wa-hls4ml: A Benchmark and Surrogate Models for hls4ml Resource and Latency Estimation

BENJAMIN HAWKS, Fermi National Accelerator Laboratory, USA
JASON WEITZ, University of California San Diego, USA
DMITRI DEMLER, University of California San Diego, USA
KARLA TAME-NARVAEZ, Fermi National Accelerator Laboratory, USA
DENNIS PLOTNIKOV, Johns Hopkins University, USA
MOHAMMAD MEHDI RAHIMIFAR, University of Sherbrooke, Canada
HAMZA EZZAOUI RAHALI, University of Sherbrooke, Canada
AUDREY C. THERRIEN, University of Sherbrooke, Canada
DONOVAN SPROULE, Columbia University, USA
ELHAM E KHODA, University of California San Diego, USA
KEEGAN A. SMITH, Texas A&M University, USA
RUSSELL MARROQUIN, University of California San Diego, USA
GIUSEPPE DI GUGLIELMO, Fermi National Accelerator Laboratory, USA
NHAN TRAN, Fermi National Accelerator Laboratory, USA
JAVIER DUARTE, University of California San Diego, USA
VLADIMIR LONCAR, European Organization for Nuclear Research (CERN), Switzerland

As machine learning (ML) is increasingly implemented in hardware to address real-time challenges in scientific applications, the development of advanced toolchains has significantly reduced the time required to iterate on various designs. These advancements have solved major obstacles, but also exposed new challenges. For example, processes that were not previously considered bottlenecks, such as hardware synthesis, are becoming limiting factors in the rapid iteration of designs. To mitigate these emerging constraints, multiple efforts have been undertaken to develop an ML-based surrogate model that estimates resource usage of ML accelerator architectures.

FERMILAB-PUB-25-0359-CSAID.
Authors' Contact Information: Benjamin Hawks, Fermi National Accelerator Laboratory, Batavia, IL, USA, bhawks@fnal.gov; Jason Weitz, University of California San Diego, La Jolla, CA, USA, jdweitz@ucsd.edu; Dmitri Demler, University of California San Diego, La Jolla, CA, USA, ddemler@ucsd.edu; Karla Tame-Narvaez, Fermi National Accelerator Laboratory, Batavia, IL, USA, karla@fnal.gov; Dennis Plotnikov, Johns Hopkins University, Baltimore, MD, USA, dennis.yuri.plotnikov@cern.ch; Mohammad Mehdi Rahimifar, University of Sherbrooke, Sherbrooke, Quebec, Canada, rahm2701@usherbrooke.ca; Hamza Ezzaoui Rahali, University of Sherbrooke, Sherbrooke, Quebec, Canada, hamza.rahali@usherbrooke.ca; Audrey C. Therrien, University of Sherbrooke, Sherbrooke, Quebec, Canada, audrey.corbeil.therrien@usherbrooke.ca; Donovan Sproule, Columbia University, New York, NY, USA, donovan.sproule@gmail.com; Elham E Khoda, University of California San Diego, La Jolla, CA, USA, ekhoda@ucsd.edu; Keegan A. Smith, Texas A&M University, College Station, TX, USA, keeganasmith2003@tamu.edu; Russell Marroquin, University of California San Diego, La Jolla, CA, USA, rdmarroquinsolares@ucsd.edu; Giuseppe Di Guglielmo, Fermi National Accelerator Laboratory, Batavia, IL, USA, gdg@fnal.gov; Nhan Tran, Fermi National Accelerator Laboratory, Batavia, IL, USA, ntran@fnal.gov; Javier Duarte, University of California San Diego, La Jolla, CA, USA, jduarte@ucsd.edu; Vladimir Loncar, European Organization for Nuclear Research (CERN), Geneva, Switzerland, vladimir.loncar@cern.ch.

Manuscript submitted to ACM

We introduce wa-hls4ml, a benchmark for ML accelerator resource and latency estimation, and its corresponding initial dataset of over 680 000 fully connected and convolutional neural networks, all synthesized using hls4ml and targeting Xilinx FPGAs. The benchmark evaluates the performance of resource and latency predictors against several common ML model architectures, primarily originating from scientific domains, as exemplar models, and the average performance across a subset of the dataset. Additionally, we introduce GNN- and transformer-based surrogate models that predict latency and resources for ML accelerators. We present the architecture and performance of the models and find that the models generally predict latency and resources for the 75% percentile within several percent of the synthesized resources on the synthetic test dataset.

CCS Concepts: • **Hardware → Hardware-software codesign**; **Hardware accelerators**; • **Computing methodologies → Supervised learning by regression**; **Neural networks**.

Additional Key Words and Phrases: surrogate model, FPGA, hls4ml, resource, latency, regression, machine learning, artificial intelligence, High-level synthesis, benchmark, edge computing, graph neural network, open source

ACM Reference Format:

Benjamin Hawks, Jason Weitz, Dmitri Demler, Karla Tame-Narvaez, Dennis Plotnikov, Mohammad Mehdi Rahimifar, Hamza Ezzaoui Rahali, Audrey C. Therrien, Donovan Sproule, Elham E Khoda, Keegan A. Smith, Russell Marroquin, Giuseppe Di Guglielmo, Nhan Tran, Javier Duarte, and Vladimir Loncar. 2025. wa-hls4ml: A Benchmark and Surrogate Models for hls4ml Resource and Latency Estimation. 1, 1 (November 2025), 30 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Domain-specific design tools leveraging high-level synthesis (HLS) have become indispensable in the design of hardware accelerators, enabling the automatic translation of high-level programming languages like C++ or Python into hardware descriptions. This shift allows developers to focus on algorithmic codesign rather than the complexities of hardware implementation, reducing both development time and the expertise required. However, one of the major challenges in using HLS tools is predicting resource utilization—such as logic elements, memory, and interconnects—during the codesign process. This requires costly and time-extensive hardware synthesis steps—both at the C and logic synthesis steps. C synthesis is required for timing estimates, its resource estimation is typically inaccurate. While logic synthesis produces accurate resource estimates, it takes a considerable amount of time.

Specialized domains like machine learning (ML), where hardware efficiency is paramount, magnify this challenge, as synthesis takes significant time especially for complex ML models. One proposed solution to greatly accelerate the resource estimation step is a neural network surrogate model predictor, which can take the resource prediction step from hours to seconds. A surrogate model is a model built to approximate a larger, more complex system. In this case it is a neural network designed to approximate the resource estimation.

Developing generalized and high-performance surrogate models is challenging, both in terms of dataset generation and model design. To address this at a more focused, initial scale, we consider dataflow FPGA architectures for embedded AI applications such as the internet of things (IoT), autonomous vehicles, and scientific sensing. Within this scope, we aim to develop a surrogate model using the hls4ml flow [15]. This allows designers to make better-informed decisions early in the development cycle, on the order of seconds from model specification, reducing the need for iterative synthesis runs and enabling more efficient hardware implementations. For hls4ml users, this is particularly beneficial, as it provides detailed feedback on the hardware requirements of neural network architectures, helping developers optimize their models for FPGA deployment.

hls4ml is an open-source framework that translates ML models into FPGA-based IP using HLS tools. It bridges the gap between ML and hardware by facilitating the development of low-latency, resource-efficient inference engines on

FPGAs. Despite these advantages, optimizing resource usage remains a complex task, requiring accurate predictions of hardware demands to effectively balance performance, power, and area constraints.

To achieve this, we introduce **wa-hls4ml**[1]: a **dataset** unprecedented in scale and features, a **benchmark** for common evaluation, and two new **surrogate models** for flexible and precise resource and latency estimation targeting the prediction of resource usage and latency for HLS tools. The combination of these three main contributions enables novel rapid codesign research at a scale beyond previously possible, by reducing the codesign loop from hours to seconds as shown in Figure 1, and is a unique community resource. Furthermore, it enables users of hls4ml and other dataflow accelerators for edge ML applications to rapidly deploy optimal FPGA implementations.

The open **dataset** is unprecedented in terms of its size, with over 680 000 fully synthesized dataflow models. The goal is to continue to grow and extend the dataset over time. We include all steps of the synthesis chain from ML model to HLS representation to register-transfer level (RTL) and save the full logs. This will enable a much broader set of applications beyond those in this paper. The **benchmark** standardizes evaluation of the performance of resource usage and latency estimators across a suite of metrics, such as the coefficient of determination (R^2), symmetric mean absolute percentage error (SMAPE), and root mean square error (RMSE), and provides sample models, both synthetic and from scientific applications, to support and encourage the continued development of better surrogate models. The **surrogate model** architectures are based on a graph NN and transformer to enable flexibility. The models predict FPGA resources: lookup tables (LUTs), flip-flops (FFs), digital signal processors (DSPs), and on-chip block random access memory (BRAM), as well as latency (clock cycles) and initiation interval (II).

The following summarizes our design rationale when it comes to building and maintaining the **benchmark**:

- **Address the need for a standard evaluation suite**: Provide exemplar benchmark models with their synthesis results, artifacts, and log files, along with comprehensive and predefined evaluation metrics.
- **Formalize the structure for optimal utility**: The benchmark will be structured to maximize its utility for the broader research community, ensuring that it remains applicable beyond resource and latency estimation.
- **Promote an open design process**: Allow contributors to propose enhancements to the benchmark and submit new surrogate models, in alignment with the latest advancements in the ML field.

In this way, we plan for the dataset and procedures laid out in this work to be a community benchmark for future avenues of study.

1.1 Related Work

Previous HLS design datasets have focused on more generic lower-level kernels, such as general matrix multiplication. For example, DB4HLS [18] targets programs from the MachSuite benchmark [31], GNN-DSE [34] targets programs from PolyBench/C [28], while HLSyn [5] includes kernels from both. In contrast, the wa-hls4ml dataset is more domain-specific, though it also targets a large variety of multilayer neural network designs generated by hls4ml, and thus incorporates higher-level programs. Concurrently with this work, HLSFactory [1] is a framework to collect and build HLS design datasets, including ML designs generated by FlowGNN [33]. Finally, rule4ml [30] is a closely related prior work from which we expand the dataset, formalize a benchmark, and explore more complex surrogate model architectures. Other datasets and surrogate models exist [13], [14], [22] but are not open source or have relatively smaller dataset sizes. Vivado/Vitis HLS also provide native estimates [3], but only after running C-synthesis, which can be

[1]Named after Wario and Waluigi who are doppelgängers of Mario and Luigi, respectively, in the Nintendo Super Mario platform game series.

Table 1. Comparison of wa-hls4ml to prior work.

Tool	Open Source	Open Dataset	Dataset Size [samples]	Vendor tools required	Is Benchmark	Input Abstraction Level
Native HLS Estimate [3]	No	N/A	N/A	Yes	N/A	HLS Code
High-Level Synthesis Performance Prediction using GNNs [38]	**Yes**	**Yes**	40, 000	Yes	**Yes**	HLS/LLVM IR Graph
Machine Learning Aided Hardware Resource Estimation for FPGA DNN Implementation [14]	No	No	N/A	**No**	No	FINN Intermediate Representation
A Graph Neural Network Model for Fast and Accurate Quality of Result Estimation for High-Level Synthesis [22]	No	No	2, 465	Yes	No	HLS/LLVM IR Graph
HLSyn [5]	**Yes**	**Yes**	42, 000	**No**	**Yes**	HLS Code
Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning [13]	No	No	1, 300	Yes	No	HLS Reports
rule4ml [30] (Related Work)	**Yes**	**Yes**	15, 000	**No**	No	hls4ml IR
wa-hls4ml (This work)	**Yes**	**Yes**	683, 176	**No**	**Yes**	hls4ml IR

time-consuming. Table 1 summarizes directly comparable studies and tools, laying out the datasets, the representation used as input, the availability of the code, and whether the approach proposes a benchmark.

Since we target higher-level hls4ml programs, it is difficult to make direct comparisons to our method. Additionally, in many cases of related work, the code is not available for evaluation. The work by Wu et al. [38] has publicly available code and most closely aligns with our approach in terms of predicting FPGA resource utilization and timing from high-level descriptions using graph neural networks (GNNs). Their framework processes intermediate representation (IR) graphs extracted after front-end compilation of C/C++ programs, enabling resource usage and timing estimation without completing the entire HLS process.

While conceptually similar, our approaches differ significantly in several key ways. First, [38] targets general C/C++ applications synthesized with Vitis HLS, with their training dataset comprising programs generated by C code generator ldrgen [6], and applications from PolyBench/C, CHStone [21], and MachSuite [32]. Conversely, our work specifically focuses on neural networks implemented using hls4ml. Second, their model necessitates running the initial stages of C synthesis to extract IR graphs, whereas our GNN- and transformer-based surrogate models operate directly on the neural network architecture description. This eliminates synthesis steps, providing faster resource prediction.

