CLAIRE: Composable Chiplet Libraries for AI Inference

Pragnya Sudershan Nalla[1], Emad Haque[2], Yaotian Liu[2],
Sachin S. Sapatnekar[1], Jeff Zhang[2], Chaitali Chakrabarti[2], Yu Cao[1]
[1]*University of Minnesota Twin Cities*, [2]*Arizona State University*

***Abstract*—Artificial intelligence has made a significant impact on fields like computer vision, Natural Language Processing (NLP), healthcare, and robotics. However, recent AI models, such as GPT-4 and LLaMAv3, demand significant number of computational resources, pushing monolithic chips to their technological and practical limits. 2.5D chiplet-based heterogeneous architectures have been proposed to address these technological and practical limits. While chiplet optimization for models like Convolutional Neural Networks (CNNs) is well-established, scaling this approach to accommodate diverse AI inference models with different computing primitives, data volumes, and different chiplet sizes is very challenging. A set of hardened IPs and chiplet libraries optimized for a broad range of AI applications is proposed in this work. We derive the set of chiplet configurations that are composable, scalable and reusable by employing an analytical framework trained on a diverse set of AI algorithms. Testing these set of library synthesized configurations on a different set of algorithms, we achieve a $1.99\times - 3.99\times$ improvement in non-recurring engineering (NRE) chiplet design costs, with minimal performance overhead compared to custom chiplet-based ASIC designs. Similar to soft IPs for SoC development, the library of chiplets improves flexibility, reusability, and efficiency for AI hardware designs.**

***Index Terms*—Optimal Chiplet Size, 2.5D, Chiplet Libraries, Hard IP, Neural Networks, Accelerator**

I. Introduction

Machine learning applications have made significant contributions in computer vision, natural language processing (NLP), healthcare, finance, robotics, and autonomous systems. For instance, GPT-4 [1] outperforms existing models in NLP and Massive Multitask Language Understanding (MMLU) benchmarks [2] [3]. However, recent neural network architecture models are very large, and the hardware required to run them demands substantial computational power. For example, the largest LLaMAv3 model contains 405 billion parameters and handles up to 128,000 tokens. While traditional hardware architectures like CPUs may have limiting performance, GPUs, though powerful, can be energy-intensive [4].

Hence, to increase energy efficiency, systems-on-chip (SoCs) have been implemented using pre-designed and pre-verified ASIC soft Intellectual Property (IP) blocks [5] [6]. However, as neural networks have grown in size, the complexity of chip design has also increased. This has led to the "area wall" problem, where monolithic chips for larger neural networks result in reduced fabrication yields and increased manufacturing costs [7]. To tackle this issue, heterogeneous 2.5D chiplet-based architectures have been proposed [8], which improve manufacturing costs, enhance fabrication yield, and enable die-level reuse for different applications.

Developing such architectures requires determining how to partition the design into chiplets, or more critically, identifying the optimal chiplet size. Some studies have tackled this by performing generic scaling and technology-design co-exploration, taking into account ESD overhead and manufacturing costs [9]. Others have concentrated on optimal mapping and partitioning through design space exploration for specific DNN workloads [7] [10]. However, these approaches often optimize chiplet designs for a limited set of neural network algorithms, such as CNNs. Implementing different neural network models may require different optimal chiplet sizes, which can increase time-to-market if redesigning chiplets becomes necessary. Therefore, to provide the flexibility needed for deploying a wide range of neural network algorithms, it is essential to offer a design solution with a faster time-to-market compared to the traditional approach of integrating soft IPs tailored for specific algorithms.

In this work, we propose and identify library-synthesized design configurations for AI applications, aiming to deliver hardened IP chiplets that are composable, scalable, and adaptable to a large set of neural networks while maintaining performance comparable to custom ASIC designs. Our approach consists of two phases: In the training phase, we identify library-synthesized configurations (C_k) based on a training set of AI algorithms (TR), with each configuration linked to a subset of these algorithms (TR_k). In the testing phase, these configurations are evaluated on a separate set of AI algorithms, referred to as the test set (TT), where each subset of test algorithms (TT_k) is mapped to the most relevant design configuration (C_k). The key contributions of this work are:

1) Identification of library-synthesized configurations (C_k) using graph construction and design space exploration.
2) Achieving a $1.99\times$ to $3.99\times$ reduction in non-recurring engineering (NRE) costs, with a performance overhead of less than 0.2% compared to custom configurations (C_i).
3) Demonstrating a $1.6\times$ to $4\times$ improvement in chiplet utilization compared to the generic configuration (C_g).

II. Motivation and Architecture

Heterogeneous 2.5D chiplet-based architectures have been proposed to address the "area wall" problem. Similar to Soft IPs for SoC development, chiplets can be pre-designed and

This work is supported in part by COCOSYS, one of six centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. This work is also partially supported by National Science Foundation under grant CCF-2403408 and CCF-2403409.

