Explainable-DSE: An Agile and Explainable Exploration of Efficient HW/SW Codesigns of Deep Learning Accelerators Using Bottleneck Analysis

Shail Dave
Arizona State University
Tempe, AZ, USA
Shail.Dave@asu.edu

Tony Nowatzki
University of California, Los Angeles
Los Angeles, CA, USA
tjn@cs.ucla.edu

Aviral Shrivastava
Arizona State University
Tempe, AZ, USA
Aviral.Shrivastava@asu.edu

ABSTRACT

Effective design space exploration (DSE) is paramount for hardware/software codesigns of deep learning accelerators that must meet strict execution constraints. For their vast search space, existing DSE techniques can require excessive trials to obtain a valid and efficient solution because they rely on black-box explorations that do not reason about design inefficiencies. In this paper, we propose Explainable-DSE – a framework for the DSE of accelerator codesigns using bottleneck analysis. By leveraging information about execution costs from bottleneck models, our DSE is able to identify bottlenecks and reason about design inefficiencies, thereby making bottleneck-mitigating acquisitions in further explorations. We describe the construction of bottleneck models for DNN accelerators. We also propose an API for expressing domain-specific bottleneck models and interfacing them with the DSE framework. Acquisitions of our DSE systematically cater to multiple bottlenecks that arise in executions of multi-functional workloads or multiple workloads with diverse execution characteristics. Evaluations for recent computer vision and language models show that Explainable-DSE mostly explores effectual candidates, achieving codesigns of 6× lower latency in 47× fewer iterations vs. non-explainable DSEs using evolutionary or ML-based optimizations. By taking minutes or tens of iterations, it enables opportunities for runtime DSEs.

CCS CONCEPTS

• **Hardware** → *Hardware accelerators*; *Electronic design automation*; • **Computer systems organization** → *Special purpose systems*; • **General and reference** → **Design**; • **Computing methodologies** → **Modeling methodologies**; • **Software and its engineering** → *Abstraction, modeling and modularity*.

KEYWORDS

design space exploration, domain-specific architectures, gray-box optimization, bottleneck model, hardware/software codesign, explainability, machine learning and systems

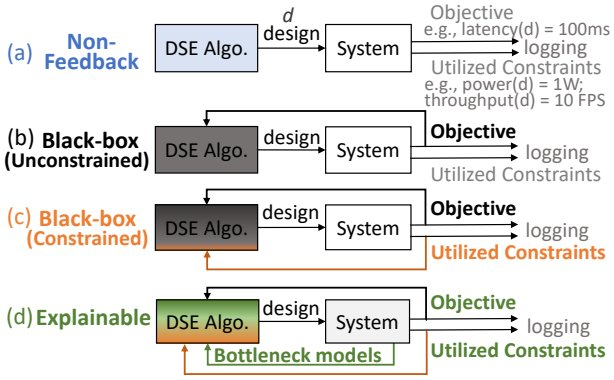

Figure 1: DSE with (a) Non-feedback, (b) Unconstrained black-box, (c) Constrained black-box, and (d) Explainable optimization, which leverages domain-specific bottleneck models.

ACM Reference Format:
Shail Dave, Tony Nowatzki, and Aviral Shrivastava. 2023. Explainable-DSE: An Agile and Explainable Exploration of Efficient HW/SW Codesigns of Deep Learning Accelerators Using Bottleneck Analysis. In *28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3623278.3624772

1 INTRODUCTION

Domain-specific accelerators, e.g., for deep learning models, are deployed from datacenters to edge. In order to meet strict constraints on execution costs (e.g., power and area) while minimizing an objective (e.g., latency), their hardware/software codesigns must be effectively explored using an effective *design space exploration* (DSE). However, the search space is vast (e.g., contain $O(10^{29})$ solutions), with each evaluation taking milliseconds or even hours [86]. For instance, [86] showed that a TPU-like architecture has 10^{14} hardware solutions with modest options for design parameters. For every hardware configuration, software space can also be huge. For example, DNN layers can be mapped on a spatial architecture in $O(10^{15})$ ways aka dataflows ([18] and Table 7), even after aggressively pruning the mapping space. Clearly, an effective exploration is needed to achieve feasible and efficient solutions quickly. [1]

Recent DSE techniques for deep learning accelerators use non-feedback or feedback-based, black-box optimizations. *Non-feedback*

[1] A feasible solution meets all constraints, and its hardware and software configurations are compatible; An efficient solution minimizes objective; Agility refers to DSE's ability to find desired solutions quickly, which becomes crucial for exploring vast space in practical DSE budgets and runtime DSEs.

ASPLOS '23, March 25–29, 2023, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0394-2/23/03…$15.00
https://doi.org/10.1145/3623278.3624772

optimizations include grid search (in [32, 57]) and random search (in [41, 53]). They evaluate different solutions for a pre-set number of iterations and terminate (Fig. 1a). *Black-box* optimizations, on the other hand, consider value of the objective before acquiring[2] the next candidates (Fig. 1b-c). Thus, they can be more effective than non-feedback approaches. They include simulated annealing [74], genetic algorithm [68, 76, 85], Bayesian optimization [34, 48, 51, 54, 58, 81, 86], and reinforcement learning [10, 36, 79, 82, 89]. These optimizations can be unconstrained or constrained.

For vast accelerator hardware/software codesign space, existing techniques require *excessive trials* for convergence or even finding a feasible solution. We argue that this is because of lack of explainability during the exploration. **By *explainability*, we imply the ability of the DSE to reason about, at each attempt, why a certain design corresponds to specific costs, and what are the underlying inefficiencies, and how they can be ameliorated.** Existing DSE approaches are non-explainable, as they lack information and reasoning about the quality of designs acquired during DSE. They may be able to figure out which of the previous trials reduced the objective but they cannot determine *why*? Consequently, in deciding the acquisition targets for the next trial, they cannot reason about and estimate the quality of the next possible candidates. In contrast, an explainable DSE would identify inefficiencies in the acquired design that incur high costs and also suggest mitigation options that would improve designs and execution further. For instance, for reducing the latency of a DNN accelerator, an explainable DSE could reason that the latency is dominated by memory access time that cannot be hidden behind the time for computation or communicating data on-chip. Therefore, it could strive to reduce the latency further by increasing off-chip bandwidth or on-chip buffer size to further exploit the available data reuse.

The goal of this paper is to develop a framework for an agile and effective DSE of hardware/software codesigns of accelerators by introducing explainablity in the DSE process.

Our approach to achieve explainable DSE is through the use of bottleneck analysis. Enabling explainability in DSE with bottleneck analysis requires ***bottleneck models***. Conventional DSEs evaluate *cost models* that provide just a single value like latency. In contrast, a bottleneck model is a graphical representation of which and how various design parameters and intermediate factors contribute to the total cost. For instance, a toy example tree in Fig. 2 illustrates how the time for computation, memory accesses, and NoC communication are intermediate factors derived from hardware/software parameters, leading to total latency. Thus, bottleneck models can provide rich information in an explicitly analyzable format. Bottleneck models can also help find mitigation, i.e., when any factor (on-chip communication) gets identified as a bottleneck, how to tune different design parameters based on the workload execution-related characteristics (e.g., increase bit-widths of NoCs by certain amount or increase physical links or time-shared unicast support). These bottleneck models can be developed based on domain-specific information, which is often embedded within experts-defined, domain-specific cost models (like [15, 44]) but *implicitly*. Having the *explicitly* analyzable bottleneck models and their driving the DSE can help DSE explain inefficiencies of acquired designs (referred to as *bottleneck analysis*) and to make mitigating acquisition decisions.

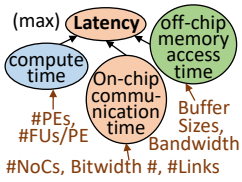

Figure 2: Example bottleneck model of DNN accelerator latency.

For enabling DSE of deep learning accelerators using bottleneck analysis, our approach overcomes the following shortcomings.

1) We develop a bottleneck model for deep learning accelerators. Taking latency minimization as an example, we describe what execution characteristics of DNN accelerators need to be leveraged, how to construct a corresponding bottleneck model, how its bottleneck graph provides insights in design/execution inefficiencies, how to pinpoint bottlenecks, and what are mitigation options for identified bottlenecks. By applying bottleneck analysis on software-optimized executions of each hardware design, our DSE co-explores both hardware-software configurations of DNN accelerators in adaptive and tightly coupled manner.

2) We propose API for specifying domain-specific bottleneck models and interfacing them with the DSE. Through proposed API, bottleneck model of an architecture/system can be described as a tree corresponding to the target cost. Navigating such tree enables the DSE to analyze the bottlenecks, relate the bottlenecks with the design parameters, and reason about the desired scaling for mitigation. For instance, by parsing a latency tree (Fig. 2), the DSE could reason that latency is a maximum value of the time taken for computations, on-chip communications, and memory accesses; if computational time exceed other factors by 3×, then the related parameters (number of functional units in PEs and number of PEs) may need to be scaled next accordingly.

The API can allow expert designers to systematically express their domain-specific bottleneck models and integrate them in DSE while leveraging constrained exploration framework. This helps overcome a limitation of previous DSEs using bottleneck analysis in other domains like multimedia or FPGA-HLS [26, 29, 67, 84] which lack such interface; as search mechanisms were defined in domain-specific ways for their bottleneck models, they could not be decoupled or reused for other domains.

3) We propose a generic framework for constrained DSE using bottleneck models, with acquisitions accounting for multiple bottlenecks in multi-workload executions. Previous DSEs using bottleneck analysis optimize only a single task at a time, i.e., consider a single cost value of executing a loop-kernel or a whole task and iteratively mitigate its bottleneck. However, when workloads involve different functions of diverse execution characteristics, e.g., a DNN with multiple layers or multiple DNNs, changing a design parameter impacts their contribution to overall cost in distinct ways; considering just a total cost could not be useful. Also, mitigation strategies to address layer-wise bottlenecks can lead to range of different values for diverse parameters. So, our framework systematically aggregates parameters predicted for mitigating bottlenecks in executions of multiple functions in one or more workloads, for making next acquisitions. Lastly, the DSE exploits awareness about constraints utilization, striving to explore among feasible solutions in the vast space and finding more efficient solutions, without quickly exhausting the constraints.

[2] Acquisition refers to a step in a DSE algorithm that selects next set of candidate designs to evaluate. §A.1 discusses the terminology for the DSE techniques.

Results: We demonstrate our explainable and agile DSE framework by exploring high-performance edge inference accelerators for recent computer vision and language processing models. By iteratively mitigating bottlenecks, Explainable-DSE reduces latency under constraints in mostly every attempt (1.3× on average). Thus, it explores effectual candidates and achieves efficient codesigns in *minutes*, while non-explainable optimizations may fail to obtain even a feasible solution over days. Explainable-DSE obtains codesigns of 6× lower latency in (36× less search time on average and up to 1675×) 47× fewer iterations vs. previous DSE approaches for DNN accelerators. By achieving highly efficient solutions in only 54 iterations, Explainable-DSE enables opportunities for cost-effective and dynamic explorations in vast space.

2 LIMITATIONS OF PRIOR DSE APPROACHES

Non-feedback DSE approaches search used by previous techniques either exhaustively over statically reduced space (e.g., grid search in [32, 49]) or randomly (e.g., in [53]). So, they do not consider any outputs like objective or utilized constraints and terminate after using a large exploration budget. It is illustrated by Fig. 1(a). On the other hand, black-box optimizations such as Bayesian Optimization (e.g., in [51, 54, 58, 81, 86]) consider values of the objective for previously tried solutions. It is illustrated by Fig. 1(b)–(c). Considering the objective helps them predict the likelihood of where the minima may lie; they acquire a candidate for the next trial accordingly. The process repeats until convergence or the number of trials exceeding a threshold. While black-box DSE could be more efficient than non-feedback DSE, they all face the following limitations:

Previous DSE techniques lack reasoning about bottlenecks incurring high costs: An efficient DSE mechanism should determine challenges hindering the reduction of objectives or utilized constraints. It should also determine which among the many parameters can help mitigate those inefficiencies and with what values. However, with objective as the only input, these black-box or system-oblivious DSEs can figure out only which prior trials reduced objective. But, they do not reason about what costs a solution could lead to and why – a crucial aspect in exploring enormous design space. This challenge is exacerbated by the fact that execution characteristics of different functions in workloads are diverse (e.g., memory- vs. compute-bounded DNN operators, energy consumption characteristics). By considering just the total cost, black-box DSEs cannot consider diverse bottlenecks in multi-modal or multi-workload executions, which need to be addressed systematically.

Implications: A major implication of excessive sampling caused by lacking explainability is **inefficiency of obtained solutions**. Fig. 4(a) illustrates this through a toy scenario, i.e., exploring the number of PEs and global buffer size for a single ResNet layer. It shows an exploration done by earlier to later trials with HyperMapper 2.0 [51] – an efficient, Bayesian-based optimizer. The figure shows that even for a tiny space, acquired solutions are inefficient (high latency), as there is no reasoning about underlying bottlenecks and their mitigation. So, even though DSE has already acquired some better solutions before, the later acquisitions correspond to inefficient solutions. As the design space becomes vast, the non-explainable DSEs can require too many trials (at least, in thousands [36, 51, 86]), and they may still not find the most efficient solutions.

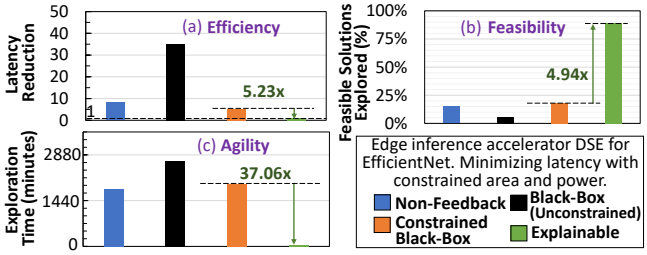

Figure 3: Effectiveness of non-explainable and explainable DSE frameworks for exploring efficient and feasible solutions in the vast space: (a) Efficiency (latency minimization), (b) Feasibility (in % of the total solutions evaluated); (c) Agility (exploration time in minutes). Analysis is shown here for exploring edge accelerator design for EfficientNetB0 model.

For example, Fig. 3(a) shows that the latency of the solutions obtained by non-explainable DSEs can be up to 35× higher, even for 2500 trials (two days of search time). This is because, practical exploration budget is fractional (thousands) compared to vast design space (quadrillions). By generating trials without understanding bottlenecks and their mitigation, most of the search budget gets spent on excessive and likely ineffectual trials.

Lacking reasoning about inefficiencies can deprive the DSE of **a tightly coupled hardware/software codesign optimization**. For instance, DSEs in [10, 36, 40, 58, 63, 82, 89] mainly explore architectural parameters with black-box DSEs and use a fixed dataflow for executions.[3] Fixing the execution methods limit the effectual utilization of architectural resources when subjected to various tensor shapes and functionalities [9, 13]. Consequently, DSEs may achieve architecture designs that are either incompatible with the dataflow (**infeasible solutions**) or **inefficient**. Likewise, in *isolated* co-optimizations, obtained HW design and dataflow are *oblivious* of each other, leading to excessive trials and inefficient solutions.