Because the wa-hls4ml dataset includes the HLS LLVM IR graphs, we can evaluate the approach presented in [38] using their publicly available code. Since our evaluation dataset is outside of their training domain, the predictions are not directly interpretable, e.g., many of the predictions are negative. However, they still show a correlation to the ground truth. To quantitatively compare the approaches, we adapted their model to our task by applying linear correction factors to their predictions when evaluating neural network models for simple 2-layer MLPs. Even after this correction, their predictions are less precise compared to our GNN- and transformer-based surrogate models. Their model achieved SMAPE values of 34.30, 36.03, and 31.26% for DSPs, LUTs, and FFs, respectively. These results indicate that domain-specific approaches like ours deliver better accuracy for ML workloads compared to general-purpose HLS estimation frameworks while also bypassing synthesis, further reducing estimation time. An interesting future study

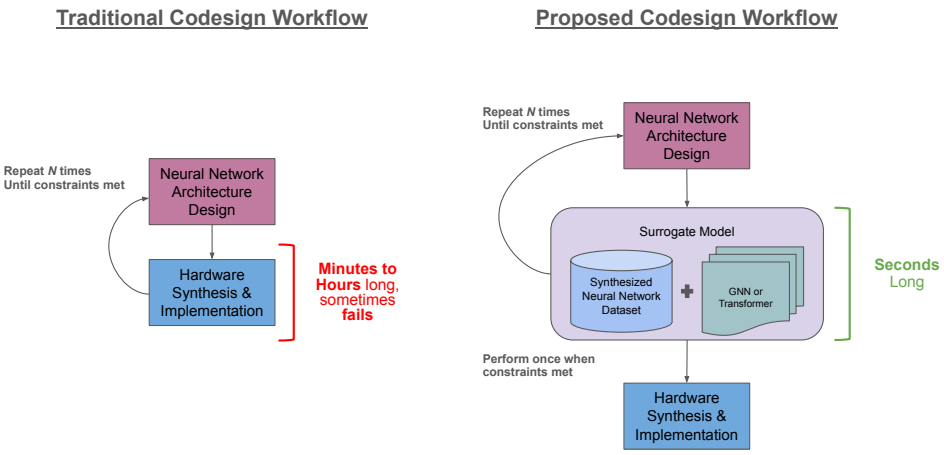

Fig. 1. The traditional codesign workflow compared to the proposed surrogate model based codesign workflow.

would be to train their model on the wa-hls4ml dataset and investigate whether their approach improves prediction accuracy.

2 Dataset

The dataset has two primary components, each designed to test different aspects of a surrogate model's performance. The first part is based on *synthetic* neural networks generated with various layer types, micro-architectures, and precisions. This synthetic dataset lets us systematically explore the FPGA resources and latencies as we vary different model parameters. The second part of the benchmark targets models from *exemplar realistic* scientific applications, requiring real-time processing at the edge, near the data sources. Models with real-time constraints constitute a primary use case for ML-to-FPGA pipelines like hls4ml. This part tests the ability of the surrogate model to extrapolate its predictions to new configurations and architectures beyond the training set, assessing the model's robustness and performance for real applications.

The training, validation, and test sets of the benchmark currently consist of 683 176 synthetic samples, consisting of data about synthesized samples of 608 679 fully-connected neural networks, 31 278 one-dimensional convolutional neural networks, and 43 219 two-dimensional convolutional neural networks. Each sample contains the model architecture, hls4ml [15] conversion parameters, the latency and resource usage numbers for that network post-logic synthesis, and associated metadata. In addition to the training, validation, and test sets, the dataset also includes 887 samples representing the successful logic synthesis of the exemplar models with varying hls4ml conversion parameters, as shown in subsubsection 2.1.1. The dataset as a whole is split, distributed, and intended to be used as follows:

- **Training set**: The set of 478 220 samples intended to be used for training a given estimator.
- **Validation set**: The set of 102 472 samples intended to be used for validation during training.
- **Test set**: The set of 102 484 samples intended to be used for testing and generating results for a given estimator.

- **Exemplar test set**: The set of 887 samples, comprising the models described in subsection 2.2, intended to be used for testing and generating results for a given estimator.

Within each subset, excluding the exemplar test set, the data is further grouped as follows. These categories explain the composition of our dataset but have no bearing on how a given estimator should be trained.

- **2_20**: The updated rule4ml dataset, containing fully-connected neural networks that were randomly generated with layer counts between 2 and 20 layers, using hls4ml resource and latency strategies.
- **2_layer**: a subset containing 2-layer deep fully-connected neural networks generated via a grid search using hls4ml resource and io_parallel strategies
- **3_layer**: a subset containing 3-layer deep fully-connected neural networks generated via a grid search using hls4ml resource and io_parallel strategies
- **conv1d**: A subset containing 3–7 layer deep 1-dimensional convolutional neural networks that were randomly generated and use hls4ml resource and io_stream strategies
- **conv2d**: A subset containing 3–7 layer deep 2-dimensional convolutional neural networks that were randomly generated and use hls4ml resource and io_stream strategies
- **latency**: a subset containing 3–7 layer deep fully-connected neural networks that were randomly generated and use hls4ml latency and io_parallel strategies
- **resource**: a subset containing 3–7 layer deep fully-connected neural networks that were randomly generated and use hls4ml resource and io_parallel strategies

2.1 Synthetic Dataset

With the introduction of ML into FPGA toolchains, e.g. for resource and latency prediction or code generation, there is a significant need for large datasets to support and train these tools. We found that existing datasets were insufficient for these needs, and therefore sought to build a dataset and a highly scalable data generation framework that is useful for a wide variety of research surrounding ML on FPGAs. This dataset serves as one of the few openly accessible, large-scale collections of synthesized neural networks available for ML research.

2.1.1 Generation. The train and test sets were created by first generating models of varying architectures in the Keras and QKeras [12] Python libraries, varying their hyperparameters. The updated rule4ml dataset follows the same generation method and hyperparameter ranges described in [30], while adding II information and logic synthesis results to the reports.

For the remaining subsets of the data, the two-layer and three-layer fully-connected models were generated using a grid search method according to the parameter ranges mentioned below, whereas larger fully-connected models and convolutional models (one- and two-dimensional) were randomly generated , where convolutional modes also contain dense, flatten, and pooling layers. The weight and bias precision was implemented in HLS as datatype `ap_fixed<`X`,1>`, where X is the specified precision and the total number of bits allocated to the weight and bias values, with one bit being reserved for the integer portion of the value. These models were then converted to HLS using hls4ml and synthesized through AMD Vitis version 2023.2 and 2024.2, targeting the AMD Xilinx Alveo U250 FPGA board [39]. The model sets have the following parameter ranges:

- *Number of layers*: 2-7 for fully-connected models; 3–7 for convolutional models
- *Activation functions*: linear for most 2-3 layer fully-connected models; ReLU, tanh, and sigmoid for all other fully connected models and convolutional models

- *Number of features/neurons*: 8–128 (step size: 8 for 2–3 layer) for fully-connected models; 32–128 for convolution models with 8–64 filters
- *Weight and bias bit precision*: 2–16 bits (step size: 2) for 2-3 layer fully-connected models, 4–16 bits (step size: powers of 2) for 3–7 layer fully-connected and convolutional models
- *hls4ml target reuse factor*: 1–4093 for fully-connected models ; 8192–32795 for convolutional models
- *hls4ml implementation strategy*: Resource strategy, which controls the degree of parallelism by explicitly specifying the number of MAC operations performed in parallel per clock cycle, is used for most fully-connected models and all convolutional models, while Latency strategy, where the computation is unrolled, is used for some 3–7 layer fully-connected models.
- *hls4ml I/O type*: The io_parallel setting, which directly wires the output of one layer to the input of the next layer, is used for all fully-connected models, and the io_stream setting, which uses FIFO buffers between layers, is used for all convolutional models.

The synthesis was repeated multiple times, varying the hls4ml *reuse factor*, a tunable setting that proportionally limits the number of multiplication operations used. The hls4ml conversion, HLS synthesis, and logic synthesis of the train and test sets were all performed in parallel on the National Research Platform Kubernetes Hypercluster and the Texas A&M ACES HPRC Cluster. On the National Research Platform, synthesis was run inside a container with a guest OS of Ubuntu 20.04.4 LTS, the containers being slightly modified versions of the xilinx-docker [25] v2023.2 “user" images, with 3 virtual CPU Cores and 16 GB of RAM per pod, with all AMD tools mounted through a Ceph [37]-based persistent volume. Jobs run on the Texas A&M ACES HPRC Cluster were run using Vitis 2024.2, each with 2 virtual CPU cores and 32 GB of RAM. The resulting projects, reports, logs, and a JSON file containing the resource/latency usage and estimates of the C and logic synthesis were collected for each sample in the dataset. The data pertaining to the resource utilization and latency, neural network architecture, and information relating to the conversion using hls4ml was then further processed into a collection of JSON files, distributed alongside this paper and described below. The full projects, which contain the generated code, logs, intermediate representations, and other related files, are also available for each sample in the primary dataset as a part of a companion dataset also released alongside this paper.

2.1.2 Dataset Analysis. We performed an analysis of the primary dataset to potentially identify trends and compare commonly used resource estimation metrics against actual resource utilization. Below are selected figures, Figure 2 and Figure 3, visualizing the dataset’s resource and latency values against bit operations (BOPs) [9] and the reuse factor of a given model.

We observe that within the dataset, the BOPs metric tends to approximate the resource and timing values of fully-connected models more closely than convolutional models. Within the convolutional models of the dataset, there is some positive correlation with BOPs, with the correlation being slightly better for timing estimates than resource information, but overall, the correlation is not as strong as it is for fully-connected models.

Additionally, we note that fully-connected models have distinct populations when grouped according to the reuse factor. These populations tend to be highly correlated for timing and resource information, with higher reuse factors exhibiting the expected behavior of larger latencies and lower resource usage in most cases. This trend is not as strong for convolutional models, where the reuse factor is less impactful than model size and complexity when implementing the network on an FPGA.

We also visualize the distribution of resource and latency features throughout the test dataset and exemplar dataset in Figure 4. We find that the distributions between the exemplar models and test dataset tend to not overlap strongly,

Table 2. The hyperparameters used in the synthesis of the exemplar benchmark models.

Hyperparameter	Values
Precision	`ap_fixed<2,1>`, `ap_fixed<8,3>`, `ap_fixed<16,6>`
Strategy	`Latency`, `Resource`
Target reuse factor	1, 128, 1024
Target board	Alveo U200, Alveo U250
Target clock	5 ns, 10 ns
Vivado version	2019.1, 2020.1

indicating that there is room to improve both the test dataset and exemplar dataset in terms of model architecture diversity, which is an area that we aim to improve upon in future works.

2.1.3 Dataset Structure. The distributed JSON files contain 683 176 total samples. The samples are split into three subsets, as described in subsection 2.1. The format across the three subsets is the same, where each sample is a single JSON file containing 9 fields:

- `meta_data`: a unique identifier, model name, and name of the corresponding gzipped tarball of the fully synthesized project, logs, and reports for the sample (contained in an accompanying dataset released alongside the primary dataset)
- `model_config`: a JSON representation of the Keras/QKeras [12] model synthesized in the sample, including the actual reuse factor as synthesized per layer.
- `hls_config`: the hls4ml configuration dictionary used to convert the model for the sample, including the target reuse factor as synthesized per layer
- `resource_report`: a report of the post-logic synthesis resources used for the sample, reported as the actual number of components used.
- `hls_resource_report`: a report of the post-hls synthesis resources estimated for the sample, reported as the actual number of components estimated.
- `latency_report`: a report of the post-hls synthesis latency estimates for the sample.
- `target_part`: the FPGA part targeted for HLS and logic synthesis for the sample.
- `vivado_version`: the version of Vivado used to synthesize the sample.
- `hls4ml_version`: the version of hls4ml used to convert the sample.

2.2 Exemplar Realistic Models

The exemplar models utilized in this study include several key architectures, each tailored for specific ML tasks and targeting scientific applications with low-latency constraints. The synthesis parameters for these models are presented in Table 2.

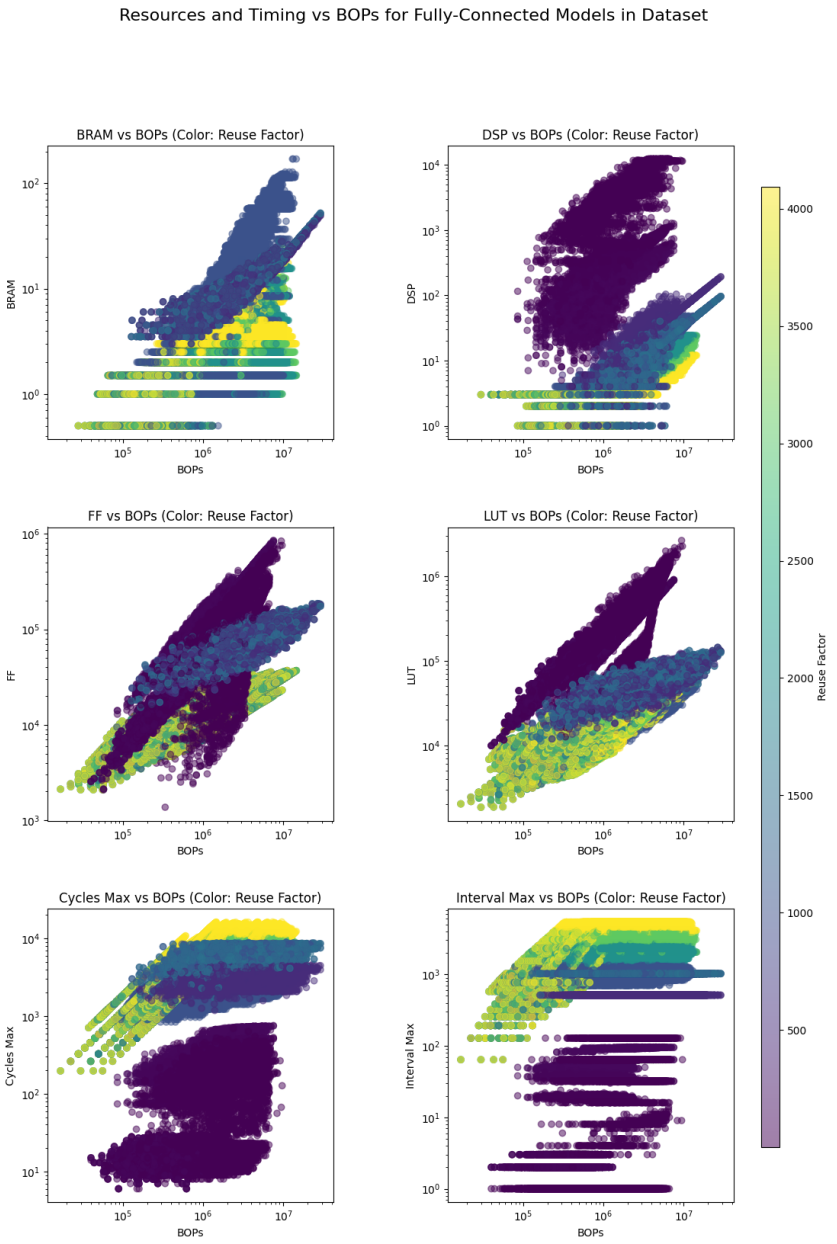

Fig. 2. All tracked output features plotted for each fully-connected model in the dataset versus Bit-Operations, with the color representing the reuse factor.