978-3-9826741-0-0/DATE25/© 2025 EDAA

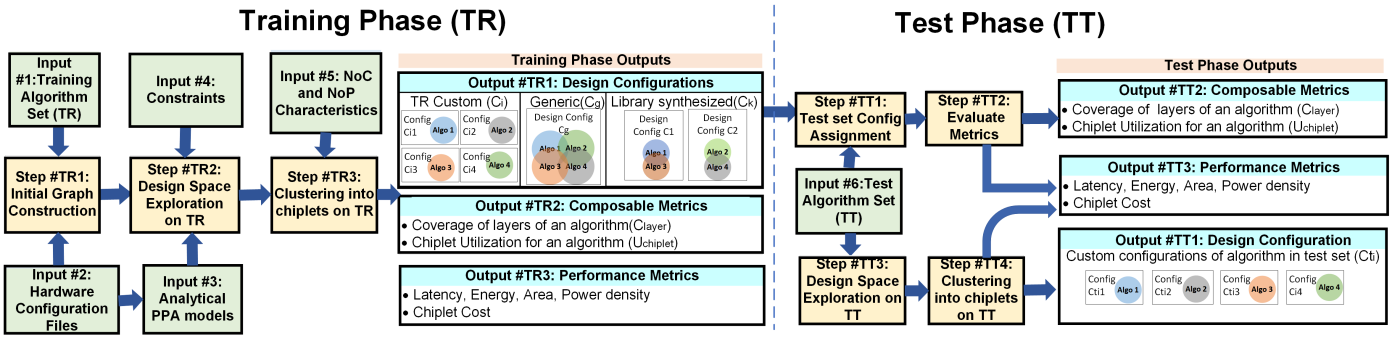

Fig. 1. Overview of the analytical framework of CLAIRE. It comprises steps: Initial Graph Construction, Design Space Exploration, Clustering, Grouping into subsets, and Evaluate Metrics. The framework integrates inputs from the Training Algorithm Set, Hardware Configuration Files, Constraints, Analytical PPA Models, NoC/NoP Characteristics, and Test Algorithm Set to determine the library synthesized configuration of chiplets.

pre-verified, reducing time-to-market. To efficiently implement diverse AI workloads it is essential to identify a set of chiplet configurations that can be easily integrated. To derive these configurations, we have developed a novel analytical framework. In this framework, algorithms are partitioned into layers and mapped into nodes and edges, where each node represents a hardware unit (e.g., systolic array, activation module) and each edge denotes connections between these units. Systolic arrays are chosen to execute convolution and linear layers due to their advantage in data reuse [11]. The nodes and edges are then used to construct a graph for different hardware configurations during the Design Space Exploration (DSE) stage. These hardware configurations are adjustable using a tunable hardware parameter file, which includes variables such as the size of the systolic array, and the number of systolic arrays, activation units, or pooling units. During this stage, performance, power, and area (PPA) metrics for each hardware unit are obtained from synthesized data at TSMC 28nm. Analytical PPA models are used to calculate the overall performance of the algorithm, and constraints are applied to find the most compact and feasible configuration. Finally, clustering is performed to partition the graphs representing the monolithic chip into chiplets. The library synthesized design configurations (C_k), derived from the training set of AI algorithms (TR), are then used in the testing phase. During testing, the most relevant configuration (C_k) is selected for a subset of algorithms in the test set (TT).

III. Analytical Framework

This section outlines the analytical framework used to identify the library-synthesized configurations of the chiplets. It details the inputs, outputs, and methodologies involved in the framework, with an overview illustrated in Figure 1.

A. Inputs

1) Input #1: Training Algorithm Set: Algorithms were selected for their relevance to AI applications and popularity. The thirteen representative algorithms in the training set (TR), along with the number of parameters, are listed in Table I.

2) Input #2: Hardware Configuration Files: Input #2 to the framework includes two categories of hardware configuration files: PPA configuration files for hardware building blocks and tunable hardware parameter files. First, we define the hardware building blocks corresponding to each torch.nn module [26] and derive their PPA metrics. For example, a systolic array is used for CONV2D module, where the processing element (PE) of the systolic array serves as the fundamental building block. The PPA values for the systolic array PE and activation functions are obtained from HISIM's synthesized data [27] at TSMC 28nm. Pooling functions are simulated with Neurosim [28], and the tanh function is derived from [29] and scaled to TSMC 28nm. The tunable hardware parameter file defines adjustable parameters for optimizing hardware unit sizes. These parameters include the systolic array size, the number of arrays, and the number of activation and pooling units. These parameters serve as the characterized components in the Design Space Exploration (DSE) process.

3) Input #3: Analytical PPA Models: Input #3 to the framework comprises analytical PPA models that serve as a bridge between hardware configuration files and the evaluation of PPA performance during the design space exploration phase. These parameterizable models, derived from HISIM [27], use configuration files from Input #2 to calculate the energy, latency, and area for each node representing a hardware unit. The resulting node-level PPA parameters are then utilized to evaluate the overall performance of the AI algorithms mapped onto specific design configurations.

