncorrelated devices, suggesting that the design choices made for HiFi-SAGE are target-agnostic and do not need to be re-designed for different uncorrelated hardware.
We proceed to evaluate the fidelity scores (ρ) of HiFi-SAGE across *LatBench*. With all estimators for individual hardware trained 100 times on 100 random seeds with 100 latency measurements, we first sort all the latency predictions for each device to produce a ranking that is then compared to the ground

truth ranking. As shown in Fig. 3, the average prediction ranks are plotted against the ground truth ranks. The average ρ of HiFi-SAGE is 0.989 on LatBench across the six devices. A direct comparison of BRP-NAS, HELP and HiFi-SAGE is presented in Tab. IV where our work clearly outperforms even the meta-trained HELP. Extending the evaluation to another dataset, HW-NAS-Bench containing NASBench-201 architectures on other hardware targets, we present in Tab. IV that HiFi-SAGE once again outperforms the SoTA latency estimators including the meta-trained ones. The mean rho of all the six devices in HW-NAS-Bench is 0.94 with HiFi-SAGE as opposed to 0.93 with meta-trained HELP and Multipredict and 0.80 with BRP-NAS.

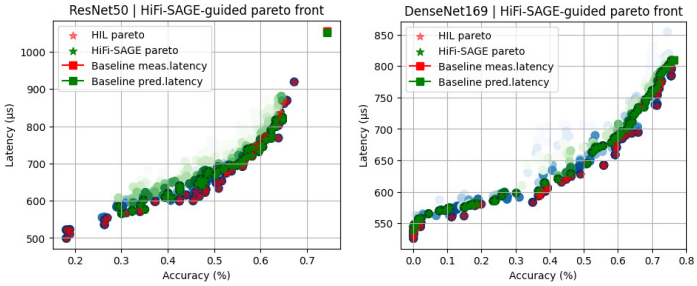
Method	Samples	HWNASBench-201			LatBench		
		Raspi4	ASIC	FPGA	D.CPU	M.DSP	M.GPU
BRP-NAS	900	0.85	0.81	0.80	0.991	0.959	0.961
HELP (Meta-test)	10	0.89	0.94	0.99	0.990	0.958	0.956
MultiPredict (Meta-test)	10	0.91	0.95	0.97	-	-	-
HiFi-SAGE (Ours)	100	**0.94**	**0.97**	**0.99**	**0.992**	**0.984**	**0.979**

TABLE IV: Performance comparison of HiFi-SAGE against SoTA for Spearman correlation (ρ) on HW-NAS-Bench-201 and LatBench datasets. HELP is meta-trained on 900 measurements from 18 different targets. The numbers for MultiPredict are taken from [12].

VIII. Application: DNN Compression Search

We use a genetic-algorithm (GA)-based [27] channel pruning algorithm [28] to explore differently pruned models for a mobile platform [29]. The GA search is initialized with a population size p, corresponding to a set of DNNs with randomly selected layer-wise pruning ratios. Then iteratively, based on the fitness criteria (latency, task accuracy), a sub-population of *fit* survivors are selected. To this set, crossover and mutation are carried out. The former indicates swapping parts of the parents' compression ratios and the latter means randomly altering a few compression ratios, both generating a new population for the next round/generation. For example, a ResNet50 compression search is carried out in 50 generations with 50 pruned models being explored per generation. This presents an opportunity to carry out HIL measurements during the first few generations so as to build HiFi-SAGE, that further replaces the HIL feedback for subsequent generations.

*a) **HIL Setup**:* We carry out the compression search experiments on a Qualcomm Snapdragon 8 Gen 2 mobile development kit [30], using the Qualcomm AI Direct SDK v2.19 [31] for compilation of the DNNs. For deriving the pruned models, we employ the AIMET tool [28] to generate channel-pruned models and the NSGA-II based Genetic-algorithm [27] to explore various pruning configurations. For training HiFi-SAGE, we collect 150 HIL latency measurements obtained during the first three generations of the GA compression search. To compare HIL-guided and HiFi-SAGE-guided search, we set the same random seed and generate comparable Pareto fronts. For the evaluations, we collect GA-compression datasets for ResNet50, DenseNet169 and InceptionV3. All the experiments were carried out for 10 generations with a population size of 50 per generation. We train HiFi-SAGE on the 100 latency measurements obtained and use the remaining 50 measurements



(a) ResNet50 (b) DenseNet169

Fig. 4: HiFi-SAGE-guided Pareto on top of a HIL-guided Pareto.

for validation. Once trained, it is restored in the GA-search to obtain latency values for the fitness metric. The biggest advantage of such a software estimator is that it is highly parallelizable and can significantly reduce the search time for network compression. Our HIL setup took 1 minute per compressed model for compilation, transfer to and from target, and sequential inference on target, while HiFi-SAGE computed 16 latencies in parallel in 1 second.

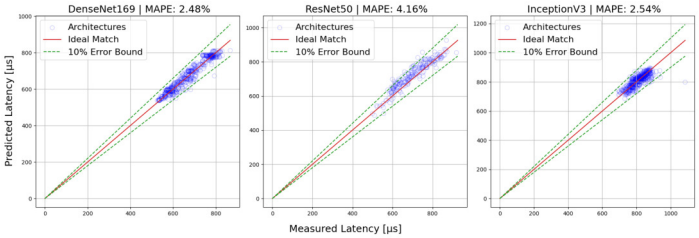


Fig. 5: Measured vs. HiFi-SAGE predicted latencies with MAPE.