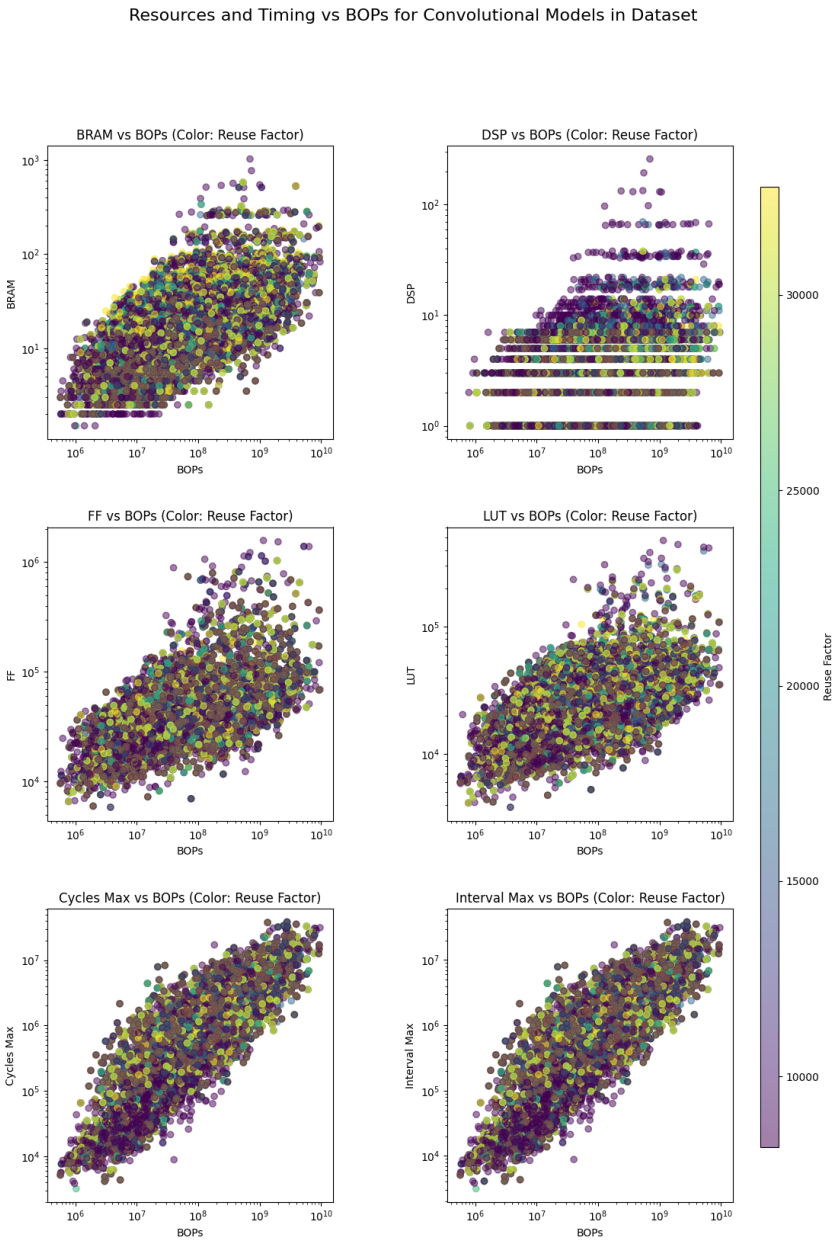

Fig. 3. All tracked output features plotted for each convolutional model in the dataset versus Bit-Operations, with the color representing the reuse factor of a given sample

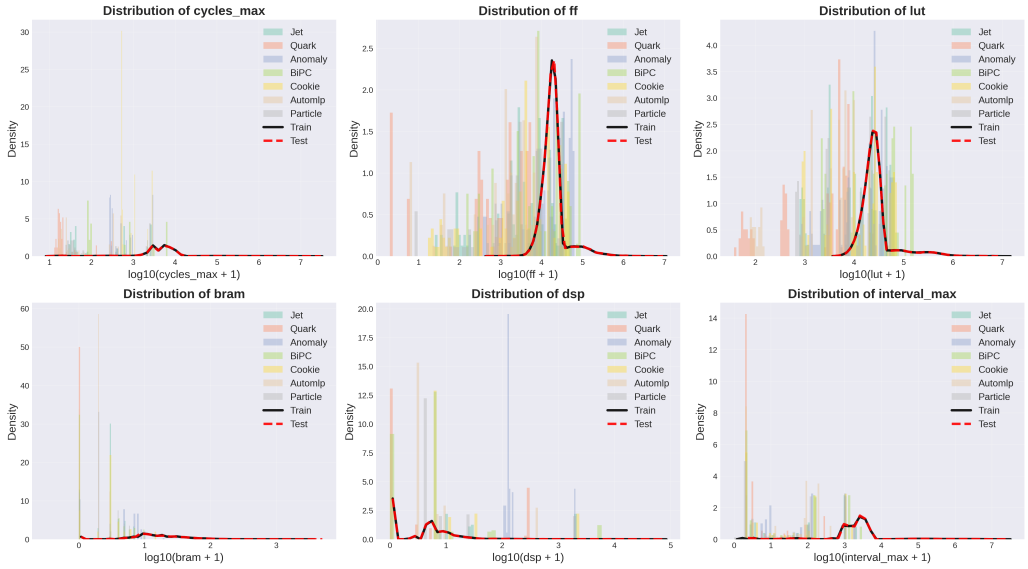

Fig. 4. Exemplar vs the train and test subset resource (label) distributions

The following gives a brief description of each of these models and their applications, while Table 3 presents their architectures. “Jet” [17] is a fully connected neural network that classifies simulated particle jets originating from one of five particle classes in high-energy physics experiments. “Quarks” [15] is a binary classifier for top quark jets. It helps probe fundamental particles and their interactions. “Anomaly” [7] is an autoencoder trained on audio data to reproduce the input spectrogram, whose loss value differentiates between normal and abnormal signals. “BiPC” [29] refers to an encoder that transforms high-resolution images, producing sparse codes for further compression. “Cookie” [20] is dedicated to real-time data acquisition for the CookieBox system, designed for advanced experimental setups requiring rapid handling of large data volumes generated by high-speed detectors. “AutoMLP” refers to a fully connected network from the AutoMLP framework [10], which focuses on accelerating MLPs on FPGAs, providing significant improvements in computational performance and energy efficiency. Lastly, “Particle Tracking” [2] tracks charged particles in real-time as they traverse silicon detectors in large-scale particle physics experiments.

3 Benchmark

3.1 Submission Guidelines

One important aspect in formalizing the benchmark structure is clearly defining the expected outputs, report format, and content guidelines for the contributors who plan to submit their surrogate models. This is vital to ensure consistent, reproducible, and fair evaluations. The following outlines the required, strongly recommended, and suggested components for a valid submission using the benchmark. The dataset and code availability for the benchmark are discussed in section 7.

When submitting surrogate models for evaluation, contributors must provide a detailed report that includes the predicted values for each FPGA metric in the benchmark. The report should also include visual comparisons between predicted and actual values, presented in the form of box plots to demonstrate the model’s accuracy. In addition,

Table 3. Architectures of the exemplar benchmark models.

Model	Size	Input	Architecture
Jet [15]	2,821	16	$\xrightarrow[\text{ReLU}]{32}\xrightarrow[\text{ReLU}]{32}\xrightarrow[\text{ReLU}]{32}\xrightarrow[\text{Softmax}]{5}$
Top Quarks [15]	385	10	$\xrightarrow[\text{ReLU}]{32}\xrightarrow[\text{Sigmoid}]{1}$
Anomaly [7]	2,864	128	$\xrightarrow[\text{ReLU}]{8}\xrightarrow[\text{ReLU}]{4}\xrightarrow[\text{ReLU}]{128}\xrightarrow[\text{ReLU}]{4}\xrightarrow[\text{Softmax}]{128}$
BiPC [29]	7,776	36	$\xrightarrow[\text{ReLU}]{36}\xrightarrow[\text{ReLU}]{36}\xrightarrow[\text{ReLU}]{36}\xrightarrow[\text{ReLU}]{36}\xrightarrow[\text{ReLU}]{36}$
CookieBox [20]	3,433	512	$\xrightarrow[\text{ReLU}]{4}\xrightarrow[\text{ReLU}]{32}\xrightarrow[\text{ReLU}]{32}\xrightarrow[\text{Softmax}]{5}$
AutoMLP [10]	534	7	$\xrightarrow[\text{ReLU}]{12}\xrightarrow[\text{ReLU}]{16}\xrightarrow[\text{ReLU}]{12}\xrightarrow[\text{Softmax}]{2}$
Particle Tracking [2]	2,691	14	$\xrightarrow[\text{ReLU}]{32}\xrightarrow[\text{ReLU}]{32}\xrightarrow[\text{ReLU}]{32}\xrightarrow[\text{Softmax}]{3}$

we strongly recommend that the report include a comprehensive description of the surrogate models' architectures, outlining key design details and hyperparameters used for training. While not required, sharing source code and trained weights is strongly encouraged to promote transparency and reproducibility of the results. We also recommend sharing the hardware specifications of the inference machine, along with the inference times. Lastly, we require documenting any further constraints, including additional training data (e.g., models, hls4ml configurations, or target boards), precision settings, or specific optimization strategies used during evaluation.

It is worth noting that we plan for the submission process to be open and ongoing, with no fixed release schedule. Instead, the benchmark will be updated to reflect significant contributions.

3.2 Benchmark Metrics

After establishing the rationale and structure of the benchmark, we formalize the various metrics we apply to assess the performance of surrogate models on both the test set and the exemplar models. The metrics we use are as follows [11]:

- Coefficient of determination (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{1}$$

- Symmetric mean absolute percentage error (SMAPE):

$$\text{SMAPE} = \frac{200\%}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_i| + |\hat{y}_i| + 1} \tag{2}$$

- Root mean square error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

In the above equations, y_i represents the ground truth, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of the ground truth values. The R^2 score evaluates the general performance of a predictor, measuring how well it captures the variability in the data. SMAPE offers insight into the relative accuracy of the predictions and is particularly useful when comparing errors across different scales. RMSE measures the magnitude of the prediction error, with sensitivity to larger outliers.

In the case of SMAPE, we use the standard formula and add a small value ϵ to the denominator to avoid division by zero. We set ϵ to the smallest strictly positive value that the resource and latency variables can have, which in our case is 1.

For the evaluation of these metrics, the latency is measured in clock cycles, and resources are measured in absolute terms, rather than percentage utilization. Furthermore, in the current version of the benchmark, each performance metric is computed separately for each regression variable, ensuring a detailed evaluation across all prediction targets. In addition to these metrics, we use a box plot per variable to visualize the distribution of relative percentage errors (RPE), which we define as:

$$\text{RPE} = \left(\frac{y_i - \hat{y}_i}{y_i + 1} \right) \times 100\% \tag{4}$$

The RPE box plots allow us to measure not only the spread of residuals but also identify trends in a surrogate model's predictions, providing a visual indication of whether a model tends to systematically underpredict or overpredict values.

4 Synthesis surrogate model

The development of the GNN and transformer surrogate models is another primary goal of this work. There is a significant challenge in designing an effective estimator that works for arbitrary neural network architectures. Many different layer structures can be used, which may have radically different implementations on an FPGA. The GNN-based approach allows for a flexible input scheme and architecture that can effectively consider many of these intricacies. The theoretical advantage of using a graph structure as our input is that underlying traits can be derived from the structure of the input models, allowing commonalities to be found while not requiring overly specific engineering of the training data.

Just as the benchmark and dataset are currently in their first iteration and set to evolve, we expect the surrogate model to see continued development beyond the scope of this work. For an initial comparison, we set the baseline MLP from rule4ml against the more structured GNN and transformer implementations.

4.1 Baseline MLP Implementation

The baseline implementation uses a trained MLP model to predict each FPGA resource and latency variable. The general MLP architecture is similar to the one introduced in the open-source rule4ml tool, with minor adaptations to support our dataset. The architecture processes both numerical and categorical data extracted from an input model. First, ordinal encoding [27] is applied to categorical inputs, such as the target board and hls4ml strategy. These encoded features are then processed through trainable embedding layers, which learn low-dimensional representations of the categorical data. Meanwhile, numerical inputs, composed mainly of statistical averages of the numerical features, are fed into a dense block, consisting of several fully connected layers with ReLU activations in between. The outputs from this block are concatenated with the embeddings, creating a unified feature vector. A final dense block processes the concatenated feature vector to produce the estimate. Following the methodology in [29], an MLP model is trained per target variable for 200 epochs using the Adam optimizer [23], minimizing a mean squared logarithmic loss.