4) Input #4: Constraints: To ensure realistic library-synthesized design configurations for cloud applications, several constraints are imposed. These include an upper limit on power density (PD_{limit}) to manage chip temperature and a maximum area for the chiplet ($A_{Chip_{limit}}$) based on the specifications in [30]. Additionally, a latency constraint (L_{limit})

TABLE I
AI ALGORITHMS SELECTED IN THE TRAINING SET

Algorithm	Type	# Params	Source
Resnet18 [12]	CNN	11.7 M	Torchvision [13]
VGG16 [14]	CNN	138 M	Torchvision [13]
Densenet121 [15]	CNN	7.98 M	Torchvision [13]
Mobilenetv2 [16]	CNN	3.5 M	Torchvision [13]
PEANUT RCNN [17]	RCNN	14.21 M	Torchvision [13]
Resnet50 [12]	CNN	25.5 M	Torchvision [13]
Mixtral-8x7B [18]	MoE LLM	46.7 B	HuggingFace [19]
GPT2 [20]	LLM	137 M	HuggingFace [19]
Meta Llama-3-8B [21]	LLM	8.03 B	HuggingFace [19]
DPT-Large [22]	Transformer	342 M	HuggingFace [19]
DINOv2-large [23]	Transformer	304 M	HuggingFace [19]
SWIN-T [24]	Transformer	29 M	Torchvision [13]
Whisperv3-large [25]	Transformer	1.54 B	HuggingFace [19]

is established for each algorithm, ensuring that the latency for execution on the design does not exceed 50% of the latency observed on a custom design solution.

5) Input #5: NoC & NoP Characteristics: For the Network-on-Chip (NoC) interface, 40 links per channel with 8 bits per link are selected, and for the Network-on-Chip(NoP) interface, one channel of the AIB 2.0 interface [31] is employed to ensure similar bandwidth with NoC, facilitating the analysis of NoP energy overhead. A 2D torus topology with a 5-port router was selected for the NoC/NoP. The power, performance, and area (PPA) characteristics of the NoC and NoP routers are sourced from [32], and analytical models are adapted from HISIM [27].

6) Input #6: Test Algorithm Set (TT): The library of synthesized design configurations (C_k) is evaluated using a test set of algorithms (TT) to assess their efficiency and composability. The test algorithms are BERT-base [33], Graphormer [34], ViT-base [35], AST [36], DETR [37], Alexnet [38] .

B. Outputs

1) Output #TR1 & #TT1: Design Configurations: Output #TR1 consists of three types of design configurations. These configurations are generated during the training process using the training set.

- Custom Configurations (C_i): Each training set algorithm (TR_i) is matched with a unique design configuration specifically tailored to its needs.
- Generic Configuration (C_g): A single design configuration that can implement all the algorithms in the training set, providing a generalized solution.
- Library-Synthesized Configurations (C_k): These configurations lie between the custom and generic approaches. Each subset of training set algorithms (TR_k) is assigned its library-synthesized design configuration (C_k).

Once these configurations are identified, the custom configurations (Ct_i) evaluated during the test phase form Output #TT1, as illustrated in Figure 1. The performance of these custom configurations (Ct_i) is compared against the previously obtained library-synthesized configurations (C_k) to demonstrate the efficiency and performance advantages of the proposed library-synthesized design configurations (C_k).

2) Output #TR2 & #TT2: Composable Metrics: To evaluate the composability of the design configuration, we use the metric algorithm coverage $C_{layer}(i,k)$. This metric is defined as the percentage of layers in algorithm i that can be implemented by design configuration C_k, divided by the total number of layers in algorithm i. A value of 100% for C_{layer} is required, meaning all layers of the algorithm must be implementable by the design configuration C_k. A second metric is chiplet utilization ($U_{chiplet}(i,k)$), which measures the fraction of modules utilized within the chiplets of the library-synthesized design configuration C_k when algorithm i is mapped onto it. Similarly, $U_{chiplet}(i,g)$ represents the fraction of modules utilized within the chiplets of the generic design configuration C_g for the same algorithm i. Higher chiplet utilization indicates a design configuration that is more customized and better tailored to the specific needs of the algorithm.

3) Output #TR3 & #TT3: Performance Metrics: We also incorporate key performance metrics such as latency, energy, area, and power density, which, together with the previously discussed constraints, provide a comprehensive assessment of the design configuration's performance. Additionally, we apply the non-recurring engineering (NRE) cost model outlined in [39] to highlight the cost advantage of library-synthesized configurations (C_k) over custom configurations (C_i). To conduct this comparison, the cost for each design configuration is first normalized relative to the cost of the generic design configuration (C_g). During the training phase, the normalized NRE cost (NRE_k) for each library-synthesized configuration (C_k) is estimated and compared to the cumulative normalized NRE cost ($NRE_{cstm}(k, TR_k)$) for custom designs (C_i) in the training subset (TR_k):

$$NRE_{cstm}(k, TR_k) = \sum_{i \in TR_k} NRE_i$$

In the testing phase, the normalized NRE cost (NRE_k) for a library-synthesized configuration (C_k) is compared to $NRE_{cstm}(k, TT_k)$ for the test subset (TT_k):

$$NRE_{cstm}(k, TT_k) = \sum_{i \in TT_k} NREt_i$$

where NRE_i and $NREt_i$ are the normalized NRE costs required to develop custom design configuration C_i and Ct_i.

C. Methodology

As shown in Figure 1, the methodology consists of several key stages: Initial Graph Construction (#TR1), Design Space Exploration (#TR2 & #TT3), clustering the graph to emulate a chiplet-based design (#TR3 & #TT4), assigning one of the identified library-synthesized configurations (C_k) to the test algorithms (#TT1), and finally, evaluating the composable and performance metrics (#TT2) for the algorithms in the test set when implemented on the identified configurations (C_k).