*b) **HIL vs HiFi-SAGE**:* As seen in Fig. 5, HiFi-SAGE estimations closely matches the HIL latency measurements with a MAPE of 4.16%, 2.54% and 2.48% of ResNet50, InceptionV3 and DenseNet169 respectively. With Fig.4, we further present the Pareto fronts of ResNet50 and DenseNet169 pruning search to demonstrate that HiFi-SAGE can indeed replace the HIL setup effectively.

IX. Conclusion

This paper introduced HiFi-SAGE, a high-fidelity GraphSAGE-based latency estimator. By leveraging a novel two-headed network design, HiFi-SAGE surpasses existing methods in prediction fidelity and measurement efficiency for training from scratch on both LatBench and HW-NAS-Bench datasets across 12 diverse targets. Moreover, we demonstrated a genetic algorithm-based compression search, with HiFi-SAGE achieving mean absolute percentage errors (MAPE) of only 2.54%, 2.75%, and 4.16% for InceptionV3, DenseNet169, and ResNet50, respectively, thereby providing an effective alternative to the expensive HIL setup. HiFi-SAGE comes with a consistent design and can be easily trained without the necessity for extensive hardware measurements which is desirable in an ever-changing landscape of hardware and software with respect to edge inference engines.

REFERENCES

[1] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," in *International Conference on Learning Representations (ICLR)*, 2020.

[2] Y. Lin, D. Hafdi, K. Wang, Z. Liu, and S. Han, "Neural-hardware architecture search," *NeurIPS WS*, 2019.

[3] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 784–800, 2018.

[4] I. Amazon Web Services, "Amazon ec2 dl2q instances," may 2024.

[5] Nvidia Corporation, "Nvidia tensorrt sdk," 2024. [Online; accessed April 29, 2024].

[6] O. R. developers, "Onnx runtime." https://onnxruntime.ai/, 2021. Version: x.y.z.

[7] C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He, and Y. Li, "A survey of graph neural networks for recommender systems: Challenges, methods, and directions," *ACM Trans. Recomm. Syst.*, vol. 1, mar 2023.

[8] T. Sterling and J. J. Irwin, "Zinc 15 – ligand discovery for everyone," *Journal of Chemical Information and Modeling*, vol. 55, no. 11, pp. 2324–2337, 2015. PMID: 26479676.

[9] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "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*, MobiSys '21, (New York, NY, USA), p. 81–93, Association for Computing Machinery, 2021.

[10] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, "Brp-nas: Prediction-based nas using gcns," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 10480–10490, Curran Associates, Inc., 2020.

[11] H. Lee, S. Lee, S. Chong, and S. J. Hwang, "Hardware-adaptive efficient latency prediction for NAS via meta-learning," in *Advances in Neural Information Processing Systems* (A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), 2021.

[12] Y. Akhauri and M. S. Abdelfattah, "Multi-predict: few shot predictors for efficient neural architecture search," *arXiv preprint arXiv:2306.02459*, 2023.

[13] B. Wu, K. Keutzer, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, and Y. Jia, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, June 2019.

[14] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, and Y. C. Lin, "{HW}-{nas}-bench: Hardware-aware neural architecture search benchmark," in *International Conference on Learning Representations*, 2021.

[15] S. Nair, S. Abbasi, A. Wong, and M. J. Shafiee, "Maple-edge: A runtime latency predictor for edge devices," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, IEEE, June 2022.

[16] L. Wu, P. Cui, J. Pei, and L. Zhao, *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Singapore, 2022.

[17] E. Ong and P. Veličković, "Learnable commutative monoids for graph neural networks," in *The First Learning on Graphs Conference*, 2022.

[18] D. Buterez, J. P. Janet, S. J. Kiddle, D. Oglic, and P. Liò, "Graph neural networks with adaptive readouts," in *Advances in Neural Information Processing Systems* (A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, eds.), 2022.

[19] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[20] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[21] A. Hagberg, P. J. Swart, and D. A. Schult, "Exploring network structure, dynamics, and function using networkx," 1 2008.

[22] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Graph neural networks with learnable structural and positional representations," *arXiv preprint arXiv:2110.07875*, 2021.

[23] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[24] M. Li, J. Zhou, J. Hu, W. Fan, Y. Zhang, Y. Gu, and G. Karypis, "Dgllifesci: An open-source toolkit for deep learning on graphs in life science," *ACS Omega*, 2021.

[25] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," May 2019.

[26] P. J. Huber, "Robust estimation of a location parameter," *The Annals of Mathematical Statistics*, vol. 35, p. 73–101, Mar. 1964.

[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[28] I. Qualcomm Technologies, "Ai model efficiency toolkit." https://quic.github.io/aimet-pages/AimetDocs/user_guide/channel_pruning.html, 2024.

[29] Qualcomm Technologies, Inc., "Snapdragon® 8 gen 2 mobile hardware development kit," 2024. [Online; accessed April 29, 2024].

[30] I. Qualcomm Technologies, "Snapdragon® 8 gen 2 mobile hardware development kit." https://developer.qualcomm.com/hardware/snapdragon-8-gen-2-hdk, 2024.

[31] Qualcomm Technologies, Inc., "Qualcomm ai engine direct sdk," 2024. [Online; accessed April 29, 2024].