4.2 Graph Neural Network

Estimating the resource usage and latency of an arbitrary ML model presents unique challenges. In particular, since a model can have any number of layers, each of which has arbitrary numbers of node connections, it is a challenge to effectively model these structures. Furthermore, even relatively simple neural networks can have structures such as skip connections, which may have nontrivial effects on the resulting resource usage and latency of the inference engine.

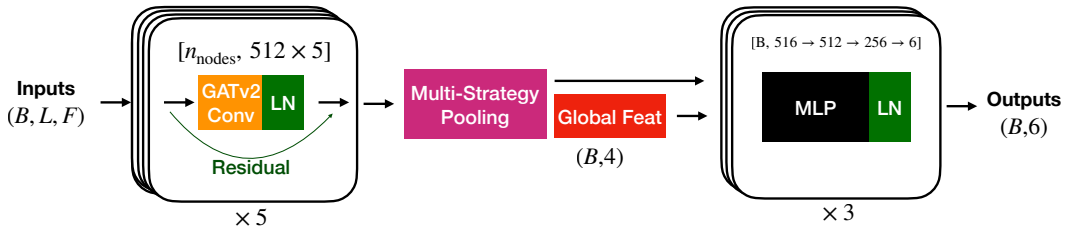

Fig. 5. The overall structure of the GNN, comprising five GATv2Conv layers. The vector (B, L, F) consists of B batch size, L layers per model, and F features per layer.

Previous attempts to resolve this used the total number of layers of the network as a feature, along with other relevant values corresponding to the layers [30]. However, this may result in a severely limited scope of input models, since very different architectures may end up sharing nearly identical input features under these constraints, while their HLS syntheses produce very different circuits. One way to handle these limitations would be to use a modeling structure with fewer limitations on how heterogeneous the input data can be. For this, we take advantage of GNNs. Since the input to such a network can be an arbitrary graph, we can convert our input models into a graph representation, allowing for the heterogeneous data to be directly handled by our surrogate model.

4.2.1 Features and Preprocessing. Each layer of the input model is treated as a graph node with an 18-dimensional feature vector consisting of three input and output dimensions, precision, reuse factor, strategy, layer or activation type, filters, kernel size, stride, padding, batch normalization, and I/O type. Numerical features like layer dimensions and synthesis parameters are normalized via z-score standardization using training set statistics. Categorical features, including layer type, activation, and padding, are one-hot encoded.

The graph is constructed by connecting nodes based on the sequential dataflow between layers. Self-loops are added to each node to allow the attention mechanism to consider the layer's own features during message passing. Global attributes that affect the entire model, such as synthesis strategy and I/O interface type, are also one-hot encoded and appended to each node's feature vector to provide consistent context.

4.2.2 Structure. The chosen GNN structure shown in Figure 5 allows for arbitrary directed graph input, with each node having a fixed 18-dimensional feature vector after preprocessing. It consists of five stacked graph attention network version 2 (GATv2) [8, 19, 36] layers, each using five attention heads to capture various facets of inter-layer relationships. The GATv2 attention mechanism assigns importance weights to edges dynamically based on the learned compatibility between node features. The attention mechanism weighs edges according to their relative importance using the GATv2 formulation, where attention weights $\alpha_{i,j}$ are calculated as

$$\mathbf{x}'_i = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{i,j} \mathbf{W} \mathbf{x}_j\right) \tag{5}$$

where $\mathbf{W}$ is a trainable weight matrix for the linear transformation applied to the node features. The attention coefficients, $\alpha_{i,j}$ which determine the importance of node j's features to node i, are computed dynamically for each edge using the GATv2 mechanism:

$$\alpha_{i,j} = \frac{\exp\left(\mathbf{a}^\top \mathrm{ELU}\left(\mathbf{W}_s \mathbf{x}_i + \mathbf{W}_t \mathbf{x}_j\right)\right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp\left(\mathbf{a}^\top \mathrm{ELU}\left(\mathbf{W}_s \mathbf{x}_i + \mathbf{W}_t \mathbf{x}_k\right)\right)} \tag{6}$$

where $\mathbf{a}$ is a learnable weight vector, and $\mathbf{W}_s$ and $\mathbf{W}_t$ are trainable weight matrices for the source and target nodes, respectively.

This allows the GNN to automatically determine which layer connections are most informative for prediction, even accounting for special structures like skip connections that may have an large impact on resource usage. The GATv2 outputs undergo layer normalization, ELU activation, and dropout regularization, with residual connections to aid gradient flow in the deep architecture.

Node embeddings from the final GATv2 layer are first reduced to a standard size by a linear projection. Then, a learnable weighted combination of additive, mean, and max pooling aggregates them into a single graph-level embedding, which is concatenated with the one-hot encoded global features. This final representation passes through an MLP to yield the hardware usage estimates.

4.2.3 Implementation. We implemented the GNN using the PyTorch [26] and PyTorch Geometric [19] libraries. The JSON dataset was converted to NumPy arrays for efficient I/O, with logarithmic scaling applied to the target hardware metrics before z-score normalization to stabilize training.

The GNN architecture is built using standard modules from these libraries, including GATv2Conv or the attention-based graph convolutions and LayerNorm [4] for stabilizing the activations between layers. The model is trained by minimizing the mean-squared error (MSE) loss between the normalized predictions and targets using the AdamW optimizer [24]. We train the GNN network using our training set, which comprises 70% of the total samples from the benchmark dataset. A dynamic learning rate is employed, and the network trained for 200 epochs. The training was performed with an NVIDIA A10 GPU.

4.3 Transformer

Similar to a GNN, a transformer architecture is a viable method to effectively estimate the resources and latency of models. With attention [35], the model complexity, scale, and relationship between layers can be better understood, by treating each layer as its own token.

4.3.1 Features and Preprocessing. As done for the GNN, the transformer follows a similar preprocessing procedure. The same 18-dimensional feature vector is produced, however no nodes are connected and no global features are created.

4.3.2 Structure. The transformer architecture is depicted in Figure 6. The inputs are dimension (B, L, F) where B is the batch size, L is the number of layers per model, and F is the number of features per layer, which is 18. The input embedding projects F into a 512-dimensional embedding. The positional encoding adds information about each layer's position relative to the others. The [CLS] token is prepended to L, increasing the dimensionality by 1, aggregating the information of the entire sequence. The two encoder blocks comprise an 8-head self-attention layer followed by a feed-forward network with normalization. The output of these blocks is the [CLS] token output, summarizing the whole model. The linear layer maps 512 to 6 outputs, corresponding to the hardware resource predictions.

4.3.3 Implementation. The transformer was implemented using PyTorch [26] for all model components. Each input model is encoded as a sequence of features per layer, with padding applied up to a maximum of 51 layers. All features

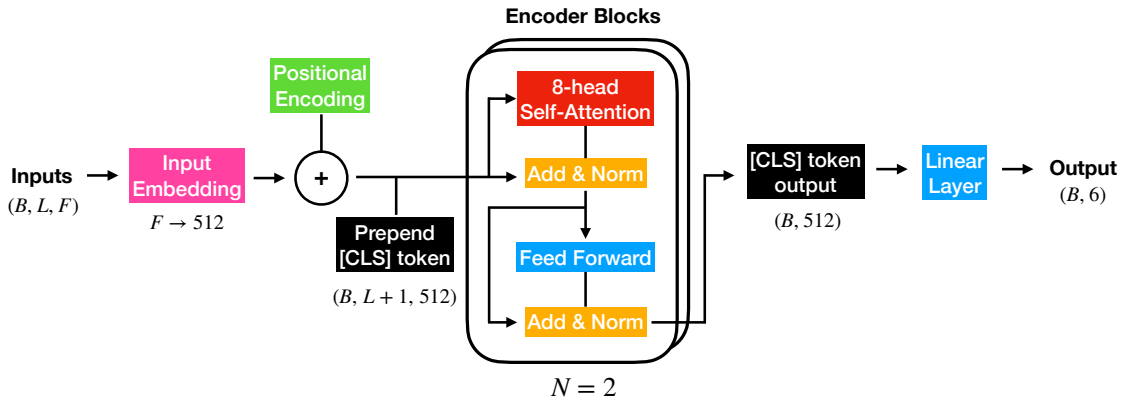

Fig. 6. The overall structure of the transformer, comprising 2 encoder blocks. The vector (B, L, F) consists of B batch size, L layers per model, and F features per layer. $N = 2$ denotes 2 sequential encoder blocks.

are normalized based on the training set's mean and standard deviation. A padding mask is generated for each sample to indicate which layers are real or padded.

Each feature vector per layer of length 18 is projected into a 512-dimensional token embedding by a learned linear layer. Learnable positional encodings are added to each token to encode the order of layers. A [CLS] token is prepended to each sequence. Padding masks are passed into the transformer to prevent attention to padded tokens.

The architecture, as defined above, uses nn.TransformerEncoder and nn.TransformerEncoderLayer. Dropout is applied within the encoder layers for regularization. The output predicts the 6 hardware metrics.

The model is trained for 250 epochs with a batch size of 1024. Training is performed on an NVIDIA A100 GPU. For inference, the outputs are rescaled to the original metrics scales for accurate predictions.

5 Results

5.1 Relative Percent Error

As discussed in subsection 3.2, we visualize results with the RPE, relative percent error, using box plots. The results are shown for each resource target (BRAM, DSP, FF, and LUT) and latency target (clock cycles) and initiation interval (II) in a box plot. We present results for the synthetic model test samples and exemplar realistic models for the baseline MLP, GNN, and transformer prediction models. Figure 7 and Figure 8 show the synthetic model test set and the exemplar models, respectively, for the baseline MLP. Figure 9 and Figure 10 show the synthetic model test set and the exemplar models, respectively, for the GNN model. Figure 11 and Figure 12 show the synthetic model test set and the exemplar models, respectively, for the transformer model. In each plot, the colored box covers the interquartile range (IQR), spanning both the first quartile (25%) and the third (75%). The dashed horizontal lines within the box show the median (orange) and mean (green) of the distribution. The whiskers extend from the box to the smallest and largest values within 1.5 times the IQR, capturing most of the data spread and considering points outside this range as outliers.

Based on the test set RPE boxplots in Figure 9 and Figure 11, the GNN and transformer models demonstrate a significant improvement over the baseline MLP. As shown in Figure 7, the MLP predictions generally have wider interquartile ranges (IQRs) and medians. The transformer model, in particular, shows very narrow IQRs for DSP and

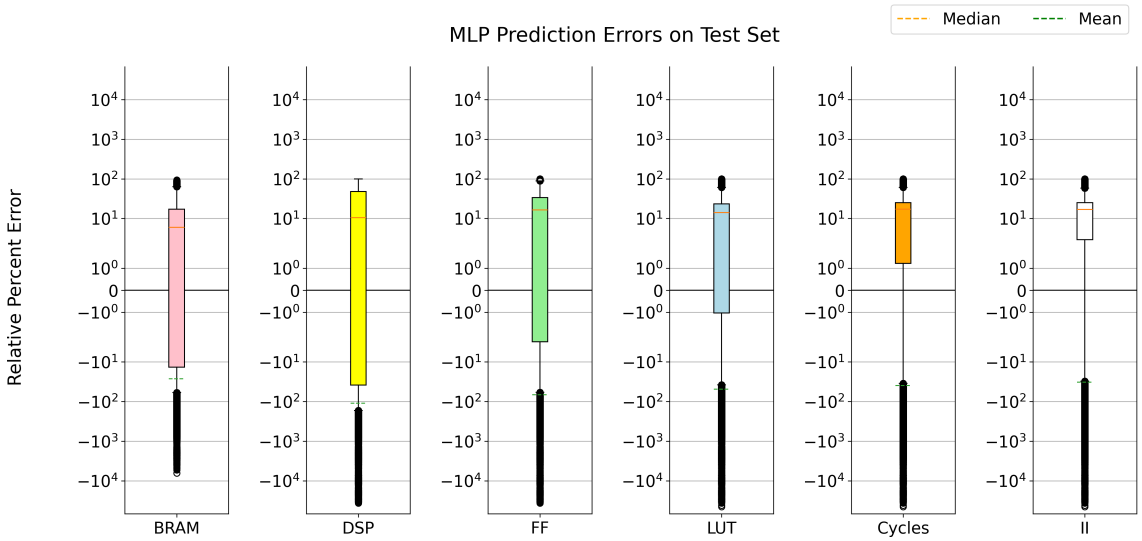

Fig. 7. Relative percentage errors of the baseline MLP on the test set. The y-axis is set to a symmetric log scale.