1) Step #TR1: Initial Graph Construction: First, layer information of AI models is extracted using the $print(model)$ command on models in TorchVision and Hugging Face. The main code reads this layer information file, parses it, and extracts details for each layer, including the layer type, input size (IFM_x, IFM_y), output size (OFM_x, OFM_y), input channels (N_{IFM}), output channels (N_{OFM}), as well as other layer-specific parameters such as kernel size (K_x, K_y), stride (Str), and padding (Pad). The layer types considered include Conv2d, Linear, Tanh, activation units, and pooling units. For each algorithm in the training set (TR_i), a graph $G_{ini_i}(N, E, w_N, w_E)$ is constructed. In this graph, each node (N) represents a hardware unit, such as a systolic array or a module designed to emulate pooling or activation functions, and each edge (E) represents the communication link between these hardware units. Each layer of the AI algorithm is mapped onto its corresponding node, such as convolution layer being mapped onto a systolic array. Layers are processed sequentially, employing intra-layer parallelism strategies to enhance efficiency. When there are insufficient systolic arrays (n_{SA}) available, the layer is partitioned into smaller sub-tasks that fit within

the available hardware resources, which are then executed sequentially. Convolutional layers are implemented using weight-stationary dataflow on the systolic array. Node weights (w_N) are assigned to represent the number of times the node needs to be executed to compute the entire layer, while edge weights (w_E) represent the volume of data communication between layers. The individual graphs constructed for each algorithm are then integrated into a universal graph $UG_{ini}(N, E, w_N, w_E)$, which consolidates information from all the algorithms used in the training phase. To determine the node weights, the size of each node or hardware unit is initialized using values from the tunable hardware parameter file. This initial size is selected such that the design configurations remain within a realistic area range of 10-100 mm².

2) Step #TR2 & #TT3: Design Space Exploration: The pseudo-code for Design Space Exploration (DSE) is illustrated in Algorithm 1.

Algorithm 1 Pseudo-code for Design Space Exploration

Input 1: Initial individual graphs $G_{ini_i}(N, E, w_N, w_E)$
Input 2-3: Analytical PPA models and constraints
Input 4: Training algorithm set (TR)
Input 5: Configurations within DSE scope ($DSE\ configs$)
Output 1: Custom design configuration outputs

1: **for** $model_i \in TR$ **do**
2: **for** $C_j \in DSE\ configs$ **do**
3: Update the initial graph to $G_{i_j}(N, E, w_N, w_E)$ for the design configuration C_j
4: Evaluate PPA to implement $model_i$ on hardware of configuration C_j
5: Apply the constraints and store the configuration (C_j, area) for next steps if the constraints are met.
6: **end for**
7: Identify the configuration (C_i) with the lowest area and store its individual graph $G_i(N, E, w_N, w_E)$.
8: **end for**
Output 2: Generic design configuration outputs
9: **for** $C_j \in DSE\ configs$ **do**
10: Sum the DSE results across all algorithms in the training set (TR) to calculate total area for configuration C_j.
11: **end for**
12: Identify the configuration (C_g) with the lowest area
13: Reconstruct individual & universal generic graph $UG(N, E, w_N, w_E)$ based on identified configuration C_g
Output 3: Library synthesized configuration outputs
14: Separate the algorithms into different subsets (TR_k) based on weighted Jaccard Similarity
15: **for** $TR_k \in TR$ **do**
16: Repeat steps 1 to 13 by replacing TR by TR_k wherein configuration (C_k) and corresponding graph $G_k(N, E, w_N, w_E)$ are obtained.
17: **end for**

The goal of DSE is to find the most compact configuration for all design setups. To begin, the custom design configurations are determined by constructing graphs $G_{i_j}(N, E, w_N, w_E)$ for each training algorithm (TR_i) under different DSE configurations (C_j). DSE is run on these graphs to obtain performance metrics, and design constraints are applied. For each algorithm (TR_i), the configuration (C_i) with the lowest area that satisfies the performance constraints is chosen as the custom design configuration. These graphs $G_i(N, E, w_N, w_E)$ represent a monolithic chip for the custom design configurations. A similar methodology is applied to obtain the custom design configurations (Ct_i) for the test set during the test phase. For the generic design configuration, DSE results of the individual graphs across all algorithms in the training set (TR) are summed, and the configuration with the smallest total area is selected. Based on this configuration (C_g), the individual and universal graphs $UG(N, E, w_N, w_E)$ are re-constructed, where the universal graph represents a monolithic chip for the generic design configuration. For the library synthesized design configurations, the training algorithm set (TR) is divided into subsets (TR_k) based on weighted Jaccard Similarity [40] between algorithm nodes, which quantifies the similarity between two algorithms by comparing the ratio of the intersection of their nodes to the union of their nodes. The most compact configuration for each subset (TR_k) becomes the library synthesized design configuration (C_k). The steps performed for the generic configuration are repeated here, with the key difference that C_g is designed to meet the needs of all algorithms in TR, while C_k is tailored for a subset TR_k. The resulting graphs $G_k(N, E, w_N, w_E)$ represent a monolithic chip for the library synthesized design configurations.