For constrained optimizations, **lack of awareness about utilization of constraints** in black-box DSEs leads to exploring infeasible and inefficient solutions and excessive trials. This is because DSEs cannot determine which constraints are violated, which regions could exhaust constraints quickly while not optimizing objective much, and how configuring different accelerator design parameters could affect all this. Fig. 3(b) illustrates this for an edge accelerator DSE subjected to power and area constraints. For constrained optimizations like HyperMapper2.0 [51], out of 2500 trials, only 18% of the evaluated solutions were feasible, and up to only 52% for constrained reinforcement learning [36].

Another implication of excessive trials is **inapplicability to dynamic DSE scenarios**. Excessive trials lead to low agility, as illustrated in Fig. 3(c). Non-explainable DSEs consume very high exploration time, even *weeks*, while obtaining solutions of lower efficiency. It makes existing DSE approaches unsuitable for dynamic explorations (e.g., convergence within a few tens to 100 iterations). For instance, unlike one-time ASIC designs, deploying accelerator overlays over FPGAs (edge/cloud; dedicated/multi-tenant) can benefit from dynamic DSEs, where constraints for DSE and resource budget may also become available just before deployment.

[3] §A.2 and §G provide background on HW/SW codesign DSE.

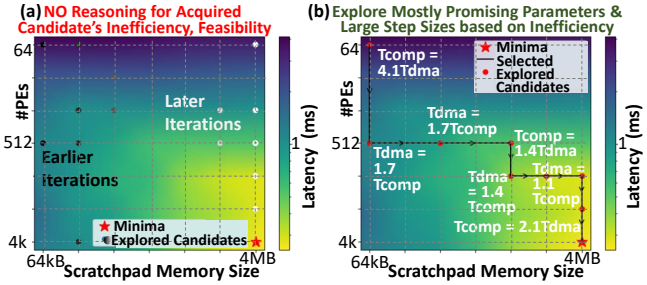

Figure 4: Example DSE of #PEs and shared memory sizes for ResNet CONV5_2b [30] with (a) Prior techniques (HyperMapper2.0 [51]), and (b) Explainable-DSE, which reasons about inefficiencies in achieved executions, limiting the search to crucial parameters and tuning accordingly.

3 DSE USING BOTTLENECK ANALYSIS: MOTIVATION AND CHALLENGES

3.1 Making DSE Explainable Through Bottleneck Analysis

In Fig. 4(b), we illustrate the same problem of designing a DNN accelerator as in Fig. 4(a), but by using bottleneck analysis in the DSE. Before acquiring new candidates, the DSE analyzes current design through the bottleneck model and pinpoints the bottleneck in achieved latency. Then, it uses mitigation obtained from the bottleneck model to make next acquisitions. A bottleneck, in the context of the latency optimization for a deep learning accelerator, can be attributed to one of the execution factors, such as time consumed by computations, communication via NoCs, and off-chip memory accesses with direct memory access (DMA) controller. For instance, after evaluating the initial point *(number of PEs, shared memory size) = (64, 64kB)*, the DSE can reason that the computation time of the design is 4.14× higher than the time taken by off/on-chip data communication. From the mitigation strategy, DSE concludes and communicates to the designers that it would scale the total number of PEs next by at least 4.14×.[4] Since this is the only mitigation suggested, the newly acquired and optimized design becomes (512, 64kB). By repeating this process, the DSE informs that the previous bottleneck got mitigated and DMA-transfers is the new bottleneck. Using the bottleneck model, the DSE considers execution characteristics (like data accessed from off-chip memory and unexploited data reuse) and mitigation for the current design point, adjusting the size of shared on-chip memory or off-chip bandwidth. This iterative process continues. It not only enables the DSE to characterize and explain DSE decisions but also reduces objectives at almost every acquisition attempt, converging to efficient solutions quickly.

3.2 Challenges in Enabling DSE of DNN Accelerators Using Bottleneck Analysis

Need bottleneck models for DNN accelerators. DSE using bottleneck analysis requires bottleneck models. Unlike cost models used in black-box DSEs that provide a single value, bottleneck models can provide rich information, in an explicitly analyzable manner,

[4]Just to note the power of explicit bottleneck mitigation strategies, if area constraint was unmet, DSE could intelligently let communication time increase but meet constraints first through reduced buffer/NoC sizes.

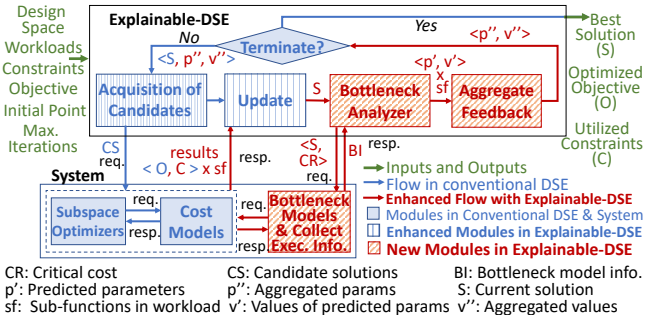

Figure 5: Explainable-DSE: A framework for exploring design space using domain-specific bottleneck models.

about 1) how design parameters contribute to different factors that lead to the total cost, and 2) mitigation options when any factor gets identified as a bottleneck. Such bottleneck/root-cause analysis have been applied for characterizing fixed designs and finding mitigation, e.g., for industry pipelines and production systems, hardware or software for specific applications [14, 70, 84], FPGA-based HLS [29, 67], overlapping microarchitectural events [24], and power outage [28]. Likewise, optimizing DNN accelerators with bottleneck analysis also require developing bottleneck models.

Need an interface to decouple domain-specific bottleneck models from a domain-independent exploration mechanism and express them to DSE. Once bottleneck models are developed, there needs to be a DSE framework that can integrate such a domain-specific bottleneck model to drive the iterative search. However, since bottleneck models are usually domain-specific, search mechanisms provided by prior DSE techniques using bottleneck analysis [26, 67, 84] are implemented too specifically for their domain. There needs to be an interface to decouple the domain-independent search mechanism from domain-specific bottleneck models so that designers can reuse and apply the same search mechanism for exploring designs in new domains like DNN acceleration.

Need acquisitions accounting for mitigation of multiple bottlenecks in multi-functional or multiple-workload executions. Prior DSE techniques using bottleneck analysis (in other domains) [26, 29, 67, 84] optimize only a single task at a time, i.e., consider a single cost value of executing a loop-kernel or whole task and iteratively mitigate arising bottleneck. However, when workloads involve different functions of diverse execution characteristics, e.g., a DNN with multiple layers or multiple DNNs, changing a design parameter impacts their contribution to the overall cost in distinct ways; considering just a total cost may not be useful. Mitigation strategies to address these layer-wise bottlenecks can lead to changing diverse parameters and a range of values possible for the same parameter. Therefore, when the DSE makes its next acquisitions, it needs to ensure that multiple bottlenecks arising from executing different functions of target workloads are mitigated systematically and effectively.

4 EXPLAINABLE-DSE: CONSTRAINTS-AWARE DSE USING BOTTLENECK ANALYSIS

This section presents Explainable-DSE – This section presents Explainable-DSE – a framework for an agile and explainable DSE

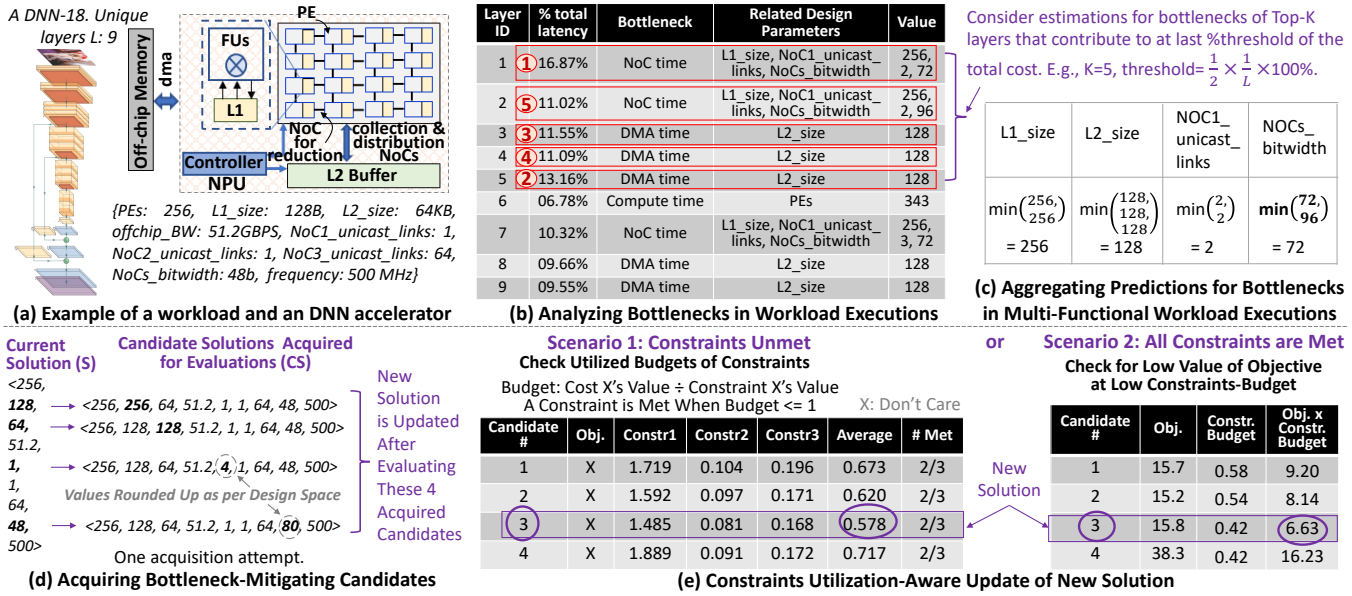

Figure 6: Example walkthrough. a) A DNN and accelerator architecture parameters, b) Analyzing bottlenecks for executing each layer/function (§4.3), c) Aggregating bottleneck mitigation for multi-functional/multiple workloads (§4.4), d) Acquiring new candidates that mitigate bottlenecks (§4.5); e) Constraints utilization-aware update of the new solution (§4.6).

using bottleneck analysis for optimizing deep learning accelerator designs. First, we discuss our framework's overall workflow and illustrate it with a walk-through example. Then, we describe how its bottleneck analyzer processes bottleneck models, i.e., determines factors incurring a high cost, parameters relevant to the bottleneck factors, and new values of parameters that can reduce the cost. We also introduce an API through which architects can specify domain-specific bottleneck models, e.g., for analyzing accelerator execution costs and bottleneck mitigation strategies. For bottleneck analysis involving the execution of multiple workloads or multiple functions within a workload, we discuss how Explainable-DSE aggregates the obtained parameters and their new values, including considering bottlenecks of only execution-critical functions. We then describe how the proposed framework considers inequality constraints when updating the obtained solutions, prioritizing exploration of feasible regions. We also provide an in-depth bottleneck model and bottlenecks mitigation strategies for exploring low-latency designs of DNN accelerators using Explainable-DSE. Lastly, we discuss how our approach can enable a tightly coupled accelerator/mappings co-explorations.

4.1 Framework Workflow

Fig. 5 illustrates the workflow of Explainable-DSE. The DSE uses bottleneck analysis to explore solutions that reduce a critical cost, denoted as *CR*. Critical cost is usually an objective *O* that needs to be minimized, and optionally an unmet inequality constraint value *C*. To reduce the cost, the bottleneck analyzer considers the current solution (*S*) and analyzes cost-related bottleneck information (*I*). The analyzer identifies the bottleneck factors incurring higher cost value and finds the *scaling* "**s**" by which the objective/constraint value needs to be reduced ("s" is internal to the analyzer, so not shown in Fig. 5). Then, the analyzer determines design parameters (p') crucial for mitigating the bottleneck and their values (v'). Workloads can be multi-modal or usually involve multiple sub-functions (sf), e.g., the accelerator needs to be optimized for different DNNs or various layers in a DNN. So, the DSE applies bottleneck analysis to the costs of each sub-function individually and aggregates the corresponding feedback obtained. This aggregation leads to a set of predicted design parameters (p'') and their respective values (v''). Based on these predictions, a new set of candidate solutions (*CS*) is derived for the subsequent acquisition. The process iterates, as depicted in Fig. 5. We refer to acquiring and evaluating candidates in the *CS* as one "*acquisition attempt*". It is analogous to z sequential DSE iterations if a *CS* contains z candidates. The current solution, *S*, is updated once (from z candidates) at every acquisition attempt. When some inequality constraint is not met, the framework considers the utilized budgets of constraints for acquired candidates in updating the current solution. This approach enables the DSE to prioritize reaching feasible subspaces. In Fig. 5, the introduction of new modules for the proposed approach and corresponding information flow is illustrated through a diagonal stride pattern and a different shade (red). The workings of these modules are described next, accompanied by a walk-through example (illustrated in Fig. 6). Additional information regarding the capabilities of the framework, current limitations, and future works for further automation and enhancements are discussed in §B and §C, respectively.

4.2 Framework Inputs and Outputs

Inputs: Information of the design space, constraints, objective, workloads, initial point, and total iterations. *Outputs* upon convergence or termination: Optimized solution and its costs.

Design Space: It defines the design parameters of type integer, real, or categorical. Their possible values can be expressed as either a list or a mathematical expression.

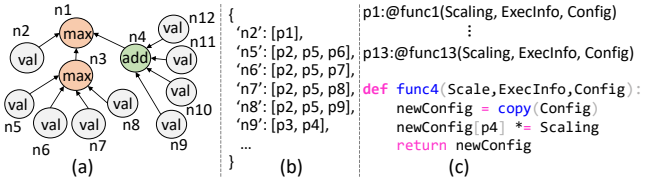

Figure 7: Proposed API through which designers or design automation tools can specify a bottleneck model of a system. The information can contain: (a) Bottleneck graph containing factors contributing to a cost, (b) Different parameters impacting the factors, and (c) Handles to subroutines that calculate new values of the parameters.

Constraints and Objective: Users can define inequality constraints on multiple costs. Our current implementation optimizes a single objective. It can be extended for multiple objectives through existing acquisition techniques.

Target System and Cost Models: System can incorporate arbitrary cost models and subspace optimizations for populating costs. It can also provide costs at sub-functions granularity, e.g., the latency of individual DNN layers. The proposed API (§4.3) enables the seamless integration of the bottleneck models.

To demonstrate DNN accelerator design explorations, we leverage existing cost models and use them to evaluate all techniques. We use Accelergy [80] to obtain the total area, energy per data access (for 45nm technology node), and maximum power. The maximum power is obtained from the maximum energy consumed by all design components in a single cycle. Accelergy provides technology-specific estimations via plugins for Aladdin [65] and CACTI [50]. We use our dMazeRunner infrastructure [15] to obtain the latency and energy consumed by mappings of DNN layers and for quick mapping optimizations for each architecture design.