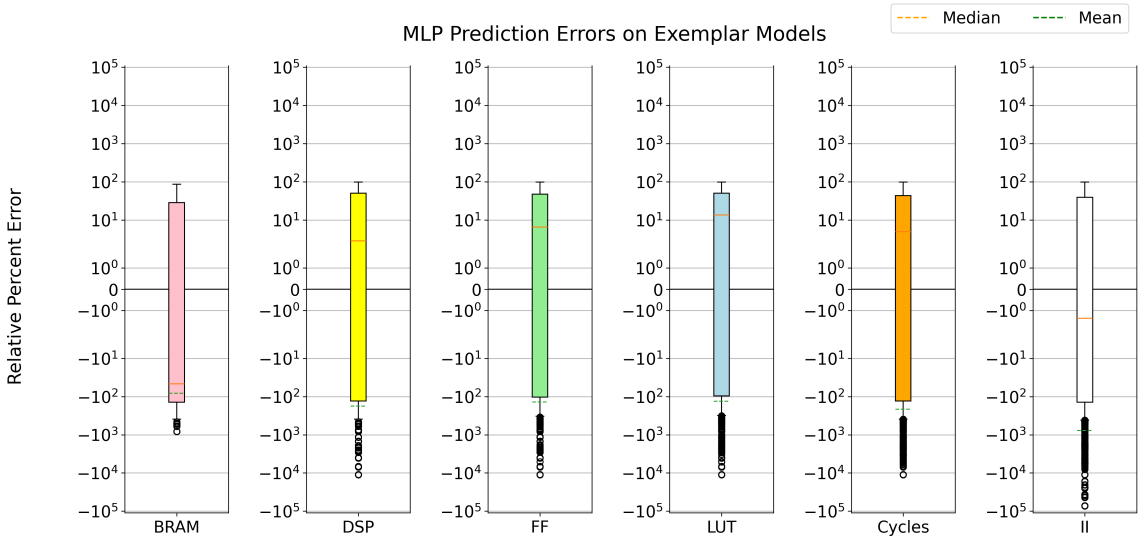

Fig. 8. Relative percentage errors of the MLP on the exemplar models. The y-axis is set to a symmetric log scale.

Cycles near zero. The GNN tends to have somewhat narrow IQRs for BRAM, DSP, and Cycles, also having a tendency to over-predict for these features, which may be a more desirable behavior than under-predicting.

When evaluated on the exemplar dataset, all three predictors show a drop in performance, highlighting the challenge of generalizing new and complex architectures not present in the training data. The RPE for the exemplar set, illustrated in Figure 8, Figure 10, and Figure 12, shows considerably wider and considerably more outliers compared to the

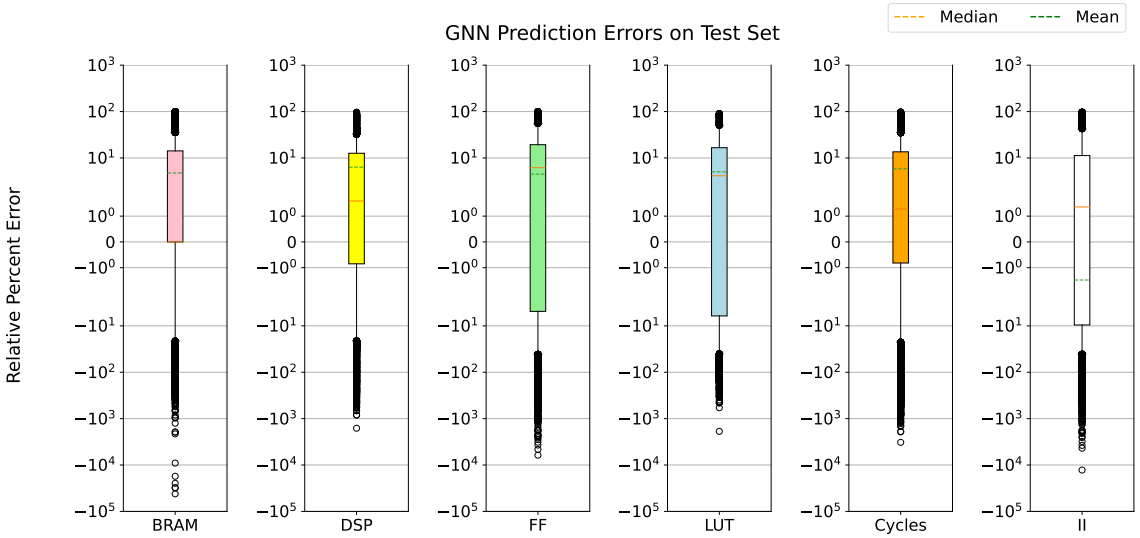

Fig. 9. Relative percent error for the GNN on the test subset. The y-axis is set to a symmetric log scale.

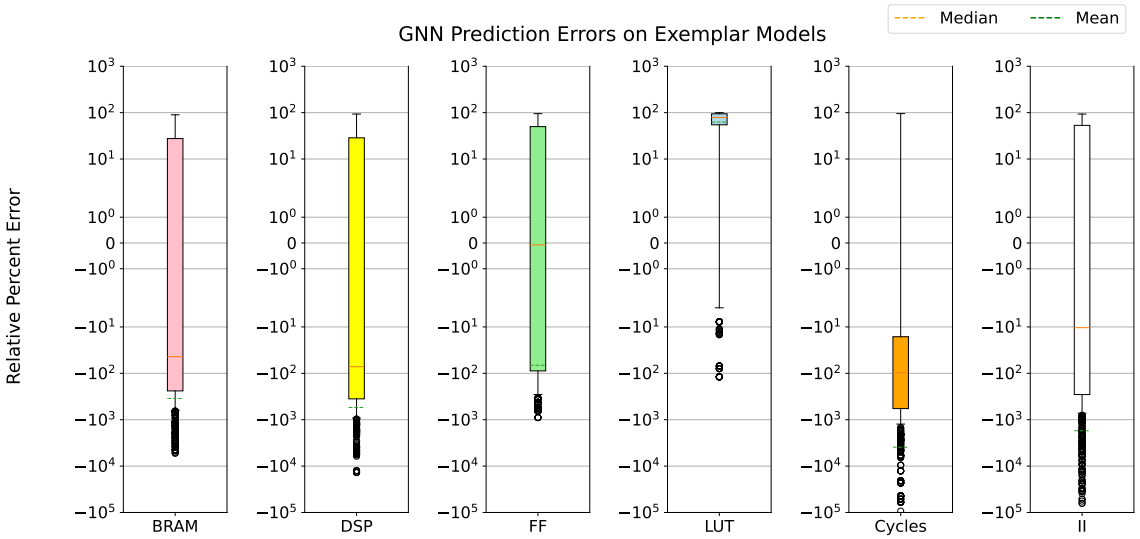

Fig. 10. Relative percent error for the GNN on the exemplar dataset. The y-axis is set to a symmetric log scale. Activation layers are removed from the exemplar set to keep a similar input structure as the GNN was trained on.

test set results. Looking at the median and mean, the MLP tends to over-predict, whereas the GNN and transformer under-predict, likely due to their log scaling pre-processing step.

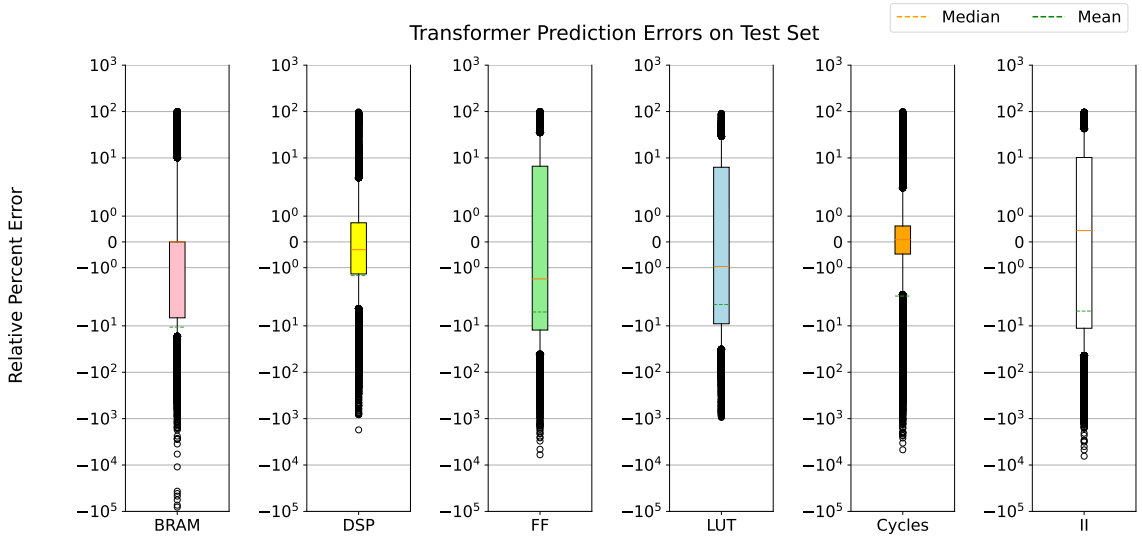

Fig. 11. Relative percent error for the transformer on the test subset. The y-axis is set to a symmetric log scale.

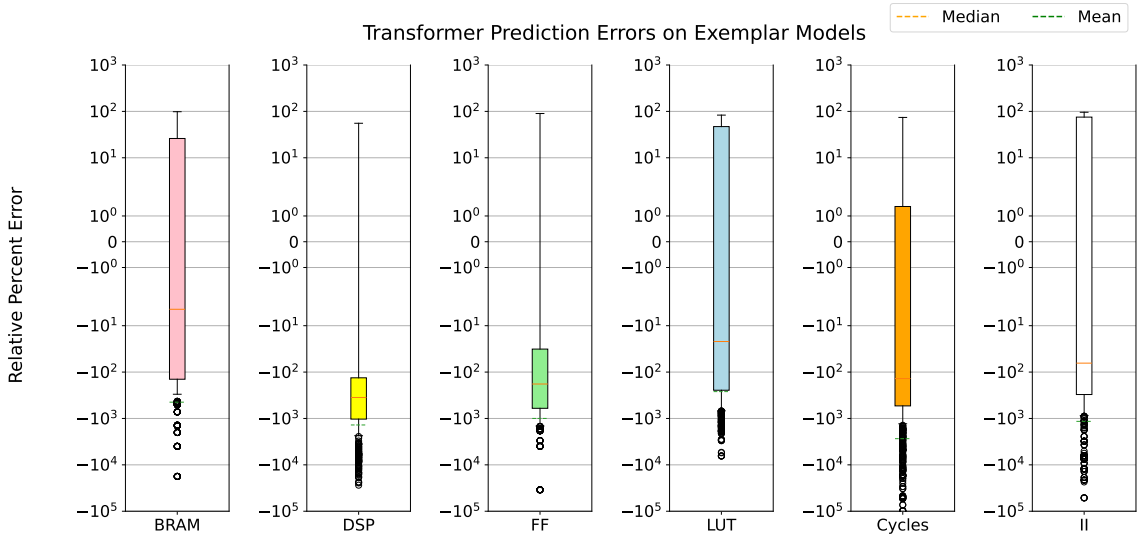

Fig. 12. Relative percent error for the transformer on the exemplar subset. The y-axis is set to a symmetric log scale.

5.2 Full evaluation metrics

For a quantitative analysis, we evaluated the R^2, SMAPE, and RMSE metrics defined in subsection 3.2 for each surrogate model. These metrics were computed for the test set and each architecture within the exemplar set. The results are summarized in Table 4 for the test set and Table 5 for the exemplar set.

Table 4. Evaluation metrics and results of the surrogate models on the test set and its subsets.

Models	Arch.	R^2 Score						SMAPE [%]						RMSE					
		BRAM	DSP	FF	LUT	Cycles	II	BRAM	DSP	FF	LUT	Cycles	II	BRAM	DSP	FF	LUT	Cycles	II
Test set (All)	MLP	0.32	0.03	0.20	0.49	0.54	0.56	33.9	105.7	24.9	15.5	31.8	25.8	**12.5**	590.4	**31897.9**	**40463.1**	450879.5	221524.7
	GNN	**0.51**	**0.89**	0.74	**0.73**	0.89	0.91	19.5	15.1	11.6	11.4	15.7	**13.4**	48.0	**580.0**	55087.6	104945.7	227987.8	201369.9
	Transformer	0.39	0.29	**0.72**	0.67	**0.95**	**0.95**	**14.1**	**10.8**	**2.9**	**2.9**	**10.1**	14.1	53.7	1472.3	57138.2	115943.6	**147137.7**	**150688.4**
Test set (Dense)	MLP	**0.47**	0.03	0.13	0.49	0.74	0.77	33.3	106.6	24.5	14.8	30.9	24.8	**9.3**	**603.0**	**31441.0**	**41221.3**	1548.5	639.8
	GNN	-0.51	-0.74	**0.73**	**0.73**	0.82	**0.91**	24.6	23.9	11.8	11.6	15.8	**13.4**	86.1	18950.8	56341.8	107544.5	1304.9	415.1
	Transformer	0.39	**0.29**	0.71	0.67	**0.95**	**0.91**	**13.6**	**10.2**	**2.7**	**2.7**	**9.9**	14.1	54.9	1509.2	58463.5	118826.6	**659.7**	**395.6**
Test set (Conv1D)	MLP	0.39	-0.07	0.31	0.24	0.38	0.40	44.7	80.5	28.7	27.2	48.4	43.8	**8.9**	2.5	14249.3	8994.6	98637.3	48849.3
	GNN	0.69	0.02	0.95	**0.96**	**0.97**	**0.97**	33.7	36.7	7.7	6.2	11.0	**11.1**	16.9	1.7	5137.6	3731.4	**21648.4**	**24199.6**
	Transformer	**0.77**	**0.41**	**0.97**	**0.96**	0.96	0.96	**29.9**	**22.7**	**7.2**	**5.8**	**10.4**	11.2	14.5	**1.3**	**4275.9**	**3421.7**	24561.6	27547.3
Test set (Conv2D)	MLP	-0.50	0.15	0.51	0.41	0.33	0.35	51.5	87.0	41.8	34.4	56.1	54.7	59.6	6.0	57190.7	18547.8	3181122.1	1562928.9
	GNN	0.44	0.51	0.92	0.95	0.84	0.88	31.2	34.8	8.8	**6.7**	19.5	18.5	27.4	5.5	24145.3	15283.9	1549792.9	1365427.3
	Transformer	**0.79**	**0.55**	**0.93**	**0.96**	**0.93**	**0.93**	**18.8**	**25.3**	**8.0**	6.8	**16.3**	**16.9**	**16.7**	**5.3**	**22659.7**	**12971.7**	**999953.4**	**1024014.3**

Table 5. Evaluation metrics and results of the surrogate models on the exemplar architectures.