3) Step #TR3 & #TT4: Clustering Into Chiplets: To convert the graphs obtained in steps #TR2 (training phase) and #TT3 (testing phase) into a chiplet-based system, the graphs are partitioned into clusters, where each cluster represents a chiplet. The clustering is performed using the Louvain clustering algorithm [41] The clustering algorithm groups nodes based on edge weights, grouping frequently communicating nodes together and placing nodes with low inter-node communication in different chiplets to reduce NoP communication energy overhead. The PPA performance of the design configurations is updated by applying NoP characteristics for inter-chiplet communication and NoC characteristics for intra-chiplet communication. In this step, the NRE chiplet cost (NRE_i), algorithm coverage ($C_{layer}(i, k)$), and chiplet utilization ($U_{chiplet}(i, k)$) are estimated for the design configurations. During the training phase, the normalized NRE cost (NRE_k) is calculated for each library-synthesized configuration (C_k) and compared with the cumulative normalized cost $NRE_{cstm}(k, TR_k)$. For the test phase, only the cumulative normalized cost $NRE_{cstm}(k, TT_k)$ is calculated.

4) Step #TT1: Test Set Configuration Assignment: In this step, the most suitable library-synthesized configuration (C_k) for each test algorithm ($model_i$) is identified by calculating the weighted Jaccard Similarity between the algorithm's nodes and the nodes of the library-synthesized configurations (C_k). The configuration with the highest similarity is selected for the algorithm.

5) Step #TT2: Evaluate Metrics: In this step, we extract performance metrics, including energy, latency, area, and power density, alongside composable metrics such as NRE chiplet cost

TABLE II
DESIGN SPECIFICATIONS OF CHIPLET LIBRARIES IDENTIFIED THROUGH LIBRARY-SYNTHESIZED DESIGN CONFIGURATIONS (C_k).

Chiplet Library	Systolic Array Size	#Systolic Arrays	Activation Units Type	#Activation Units	Pooling Units Type	#Pooling Units	FLATTEN	PERMUTE
L1	32×32	32	RELU, RELU6	16	MAXPOOL, AVGPOOL	16	No	No
L2	32×32	32	GELU	16	ADAPTIVEAVGPOOL	16	Yes	Yes
L3	32×32	32	RELU, GELU, SILU	16	None	-	No	No
L4	32×32	32	None	-	LASTLEVELMAXPOOL	16	No	No
L5	32×32	64	RELU	16	ROIALIGN	16	Yes	No
L6	32×32	64	None	-	None	-	No	No
L7	32×32	32	GELU	16	None	-	No	No

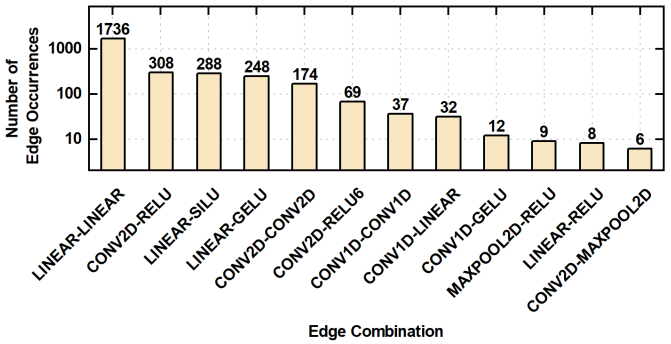

Fig. 2. Number of edge occurrences for edge combinations/layer connections in the training set algorithms.

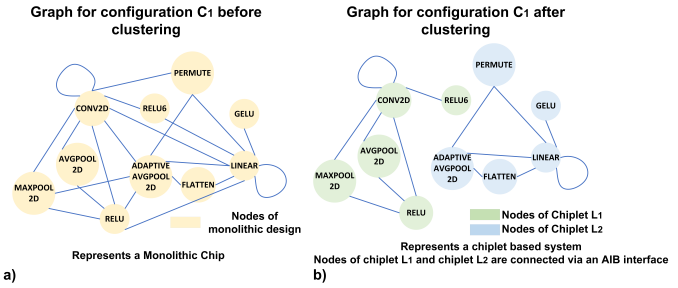

Fig. 3. Intermediary graphs for the identified library-synthesized design configuration C_1 a) before clustering, and b) after clustering

(NRE_i), algorithm coverage ($C_{layer}(i,k)$), chiplet utilization ($U_{chiplet}(i,k)$) for the library-synthesized design configurations, and chiplet utilization ($U_{chiplet}(i,g)$) for the generic design configuration. Subsequently, we compute the normalized NRE cost (NRE_k) for each library-synthesized configuration (C_k) employed by the algorithms in the test set.

IV. CASE STUDY

In this section, we present the computing profile and intermediary outputs for one of the identified library-synthesized configurations, C_1, as a case study. We begin by outlining the computing profile of the training set, focusing on common layer connections (edges) shared across the training algorithm set (TR). Figure 2 illustrates the top 12 most frequent edge combinations. The LINEAR-LINEAR connection is the most dominant, largely due to the Q, K, V operations in Transformer-based algorithms. Next is the CONV2D-RELU connection, which is prevalent due to its frequent use in CNNs. These observations suggest that by grouping algorithms based on the commonality of their layer types and connections, we can develop efficient library-synthesized configurations for specific classes of algorithms. Following this, we present the intermediary graphs for configuration C_1 before and after clustering, as shown in Figure 3. The graph before clustering represents the monolithic chip design, while the graph after clustering illustrates the chiplet-based system. The monolithic chip 3a is partitioned into two chiplets during the clustering step (Step #TR3), with the resulting chiplets (Chiplet L_1 and Chiplet L_2) shown in Figure 3b. The nodes of chiplets L_1 and L_2 are connected via an AIB Interface.