4.3 Bottleneck Analyzer

Before each acquisition attempt, Explainable-DSE conducts bottleneck analysis on the obtained solution from previous attempt. It uses the bottleneck model, which helps pinpoint the execution bottlenecks and suggests options to mitigate them, ultimately reducing costs. For instance, Fig. 6 demonstrates this exploration process for an 18-layer DNN, where nine layers have unique tensor shapes for execution-critical operators (CONV and GEMM). Fig. 6(a) shows the architectural template and parameter values of the current solution during the DSE. Fig. 6(b) displays the bottleneck analyzer's ability to identify bottlenecks for each DNN layer and estimate which parameters should be updated with what specific values. This section further explains how the analyzer works and presents an API through which designers can specify their domain-specific bottleneck models for the DSE.

By evaluating the bottleneck model, the bottleneck analyzer determines (a) bottleneck factors, (b) parameters that are most critical for reducing the costs of these bottleneck factors, and (c) values of these critical parameters. Designers can provide the information for bottleneck models through an API that comprises up to three data structures, as illustrated in Fig. 7. The first and the key data structure is a graph of the bottleneck model, which outlines the underlying factors contributing to the total cost. The second includes a

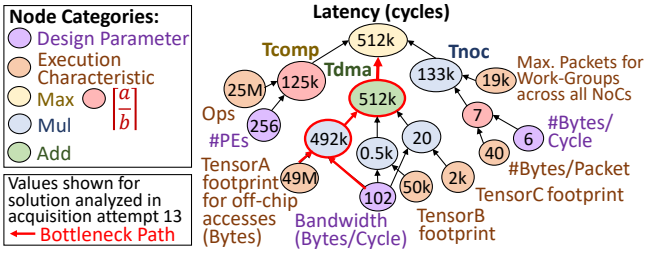

Figure 8: A simplified bottleneck model for analyzing the latency of a DNN layer execution on a DNN accelerator. As compared to conventional cost models that provide a single value, the graph-based bottleneck models can provide richer information in an explicitly analyzable format and outline how hardware/software parameters relate to total cost, allowing designers and the DSE to make informed decisions.

list of related parameters for each factor. The third contains handles to subroutines that predict the next values of parameters. When some information is unavailable, such as how to predict the value of a parameter, Explainable-DSE resorts to its black-box counterpart (e.g., sampling neighboring values).

(a) Determining bottleneck factors from bottleneck model graph: A bottleneck model is a graphical representation of which and how various factors contribute to the cost of executing a workload on an accelerator, as depicted in Fig. 7(a). It is represented as a tree whose nodes are mathematical functions like addition, multiplication, division, and maximum. Each node typically represents a cost factor, which is calculated from values of its children by applying the corresponding mathematical function. Thus, the root node of the bottleneck model represents the total cost and leaf nodes are hardware, software, or execution related design parameters.

For example, Fig. 8 shows a simplified bottleneck model for a DNN layer execution, where the root corresponds to the overall cost (e.g., latency). The total cost depends on child nodes representing underlying cost factors. For example, the total latency is determined as the maximum value among the computational time, the total on-chip communication time, and the total DMA time for off-chip memory accesses. The total DMA time, in turn, is additive and depends on the off-chip footprint of different tensors and the bandwidth. Similarly, the time for communicating data from on-chip buffers to PEs via NoCs is approximated with the total data packets communicated to different workgroups and NoC bus widths. Leaf nodes in a bottleneck graph typically represent values of the *design parameters* from the design space, such as hardware design parameters, application parameters like tensor shapes and quantization bit-width, and the accelerator's execution characteristics for a given workload or application or the code optimization parameters. Execution characteristics include data allocation to buffers, on/off-chip communication of data, and unexploited reuse, etc. (§4.7) and obtained from mapping of a workload on the accelerator.

During each acquisition attempt, the analyzer considers current solution and populates the graph with the corresponding actual values. For each cost factor, which is an intermediate node, the analyzer calculates its contribution to the total cost as the ratio of its value to the total cost. The analyzer traverses the graph

and computes contribution of each factor based on the associated mathematical operation. For instance, at a max node, it traces back to the maximum value; at an add node, it counts contributions proportionally. It identifies the factor with the highest contribution as the primary bottleneck. The analyzer then calculates the **scaling "s"**, which is the ratio by which the cost of the bottleneck factor should be reduced to alleviate bottleneck. In Fig. 8, DMA time dominates the total latency, whereas the computational and on-chip communication time contributes to only 24.4% and 25.9% of the total latency, respectively. The analyzer finds that the later factors can be balanced by scaling down the DMA time, e.g., by a factor of 100% ÷ 25.9% or 3.85×. Through traversal, the analyzer identifies the memory footprint of tensor A as the primary bottleneck operand. The analyzer may also determine multiple bottlenecks (based on decreasing order of their contributions) so that the acquisition function can generate an adequate number of candidates.

(b) Selecting parameters associated with the bottleneck: To determine which parameters impact specific bottleneck factors, the analyzer can traverse the bottleneck graph, or designers can provide this information through a dictionary that maps the node names/numbers to relevant parameters (Fig. 7b). In the example bottleneck graph of Fig. 7(a), nodes '$n4$' and '$n9$' correspond to DMA time and the off-chip footprint of Tensor A, respectively. They are associated with parameters '$p3$' and '$p4$' (e.g., '$L2\_size$' and '$offchip\_BW$' in Fig. 6a). Once the bottleneck factor and mitigating parameters are identified, DSE can obtain new values from supporting subroutines or evaluating the bottleneck path.

(c) Obtaining values of critical parameters for bottleneck mitigation: Designers can provide handles to domain-specific subroutines that contain mitigation strategies for different design parameters, as shown in Fig. 7(c). Each subroutine calculates the new value of a parameter based on the current parameter value, the scaling s required for reducing the bottleneck factor, and the execution characteristics of the current design configuration (§4.7). For example, the function '`func4`' can scale the off-chip bandwidth to reduce DMA time, and functions '`func5`' to '`func8`' can scale the bus width or NoC links to lower on-chip communication time. The DSE can leverage these subroutines to predict bottleneck-mitigating values for acquiring the next candidates.

While accelerator designers can specify bottleneck models in the proposed graphical representation, design tools or machine learning-based approaches can be developed for automatic construction of bottleneck models or mitigation options for designing new processors and off-the-shelf or large-scale architectures (§C).

4.4 Addressing Bottlenecks in Multi-Functional and Multi-Workload Executions

As Fig. 6(b) illustrates, the analyzer performs bottleneck analysis on each sub-function of workloads (DNN layer) one by one. Due to the diverse execution characteristics of these functionalities, the predictions obtained for each sub-function can be distinct, depending on the factors like available reuse and parallelism. Additionally, mitigation options for multiple bottlenecks in executions of various DNN layers may involve multiple values for the same parameter. Hence, an aggregation is required to determine the next set of parameters and their values (Fig. 6c). The DSE employs two methods for the aggregation/filtering of the predicted parameters and values:

(i) *Aggregating different values of the same parameter:* After analyzing the solution S for multiple sub-functions (identifying bottlenecks and predicting mitigation), there can be different predicted values of the same parameter. So, the final prediction can be obtained by either iterating over some of these predicted values or applying a function (maximum, minimum, average) on them. Choosing the maximum value can lead to faster convergence, but it can favor a single sub-function and be overly aggressive for others. For instance, selecting a new value as 16× (from options like 4×, 8×, 16×) of the current number of PEs can significantly reduce latency of a non-performance-critical DNN layer but not of other layers, while consuming higher area and power. Thus, exploration can quickly exhaust the budget for constraints without getting a chance to explore a considerable range of intermediate candidates that could minimize the overall cost. Instead, we opt for selecting the minimum value as the final prediction (shown in Fig. 6c).

(ii) *Aggregating parameters from only bottleneck sub-functions:* Not all the sub-functions or cost factors require improvement. Hence, Explainable-DSE allows focusing on only the bottleneck ones, i.e., those contributing the most to the total cost. This capability is achieved through two tunable parameters: K and $threshold$. The DSE considers predictions from up to top-K sub-functions whose fractional contributions to the total cost exceed a certain $threshold$. In target DNNs, the number of layers with unique tensor shapes (l) can range from a few to several tens. So, K is arbitrarily set to five and the $threshold$ to 0.5*(1/l)*100%, considering predictions from layers that consume higher portions of the cost. For the example in Fig. 6, the analyzer considers mitigating bottlenecks from the top-5 layers that contribute at least 5.5% to the total latency.

4.5 Bottlenecks-Guided Acquisitions of New Candidates

After aggregating predicted parameter values for mitigating bottlenecks, the DSE populates the candidates CS to be acquired next. For simplicity, our acquisition function samples a new candidate for each new parameter value. As Fig. 6(d) shows, all but one parameter of the candidate has the same value as in the current solution. This mechanism naturally facilitates an iterative search that adaptively tunes among bottleneck parameters. It avoids a greedy local search [56] by the following means. i) It limits exploration parameters to only a few (critical for addressing the bottleneck); ii) It can predict values of larger step-size (non-neighbors) based on bottleneck mitigation analysis (whereas local search explores p immediate neighboring values for all p parameters in the selected solution). Acquisitions by addressing multiple, dynamic bottlenecks (different parameters to be optimized at each DSE iteration) and exploring larger step sizes usually help avoid over-optimization within the local neighborhood (converging to local optimal). §C further discusses workarounds for overcoming the bottleneck-oriented greediness in the search. With modular framework, the designers may also specify other acquisition/update functions that act upon bottlenecks-mitigating parameters. When acquiring a candidate, if a predicted value is not present in the defined design space (e.g., non-power-of-2), the DSE rounds it up to the closest value.

4.6 Constraints-Budget Awareness in Updating the New Solution

When exploring a vast space under tight constraints, initially acquired solutions usually fail to meet some constraints (e.g., low-area, high-latency region). To effectively explore the space, the DSE accounts for the *constraints budget* when selecting the new solution, which, in turn, impacts the acquisitions of new candidates. In determining the new solution among the explored candidates, the DSE first checks whether the acquired candidates meet all constraints and by what margin. If any candidate does not meet all constraints, it selects a candidate that uses the least *constraints budget* as the new solution. The constraints budget is calculated as the average of the utilized constraint values that are normalized to the constraint thresholds. Such accounting is illustrated in Fig. 6(e) - scenario 1. Further, for monomodal cost models, when a candidate (corresponding to the new value of some parameter) violates more constraints than the obtained solution, the DSE can stop further exploration for that parameter's range. Thus, by prioritizing the feasibility of solutions, the DSE limits acquiring solutions that optimize the objective at the expense of violating constraints. When multiple candidates satisfy all constraints (scenario 2), the DSE selects the one (as the new solution) that achieves the lowest objective value with a lower constraints budget, i.e., the smallest value for *objective×constraints budget*. Such a strategy can help avoid greedy optimization that chases marginal objective reduction, seeking more promising solutions without quickly exhausting the constraints.

4.7 Bottleneck Mitigation for Designing Deep Learning Accelerators

We use the latency of executing a DNN as an example cost for describing bottleneck mitigation for optimizing DNN accelerator/mapping codesigns. We describe what information about the latency can be analyzed for constructing a bottleneck model and predicting new values to mitigate various bottlenecks.

Information embedded in bottleneck model: The bottleneck model incorporates execution characteristics of an optimized mapping of a DNN layer onto an architecture design. They include:
• T_comp T_comm, T_dma: Total time consumed by computations on PEs, communicating data via NoCs, and accessing data from off-chip memory via DMA, respectively. • $Accel\_freq$: Frequency of the accelerator (MHz) • $data\_offchip$: Data (bytes) accessed from off-chip, per operand • $data\_noc$: Data (bytes) communicated via NoC, per operand • NoC_groups_needed: Maximum number of concurrent links that can be provided for communicating unique data to different PE-groups; one variable per operand. • $NoC\_bytes\_per\_group$: Size of the data that can be broadcast to PEs within every workgroup of PEs; one variable per operand.

Using above information, a bottleneck graph can be created as illustrated in Fig. 8. Typically, this information is available from experts-defined cost models like [15, 44, 80]. If not, it may be obtained through similar analysis, hardware counters, or ML models.

Dictionary of affected parameters: It contains different factors contributing to the latency as keys and a list of relevant parameters as values. For example, the computation time is affected by the number of PEs and functional units in PEs. The time consumed by NoC communication is affected by the concurrent unicast links in NoCs, bit-widths of NoCs, and size of the local buffer or RF. The buffer size impacts the exploited reuse and the size of the data to be communicated. DMA time is affected by the bandwidth for off-chip memory accesses and the size of the shared memory.

Determining new values of accelerator design parameters: For a design configuration, analyzing the bottleneck model of a cost provides s, which is the scaling to be achieved by reducing a bottleneck factor's cost. $X\_current$ and X_new indicates the current and predicted value of a parameter X, respectively. X is a parameter impacting the bottleneck factor (obtained from dictionary). We next describe the calculations for values of various design parameters.

• **PEs:** The number of PEs required can be calculated directly from the needed scaling. $PEs\_new = s * PEs\_current$.

• **Off-chip BW:** Bandwidth (BW) for off-chip and on-chip communication is obtained from the number of data elements communicated per operand and the target scaling factor. E.g.,
$scaled\_T\_dma = T\_dma \div s$;
$footprint = sum(data\_offchip)$;
$bytes\_per\_cycle = footprint \div scaled\_T\_dma$
$offchip\_BW\_new = bytes\_per\_cycle * Accelerator\_freq$

• **NoC Links and Bit-width:** For DNN accelerators, separate NoCs communicate different operands, each with multiple concurrent links for various PE groups. For every NoC, the maximum number of PE-groups with simultaneous access and the total bytes broadcast to each group are obtained from the cost model [15]. If communication time is a bottleneck, the operand causing it ('op') is available from the bottleneck analysis of the graph. Then, for the corresponding NoC, its width (bits) is scaled to make the broadcast faster based on the needed scaling. The new value is clamped to avoid exceeding the maximum width feasible for a one-shot broadcast.
$max\_width\_feasible = exec\_info[noc\_bytes\_per\_group][op] * 8$
$width\_scaled = noc\_width\_current * s$
$noc\_width\_new = min(width\_scaled, max\_width\_feasible)$

Similarly, total unicast links needed by the NoC for op are calculated from required concurrent accesses by PE groups.
$max\_links\_feasible = exec\_info[noc\_groups\_needed][op]$
$links\_scaled = noc\_unicast\_links\_current[op] * s$
$unicast\_links\_new[op] = min(links\_scaled, max\_links\_feasible)$

Whenever the number of PE-groups requiring different data elements exceeds the available unicast links (by $V\times$), the data is unicast with time-sharing (V times) over configurable NoC (as in Eyeriss [8]) to facilitate the mapping. Parameter $virtual\_unicast\_links$ indicates time sharing over a unicast link, which can be set as number of time sharing instances (V).