Models	Arch.	R^2 Score						SMAPE [%]						RMSE					
		BRAM	DSP	FF	LUT	Cycles	II	BRAM	DSP	FF	LUT	Cycles	II	BRAM	DSP	FF	LUT	Cycles	II
Jet	MLP	**-1.22**	0.27	0.33	-0.12	**0.53**	**0.49**	**65.0**	86.0	**58.8**	**74.1**	89.1	**66.4**	**2.2**	569.8	8654.0	18214.8	**711.4**	**393.6**
	GNN	-1.29	-0.14	0.12	**0.19**	0.43	0.41	143.8	170.3	86.3	80.8	**86.6**	103.6	2.3	712.2	9917.7	**15516.7**	780.3	424.2
	Transformer	-23.19	**0.32**	**0.46**	-0.03	0.17	0.04	110.3	**77.7**	80.6	90.1	103.4	120.4	7.4	**550.8**	**7764.8**	17520.1	944.2	542.0
Quarks	MLP	N/A*	**0.19**	**0.51**	**-0.41**	-36.68	-13.18	**129.9**	**118.1**	**83.5**	**91.9**	158.4	**136.3**	2.3	108.6	**1327.1**	**3298.1**	441.1	266.6
	GNN	N/A*	-0.31	-0.64	-11.33	-95.13	-33.73	200.0	119.9	105.8	113.6	171.1	170.2	**1.2**	137.9	2432.5	9756.8	704.5	417.3
	Transformer	N/A*	-0.28	0.24	-12.46	**-4.21**	**-2.61**	200.0	138.4	92.0	117.4	**144.2**	152.5	119.9	**136.2**	1655.9	10193.2	**164.0**	**134.5**
Anomaly	MLP	-0.80	0.26	**0.59**	**0.45**	0.42	**0.49**	**104.3**	**59.2**	**36.9**	**49.2**	51.5	107.9	4.4	500.7	**12774.2**	**14251.0**	761.2	**376.7**
	GNN	**-0.64**	0.17	-0.54	-6.91	**0.43**	0.46	117.5	185.8	74.8	91.5	76.2	**86.7**	**4.2**	531.8	24620.3	53979.5	**755.7**	384.6
	Transformer	-185.38	**0.71**	-10.84	-135.03	0.32	0.32	168.9	72.3	121.3	160.7	**45.8**	117.3	44.8	**312.1**	68319.0	223829.6	820.9	434.9
BiPC	MLP	**-0.71**	-0.04	**0.16**	**0.03**	**0.44**	0.43	**107.7**	127.0	**77.2**	**82.3**	75.9	**70.6**	**3.4**	1719.2	**24472.0**	**48867.4**	**1821.4**	573.6
	GNN	-0.93	-0.10	-0.35	-0.24	0.26	**0.45**	145.2	**118.5**	111.5	89.6	87.7	119.1	3.6	1776.3	30980.1	55222.2	2090.1	**559.7**
	Transformer	-17.38	**0.12**	-16.70	-12.16	0.43	**0.45**	135.1	136.2	124.6	120.4	**69.3**	101.3	11.1	**1588.1**	112115.9	179918.2	1832.6	563.7
Cookie-box	MLP	**-0.54**	0.20	**0.34**	**0.13**	**0.45**	**0.53**	**64.4**	91.3	**66.8**	**67.9**	**32.2**	**72.3**	**3.5**	657.9	11675.1	**21420.7**	**653.5**	**341.0**
	GNN	-19.23	**0.98**	-1.92	-90.18	0.21	0.26	157.2	137.6	114.4	118.4	45.7	82.6	12.7	**104.5**	24559.3	219369.3	784.5	429.1
	Transformer	-37.91	0.64	**0.34**	-19.36	0.26	-0.02	141.9	**88.0**	93.3	138.4	39.0	130.7	17.6	438.8	**11657.6**	103672.4	762.4	503.6
AutoMLP	MLP	-1.09	**0.41**	**0.69**	**-0.22**	-0.33	-1.71	**56.4**	**72.1**	**61.4**	**68.9**	86.8	**86.0**	0.9	**104.4**	**1524.8**	**3586.5**	226.5	163.0
	GNN	**-1.04**	-0.15	0.19	-0.56	-0.08	-0.34	161.8	180.8	79.9	78.9	**79.0**	96.2	**0.9**	145.1	2459.0	4056.9	**205.6**	114.7
	Transformer	-59.19	0.01	0.15	-13.91	**0.20**	**0.22**	118.1	95.4	74.2	120.8	101.8	113.9	4.9	134.6	2528.0	12540.5	177.1	**87.5**
Particle Tracking	MLP	-1.41	0.28	0.33	-0.08	**0.52**	**0.50**	**65.6**	**75.2**	**58.1**	**71.2**	87.5	**61.6**	2.1	536.1	8093.3	16695.1	**692.9**	**382.0**
	GNN	**-1.03**	-0.13	0.19	**0.14**	0.45	0.40	144.4	158.5	84.8	81.4	**83.5**	100.9	**2.0**	670.2	9100.7	**15013.8**	745.5	419.1
	Transformer	-24.92	**0.34**	**0.47**	-0.01	0.15	0.04	118.5	83.7	80.6	89.1	108.0	124.5	7.0	**512.5**	**7384.7**	16204.8	927.1	529.5

* R^2 score calculation is skipped since all true values are 0.

On the test set, Table 4, the three models show distinct performance patterns. The transformer has the highest R^2 scores for Cycles (0.95) and II (0.95), while the GNN performs best for DSP (0.89) and competitive scores for other resources. The MLP shows lower R^2 values across most metrics, particularly for DSP (0.03). In terms of SMAPE, the transformer achieves lower errors for FF (2.9%) and LUT (2.9%), while the GNN shows the best performance for II (13.4%). RMSE results vary considerably, with the MLP showing the lowest values for BRAM, FF, and LUT, though this may reflect its tendency towards smaller absolute predictions rather than better accuracy.

Performance varies significantly across the layer types of the test set. For dense layers, all models show improved performance compared to the overall set, likely due to their relative simplicity and high representation in the overall dataset. For Conv1D layers, both the GNN and transformer show strong prediction with R^2 values exceeding 0.95 for most metrics, while the MLP struggles particularly with DSP ($R^2 = -0.7$). Conv2D layers present the greatest challenge, though the transformer still achieves R^2 values above 0.90 for Cycles and II.

The exemplar architectures Table 5 highlight the generalization challenges faced by all models. Negative R^2 values are common across all models, indicating predictions worse than the mean. The MLP shows negative R^2 for BRAM in most cases but performs relatively better for Cycles and II in some architectures (Jet: 0.53, 0.49). The GNN shows mixed performance, with particularly poor performance on Quarks and Anomaly architectures but reasonable performance for specific metrics in other cases. The transformer shows the most variability, achieving the best DSP predictions for several architectures but struggling with BRAM predictions.

SMAPE values for the exemplar are also substantially higher than the test set across all models. The transformer generally achieves lower SMPAPE values for resource utilization (particularly DSP), while showing competitive performance for timing metrics.

As seen in Appendix A, all models are better at predicting resources and latency for the test set compared to the exemplar set. Relative to the MLP, the GNN and transformer are able to consistently predict DSP and Cycles on the test set. Overall, the discrepancy between the exemplar and test data performance we see among all models can be attributed to the distribution shown in Figure 4, where the exemplar data is not reflected by the training and test subsets. This highlights the need to further improve the diversity of the dataset to include a wider variety of model architectures.

6 Summary and Outlook

We developed wa-hls4ml as a benchmark to provide a standardized method to evaluate the performance of neural-network-based FPGA resource estimation tools. Alongside the associated dataset, we hope to provide a comprehensive performance evaluation scheme and a basis for the further development of similar tools, in line with previous benchmark efforts [7, 16].

In presenting the GNN and transformer-based neural networks, we seek to demonstrate the performance of novel architectures for estimating latency and resource utilization of neural networks on FPGAs through hls4ml. The results demonstrate that the surrogate models perform well on the test dataset, indicating that our approach toward estimating resources and latency is viable and that further research into these methods is warranted.

While the estimators perform well on test data similar to their training set, their performance on the realistic exemplar set, which contains varying architectures and different configurations not present in the training data, is lacking. The ability of the GNN and transformer to handle such varied architectures implicitly still offers great potential, especially with the development of a more robust dataset.

This work demonstrates one concrete application where the surrogate model rapidly predicts resource/latency estimates, but the wa-hls4ml dataset is intentionally broader. Including the full project for each synthesized model (HLS code, IR and multi-stage reports) opens up other applications such as code/IR-driven learning (code2vec embeddings, code autocomplete), budget-aware architecture recommendation, and LLM-based assistants tailored to HLS.

In future work, we intend to expand the provided dataset to include larger neural networks, more intricate architectures that include features like skip connections, a larger variety of hardware settings like reuse factors, and a larger number of samples in the dataset. As hls4ml evolves to support more architectures, we intend to continuously extend the dataset to reflect the added features.

We additionally intend to further improve the GNN and transformer models not only with an improved training dataset, but through further refinements of their architecture, and the exploration of techniques such as preferring overestimation when calculating the loss. Similarly, we intend to continue to develop and update the benchmark periodically, incorporating new metrics, tracked features, and dataset improvements as appropriate.

7 Dataset and Code Availability

The training, test, and exemplar test datasets discussed in this work are available here: https://huggingface.co/datasets/fastmachinelearning/wa-hls4ml, licensed under the CC-BY-NC 4.0 license. Additionally, a dataset containing the corresponding synthesized project files and logs for the samples in the training, test, and exemplar datasets is available here: https://huggingface.co/datasets/fastmachinelearning/wa-hls4ml-projects, licensed under the CC-BY-NC 4.0 license. The code used to generate the datasets, plots, models, and other associated results are available at the following meta repository, licensed under the respective licenses as mentioned in the repositories: https://github.com/fastmachinelearning/wa-hls4ml-paper.

Acknowledgments

This manuscript has been authored by FermiForward Discovery Group, LLC under Contract No. 89243024CSC000002 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI).

Compute was provided in part by the Elastic Analysis Facility at Fermilab.

Portions of this research were conducted with the advanced computing resources provided by Texas A&M High Performance Research Computing.

This work was supported in part by National Science Foundation (NSF) awards CNS-1730158, ACI-1540112, ACI-1541349, OAC-1826967, OAC-2112167, CNS-2100237, CNS-2120019, the University of California Office of the President, and the University of California San Diego's California Institute for Telecommunications and Information Technology/Qualcomm Institute. Thanks to CENIC for the 100 Gbps networks.

KS is supported by National Science Foundation (NSF) Grants 2112356 and 2411377

BH, DP, KT, GG, and NT are supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy (DOE), Office of Science, Office of High Energy Physics. BH and NT are also supported under the DOE Early Career Research program under Award No. DE-0000247070. KT is also supported by DOE Grant KA2401045. BH, JD, and NT are supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under the "Real-time Data Reduction Codesign at the Extreme Edge for Science" Project(DE-FOA-0002501).

JD is supported by the Research Corporation for Science Advancement (RCSA) under grant #CS-CSA-2023-109, Alfred P. Sloan Foundation under grant #FG-2023-20452, U.S. Department of Energy (DOE), Office of Science, Office of High Energy Physics Early Career Research program under Award No. DE-SC0021187, and the U.S. National Science Foundation (NSF) Harnessing the Data Revolution (HDR) Institute for Accelerating AI Algorithms for Data Driven Discovery (A3D3) under Cooperative Agreement PHY-2117997.

Thank you to Prof. Luca Carloni at Columbia University for for supporting the work of DS through the CSEE–E6868: Embedded Scalable Platforms – Spring '25 course.

JW is supported by a WATCHEP fellowship sponsored by the DOE, Office of High-Energy Physics under Award No. DE-SC-0023527.

HER, MMR, and ACT are supported by funding from the Canada Research Chairs Program. ACT holds the Canada Research Chair in Real-Time Intelligence Embedded for High-Speed Sensors.

References

[1] Stefan Abi-Karam, Rishov Sarkar, Allison Seigler, Sean Lowe, Zhigang Wei, Hanqiu Chen, Nanditha Rao, Lizy John, Aman Arora, and Cong Hao. 2024. HLSFactory: A Framework Empowering High-Level Synthesis Datasets for Machine Learning and Beyond. (2024). arXiv:2405.00820

[2] H Abidi, A Boveia, V Cavaliere, D Furletov, A Gekow, CW Kalderon, and S Yoo. 2022. Charged Particle Tracking with Machine Learning on FPGAs. (2022). arXiv:2212.02348

[3] Advanced Micro Devices (Xilinx). 2024. *Vitis High-Level Synthesis User Guide (UG1399)* (2024.2 ed.). Advanced Micro Devices. https://docs.amd.com/r/en-US/ug1399-vitis-hls

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer Normalization.