TABLE III
IDENTIFIED LIBRARY-SYNTHESIZED DESIGN CONFIGURATIONS (C_k) AND THEIR CORRESPONDING ALGORITHM SUBSETS

Config (C_k)	Training Algorithm Subsets (TR_k)	Test Algorithm Subsets (TT_k)
C_1	VGG16, Mobilenet,Densenet, Resnet50, SWIN-T, Resnet18	DETR, Alexnet
C_2	PEANUT RCNN	No test set algorithm assigned
C_3	DPT, DINOv2, Mixtral, LLama3	BERT, Graphormer, ViT, AST
C_4	Whisperv3	No test set algorithm assigned
C_5	GPT2	No test set algorithm assigned

V. RESULTS

A. Training Phase

During the training phase, custom (C_i), generic (C_g), and library-synthesized design configurations (C_k) were identified, with a total convergence time of eight minutes. The DSE run encompassed 81 configurations. In total, five different library-synthesized design configurations (C_k) were selected, with their corresponding algorithm subsets detailed in Table III. The chiplet libraries inside these configurations are listed in Table II. Each training set algorithm achieved full algorithm coverage ($C_{layer} = 100\%$) when mapped onto its designated design configuration. However, chiplet utilization ($U_{chiplet}$) varied, with custom design configurations achieving full utilization ($U_{chiplet} = 1$), but progressively lower utilization was observed as the designs transitioned from custom to library-synthesized and then to generic configurations. The performance metrics collected during the training phase for different design configurations are compared in Figure 4. The generic design configuration's area was strongly influenced by the PEANUT-RCNN algorithm, which has the most diverse set of layer types among the algorithms in the training set.

TABLE IV
NRE COSTS OF IDENTIFIED LIBRARY-SYNTHESIZED CONFIGURATIONS (NRE_k) AND CUSTOM CONFIGURATIONS ($NRE_{cstm}(k, TR_k)$)

Config (C_k)	Training Algorithm Subsets (TR_k)	$NRE_{cstm}(k, TR_k)$	NRE_k	Cost benefit
C_1	VGG16,Mobilenet,Densenet Resnets, SWIN-T	2.998	0.5	5.99×
C_3	DPT, DINOv2, Mixtral, LLama3	0.999	0.25	3.99 ×

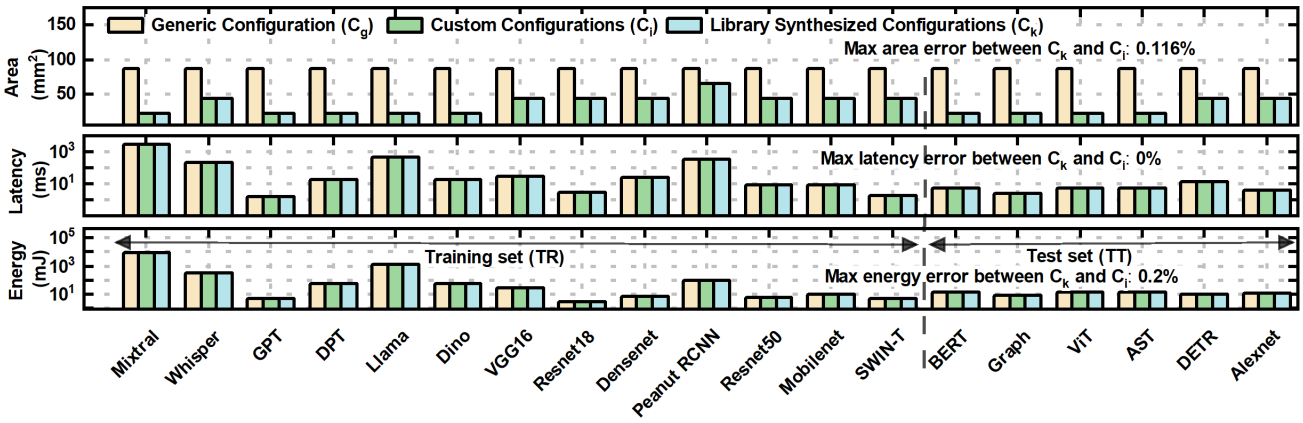

Fig. 4. Area, latency, and energy comparison of generic configuration (C_g), custom configurations (C_i), and library-synthesized configurations (C_k)

TABLE V
CHIPLET UTILIZATION WHEN TEST SET ALGORITHMS ARE DEPLOYED ON THE GENERIC CONFIGURATION (C_g) AND ON IDENTIFIED LIBRARY-SYNTHESIZED DESIGN CONFIGURATIONS (C_k)

Test Algorithm (TT_i)	$U_{chiplet}(i,g)$	Config (C_k)	$U_{chiplet}(i,k)$
BERT-base	0.188	C_3	0.75
Graphormer	0.125		0.5
ViT-base	0.188		0.75
AST	0.125		0.5
DETR	0.25	C_1	0.4
Alexnet	0.31		0.5

As expected, custom configurations had smaller areas than the generic design since the generic configuration accommodates all training algorithms and thus includes additional hardware modules. Interestingly, the area of the library-synthesized configurations deviated by only 0.116% from that of the custom configuration for any algorithm, a result made possible by effectively partitioning the training set into subsets (TR_k) using weighted Jaccard Similarity. In addition, the NoC interface and NoP interface (AIB 2.0) were configured to maintain equal bandwidth across networks, resulting in comparable latency values across all design configurations. Since power gating for underutilized units was not applied, the energy consumption varied by only 0.2% across the configurations. The cumulative NRE cost $NRE_{cstm}(k, TR_k)$ for the custom configurations of algorithms in TR_k was compared against the normalized NRE cost (NRE_k) for the library-synthesized configurations (C_k). As listed in Table IV, It was observed that library-synthesized configurations C_1 and C_3 exhibited significant cost reductions, with $5.99\times$ and $3.99\times$ lower NRE costs, respectively, compared to the cumulative NRE costs for custom configurations in the training phase.