• **Sizing RFs and Memory:** The total NoC communication time can be reduced by increasing the bottleneck operand (op)'s reuse in the RF (register file or local buffer) of the PEs. Increasing the reuse by R requires (R×) larger chunks of non-bottleneck operands, which need to be stored in the RF and communicated via other NoCs. Using the information about non-exploited (available) reuse of the bottleneck operand and the required scaling, the new RF size can be calculated as:
$target\_scaling = min(max\_reuse\_available\_RF[op], S)$
$RF\_size\_new = \sum_{op_i} \lceil exec\_info[data\_RF][op_i] *$
$target\_scaling \div reuse\_available\_RF[op_i] \rceil$

The calculation is similar for the global buffer (scratchpad memory), except for the targeted scaling. In off-chip data communication, multiple operands are communicated one by one via DMA (unlike simultaneously by NoCs per operand). So, the targeted scaling of the scratchpad depends on the bottleneck operand's (with remaining reuse) contribution (f) to the total off-chip footprint. The speedup/scaling achievable through exploiting reuse (A) can be approximated with the Amdahl's law as:

$A = (s * f) \div (1 - s + (s * f))$

$target\_scaling = min(max\_reuse\_available\_SPM[op], A)$

$SPM\_size\_new = \sum_{op_i} \lceil exec\_info[data\_SPM][op_i] * target\_scaling \div reuse\_available\_SPM[op_i] \rceil$

We implemented Explainable-DSE workflow and bottleneck analysis and mitigation for DNN accelerators in python. It allows easy interfacing with the cost models for DNN accelerators. Since the implementation of the bottleneck analysis module and the bottleneck-guided DSE is external to the cost model, they could be extended to interface with other accelerator cost models like MAESTRO [44] that make the execution characteristics available (e.g., bandwidth, Ops, data packets to be communicated). §C and §D discuss such specification efforts for bottleneck models.

4.8 Tightly Coupled Hardware/Software Codesign Explorations

Efficient codesign requires optimizing both the hardware configurations and mappings in a coordinated manner. However, when using black-box DSEs, these configurations are typically explored in a loosely coupled manner. In other words, the acquired candidates usually do not address inefficiencies in the achieved execution with their co-optimization counterparts. For example, the acquired values of the off-chip/NoC bandwidth may be inefficient or incompatible with the selected loop tile configuration (in the same/previous trials in the mapping optimization), resulting in significantly higher communication time and total latency.

To address these inefficiencies, the DSE integrates mapping space optimizations and explores HW/SW codesign in a tightly coupled manner through bottleneck-guided exploration. The usage of bottleneck models allows reasoning about design inefficiencies for the objective optimized by a co-optimization counterpart. For example, the DSE considers software optimization as a subspace for iteratively optimizing hardware configurations. For a hardware configuration, when the DSE optimizes mappings through explorations or even a fixed schema, it mostly leads to efficient executions that can adapt to the tensor shapes and workload characteristics (reuse, batching, parallelism, etc.) for the selected hardware configuration. Then, the DSE uses bottleneck models that consist of both hardware and software/execution parameters. The DSE finds bottlenecks in the executions optimized by the mapping optimizer. Then, in the next attempt, the DSE acquires new hardware candidates such that they address bottlenecks in the executions optimized previously through software configurations. Once a new hardware design is updated as the current solution, software configurations are optimized again in tandem. Consequently, this approach leads to an efficient codesign for diverse tensor shapes and workload characteristics.

For efficient exploration of hardware/mapping codesign within practical budgets, DSE needs to explore quality mappings quickly.

Table 1: Design space for edge DNN accelerators.

Data: int16; Freq. 500 MHz; Constraints: Throughput>=40/10 FPS (vision light/large), 120/530/176k samples/second (NLP: Transformer/BERT/wav2vec2); Area < 75 mm^2; Max. power < 4W. Objective: Minimize latency.

Parameter	Values	Options
PEs	64, 128, ..., 4096	7
L1 buffer (B)	8, 16, ..., 1024	8
L2 buffer (kB)	64, 128, ..., 4096	7
Offchip bandwidth (MBPS)	1024, 2048, 4096, 6400, 8192, 12800, 19200, 25600, 38400, 51200	10
NOC datawidth	16*i; i: [1, 16]	16
Physical unicast (×4)	PEs*i / 64; i: [1, 64]	64^4
Virtual unicast (×4)	2^{3i}; i: [0, 3]	4^4

Our approach builds on previous research on mappers for DNN accelerators that eliminate infeasible and ineffective mappings by pruning loop tilings and orderings (detailed in §7, §F). For fast mapping optimizations, we have integrated and extended dMazeRunner [15], which can find near-optimal solutions within seconds. Mappers like dMazeRunner [15], Interstellar [83], or ZigZag [49] consider comprehensive space, optimally prune loop orderings, and prune tilings based on the utilization of architectural resources (PEs, buffers, non-contiguous memory accesses). Then, they linearly explore the pruned space. However, one challenge with their fixed utilization thresholds for pruning is that it may lead to a search space that contains either too few mappings (e.g., tens) for some DNN layers or too many (many thousands) for others. To address this challenge, we automatically adjust these search hyperparameters of dMazeRunner to formulate the mapping search space that contains up to the top-N mappings based on utilization thresholds. N is the size of pruned mapping space formulated by adjusting thresholds for pruning the search space iteratively, which must be within a user-specified range, such as [10, 10000]. These mappings are then evaluated linearly, as in dMazeRunner [15] or Timeloop [53]. This approach helps achieve quality mappings by pruning ineffectual methods like in dMazeRunner/Interstellar, while also ensuring a reasonably large space of high-quality mappings as per specified exploration budget.

5 EXPERIMENTAL METHODOLOGY

• **Benchmarks:** We evaluate 11 DNNs for Computer Vision (CV) and Natural Langugage Processing (NLP) tasks [59]. CV models include ResNet18, MobileNetV2, and EfficientNetB0 [69] (light) and VGG16, ResNet50, and Vision Transformer [22] (large) for classifying ImageNet images. The light and large labels differentiate models based on inference latency and total computations. For object detection, we evaluated recent models FasterRCNN-MobileNetV3 [33] and YOLOv5 [4] (large). NLP models include Transformer for English-German sentence translation [73] and BERT-base-uncased [20] for Q&A on SQuAD dataset. We also evaluated Facebook wav2vec 2.0 [5] for automatic speech recognition (ASR). Their DNN layers are 18, 53, 82, 16, 54, 86, 79, 60, 163, 85, and 109 respectively. We obtained models from PyTorch and Hugging Face [78].

• **Design space:** Table 1 lists the design space of a DNN accelerator for inference at the edge. Like existing accelerators, we considered four dedicated NoCs for a total of four read/write operands [13]. The

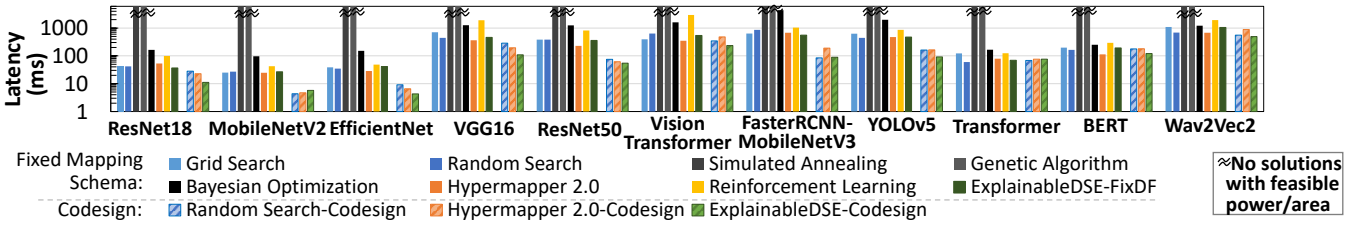

Figure 9: Explainable-DSE obtained codesigns of 6× lower latency.

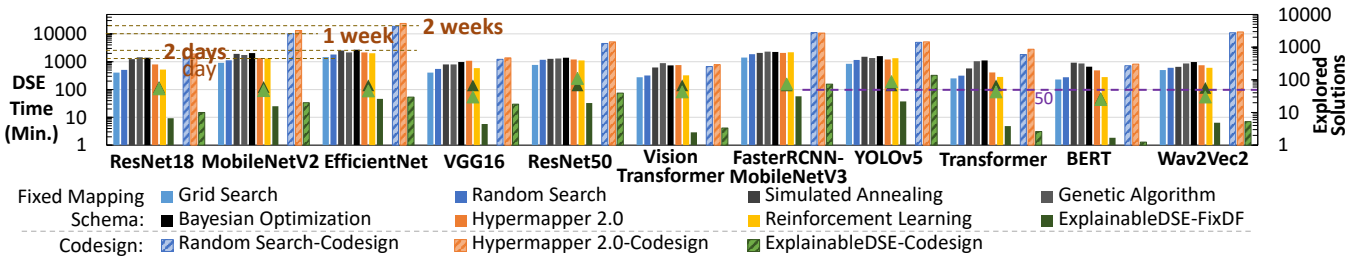

Figure 10: Explainable-DSE with fixed dataflow and codesigns reduce search time by 53× and 103× (minutes vs. days–weeks).

number of links for concurrent or time-shared unicasting is per each NoC. To limit the design space for related techniques, we considered expressing the number of unicast links as a fraction of total PEs. We selected execution constraints based on the requirements for ML benchmarks [59] and designs of industrial edge accelerators for ML inference, e.g., [1, 2]. We set the objective as minimizing the latency of the single-stream execution [59].

• **DSE techniques:** We evaluated Explainable-DSE against previous accelerator DSE frameworks using constrained optimizations - Hypermapper 2.0 [51] and Confuciux [36] for reinforcement learning (RL). Confuciux limits the total parameters to two, works with a single constraint, and requires the same number of values for all parameters. So, we generalized its implementation for evaluations. We also evaluated our approach against non-feedback or black-box approaches like Grid search, Random search, Simulated annealing (Scipy [75]), Genetic algorithm (Scikit-Opt [3]), and Bayesian optimization [52]. We evaluated all techniques on a Dell precision 5820 tower workstation. Like previous DNN accelerator DSEs, we used the cost models [15, 80]. The system for evaluating the candidates with cost models was the same for all techniques.

• **Mapping optimizations and codesign explorations:** Prior works mostly used a fixed dataflow, such that exploration time is primarily spent on optimizing hardware configurations, while getting efficient mappings with fixed schema. So, we first fixed the mapping technique as an optimized output stationary dataflow (SOC-MOP) [7] for all approaches. Then, we demonstrate the codesign with Explainable-DSE by a tightly coupled optimization of both the hardware and mapping configurations. We also compare obtained codesigns with those obtained by black-box approaches. Black-box codesign DSE explores hardware configurations with two techniques that were found effective: random search and HyperMapper 2.0. §F details setup for an effective black-box exploration of mappings in a comprehensive yet highly pruned space of feasible/effectual mappings. For mapping each DNN layer on every hardware configuration, black-box DSE evaluations use Timeloop-like random search for 10,000 mapping trials, as it was found effective in quickly obtaining high-quality mappings (§F).

• **Exploration budget:** We consider 2500 iterations for statically finding the best solutions. We also analyze dynamic DSE capabilities by explorations in 100 iterations.

6 RESULTS AND ANALYSIS

6.1 Explainable-DSE Obtained Codesigns of 6× Low Latency in 47× Less Iterations

Fig. 9 illustrates the latency obtained by different techniques for static exploration. By exploring among quality solutions, Explainable-DSE obtained 6× more efficient solutions, on average, as compared to previous approaches, and up to 9.6× over random search and 49.3× over Bayesian optimization. Even when dataflow (schema for optimized mappings) was fixed for all techniques, it obtained 1.77× lower latency on average and up to 7.89×. By applying bottleneck analysis on workload executions at every acquisition attempt, Explainable-DSE could determine parameters critical for improving efficiency. Thus, it can effectively reach high-reward subspaces among the vast space. Fig. 11 illustrates this with latency reduction obtained over iterations by taking examples of two models, EfficientNet for CV and Transformer for NLP. With objective reduction at almost every attempt, the Explainable-DSE converges to quality solutions early on (some tens of iterations) and usually of better efficiency. For instance, obtained solutions have 6.6×-35.1× lower latency for EfficientNet, as compared to the DSEs with fixed dataflow and 2.1×-9.7× as compared to black-box co-optimizations. Overall, at every attempt, it reduced the values of objective for feasible acquisitions by geomean 1.30× and 1.32× for fixed and co-explored mappings (as shown in Table 3). Acquisitions by non-explainable techniques, being bottlenecks-unaware, do not focus much on de-facto promising subspaces. In fact, in some of our evaluations for Bayesian optimization, random search, and constrained RL, the reduction in the objective throughout the DSE iterations was negative (Table 3). They acquired candidates without understanding

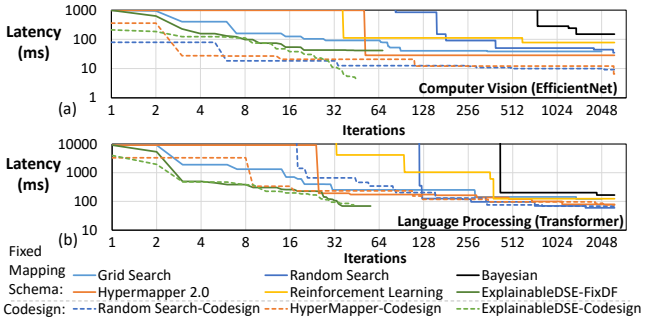

Figure 11: Latency reduced over iterations for (a) EfficientNet; (b) Transformer.

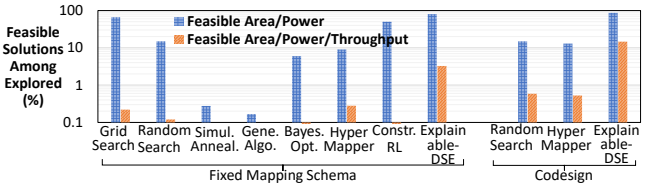

Figure 12: Most acquisitions by Explainable-DSE met area and power constraints, as compared to non-explainable techniques. Solutions obtained by all but Explainable-DSE mostly did not meet strict throughput requirements.

bottlenecks, out of which many were feasible but corresponded to lower efficiencies than the previously encountered best solutions.

Fig. 10 shows the total time (bars) taken by DSE techniques. Through constraints accommodation and systematically mitigating bottlenecks in multi-functional workload executions, explorations quickly converged or terminated while achieving even more efficient solutions. For example, Explainable-DSE with fixed and optimized mappings explored about only 59 and 54 designs, respectively (shown by triangles; ~2500 for other techniques). It led to search time reduction of 53× and 103× on average over black-box explorations, when using fixed dataflow for all techniques and hardware/mapping co-optimization, respectively. Maximum reduction in the search time was up to 501× and 1675×, respectively. Using modest information on mitigating bottlenecks, explainable DSEs consumed only 21 and 64 minutes, on average. In fact, they achieved the most efficient solutions for BERT under just two minutes!

6.2 Including Software Design Space in the Exploration Enables 4.24× Better Solutions

With the availability of exploration budget (by a drastic reduction in the search time), hardware/software codesigns can truly be enabled by optimizing both of them in a tightly coupled manner. Codesigns obtained with Explainable-DSE reduced objective by 4.24× on average as compared to using a single optimized mapping per DNN operator. The higher efficiency emanates from achieving better mappings tailored for processing various DNN layers (different functionality and tensor shapes of DNN operators) on the selected hardware configuration. They leverage higher spatial parallelism and more effectively hide data communication latency behind computations as compared to a pre-set dataflow. Further, mapping optimizations reduce the objective considerably without necessarily increasing hardware resources. Thus, by having a more constraints-budget on hand, the DSE was able to reduce the objective further (also evident in Fig. 11a).