[5] Yunsheng Bai, Atefeh Sohrabizadeh, Zongyue Qin, Ziniu Hu, Yizhou Sun, and Jason Cong. 2023. Towards a Comprehensive Benchmark for High-Level Synthesis Targeted to FPGAs. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 45288. https://proceedings.neurips.cc/paper_files/paper/2023/file/8dfc3a2720a4112243a285b98e0d4415-Paper-Datasets_and_Benchmarks.pdf

[6] Gergö Barany. 2017. Liveness-driven random program generation. In *International Symposium on Logic-Based Program Synthesis and Transformation*. Springer, 112–127.

[7] Hendrik Borras, Giuseppe Di Guglielmo, Javier Duarte, Nicolò Ghielmetti, Ben Hawks, Scott Hauck, Shih-Chieh Hsu, Ryan Kastner, Jason Liang, Andres Meza, et al. 2022. Open-source FPGA-ML codesign for the MLPerf Tiny Benchmark. In *3rd Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware (MLBench) at 5th Conference on Machine Learning and Systems (MLSys)*. arXiv:2206.11791

[8] Shaked Brody, Uri Alon, and Eran Yahav. 2022. How Attentive are Graph Attention Networks?. In *International Conference on Learning Representations*. arXiv. arXiv:2105.14491 https://openreview.net/forum?id=F72ximsx7C1

[9] Javier Campos, Jovan Mitrevski, Nhan Tran, Zhen Dong, Amir Gholaminejad, Michael W Mahoney, and Javier Duarte. 2024. End-to-end codesign of Hessian-aware quantized neural networks for FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 17, 3 (2024), 1. arXiv:2304.06745 doi:10.1145/3662000

[10] Chao Chen, Bruno da Silva, Chenxi Yang, Caiyun Ma, Jianqing Li, and Chengyu Liu. 2023. AutoMLP: A framework for the acceleration of multi-layer perceptron models on FPGAs for real-time atrial fibrillation disease detection. *IEEE Transactions on Biomedical Circuits and Systems* (2023).

[11] Davide Chicco, Matthijs J Warrens, and Giuseppe Jurman. 2021. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *PeerJ Comput. Sci.* 7 (2021), e623.

[12] Claudionor N. Coelho Jr., Aki Kuusela, Shan Li, Hao Zhuang, Thea Aarrestad, Vladimir Loncar, Jennifer Ngadiuba, Maurizio Pierini, Adrian Alan Pol, and Sioni Summers. 2021. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nat. Mach. Intell.* 3 (2021), 675. arXiv:2006.10159 doi:10.1038/s42256-021-00356-5

[13] Steve Dai, Yuan Zhou, Hang Zhang, Ecenur Ustun, Evangeline F.Y. Young, and Zhiru Zhang. 2018. Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 129–132. doi:10.1109/FCCM.2018.00029

[14] Dana Diaconu, Lucian Petrica, Michaela Blott, and Miriam Leeser. 2022. Machine Learning Aided Hardware Resource Estimation for FPGA DNN Implementations. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 77–83. doi:10.1109/IPDPSW55747.2022.00022

[15] Javier Duarte, Song Han, Philip Harris, Sergo Jindariani, Edward Kreinar, Benjamin Kreis, Jennifer Ngadiuba, Maurizio Pierini, Ryan Rivera, Nhan Tran, et al. 2018. Fast inference of deep neural networks in FPGAs for particle physics. *JINST* 13 (2018), P07027. arXiv:1804.06913 doi:10.1088/1748-0221/13/07/P07027

[16] Javier Duarte, Nhan Tran, Ben Hawks, Christian Herwig, Jules Muhizi, Shvetank Prakash, and Vijay Janapa Reddi. 2022. FastML Science Benchmarks: Accelerating Real-Time Scientific Edge Machine Learning. In *3rd Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware (MLBench) at 5th Conference on Machine Learning and Systems (MLSys)*. arXiv:2207.07958

[17] Farah et al Fahim. 2021. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. In *1st TinyML Research Symposium*. arXiv:2103.05579

[18] Lorenzo Ferretti, Jihye Kwon, Giovanni Ansaloni, Giuseppe Di Guglielmo, Luca Carloni, and Laura Pozzi. 2021. DB4HLS: A Database of High-Level Synthesis Design Space Explorations. arXiv:2101.00587 https://arxiv.org/abs/2101.00587

[19] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR 2019 (RLGM Workshop)*. arXiv. arXiv:1903.02428 doi:10.48550/arXiv.1903.02428

[20] Berthié Gouin-Ferland, Mohammad Mehdi Rahimifar, Charles-Étienne Granger, Quentin Wingering, Ryan Coffee, and Audrey Corbeil Therrien. 2022. Combining optimized quantization and machine learning for real-time data reduction at the edge. In *2022 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*. IEEE, 1.

[21] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, and Hiroaki Takada. 2009. Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis. *journal of information processing* 17 (2009), 242–254.

[22] M. Usman Jamal, Zhuowei Li, Mihai T. Lazarescu, and Luciano Lavagno. 2023. A Graph Neural Network Model for Fast and Accurate Quality of Result Estimation for High-Level Synthesis. *IEEE Access* 11 (2023), 85785–85798. doi:10.1109/ACCESS.2023.3303840

[23] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference for Learning Representations*. arXiv:1412.6980

[24] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. https://openreview.net/forum?id=Bkg6RiCqY7

[25] Jason Moss. 2024. xilinx-docker. https://gitlab.com/rjmoss/xilinx-docker

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. arXiv:1912.01703

[27] Kedar Potdar, Taher Pardawala, and Chinmay Pai. 2017. A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *Int. J. Comput. Appl.* 175 (2017), 7.

[28] Louis-Noel Pouchet and Tomofumi Yuki. 2018. PolyBench/C. https://sourceforge.net/projects/polybench/

[29] Hamza Ezzaoui Rahali, Mohammad Mehdi Rahimifar, Charles-Étienne Granger, Zhehui Wang, and Audrey C Therrien. 2024. Efficient compression at the edge for real-time data acquisition in a billion-pixel X-ray camera. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1058 (2024), 168829.

[30] Mohammad Mehdi Rahimifar, Hamza Ezzaoui Rahali, and Audrey C Therrien. 2024. rule4ml: An Open-Source Tool for Resource Utilization and Latency Estimation for ML Models on FPGA. (2024). arXiv:2408.05314

[31] Brandon Reagen, Robert Adolf, Yakun Sophia Shao, Gu-Yeon Wei, and David Brooks. 2014. MachSuite: Benchmarks for accelerator design and customized architectures. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 110. doi:10.1109/IISWC.2014.6983050

[32] Brandon Reagen, Robert Adolf, Yakun Sophia Shao, Gu-Yeon Wei, and David Brooks. 2014. Machsuite: Benchmarks for accelerator design and customized architectures. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 110–119.

[33] R. Sarkar, S. Abi-Karam, Y. He, L. Sathidevi, and C. Hao. 2023. FlowGNN: A Dataflow Architecture for Real-Time Workload-Agnostic Graph Neural Network Inference. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE Computer Society, Los Alamitos, CA, USA. arXiv:2204.13103 doi:10.1109/HPCA56546.2023.10071015

[34] Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. 2022. Automated Accelerator Optimization Aided by Graph Neural Networks. In *2022 59th ACM/IEEE Design Automation Conference (DAC)*.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL] https://arxiv.org/abs/1706.03762

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*. arXiv:1710.10903 doi:10.48550/arXiv.1710.10903

[37] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. 2006. Ceph: a scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. USENIX Association, USA, 307.

[38] Nan Wu, Hang Yang, Yuan Xie, Pan Li, and Cong Hao. 2022. High-level synthesis performance prediction using GNNs: benchmarking, modeling, and advancing. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. Association for Computing Machinery, New York, NY, USA, 49–54. doi:10.1145/3489517.3530408

[39] AMD Xilinx. 2023. Alveo U250 Datasheet. https://docs.amd.com/r/en-US/ds962-u200-u250/Alveo-Product-Details. Accessed: 2024-10-06.

Appendices

A Scatter Plots

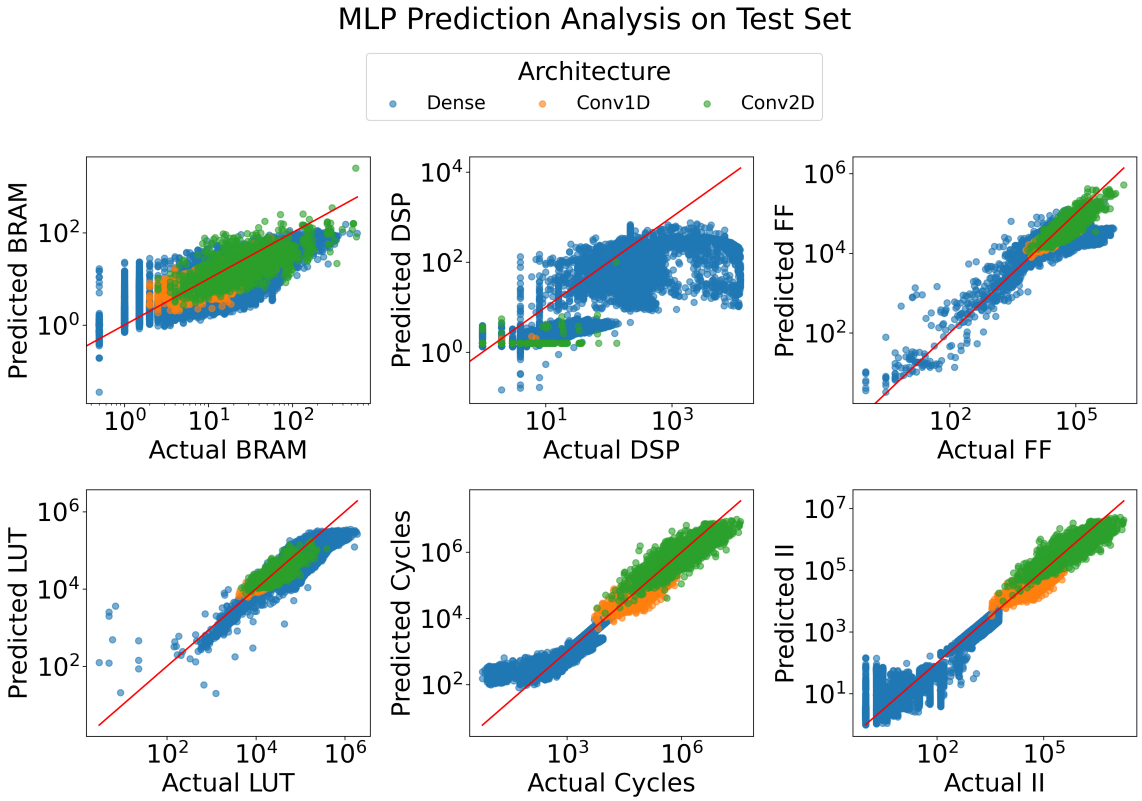

Fig. 13. Scatter plots of MLP predictions on the test set. The red line shows the deviation from true values. Both axes are set to a logarithmic scale.

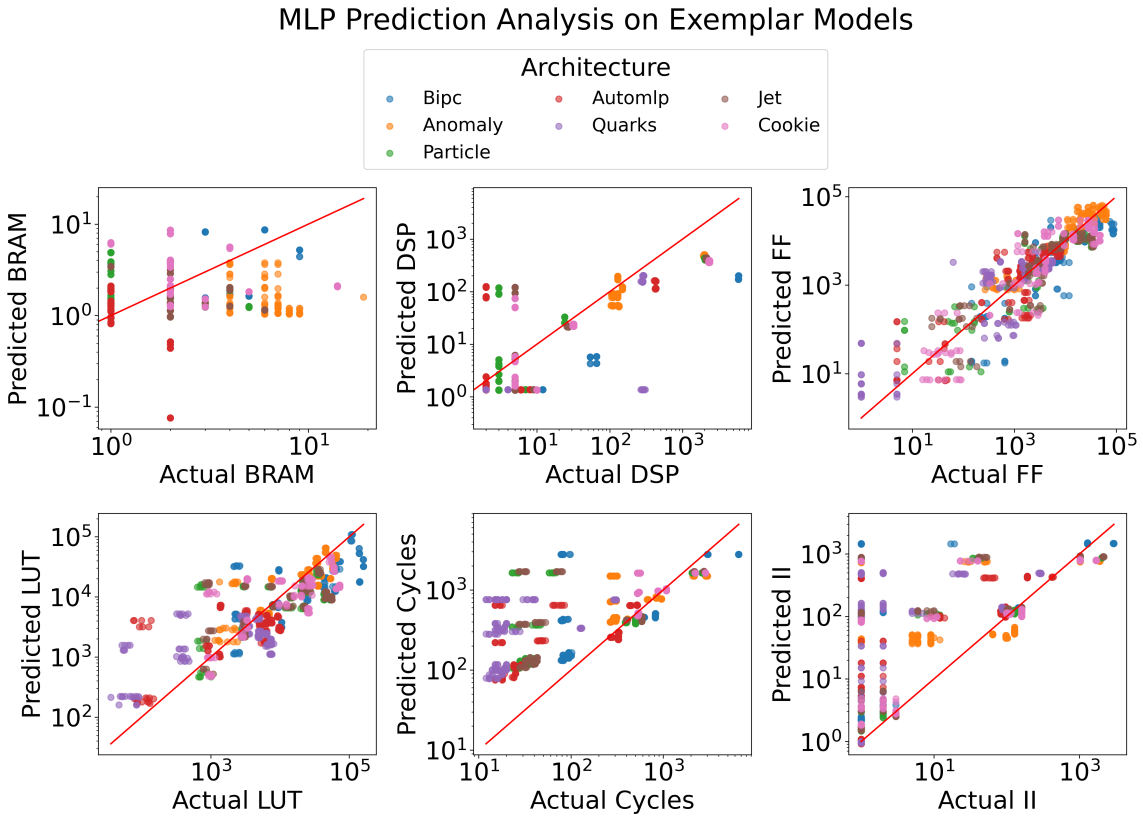

Fig. 14. Scatter plots of MLP predictions on the exemplar models. The red line shows the deviation from true values. Both axes are set to a logarithmic scale.