B. Evaluation on Test Set

During step #TT1 in the test phase, the library-synthesized design configurations (C_k) selected for each algorithm are listed in Table III. For the algorithms in the test set, the algorithm coverage (C_{layer}) for these configurations is (100%), as required. Additionally, chiplet utilization is evaluated for both generic and library-synthesized design configurations and is summarized in Table V. The chiplet utilization for algorithms mapped to configuration C_3 demonstrates a $4\times$ improvement in chiplet utilization compared to the generic design configuration (C_g).

TABLE VI
NRE COSTS OF IDENTIFIED LIBRARY-SYNTHESIZED CONFIGURATIONS (NRE_k) AND CUSTOM CONFIGURATIONS ($NRE_{cstm}(k, TT_k)$)

Config (C_k)	Test Algorithm Subsets (TT_k)	$NRE_{cstm}(k, TT_k)$	NRE_k	NRE cost benefit
C_1	DETR, Alexnet	0.999	0.5	$1.99\times$
C_3	BERT, Graphormer, ViT, AST	0.999	0.25	$3.99\times$

This highlights the importance of using library-synthesized configurations tailored to subsets of algorithms rather than relying on a single generic configuration to enhance utilization. Additionally, in the test phase, the library-synthesized configurations (C_k) demonstrated a NRE cost benefit of $1.99\times$ to $3.99\times$ over custom design configurations, as depicted in Table VI. Furthermore, the performance metrics of the test set algorithms are evaluated when deployed on their respective library-synthesized configurations (C_k), and compared against the generic configuration (C_g) and custom configurations (C_i). These results are plotted in Figure 4. As shown, the performance of the library-synthesized configurations (C_k) closely matches that of the custom design configurations, with a maximum deviation of only 0.116%. The new models, such as GPT2 and Whisper, use a 1D convolution module, differing from traditional architectures, and are grouped separately. A comprehensive algorithm test set with similar architectures will address the unassigned cases in Table III.

VI. CONCLUSION

In this work, we successfully identified library-synthesized configurations by employing graph construction and design space exploration using a training set of AI algorithms. When tested on a separate set of test algorithms, the proposed configurations achieved a $1.99\times$ to $3.99\times$ reduction in non-recurring engineering (NRE) costs and provided 100% algorithm coverage, with performance overhead of less than 0.2% compared to custom ASIC designs. Additionally, these configurations demonstrated a $1.6\times$ to $4\times$ increase in chiplet utilization compared to generic configurations during testing. Thus, this solution offers a flexible and efficient solution for AI chip development, benefiting both system designers and AI algorithm developers.

REFERENCES

[1] OpenAI, "GPT-4 Technical Report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[2] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring Massive Multitask Language Understanding," 2021. [Online]. Available: https://arxiv.org/abs/2009.03300

[3] D. Hendrycks, C. Burns, S. Basart, A. Critch, J. Li, D. Song, and J. Steinhardt, "Aligning AI With Shared Human Values," 2023. [Online]. Available: https://arxiv.org/abs/2008.02275

[4] A. Boutros, S. Yazdanshenas, and V. Betz, "You Cannot Improve What You Do not Measure: FPGA vs. ASIC Efficiency Gaps for Convolutional Neural Network Inference," *ACM Trans. Reconfigurable Technol. Syst.*, 2018. [Online]. Available: https://doi.org/10.1145/3242898

[5] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. Pande, C. Grecu, and A. Ivanov, "System-on-Chip: Reuse and Integration," *Proc. IEEE*, 2006. [Online]. Available: https://doi.org/10.1109/JPROC.2006.873611

[6] H. Liao, J. Tu, J. Xia, and X. Zhou, "DaVinci: A Scalable Architecture for Neural Network Computing," in *IEEE Hot Chips 31 Symp. (HCS)*, 2019. [Online]. Available: https://doi.org/10.1109/HOTCHIPS.2019.8875654

[7] Z. Tan, H. Cai, R. Dong, and K. Ma, "NN-Baton: DNN Workload Orchestration and Chiplet Granularity Exploration for Multichip Accelerators," in *ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2021. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00083

[8] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture," in *IEEE/ACM 52nd Annu. Int. Symp. Microarch. (MICRO)*, 2019. [Online]. Available: https://doi.org/10.1145/3352460.3358302

[9] A. Graening, S. Pal, and P. Gupta, "Chiplets: How Small is too Small?" in *ACM/IEEE 60th Annu. Des. Autom. Conf. (DAC)*, 2023. [Online]. Available: https://doi.org/10.1109/DAC56929.2023.10247947

[10] X. Hao, Z. Ding, J. Yin, Y. Wang, and Y. Liang, "Monad: Towards Cost-Effective Specialization for Chiplet-Based Spatial Accelerators," in *IEEE/ACM Int. Conf. Comput. Aided Des. (ICCAD)*, 2023. [Online]. Available: https://doi.org/10.1109/ICCAD57390.2023.10323880

[11] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE J. Solid-State Circuits*, 2017. [Online]. Available: https://doi.org/10.1109/JSSC.2016.2616357

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[13] T. maintainers and contributors, "TorchVision: PyTorch's Computer Vision Library," https://github.com/pytorch/vision, 2016.