For exploring comprehensively defined vast space of architectural configurations with non-explainable DSEs, presetting dataflow can lead to many infeasible solutions (§6.3). Note that infeasible solutions are not just hardware configurations with exceeding constraints like area or power. The designs can also be infeasible when a generated hardware configuration is incompatible with the used software, i.e., dataflow for mapping. For instance, in configurations generated by non-explainable DSEs, the total number of links for time-shared unicast was often lower than that needed by spatial parallelism in the dataflow used for mapping. That is exactly why a codesign or joint exploration with the software is important.

Black-box co-optimizations incorporated mapping explorations and reduced latency of obtained solutions further by 2.33× for HyperMapper 2.0 and 2.63× for random search, as compared to their DSEs using a fixed schema for optimized mappings. It is primarily because of the availability of more constraints-budget at hand, as discussed before. The co-optimizations also alleviated aforementioned challenge of mapping-hardware incompatibility. As Fig. 12 shows, with software optimization in the loop, the black-box co-optimizations find more feasible designs than black-box DSEs using a fixed mapping schema, when allowed to explore hardware design configurations for the same number of trials. However, even after 2500 trials for exploring hardware configurations and 10,000 trials for exploring mappings of each DNN layer on every hardware configuration, the latency of codesigns obtained by black-box approaches are still 1.6× higher than the codesigns obtained by Explainable-DSE, while consuming 103× more search time (taking 7-16 days for four workloads). Key reasons for such effective explorations by Explainable-DSE include generating fewer yet objective-reducing trials and tightly coupled codesigns. As Explainable-DSE leverages the domain knowledge, its generated designs continually address arising execution inefficiencies, converging in 47× less iterations. In black-box co-optimizations, the DSE is loosely coupled, as the generated hardware configuration is not necessarily tailored to work best with the optimized mappings (from the previous/same trial for the hardware DSE). In contrast, tightly coupled codesign exploration in Explainable-DSE finds hardware configurations that alleviate inefficiencies in the workload executions optimized previously by mappings; Once new hardware configuration is generated, mapping exploration strives to utilize hardware resources effectively, lowering costs further. And, this repeats. Thus, optimizations for both hardware and software configurations strive to reduce inefficiencies in the execution optimized by their counterpart.

Although optimizing the mappings for every hardware design requires additional search time, the overall increase for exploring codesigns with Explainable-DSE was only 3× on average (from 21 minutes to 64). In fact, for all except large object detection models, the DSE time increased from 16 minutes to only 26 minutes. One reason is that the mappings can be quickly evaluated with analytical performance models (e.g., a minute each for several hundred to a few thousand mappings) and concurrent execution with multiple threads [15] (subjected to execution on four cores at maximum in our evaluations). Moreover, applying bottleneck analysis

Table 2: Latency minimized by DSE techniques in 100 iterations.

Explainable-DSE evaluated ~54 solutions. Designs obtained by Non-Explainable DSEs were *low-throughput* (shaded values) and *incompatible* with used dataflow (dashes). More importantly, * denotes that none of the obtained candidates by a non-explainable DSE met even area/power constraints.

DSE Technique	ResNet18	MobileNetv2	EfficientNet	VGG16	ResNet50	Vision Transformer	FasterRCNN-MobileNetv3	YOLOv5	Transformer	BERT	Wav2Vec2
Grid Search-FixDF	278	73.4	92.0	3650	747	1973	1625	1477	251	780	1933
Random Search-FixDF	-*	197	694	41912	626	1376	3152	7754	157	1044	2357
Simulated Annealing-FixDF	-*	-*	-*	-*	-*	-*	-*	-*	-*	-*	-*
Genetic Algorithm-FixDF	-*	-*	-*	-	-*	-	-	-*	-*	-*	-*
Bayesian Optimization-FixDF	-	-	-	-	-	-	-	-	-	-	-
HyperMapper 2.0-FixDF	53.3	46.5	135	1339	493	1308	13582	1142	171	663	912
Reinforcement Learning-FixDF	-	-	360	-	-	-	21150	18082	143	1428	1428
Random Search-Codesign	69.6	12.7	9.5	870	209	857	224	218	244	240	1427
HyperMapper 2.0-Codesign	63.1	5.1	10.3	1233	87.3	1084	830	348	133	637	1945
ExplainableDSE-Codesign	11.2	5.7	4.3	109	54.9	233	89.2	92.1	76.2	121	494

on efficient mappings helped obtain efficient designs faster (1.1× lower iterations for hardware designs on average, and up to 1.9×). Whenever the DSE for codesigns evaluated a similar number of architecture designs as our DSE with fixed dataflow, it went on to explore even more efficient solutions (e.g., 2.33× lower latency for Vision Transformer).

6.3 By Considering Utilization of Constraints, DSE Mostly Acquires Feasible Solutions Without Exhausting Constraints Quickly

Non-explainable black-box optimization approaches, e.g., with Genetic Algorithm or Bayesian Optimization, did not know which configurations could likely lead to feasible subspaces. Therefore, even after exploring over days, they almost did not obtain a single feasible solution. When considering only area and power constraints, feasibility of the explored solutions was higher for mostly all techniques (Fig. 12), e.g., 15% for random search and 50% for constraints-aware reinforcement learning. However, when considering throughput requirement for DNN inference, the feasibility of the explored solutions was barely ~0.1%–0.3%. By exploring mappings, the black-box codesign optimizations addressed the challenge of mappings being incompatible for the obtained hardware configurations. Thus, they improved feasibility by 2×-5×, but the overall feasibility was still ~0.6%. Such low feasibility for DSE in humongous space is presumably caused by not accommodating constraints during exploration and bottlenecks-unaware acquisition trials. Contrarily, Explainable-DSE prioritized to meet the constraints for its acquisitions and update of the new solutions, which helped avoid infeasible subspaces. Plus, addressing bottlenecks in executions helped acquiring high-performance solutions. Hence, 87% and 15% of solutions explored by Explainable-DSE codesigns were feasible when considering area and power constraints and all the three constraints, respectively. For DNNs like BERT and MobileNetV2, 89%–98% of the explored solutions met area and power constraints. Once Explainable-DSE achieved a solution that met all constraints, it always ensured to optimize further with a feasible solution.

6.4 Enabling Efficient Dynamic Exploration in the Vast Space

Table 2 shows latency of solutions achieved in 100 iterations by different techniques. Under a short exploration budget, non-explainable techniques did not find a feasible solution (shaded values). Even after ignoring intense throughput requirements, most techniques could not find feasible solutions. Contrarily, by exploring spaces where candidates utilize low budget of constraints, Explainable-DSE quickly landed feasible solutions. Black-box approaches explored feasible codesigns, but they did not meet throughput requirements. On the other hand, by addressing the bottlenecks in multi-functional executions, Explainable-DSE achieved solutions of one to two orders of magnitude lower latency over other techniques.

7 ADDITIONAL RELATED WORKS

In this section, we discuss additional related work beyond the background on DNN accelerator DSE techniques described in §A.2, their limitations in §2, and previous DSEs using bottleneck analysis for different domains in §3.

• **Execution cost models of DNN accelerators:** The cost models of SECDA [29] and TVM/VTA [6] support end-to-end simulation and synthesis, while faster analytical models are more commonly used to optimize mappings and accelerator design configurations. Their examples include MAESTRO [44], Accelergy [80], SCALE-Sim [61], and those of Timeloop [53], dMazeRunner [15], and Interstellar [83] infrastructures. Most of these models estimate both latency/throughput and energy. In addition to computational cycles, MAESTRO, dMazeRunner, and Timeloop account for on-chip and off-chip communication latency. Table 5 compares their execution modeling features. For the DSE, we used the cost model of our dMazeRunner infrastructure, which also considers the performance overheads of non-contiguous memory accesses and allows explicit specification of NoC bandwidths and flexibly specifying mappings through loop nest configurations.

• **Mappers for DNN accelerators:** Mappers typically target the space of all valid loop tilings and orderings. For tensor shapes of a layer, there can be many factors of loop iteration counts, and just populating the space of valid mappings could be time-consuming (*micro*seconds–several seconds) [60]. Table 6 compares different mappers. Timeloop [53], a commonly used mapper, explores mappings through random sampling, while GAMMA [37] uses a genetic algorithm. However, GAMMA limits the number of loops that can be executed spatially and does not prune invalid tilings before exploration, requiring several-fold more trials for convergence [38]. Without eliminating ineffectual loop tilings and orderings beforehand, black-box explorations typically require thousands of trials,

generating many invalid mappings, and take hours to map a single DNN layer once [35]. Mind Mappings [31] reduces the search time by training a surrogate model that estimates costs faster than analytical models. CoSA [35] uses a prime factorization-based approach to construct the tiling space for a mixed-integer programming solver. But, many tilings corresponding to combinations of prime factors remain unexplored, potentially resulting in sub-optimal solutions. Additionally, most mappers do not support depthwise-convolutions, invoking convolutions channel-by-channel. So, they miss opportunities for exploiting parallelism across multiple channels and reducing miss penalties for accessing contiguous data of consecutive channels from the off-chip memory.

Interstellar [83] prunes ineffectual tilings by constraining the search to pre-set resource utilization thresholds. dMazeRunner [15] goes further and prunes loop orderings for unique/maximum reuse of operands and proposes heuristics that reduce the space to highly efficient mappings, which can be explored in second(s). Hence, we utilize our dMazeRunner infrastructure in our codesign and extend it to construct the space of up to top-N mappings, where N is the maximum mapping trials allowed. ZigZag [49] and follow-up mappers build upon such pruning strategies. ZigZag allows uneven blockings of loops for processing different tensors, which may partially improve efficiency. However, ZigZag's search time for a DNN layer is nearly hours [49]. While works such as [45, 49, 71, 87, 88] optimize DNN mappings on one or more hardware accelerators, they require exploring hardware parameters exhaustively or with black-box optimizations.

• **Hardware/software codesign explorations of DNN accelerators:** Previous DNN accelerator DSEs, such as [46, 58, 76, 81, 86], used black-box optimizations. They incur excessive trials and ineffectual solutions, as they lack reasoning about the higher costs of obtained candidates and the potential efficiency of candidates to be acquired next (§2). Further, DSEs of [10, 36, 40, 58, 63, 82, 89] used a fixed dataflow in explorations. It obviates increasing search time further but may not lead to the most efficient solutions compared to codesigns.

Recent approaches HASCO [81] and DiGamma [39] optimize both hardware and mapping configurations in a black-box manner, encountering the same challenges of ineffectual and excessive trials due to non-explainability (§2). Secondly, with a loosely coupled codesign exploration (§4.8), they acquire HW/SW configurations that may not be effective or suitable for the counterpart. Furthermore, they target a limited hardware design space comprising only buffers and PEs. Finally, they typically do not explore a single accelerator design that addresses inefficiencies in executing DNNs with many layers.

• **DSE using bottleneck analysis:** DSEs of [26, 84] use bottleneck analysis, but they are unaware of constraints utilization and optimize only a single loop-kernel. Plus, they explored only neighboring values of parameters (instead of scaling them to mitigate bottleneck in one shot). It leads to search time comparable to black-box DSEs [84]. AutoDSE [67] and SECDA [29] proposed bottleneck analysis specific to FPGA-based HLS, and their search optimizes a single loop-kernel/task of a single workload at a time. We propose using bottleneck models for DNN accelerator designs; our DSE framework generalizes prior DSEs to the case of multiple loop-nests, multimodal workloads, and multiple workloads through aggregation of various bottleneck mitigation (for new acquisitions of promising designs). Further, via proposed API and data structures, our framework decouples bottleneck models from search algorithms, allowing designers to systematically express the bottleneck models for their domain-specific architectures/systems and interface with the bottleneck-guided, explainable DSE.

8 CONCLUSIONS

Agile and efficient exploration in the vast design space, e.g., for hardware/software codesigns of deep learning accelerators, require techniques that not just should consider objectives and constraints but are also explainable. They need to reason about obtained costs for acquired solutions and how to improve underlying execution inefficiencies. Non-explainable DSE with black-box optimizations (evolutionary, ML-based) lack such capability; obtaining efficient solutions even after thousands of trials or days can be challenging. To overcome such challenges, we proposed Explainable-DSE, which analyzes execution through bottleneck models. As compared to cost models that provide a total value, a bottleneck model can graphically express which and how various design parameters and intermediate factors contribute to the total cost. Thus, it can provide rich information in an explicitly analyzable format, allowing the designers and DSE to identify the bottleneck factors for the obtained costs and acquire mitigating solutions. Proposed API can allow designers and/or automation tools to express their domain-specific bottleneck models and interface with the DSE. Through aggregation of predictions for bottleneck mitigation, the DSE facilitates a single effective solution for multi-functional or multiple workloads. In addition, awareness of utilized constraints in the decision making allows the DSE to prioritize exploration among feasible solutions and find more efficient solutions without quickly exhausting the constraints. Our demonstration of optimizing codesigns of DNN accelerators showed how Explainable-DSE could effectively explore feasible and efficient candidates (6× low-latency solutions). By obtaining most efficient solutions in short exploration budgets (47× fewer iterations or minutes/hours vs. days/weeks), it opens up opportunities for cost-effective and dynamic explorations.

ACKNOWLEDGEMENTS

We thank anonymous reviewers for their valuable feedback and suggestions. This work was partially supported by National Science Foundation (NSF) grant #1645578 and Graduate College Fellowship at Arizona State University. This work is done in part for the Artificial Intelligence Hardware (AIHW) program of the Semiconductor Research Corporation (SRC).

REFERENCES

[1] 2018. ARM Machine Learning Processor. https://en.wikichip.org/wiki/arm_holdings/microarchitectures/mlp.
[2] 2019. Intel Nervana NNP-I 100. https://en.wikichip.org/wiki/nervana/nnp/nnp-i_1100.
[3] 2019. scikit-opt. github.com/guofei9987/scikit-opt/.
[4] 2019. YOLOv5 Classification. https://pytorch.org/hub/ultralytics_yolov5.
[5] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems* 33 (2020), 12449–12460.
[6] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Q. Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos

Guestrin, and Arvind Krishnamurthy. 2018. {TVM}: An Automated {End-to-End} Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 578–594.

[7] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*.

[8] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[9] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019).

[10] Kanghyun Choi, Deokki Hong, Hojae Yoon, Joonsang Yu, Youngsok Kim, and Jinho Lee. 2021. Dance: Differentiable accelerator/network co-exploration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 337–342.

[11] Coral. [n. d.]. Edge TPU Performance Benchmarks. https://coral.ai/docs/edgetpu/benchmarks/.

[12] Ayse K. Coskun, Jose L. Ayala, David Atienza, Tajana Simunic Rosing, and Yusuf Leblebici. 2009. Dynamic thermal management in 3D multicore architectures. In *2009 Design, Automation Test in Europe Conference Exhibition*. 1410–1415.