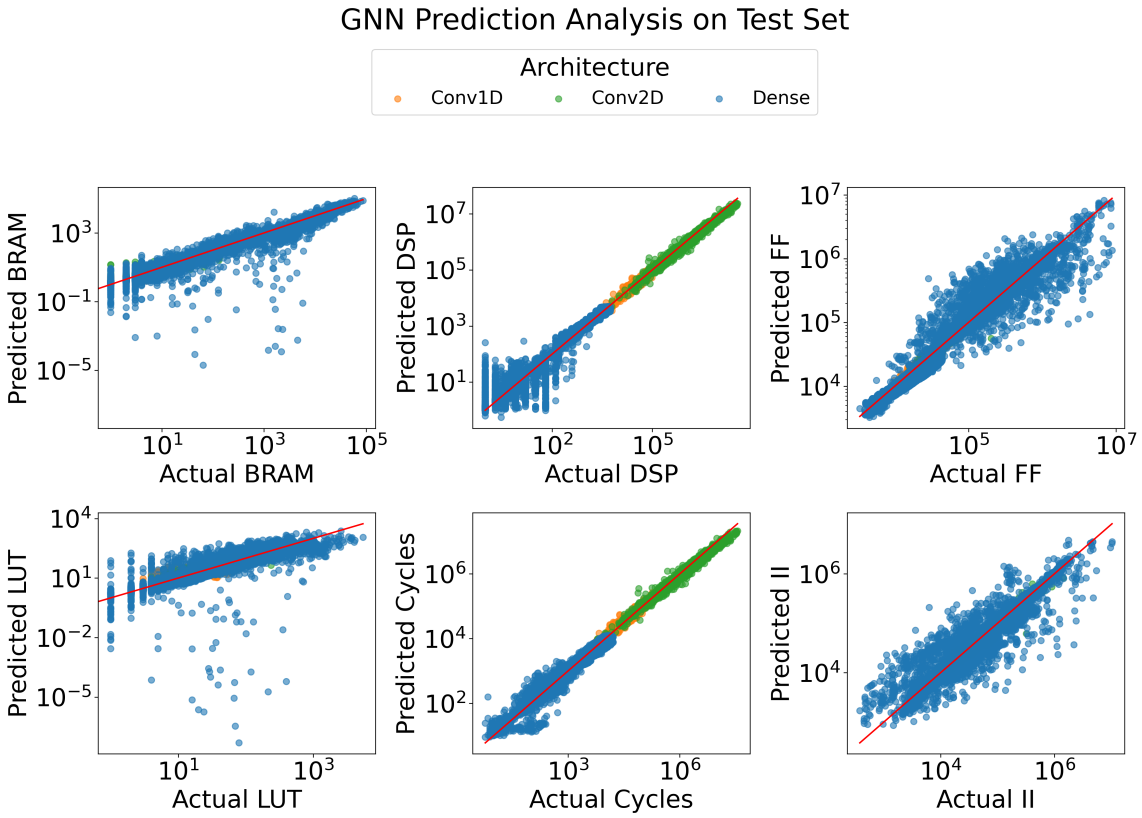

Fig. 15. Scatter plots of GNN predictions on the test set. The red line shows the deviation from true values. Both axes are set to a logarithmic scale.

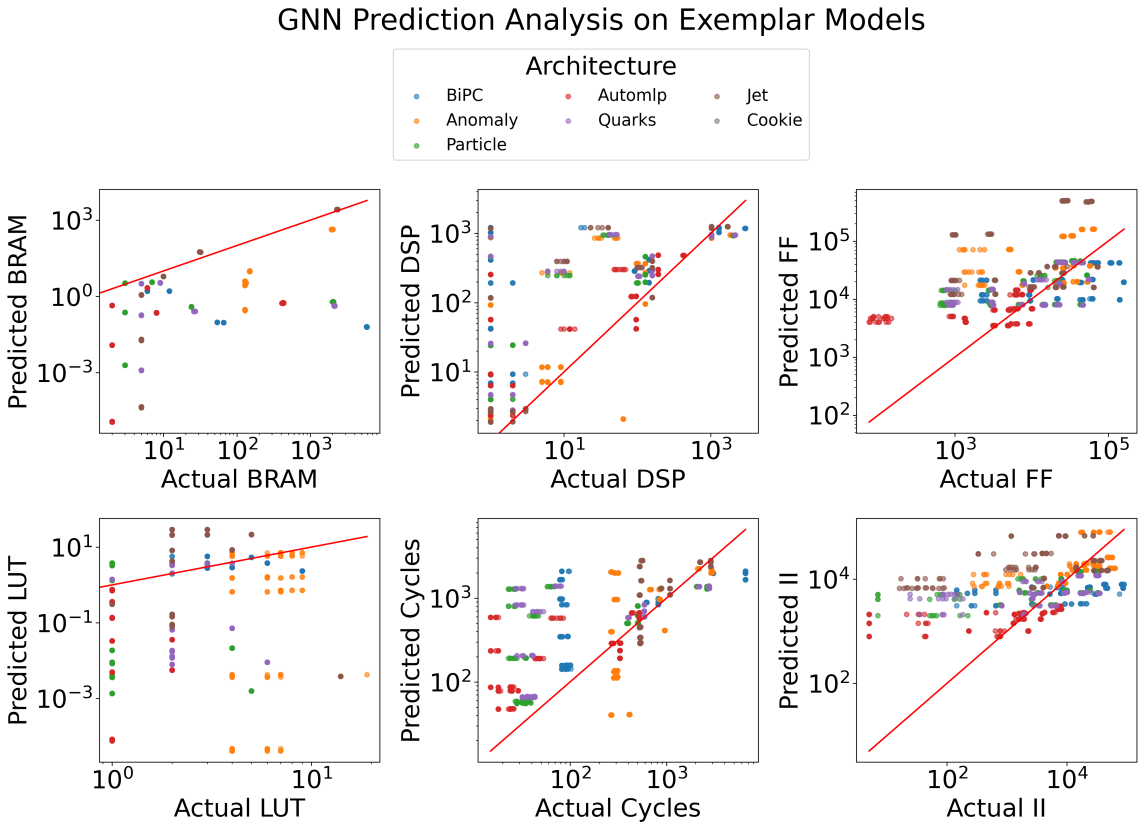

Fig. 16. Scatter plots of GNN predictions on the exemplar models. The red line shows the deviation from true values. Both axes are set to a logarithmic scale.

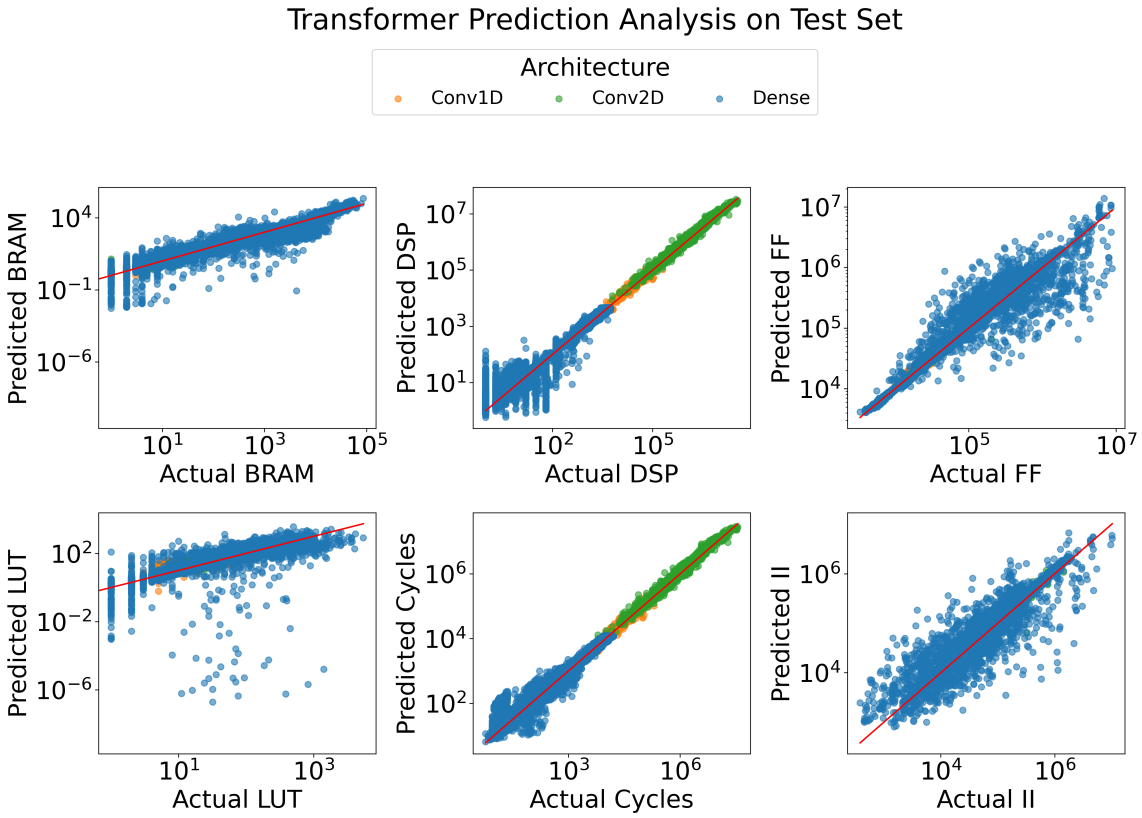

Fig. 17. Scatter plots of transformer predictions on the test set. The red line shows the deviation from true values. Both axes are set to a logarithmic scale.

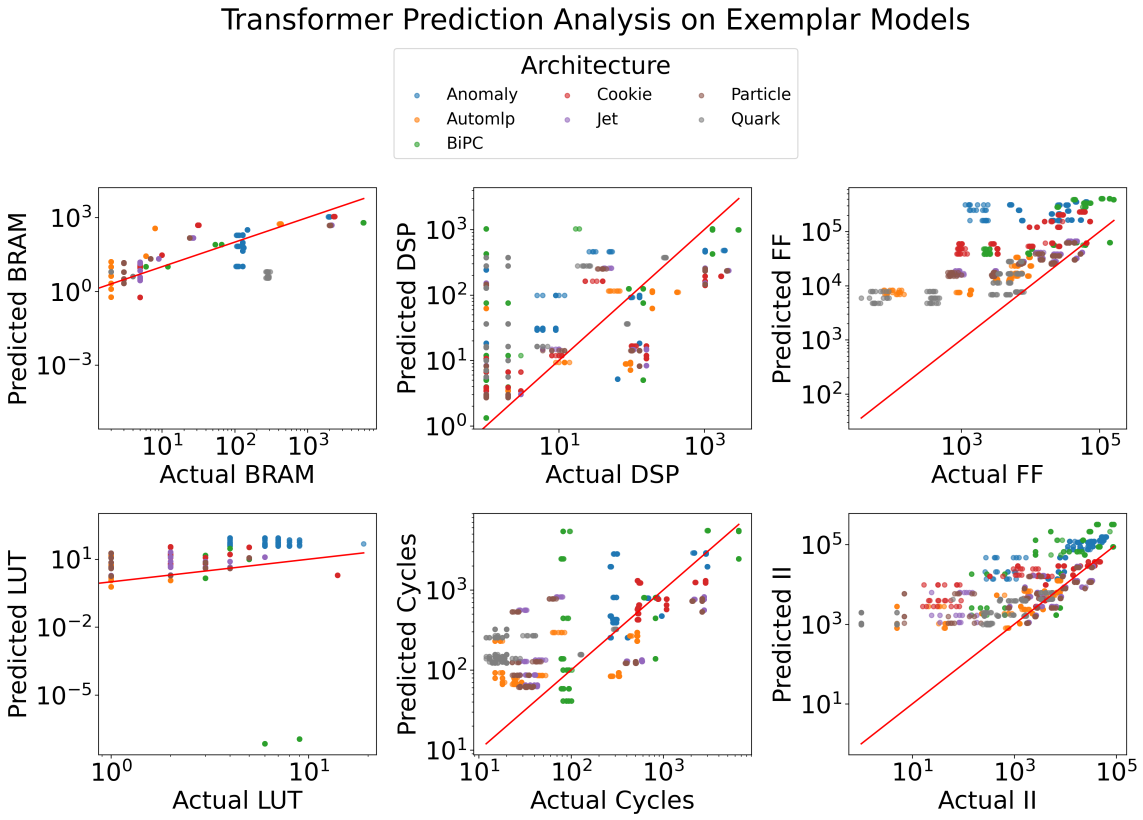

Fig. 18. Scatter plots of transformer predictions on the exemplar models. The red line shows the deviation from true values. Both axes are set to a logarithmic scale.