[14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[15] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," 2018. [Online]. Available: https://arxiv.org/abs/1608.06993

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2019. [Online]. Available: https://arxiv.org/abs/1801.04381

[17] A. J. Zhai and S. Wang, "PEANUT: Predicting and Navigating to Unseen Targets," 2022. [Online]. Available: https://arxiv.org/abs/2212.02497

[18] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mixtral of Experts," 2024. [Online]. Available: https://arxiv.org/abs/2401.04088

[19] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "HuggingFace's Transformers: State-of-the-art Natural Language Processing," 2020. [Online]. Available: https://arxiv.org/abs/1910.03771

[20] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Language Models Are Unsupervised Multitask Learners," *OpenAI Blog*, 2019. [Online]. Available: https://openai.com/blog/language-unsupervised

[21] AI@Meta, "Llama 3 Model Card," 2024. [Online]. Available: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

[22] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision Transformers for Dense Prediction," 2021. [Online]. Available: https://arxiv.org/abs/2103.13413

[23] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "DINOv2: Learning Robust Visual Features without Supervision," 2024. [Online]. Available: https://arxiv.org/abs/2304.07193

[24] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," 2021. [Online]. Available: https://arxiv.org/abs/2103.14030

[25] Alec Radford and Jong Wook Kim and Tao Xu and Greg Brockman and Christine McLeavey and Ilya Sutskever, "Robust Speech Recognition via Large-Scale Weak Supervision," 2022. [Online]. Available: https://arxiv.org/abs/2212.04356

[26] Pytorch, "Torch.nn Modules Technical Documentation," 2023, https://pytorch.org/docs/stable/nn.html.

[27] Z. Wang, J. Sun, A. Goksoy, S. K. Mandal, Y. Liu, J.-S. Seo, C. Chakrabarti, U. Y. Ogras, V. Chhabria, J. Zhang, and Y. Cao, "Exploiting 2.5D/3D Heterogeneous Integration for AI Computing," in *29th Asia South Pacific Des. Autom. Conf. (ASP-DAC)*, 2024. [Online]. Available: https://doi.org/10.1109/ASP-DAC58780.2024.10473875

[28] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2018. [Online]. Available: https://doi.org/10.1109/TCAD.2018.2789723

[29] V.-T. Nguyen, T.-K. Luong, H. Le Duc, and V.-P. Hoang, "An Efficient Hardware Implementation of Activation Functions Using Stochastic Computing for Deep Neural Networks," in *IEEE 12th Int. Symp. Embed. Multicore/Many-core Syst.-on-Chip (MCSoC)*, 2018. [Online]. Available: https://doi.org/10.1109/MCSoC2018.2018.00045

[30] I. Magaki, M. Khazraee, L. V. Gutierrez, and M. B. Taylor, "ASIC Clouds: Specializing the Datacenter," in *ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016. [Online]. Available: https://doi.org/10.1109/ISCA.2016.25

[31] Intel, "AIB 2.0 Technical Specification," 2019, https://github.com/chipsalliance/AIB-specification.

[32] P. Vivet, Y. Thonnart, R. Lemaire, C. Santos, E. Beigné, C. Bernard, F. Darve, D. Lattard, I. Miro-Panadès, D. Dutoit, F. Clermidy, S. Cheramy, A. Sheibanyrad, F. Pétrot, E. Flamand, J. Michailos, A. Arriordaz, L. Wang, and J. Schloeffel, "A $4 \times 4 \times 2$ Homogeneous Scalable 3D Network-on-Chip Circuit With 326 MFlit/s 0.66 pJ/b Robust and Fault Tolerant Asynchronous 3D Links," *IEEE J. Solid-State Circuits*, 2017. [Online]. Available: https://doi.org/10.1109/JSSC.2016.2611497

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[34] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do Transformers Really Perform Bad for Graph Representation?" 2021. [Online]. Available: https://arxiv.org/abs/2106.05234

[35] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual Transformers: Token-based Image Representation and Processing for Computer Vision," 2020. [Online]. Available: https://arxiv.org/abs/2006.03677

[36] Y. Gong, Y.-A. Chung, and J. Glass, "AST: Audio Spectrogram Transformer," 2021. [Online]. Available: https://arxiv.org/abs/2104.01778

[37] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," 2020. [Online]. Available: https://arxiv.org/abs/2005.12872

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, 2017. [Online]. Available: https://doi.org/10.1145/3065386

[39] Y. Feng and K. Ma, "Chiplet Actuary: A Quantitative Cost Model and Multi-Chiplet Architecture Exploration," in *Proc. 59th ACM/IEEE Des. Autom. Conf.*, 2022. [Online]. Available: https://doi.org/10.1145/3489517.3530428

[40] P. Jaccard, "Etude de la distribution florale dans une portion des Alpes et du Jura," *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 1901. [Online]. Available: https://doi.org/10.5169/seals-266450

[41] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," 2008. [Online]. Available: https://arxiv.org/abs/0803.0476