[13] Shail Dave, Riyadh Baghdadi, Tony Nowatzki, Sasikanth Avancha, Aviral Shrivastava, and Baoxin Li. 2021. Hardware Acceleration of Sparse and Irregular Tensor Computations of ML Models: A Survey and Insights. *Proc. IEEE* 109, 10 (2021), 1706–1752.

[14] Shail Dave, Mahesh Balasubramanian, and Aviral Shrivastava. 2018. RAMP: Resource-Aware Mapping for CGRAs. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6.

[15] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. 2019. DMazerunner: Executing perfectly nested loops on dataflow accelerators. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–27.

[16] Shail Dave, Alberto Marchisio, Muhammad Abdullah Hanif, Amira Guesmi, Aviral Shrivastava, Ihsen Alouani, and Muhammad Shafique. 2022. Special Session: Towards an Agile Design Methodology for Efficient, Reliable, and Secure ML Systems. In *2022 IEEE 40th VLSI Test Symposium (VTS)*. IEEE, 1–14.

[17] Shail Dave and Aviral Shrivastava. 2022. Design Space Description Language for Automated and Comprehensive Exploration of Next-Gen Hardware Accelerators. *in Workshop on Languages, Tools, and Techniques for Accelerator Design (LATTE'22)* (2022). co-located with the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2022).

[18] Shail Dave, Aviral Shrivastava, Youngbin Kim, Sasikanth Avancha, and Kyoungwoo Lee. 2020. dMazeRunner: Optimizing Convolutions on Dataflow Accelerators. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1544–1548.

[19] Aryan Deshwal, Nitthilan Kanappan Jayakodi, Biresh Kumar Joardar, Janardhan Rao Doppa, and Partha Pratim Pande. 2019. MOOS: A multi-objective design space exploration and optimization framework for NoC enabled manycore systems. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019).

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. [n. d.]. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*.

[21] Joydeep Dey and Sudeep Pasricha. 2022. Robust Perception Architecture Design for Automotive Cyber-Physical Systems. *arXiv preprint arXiv:2205.08067* (2022).

[22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[23] Lorenzo Ferretti, Giovanni Ansaloni, and Laura Pozzi. 2018. Lattice-traversing design space exploration for high level synthesis. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 210–217.

[24] Brian A Fields, Rastislav Bodik, Mark D Hill, and Chris J Newburn. 2003. Using interaction costs for microarchitectural bottleneck analysis. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* IEEE, 228–239.

[25] Björn Forsberg, Maxim Mattheeuws, Andreas Kurth, Andrea Marongiu, and Luca Benini. 2020. A synergistic approach to predictable compilation and scheduling on commodity multi-cores. In *The 21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. 108–118.

[26] Gennette Gill and Montek Singh. 2009. Bottleneck analysis and alleviation in pipelined systems: A fast hierarchical approach. In *2009 15th IEEE Symposium on Asynchronous Circuits and Systems*. IEEE, 195–205.

[27] Soonhoi Ha, Jürgen Teich, Christian Haubelt, Michael Glaß, Tulika Mitra, Rainer Dömer, Petru Eles, Aviral Shrivastava, Andreas Gerstlauer, and Shuvra S Bhattacharyya. 2017. Introduction to hardware/software codesign. *Handbook of Hardware/Software Codesign* (2017), 3–26.

[28] Di Han, Wei Chen, Bo Bai, and Yuguang Fang. 2019. Offloading optimization and bottleneck analysis for mobile cloud computing. *IEEE Transactions on Communications* 67, 9 (2019), 6153–6167.

[29] Jude Haris, Perry Gibson, José Cano, Nicolas Bohm Agostini, and David Kaeli. 2021. SECDA: Efficient Hardware/Software Co-Design of FPGA-based DNN Accelerators for Edge Inference. In *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[31] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W Fletcher. 2021. Mind mappings: enabling efficient algorithm-accelerator mapping space search. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[32] Christian Heidorn, Frank Hannig, and Jürgen Teich. 2020. Design space exploration for layer-parallel execution of convolutional neural networks on CGRAs. In *Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems*. 26–31.

[33] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. [n. d.]. Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 1314–1324.

[34] Qijing Huang, Charles Hong, John Wawrzynek, Mahesh Subedar, and Yakun Sophia Shao. 2022. Learning A Continuous and Reconstructible Latent Space for Hardware Accelerator Design. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 277–287.

[35] Qijing Huang, Aravind Kalaiah, Minwoo Kang, James Demmel, Grace Dinh, John Wawrzynek, Thomas Norell, and Yakun Sophia Shao. 2021. Cosa: Scheduling by constrained optimization for spatial accelerators. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 554–566.

[36] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 622–636.

[37] Sheng-Chun Kao and Tushar Krishna. 2020. Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.

[38] Sheng-Chun Kao, Angshuman Parashar, Po-An Tsai, and Tushar Krishna. 2022. Demystifying Map Space Exploration for NPUs. *arXiv preprint arXiv:2210.03731* (2022).

[39] Sheng-Chun Kao, Michael Pellauer, Angshuman Parashar, and Tushar Krishna. 2022. Digamma: Domain-aware genetic algorithm for hw-mapping co-optimization for dnn accelerators. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 232–237.

[40] Liu Ke, Xin He, and Xuan Zhang. 2018. Nnest: Early-stage design space exploration tool for neural network inference accelerators. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 1–6.

[41] David Koeplinger, Christina Delimitrou, Raghu Prabhakar, Christos Kozyrakis, Yaqi Zhang, and Kunle Olukotun. 2016. Automatic Generation of Efficient Accelerators for Reconfigurable Hardware. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 115–127.

[42] Takuya Kojima, Nguyen Anh Vu Doan, and Hideharu Amano. 2020. GenMap: A genetic algorithmic approach for optimizing spatial mapping of coarse-grained reconfigurable architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 11 (2020), 2383–2396.

[43] Aviral Kumar, Amir Yazdanbakhsh, Milad Hashemi, Kevin Swersky, and Sergey Levine. 2021. Data-Driven Offline Optimization for Architecting Hardware Accelerators. In *International Conference on Learning Representations*.

[44] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 754–768.

[45] Hyoukjun Kwon, Liangzhen Lai, Michael Pellauer, Tushar Krishna, Yu-Hsin Chen, and Vikas Chandra. 2021. Heterogeneous dataflow accelerators for multi-DNN workloads. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 71–83.

[46] Yujun Lin, Mengtian Yang, and Song Han. 2021. NAAS: Neural accelerator architecture search. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1051–1056.

[47] Google LLC. [n. d.]. Coral Edge TPU Accelerator. https://coral.ai/products/accelerator-module.

[48] Atefeh Mehrabi, Aninda Manocha, Benjamin C Lee, and Daniel J Sorin. 2020. Prospector: Synthesizing efficient accelerators via statistical learning. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE.

[49] Linyan Mei, Pouya Houshmand, Vikram Jain, Sebastian Giraldo, and Marian Verhelst. 2021. ZigZag: Enlarging joint architecture-mapping design space exploration for DNN accelerators. *IEEE Trans. Comput.* 70, 8 (2021), 1160–1174.

[50] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.

[51] Luigi Nardi, David Koeplinger, and Kunle Olukotun. 2019. Practical design space exploration. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 347–358.

[52] Fernando Nogueira. 2014–. Bayesian Optimization: Open source constrained global optimization tool for Python. https://github.com/fmfn/BayesianOptimization

[53] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. 2019. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 304–315.

[54] Maryam Parsa, Aayush Ankit, Amirkoushyar Ziabari, and Kaushik Roy. 2019. Pabo: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[55] Andy D. Pimentel. 2017. Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration. *IEEE Design & Test* 34, 1 (2017).

[56] David L Poole and Alan K Mackworth. 2010. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press.

[57] Nirmal Prajapati, Sanjay Rajopadhye, Hristo Djidjev, Nandakishore Santhi, Tobias Grosser, and Rumen Andonov. 2019. Optimization Approach to Accelerator Codesign. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 6 (2019), 1300–1313.

[58] Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2017. A case for efficient accelerator design space exploration via bayesian optimization. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE.

[59] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[60] Enrico Russo, Maurizio Palesi, Davide Patti, Salvatore Monteleone, Giuseppe Ascia, and Vincenzo Catania. 2022. Multi-Objective End-to-End Design Space Exploration of Parameterized DNN Accelerators. *IEEE Internet of Things Journal* (2022).

[61] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).

[62] Roberto Santana. 2017. Gray-box optimization and factorized distribution algorithms: where two worlds collide. *arXiv preprint arXiv:1707.03093* (2017).

[63] Giulia Santoro, Mario R Casu, Valentino Peluso, Andrea Calimera, and Massimo Alioto. 2018. Energy-performance design exploration of a low-power microprogrammed deep-learning accelerator. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1151–1154.

[64] Kiran Seshadri, Berkin Akin, James Laudon, Ravi Narayanaswami, and Amir Yazdanbakhsh. 2022. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 79–91.

[65] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2014. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 97–108.

[66] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. 2013. Mapping on multi/many-core systems: Survey of current and emerging trends. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–10.

[67] Atefeh Sohrabizadeh, Cody Hao Yu, Min Gao, and Jason Cong. 2022. AutoDSE: Enabling Software Programmers to Design Efficient FPGA Accelerators. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 27, 4 (2022).

[68] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-Optimized OpenCL-Based FPGA Accelerator for Large-Scale Convolutional Neural Networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Association for Computing Machinery, 16–25.

[69] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[70] Catia Trubiani, Antinisca Di Marco, Vittorio Cortellessa, Nariman Mani, and Dorina Petriu. 2014. Exploring synergies between bottleneck analysis and performance antipatterns. In *Proceedings of the 5th ACM/SPEC International Conference on Performance engineering*. 75–86.

[71] Miheer Vaidya, Aravind Sukumaran-Rajam, Atanas Rountev, and P Sadayappan. [n. d.]. Comprehensive accelerator-dataflow co-design optimization for convolutional neural networks. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 325–335.

[72] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. 2018. Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions. *arXiv preprint arXiv:1802.04730* (2018).

[73] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[74] Stylianos I Venieris and Christos-Savvas Bouganis. 2016. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 40–47.

[75] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods* 17, 3 (2020), 261–272.

[76] Jie Wang and Jason Cong. 2021. Search for Optimal Systolic Arrays: A Comprehensive Automated Exploration Framework and Lessons Learned. *arXiv preprint arXiv:2111.14252* (2021).

[77] Jian Weng, Sihao Liu, Vidushi Dadu, Zhengrong Wang, Preyas Shah, and Tony Nowatzki. 2020. Dsagen: Synthesizing programmable spatial accelerators. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 268–281.

[78] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, et al. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 38–45.

[79] Nan Wu, Yuan Xie, and Cong Hao. 2021. Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI*. 39–44.

[80] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. 2019. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[81] Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. 2021. Hasco: Towards agile hardware and software co-design for tensor computation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1055–1068.

[82] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. 2020. Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[83] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, Christos Kozyrakis, and Mark Horowitz. 2020. Interstellar: Using halide's scheduling language to analyze dnn accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 369–383.

[84] Yang Yang, Marc Geilen, Twan Basten, Sander Stuijk, and Henk Corporaal. [n. d.]. Automated bottleneck-driven design-space exploration of media processing systems. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. 1041–1046.

[85] Ye Yu, Yingmin Li, Shuai Che, Niraj K Jha, and Weifeng Zhang. 2020. Software-defined design space exploration for an efficient dnn accelerator architecture. *IEEE Trans. Comput.* 70, 1 (2020), 45–56.

[86] Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. 2022. A full-stack search technique for domain optimized deep learning accelerators. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 27–42.

[87] Xiaofan Zhang, Yuan Ma, Jinjun Xiong, Wen-Mei W. Hwu, Volodymyr Kindratenko, and Deming Chen. 2022. Exploring HW/SW Co-Design for Video Analysis on CPU-FPGA Heterogeneous Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 6 (2022), 1606–1619.

[88] Shixuan Zheng, Xianjue Zhang, Leibo Liu, Shaojun Wei, and Shouyi Yin. [n. d.]. Atomic Dataflow based Graph-Level Workload Orchestration for Scalable DNN Accelerators. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 475–489.

[89] Yanqi Zhou, Xuanyi Dong, Tianjian Meng, Mingxing Tan, Berkin Akin, Daiyi Peng, Amir Yazdanbakhsh, Da Huang, Ravi Narayanaswami, and James Laudon. 2022. Towards the Co-design of Neural Networks and Accelerators. *Proceedings of Machine Learning and Systems* 4 (2022), 141–152.

A ACCELERATOR HARDWARE/SOFTWARE CODESIGN EXPLORATION

A.1 DSE Problem Formulation and Terminology

Exploration of accelerator designs is a *constrained minimization* problem, where the most efficient *solution*[5] corresponds to minimized *objective* (e.g., latency), subjected to *inequality constraints* on some *costs* (e.g., area, power) and *parameters* p of accelerator design [55]. Fig. 13 illustrates the DSE problem formulation. During the optimization, every solution gets evaluated by *cost models* for objectives and inequality constraints. The DSE technique needs to consider only *feasible* solutions and determine the most efficient solution by processing several *iterations* (*trials*). It is a discrete optimization since the *search space* is usually confined to presumably effective solutions, e.g., power-of-two or categorical values of parameters. It is also *derivative-free optimization* [51].

$$\begin{aligned} &min && obj(p), \quad p = (p_1, p_2, ..., p_n) \in \mathbb{R}^n \\ &subject\ to && cost_i(p) \leq constraint_i; \quad for\ i = 1, 2, ..., m \end{aligned}$$

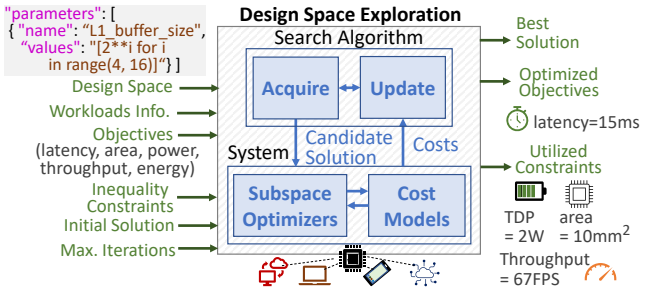

Figure 13: Accelerator DSE as constrained minimization.

A.2 DNN Accelerator Hardware/Software Codesigns DSE

Hardware/software codesigns can be explored by partitioning the search space and optimizing *software* space as a *subspace* in a loop. So, the DSE technique needs to find the best mapping of a task onto architecture and repeat the search with different architectural configurations [27]. Partitioning enables exploration in reduced space compared to exploring parameters from multiple spaces altogether. DSE techniques for DNN accelerators explore hardware designs through non-feedback or black-box optimizations like evolutionary or ML-based [10, 34, 36, 54, 58, 68, 76, 81, 82, 85, 86, 89]. Such approaches are also commonly used for designing or optimizing computing systems in general [12, 19, 21, 23, 25, 42, 51, 66, 72]. For mapping DNNs on a design (subspace optimization), they typically fix the way of execution or dataflow, e.g., in [10, 36, 40, 58, 63, 82, 84]. Hence, for processing each functionality (nested loop such as a DNN layer), these techniques usually have just one mapping. Thus, they primarily optimize designs of accelerator architecture, i.e., parameters for buffers, processing elements (PEs), and NoCs.

[5]In the accelerator design space exploration context, we use terms "solutions", "designs", and "configurations" interchangeably.

B CAPABILITIES AND DISTINGUISHED FEATURES

In this section, we highlight the capabilities of the proposed DSE using bottleneck models for agile and explainable explorations.

• **Efficient designs.** Explainable-DSE finds better solutions since it investigates costs and bottlenecks that incur higher costs; by exploring candidates that can mitigate inefficiencies in obtained designs, DSE provides efficient designs.

• **Quick/runtime DSE.** The DSE can reduce objective values at almost every acquisition attempt; it searches mostly in feasible/effectual solution spaces. Thus, DSE achieves efficient solutions quickly, which is beneficial for early design phase and for dynamic DSEs, e.g., deploying accelerator overlays at run time. Additionally, it can help when acquisition budgets are limited, e.g., due to evaluation of a solution consuming minutes to hours [86]. Further, when designers optimize designs offline with hybrid optimization methodologies [36] comprising multiple optimizations, quickly found efficient solutions can serve as high-quality initial points.

• **Explainability in the DSE and design process.** This work shows the need for explainability in the design process, e.g., in exploring the vast design space of deep learning accelerators and how DSE driven by bottleneck models can achieve explainability. Unlike cost models that provide a single value, bottleneck models can provide rich information in an explicitly analyzable format. Consequently, explorations based on bottleneck analysis can help explain why designs perform well/poorly and which regions are well-explored/unexplored in the vast space and why.

• **Generalized bottleneck-driven DSE for multiple micro-benchmarks and workloads.** In acquiring new candidates, DSE accounts for various bottlenecks in executing multiple loop nests (e.g., DNN layers) of diverse characteristics. Thus, the DSE can provide a single solution that is most effective overall, in contrast to previous DSEs that provide loop-kernel-specific solutions.

• **Specification for expressing domain-specific bottleneck models to the DSE.** This work proposes an API for expressing domain-specific bottleneck models so that the designers and/or design automation tools can integrate them to bottleneck-driven DSE frameworks and reuse the DSE.

• **Comprehensive design space specification.** In the DSE, appropriate values of a parameter is selected through bottleneck models. Thus, the DSE can alleviate the need for fine-tuning the design space; users can comprehensively define/explore vast space, e.g., more parameters and large ranges of values (arbitrary instead of power-of-two).

• **Bottleneck analysis for hardware/software codesign of deep learning accelerators.** By taking the latency of accelerators as an example, this work shows how to construct bottleneck models (for designing deep learning accelerators) and bottleneck analysis for improving the accelerator designs based on their execution characteristics. It shows how bottleneck models, as compared to conventional cost models, can be inherently analyzable and information-rich, allowing to make informed decisions for the design optimizations.

C OPPORTUNITIES AND FUTURE WORK

• **Improving efficiency further through better acquisitions:** By using bottleneck models and considering available budgets of

Table 3: At every acquisition attempt, Explainable-DSE reduces objective by 30% vs. ~1.4% by non-explainable techniques. N/A is indicated when a technique could not find a single feasible hardware solution.

DSE Technique	ResNet18	MobileNetv2	EfficientNet	VGG16	ResNet50	Vision Transformer	FasterRCNN-MobileNetv3	YOLOv5	Transformer	BERT	Wav2Vec2	Average
Grid Search-FixDF	1.71%	1.03%	1.07%	1.21%	1.25%	1.41%	0.71%	0.55%	0.98%	1.04%	1.07%	1.09%
Random Search-FixDF	0.52%	-0.87%	7.34%	-2.26%	4.69%	-4.29%	-1.41%	-0.90%	0.01%	0.97%	-1.45%	0.21%
Simulated Annealing-FixDF	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Genetic Algorithm-FixDF	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Bayesian Optimization-FixDF	11.26%	26.57%	19.57%	19.22%	-1.09%	-4.89%	-10.01%	-12.28%	10.15%	-0.27%	11.33%	6.32%
HyperMapper 2.0-FixDF	5.32%	1.21%	0.44%	2.67%	4.94%	4.86%	-0.20%	0.87%	1.40%	3.35%	1.18%	2.37%
Reinforcement Learning-FixDF	-0.75%	-4.13%	5.18%	-2.51%	-2.97%	-10.47%	0.67%	1.66%	4.62%	0.50%	-0.50%	-0.79%
Random Search-Codesign	-0.07%	-0.29%	0.14%	0.44%	0.33%	0.57%	0.23%	0.91%	0.48%	-0.24%	0.02%	0.23%
HyperMapper 2.0-Codesign	0.56%	0.46%	0.59%	0.64%	0.68%	0.68%	0.72%	0.76%	0.43%	0.52%	0.73%	0.62%
ExplainableDSE-FixDF	53.54%	21.92%	20.48%	52.42%	15.32%	31.74%	23.73%	21.54%	30.96%	40.66%	21.44%	30.34%
ExplainableDSE-Codesign	30.50%	23.45%	32.10%	32.03%	18.77%	46.29%	27.03%	18.78%	26.19%	47.30%	46.70%	31.74%

constraints, the DSE can find outperforming solutions. However, it may still converge to a suboptimal solution (e.g., for VGG-16) due to the greed for resolving the bottlenecks for further optimizations. This challenge can be addressed by making the acquisitions more exploratory, e.g., exploring distant promising subspaces. It can be achieved by exploring multiple spaces side-by-side by targeting a pool of various initial points [51]. Alternatively, the acquisition function can incorporate both self-supervised learning or inducing randomness [31] and bottleneck/constraint considerations.

• **Exploring and addressing implications of specifying ineffectual bottleneck analysis for an arbitrary system:** In Explainable DSE, the bottleneck model helps explain design cost and guide the DSE. It considers which factors constitute overall cost and how design parameters and their scaling could impact each factor. For various domain-specific systems, designers usually characterize them manually and develop domain-specific bottleneck models/analyses or have first-hand information [24, 26, 28, 67, 70, 84]. Even for designing deep learning accelerators, designers already develop cost models or closely work with them [15, 44, 53, 54, 58, 68, 74, 76, 81, 85, 86]. Thus, they can determine bottleneck model and mitigation in various ways. For instance, designers can obtain a bottleneck model by simplifying their analytical cost model, which they can provide along with the cost model. Alternately, designers could estimate bottleneck mitigation through characterization or sensitivity analysis of design parameters. Designers can also opt for automation techniques, as discussed later.

A domain-specific bottleneck model constructed by domain experts or possibly learned from domain-specific data can lead to an effective DSE. However, in the absence of an effectual analysis for an arbitrary system, the resulting exploration may be slower or suboptimal. For instance, the DSE may require more acquisitions and be slower if the designer-provided mitigation scales the parameter values conservatively or associates irrelevant parameters with bottleneck factors (e.g., the total number of PEs to the DMA time). Conversely, the DSE may converge to suboptimal solutions if the designer either skips associating a critical parameter with a bottleneck factor or scales values aggressively. The examples include a) a user not suggesting explorations of NOC links or bit-widths when on-chip communication time is the bottleneck and b) the DSE scaling the number of PEs by 2× or more, even if the required scaling was only 1.2×. Either can cause the DSE to miss a range of efficient and constraints-satisfying solutions.

• **Automating construction of bottleneck models and bottleneck mitigation for arbitrary or large-scale domain-specific systems:** For expert-defined cost models, such as those of DNN accelerators, manual bottleneck analysis by designers with first-hand domain information can be possible. In general, when domain-specific architectures can be described and evaluated as flow graphs, the analysis of costs/bottlenecks may be automated by parsing execution information for architectural components [16, 17, 77].

There can be scenarios where designers may want to optimize the application processing for off-the-shelf processors or can only access pre-existing design models and simulators that are large-scale or complicated. In such cases, designers may not be able to provide the domain-information for constructing bottleneck models and available designs or large-scale simulation models (e.g., in C++/RTL) need to be used for deriving bottleneck models. Hence, learning-based approaches can be developed [43], which could be broadly applicable, while still leveraging the proposed structure/organization of the bottleneck models and their usage in gray-box/white-box design space exploration. Graph-based ML models or self-supervised ML models can be more suitable, including but not limited to decision trees, graph neural networks, and reinforcement learning. Likewise, bottleneck mitigation in complicated scenarios can be estimated using gray-box optimization functions that approximate the relevance and contributions of each parameter to the total cost [62] or through surrogates [31].

Our dynamic DSE evaluation demonstrates the potential of incorporating explainability into the DSE. Efficiency, agility, and generalization of the DSE can be improved even further through improving the predictions for bottleneck mitigation, the decision mechanism that uses the feedback, aggregation and acquisition functions, and parallel evaluation/exploration of candidates and promising subspaces.

D SPECIFICATION NEEDED BY DSE APPROACHES

Black-box optimizations such as random search, simulated annealing, or Bayesian optimization are easy to deploy and require minimal tuning and specification for meaningfully curating and constraining the design space, which may take some hours or days. However, due to their non-explainable nature, they may mostly explore inefficient or infeasible solutions while consuming significant search time for excessive sampling. Applying these black-box

exploration approaches to a domain-specific design optimization problem may still need some additional specification efforts [36, 37], depending on their implementation. For instance, Confuciux [36] is a reinforcement learning-based DSE framework that uses an LSTM/MLP-based policy network. For the targeted evaluations, we extended it to allow an arbitrary number of parameters, a different environment (target setup), different numbers of possible options for different parameters (different list sizes), and an arbitrary number of constraints. We also extended its reward calculation to consider an arbitrary number of constraints and their utilization. This extension required adding/modifying a few tens of lines of code (LoC) and a few days of work.

For bottleneck-guided DSE, we developed a bottleneck model for the target domain of DNN accelerator's hardware/mapping codesign. This is similar to efforts made previously in other domains [24, 26, 28, 67, 84]. The bottleneck model was expressed to the DSE framework via proposed API. It led to about tens of LoC that specified how different factors contribute to overall cost and a few to several lines for integrating first-hand information about bottleneck mitigation that provided predictions for new values of each parameter. It is worth noting that this is significantly less code and development efforts compared to domain-specific analytical cost models [15, 44], which typically require thousands of lines of code and provide only a single value for the total cost. The bottleneck model used in the DSE is generally simpler than the full analytical model, as it only considers major execution-related factors. And to infer new parameter values for bottleneck mitigation, it incorporates simple estimates or performs a walk-through of the populated values/paths in the bottleneck graph. In general, domain experts can derive the bottleneck model from either the graphical representation of the analytical model they develop/use or the sensitivity of a cost to the design parameters, which may take several days. §C discusses how bottleneck mitigation may be generalized for arbitrary systems.

E CASE STUDY: EFFICIENCY OF THE DESIGNS ACHIEVED BY DSE

Methodology: In this study, we compare the efficiency of designs obtained by our DSE (§6; Fig. 9) to those of edge AI accelerators. The DSE can only explore designs for the spatial architecture templates used by the cost models (e.g., in [15, 44, 53, 80]). Therefore, to make a fair comparison, we only considered edge accelerators with similar architectural features. Specifically, we compare with two edge accelerators: 1) Coral Edge TPU [47], a state-of-the-art industrial edge accelerator platform developed by Google, and 2) Eyeriss [7], an efficient edge accelerator that incorporates several special optimizations for high energy efficiency and low latency.

While novel edge accelerators have been developed recently, they typically contain several (micro)architectural features for specialization (e.g., mixed-precision computations or sparsity exploiting features [13]) that are not modeled (accurately) by the commonly used/available latency models for edge AI accelerators. The architectures considered here are suitable for the comparison, as they are commonly used as a template in the system-level tools for design and execution modeling [15, 31, 36, 44, 53, 83]. Table 4 compares the architectural features and execution optimizations for the Edge TPU, Eyeriss, and the template architecture used by the DSE cost models. We obtained the results for Edge TPU from the performance benchmarking of TPU-optimized models [11], and the results for Eyeriss chip from its evaluations [7].

Table 4: Comparison of architectural features and execution optimization for edge AI acceleration.

Feature	Edge TPU	Eyeriss	DSE
Data Precision	8b*	16b	16b
Technology	-	65nm	45nm
PEs	Scalar MAC	Scalar MAC	Scalar MAC
Temporal Reuse	Yes	Yes	Yes
Spatial Reuse	Data Distribution and Reduction	Data Distribution and Reduction	Data Distribution
Mapping Optimizations	Automatic (TFLite)	Fixed-Style	Automatic
Hardware-Mapping Co-optimization	No	Yes	Yes
Accelerator-DNN Codesign	Yes†	No	No
Frequency (MHz)	125-500	100-250	500

∗ Edge TPU results are scaled to match 16b precision used in comparison.
† For Edge TPU, the EfficientNet-EdgeTPU model is codesigned.
¶ In lack of information about the actual power consumed by edge TPU for different models, 1.4W power is considered (as reported for MobileNetV2 in the edge TPU datasheet).

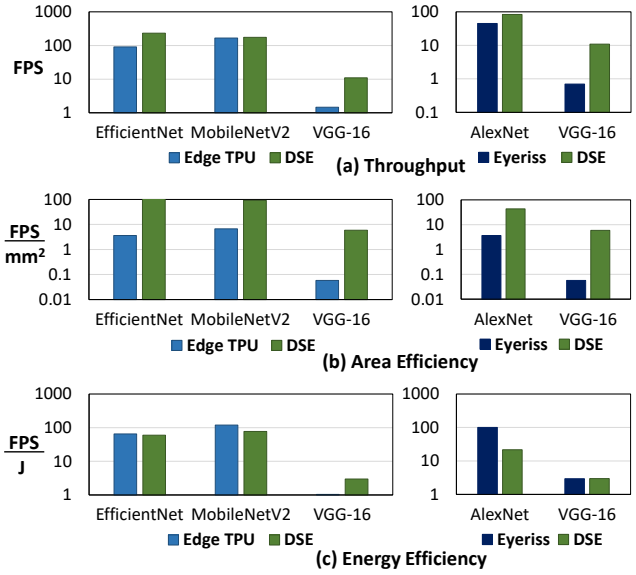

Figure 14: Comparison of efficiency of the designs obtained by the DSE with that of the edge accelerators Google Coral Edge TPU [47] and Eyeriss [8]: (a) Throughput (frames per second i.e., FPS) [3.7×, 8.7×], (b) Area efficiency (FPS/mm^2) [49×, 57×], and (c) Energy efficiency (FPS/Joule) [1.5×, 0.6×].

Results: Fig. 14 shows performance achieved by all three designs and the resultant energy efficiency and area efficiency. The results demonstrate that, on average, codesigns obtained by the DSE attain 3.7× higher throughput than the Edge TPU. It is due to the DSE's ability to analyze execution bottlenecks and optimize

Table 5: Execution cost models for deep learning accelerators.

Features	Timeloop [53]	dMazeRunner [15]	MAESTRO [44]	Interstellar [83]	SCALE-Sim [61]	Accelergy [80]
Performance Estimate	Yes	Yes	Yes	No	Yes	No
Energy Estimate	Yes	Yes	Yes	Yes	No	Yes
Integrated Support for ML Libraries	No	Yes	Yes	No	No	N/A
Layers Supported	GEMM, CONV	GEMM, CONV, DWCONV	GEMM, CONV, DWCONV	GEMM, CONV	GEMM, CONV	N/A
Data Precision	Variable	Variable	Variable	Variable	Fixed	Variable
Mapping Specification	Loop Nest Configuration	Loop Nest Configuration	Directives	Loop Nest Configuration	N/A	N/A
Dataflow	All	All	All	All	WS, OS, IS	N/A
Spatial and Temporal Data Reuse	Yes	Yes	Yes	Yes	Yes	N/A
Data Reuse with Striding Convolution	Yes	No	Yes	N/A	N/A	N/A
Considers On-Chip Communication Latency	Yes	Yes	Yes	No	N/A	N/A
Memory Hierarchy	Arbitrary	3-level	3-level	3/4-level	Fixed	Arbitrary
Models overhead of non-contiguous accesses	No	Yes	No	No	No	No

Table 6: Mappers for deep learning accelerators.

Features	Timeloop [53]	Gamma [37]	Mind Mappings [31]	CoSA [35]	Interstellar [83]	ZigZag [49]	dMazeRunner [15]
Comprehensive Mapping Space	Yes	No	Yes	No	Yes	Yes	Yes
Discard Invalid Mappings for Mapping-Space Construction	Yes	No	No	Yes	Yes	Yes	Yes
Prune Inefficient Tilings	No	No	No	No	No	Yes	Yes
Prune Inefficient Orderings	No	No	No	No	Fixed	Yes	Yes
Layers Supported	CONV, GEMM	CONV, GEMM, DWCONV	CONV, MTTKRP	CONV, GEMM	CONV, GEMM	CONV, GEMM	CONV, GEMM, DWCONV
Exploration Approach	Random	Genetic Algorithm	Gradient Descent	Mixed Integer Programming	Heuristic	Heuristic	Heuristic
Uneven Tiling for Tensors	No	No	No	No	No	Yes	No
Search Time	Minutes	Minutes	Minutes	Seconds	Minutes	Hours	Seconds
Off-line Training	No	No	Yes	No	No	No	No

both mappings and hardware configurations, such as NoC configurations/bandwidth and buffer sizes. The DSE-achieved designs also required more than an order of magnitude less on-chip area, presumably due to allocating significantly smaller buffers and fewer MAC units (e.g., considering Edge TPU configurations studied in [64]). The overall area efficiency is higher by about 14× for MobileNetV2 and 49× on average due to a high-throughput execution for VGG-16. Although the DSE was focused on minimizing latency, energy efficiency of its designs matches that of the EfficientNet-Edge TPU codesign, and is even higher by 2.9× for VGG-16.

When compared to Eyeriss, the DSE achieves designs of lower latency (with similar area/power budgets), and improves area efficiency by up to 11.84× and energy efficiency of VGG-16 by up to 1.33×—while not incorporating additional efficiency-gaining features as in Eyeriss. For instance, Eyeriss leverages frequency scaling, power gating, zero skipping, and compression (for up to 86% sparsity in AlexNet layers). Thus, Eyeriss achieves about 2×–3.75× higher energy efficiency when compared to the designs obtained by DSE that minimized latency. In most cases, the codesigns obtained by DSE in a tightly coupled manner outperform the Eyeriss-like designs. These comparisons demonstrate the potential of the proposed capabilities for the accelerator system design. With the continued development of automated execution modeling and inclusion of more template architectures and specialization components, proposed methodology can be expected to enhance efficiency further.

F CONSTRUCTING EFFECTUAL MAPPING SPACE AND BLACK-BOX MAPPERS

Background: The process of optimizing mappings of a DNN layer on an accelerator design involves exploring an enormous search space, as demonstrated by previous works [18, 37] and later summarized in Table 7. This search space is primarily composed of configurations for loop tile sizings and loop orderings. Loop tilings determine the data bursts that need to be accessed via memory hierarchy and spatial parallelism, whereas loop orderings determine the data reuse and impact the total memory accesses. For

Table 7: Analyzing size of the mapping space for deep learning accelerators.

Model	Layer	Tile Sizings	Tile Sizings with Valid Factors	Valid Tilings w.r.t. Hardware	Orderings at a Memory Level	Orderings with Unique/Max Data Reuse	Full Map. Space	Factorization-Constrained Mapping Space	Factorization-Constrained Reuse-Aware Map. Space
		A	B	C	D	E	F: A*D^2	G: B*D^2	H: B*E^2
ResNet18	CONV_2_1a	$O(10^{25})$	$O(10^{13})$	$O(10^{7})$	$O(10^{4})$	15/3	$O(10^{32})$	$O(10^{20})$	$O(10^{14})$
MobileNetV2	features.2.conv.0	$O(10^{22})$	$O(10^{12})$	$O(10^{6})$	$O(10^{4})$	15/3	$O(10^{30})$	$O(10^{19})$	$O(10^{13})$
EfficientNetB0	blks.2.expand	$O(10^{22})$	$O(10^{12})$	$O(10^{6})$	$O(10^{4})$	15/3	$O(10^{29})$	$O(10^{20})$	$O(10^{13})$
VGG-16	CONV_1_2	$O(10^{28})$	$O(10^{14})$	$O(10^{7})$	$O(10^{4})$	15/3	$O(10^{36})$	$O(10^{21})$	$O(10^{15})$
ResNet50	CONV_2_1b	$O(10^{25})$	$O(10^{13})$	$O(10^{7})$	$O(10^{4})$	15/3	$O(10^{32})$	$O(10^{20})$	$O(10^{14})$
Vision Transformer	patchembeddings. CONV2D	$O(10^{25})$	$O(10^{13})$	$O(10^{6})$	$O(10^{4})$	15/3	$O(10^{32})$	$O(10^{20})$	$O(10^{14})$
FasterRCNN-MobileNetV3	features.12. conv2.excite	$O(10^{26})$	$O(10^{13})$	$O(10^{6})$	$O(10^{4})$	15/3	$O(10^{33})$	$O(10^{20})$	$O(10^{14})$
YOLOv5	features.1.conv	$O(10^{27})$	$O(10^{14})$	$O(10^{7})$	$O(10^{4})$	15/3	$O(10^{34})$	$O(10^{21})$	$O(10^{15})$
Transformer	decoder. output_projection	$O(10^{27})$	$O(10^{9})$	$O(10^{4})$	$O(10^{1})$	3/3	$O(10^{28})$	$O(10^{10})$	$O(10^{10})$
BERT	encoder.layer.0. output.dense	$O(10^{26})$	$O(10^{9})$	$O(10^{5})$	$O(10^{1})$	3/3	$O(10^{27})$	$O(10^{11})$	$O(10^{10})$
Wav2Vec2	encoder.layers.0. intermediate.dense	$O(10^{28})$	$O(10^{12})$	$O(10^{6})$	$O(10^{1})$	3/3	$O(10^{29})$	$O(10^{13})$	$O(10^{12})$

mapping DNN operators such as convolutions and multi-layer perceptrons on an accelerator with a 3-level buffer/memory hierarchy and 1-level spatial parallelism [8], the mapping space corresponds to 28-deep and 12-deep nested loops, respectively [15]. Tile sizings are configurations that represent the values for loop iterations at each temporal/spatial level in the architectural processing hierarchy. For a selected tiling configuration, processing of a 7-deep nested loop at a buffer level corresponds to 7! different loop orderings. Table 7 lists the DNNs and their layers with the largest search space, which can contain up to $O(10^{36})$ configurations.

Pruning invalid loop tilings: To optimize the tile sizes of a DNN layer using a black-box optimizer, designers often set the index variables of tiled loops as design parameters and specify lower/upper bounds (loop iterations) to define the range of values that the optimizer can explore [31, 37]. However, the search space for tile sizes can be enormous, containing as many as $O(10^{28})$ configurations for certain DNN layers, as shown in Table 7. A black-box optimizer is unlikely to find more than a few valid mappings under practical exploration budgets for exploring such a large search space [35]. This is because, for each loop in a DNN operator, only a small subset of factors of loop iterations lead to valid tile sizes. For example, a set (8, 4, 2, 16) makes a valid 4-level tiling configuration for 1024 loop iterations over output activations/filters, while many more do not. Therefore, designers formulate the space of valid tiling sizes based on factorization of loop iterations [35, 53]. As shown in Table 7, this prunes the search space of tiling configurations by a square root or even a cube root, e.g., from $O(10^{22})$–$O(10^{28})$ configurations to a much smaller range of $O(10^{9})$–$O(10^{14})$ configurations. Invalid tiling configurations, which require more architectural resources than are available in the target hardware configuration, are usually discarded by black-box optimizers during the exploration (as indicated in column *C* of Table 7).

Pruning ineffectual loop orderings: We further reduced the space by discarding ineffective loop orderings. Previous black-box mappers explored all orderings, resulting in a large number of configurations (7! or $O(10^{4})$) for processing a convolution at a memory level in the memory hierarchy. Instead, our approach build upon previous research that has shown only a handful of loop-orderings having unique data reuse of tensors (15 for convolutions) and a few with maximum reuse of various tensors [15, 49].

Overall mapping space: The GAMMA-like mapper considers full (non-factorized) tiling space and all loop-orderings [37] (column *F*), while Timeloop [53] and CoSA [35] consider factorized tile sizes but all loop orderings. For evaluations with black-box mappers, we used factorized tile sizes (all valid factorization) and considered only loop orderings with unique data reuse. In practice, there are only a few unique data reuse scenarios (vs. 15/3) for each tiling configuration, so all of them can be explored linearly [15]. Therefore, we invoked black-box optimizations with 10,000 trials for mapping each layer on a hardware configuration. During each trial, the optimization acquired a tiling configuration and evaluated all effectual loop-orderings through the cost model, selecting the one that minimized the objective. Thus, the mapping space formulation discarded a large number of invalid and ineffective configurations (column *H*) without compromising the optimality.

Selection of the mapping optimization technique: Evaluations comparing black-box DSEs of hardware configurations with fixed mapping schema showed that random search and Bayesian optimization-based HyperMapper 2.0 [51] were the most effective, as depicted in Figure 9. As a result, we selected these two techniques for optimizing both the hardware and mapping configurations. However, when we applied them to optimize mappings from the pruned space (column H in Table 7), we found that random search obtained efficient mappings within several seconds. By contrast, HyperMapper 2.0 generated efficient mappings, but its search overhead was prohibitively high, requiring a few hours to evaluate just a single DNN layer. Consequently, we used random search for optimizing mappings during the codesign DSE.

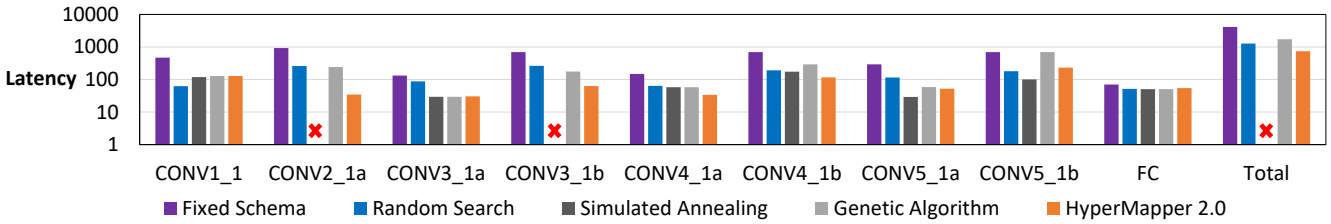

Figure 15: Efficiency of mappings obtained by different black-box optimizations for accelerating ResNet layers.

We also compared the quality of mappings obtained by random search with those obtained by simulated annealing, genetic algorithm, and Bayesian optimization for ResNet18 layers, as shown in Fig. 15.[6] The random search successfully achieved low-latency mappings for all layers, whereas simulated annealing [75] failed to map a few layers (in 10,000 trials), and genetic algorithm [3] led to a higher overall latency than random search and took almost four hours to optimize mappings for the nine layers, making it impractical for codesign search. Therefore, we opted to use a Timeloop-like random search to quickly and efficiently explore the highly pruned mapping space.

G CODESIGN OPTIMIZATION: MULTI-STAGE OR JOINT?

The optimization of hardware and software codesigns can be done either by exploring partitioned sub-spaces in a sequential manner or simultaneously. In a partitioned or a two-stage optimization, an outer loop iterates over different hardware configurations, and an inner loop optimizes the software space for each hardware configuration selected. On the other hand, the joint or simultaneous exploration involves finding a new configuration for both the hardware and software parameters at the same time in a trial. Although approaches using simultaneous search have been proposed, they are often infeasible to apply to a multi-workload exploration, target system with diverse and time-consuming cost evaluations, and huge collective search space. Therefore, partitioned sub-space exploration is commonly used for optimizing codesigns (§A.2). For demonstration of Explainable-DSE, all our DSE evaluations also follow two-stage optimization.

Firstly, approaches using simultaneous search [39, 51] typically optimize configurations for individual loop kernels such as a single DNN layer, as they optimize both the hardware and software parameters at every search attempt. It does not necessarily lead to a single accelerator design that is most efficient for the entire workload or a set of workloads, as layer-specific designs may not be optimal overall for the entire DNN or multiple DNNs.

Furthermore, simultaneously optimizing both hardware and software parameters can be very time-consuming. A target system often involves different cost functions or modules for different metrics that could consume different evaluation times. For example, evaluating area and power of each hardware configuration via Accelergy [80] alone could take a few seconds, whereas the cost models of dMazeRunner [15] or Timeloop [53] could estimate latency/energy for hundreds–thousands of mappings in a second. For exploring codesigns for a DNN with L=50 unique layers, consider a black-box DSE that is budgeted H=2,500 trials for hardware configurations and M = 10,000 trials for mapping each DNN layer on each hardware configuration. Simultaneous exploration of hardware and software configurations in $H \times M$ trials for each of the L layers requires the system to evaluate power/area costs for $H \times M \times L$ times, which would take more than 0.7 million hours, or 79 years! In contrast, a two-stage partitioned exploration evaluates power/area costs only for H trials and if the DSE samples infeasible mappings for a hardware configuration, they can be discarded promptly without further detailed evaluation. Our experiments show that the black-box DSEs obtained codesigns in a few days to a few weeks with exploring the partitioned subspaces.

Finally, in addition to the design parameters such as the total PEs or buffer sizes, hardware configurations can have various parameters, such as bandwidth, reconfiguration of NoCs (time-multiplexed communication of data packets, bus widths), and those for architectural specialization/heterogeneity, which further increase the search space for both the hardware and software/mapping configurations. With the vast space for both the hardware and software/mapping configurations, the collective search space becomes huge, compounding the already challenging exploration of feasible and effective solutions for either of the hardware and software parameters. Additionally, in the DSE trials, simultaneously acquired hardware and software configurations may not be compatible with each other or may not mitigate execution inefficiencies corresponding to their counterpart.

Received 20 October 2022; revised 2 March 2023; accepted 27 April 2023

[6] The mappings were evaluated for the initial hardware configuration, corresponding to the lowest values of design parameters in Table 1.