Automotive Science and Engineering, Vol. 16, No. 3, (2026), 5161-5177

Iran University of Science and Technology

Automotive Science and Engineering

Journal Homepage: ase.iust.ac.ir

ISSN: 2008-9899

Deep Learning-Based Cache Configuration Prediction for Field Programmable Gate Array-Accelerated Smart Vehicles

Mohammad Cheshfar[1], Mohammad Hossein Maghami[1*], Hossein Gharaee Garakani[2] and Parviz Amiri[1]

[1]Faculty of Electrical Engineering, Shahid Rajaee Teacher Training University, Tehran, Iran

[2]Department of Information Technology, ICT Research Institute, Tehran, Iran

ARTICLE INFO

Article history:

Received: 11 Jan 2026

Accepted: 7 Aug 2026

Published: 5 Sep 2026

Keywords:

Smart Vehicles

Cache design

FPGA

Deep Learning

A B S T R A C T

This paper presents a machine-learning framework for cache-configuration prediction in embedded Field Programmable Gate Array (FPGA) systems, with a particular focus on smart-vehicle and resource-constrained computing applications. The proposed method formulates cache tuning as a regression problem and predicts the number of sets, associativity, and block size from memory-access traces together with FPGA resource budgets. Unlike classification-based approaches limited to predefined cache classes, the proposed CNN–LSTM model supports continuous cache-space exploration and produces hardware-feasible configurations after discretization. Experimental results on MachSuite, PolyBench, and CHStone benchmarks demonstrate effective generalization to unseen workloads, achieving a validation MAE of 0.085 and a test-set miss-rate MAE of 0.064. On a Xilinx Zynq-7020 target, the predicted configurations reduced average CLB utilization from 19.13% to 15.38% of available resources, increased BRAM utilization from 59.86% to 70.00%, and lowered estimated on-chip power from 0.201 W to 0.196 W under XPE-based pre-implementation analysis. These results indicate that the proposed approach can significantly reduce the cost of exhaustive design-space exploration while preserving cache performance and hardware efficiency, making it a promising solution for embedded FPGA platforms used in smart-vehicle systems.

1. Introduction

Smart vehicles are increasingly viewed as a key enabling technology for modern transportation systems because they can improve road safety, reduce congestion, and support more efficient mobility [1], [2], [3]. As vehicle intelligence advances, applications such as autonomous driving, driver assistance, and in-vehicle perception must process large amounts of sensor and computational data under strict latency, energy, and area constraints [4], [5]. In this setting, embedded hardware platforms must provide both performance and efficiency, since these systems often operate in resource-constrained environments where real-time response is essential [6].

*Corresponding Author
Email Address: mhmaghami@sru.ac.ir
https://doi.org/10.22068/ase.2026.733

"*Automotive Science and Engineering*" is licensed under a Creative Commons Attribution-Noncommercial 4.0 International License.

[Downloaded from idea.iust.ac.ir on 2026-11-16]

Machine learning and deep learning have become central to many smart-vehicle functions, including object detection, scene interpretation, and control support. These workloads are computationally intensive and place substantial pressure on the memory hierarchy, especially when processing continuous streams of data from cameras, radar, lidar, and other sensors. Field-Programmable Gate Arrays (FPGAs) offer an attractive implementation platform for such tasks because they combine parallel execution, reconfigurability, and energy efficiency. However, FPGA-based accelerator design remains challenging due to limited on-chip resources such as BRAM, CLB, and DSP blocks, as well as the difficulty of exploring a large design space under multiple constraints.

A major bottleneck in FPGA accelerators is the memory subsystem. In many data-intensive applications, memory movement and cache behavior dominate execution time and energy consumption. As a result, selecting appropriate cache parameters, including the number of sets, associativity, and block size, is critical for achieving a good balance between performance and hardware cost. At the same time, cache tuning must account for application-specific access patterns and device-specific resource budgets, which makes manual design-space exploration both time-consuming and difficult to scale.

Although prior studies have investigated cache optimization and machine-learning-assisted design-space exploration, several limitations remain in the context of FPGA-based embedded platforms and smart-vehicle applications. In particular:

- Existing studies often rely on predefined cache classes or discrete configuration spaces, which limits flexibility and makes it difficult to adapt to unseen workloads.
- Hardware-resource constraints are rarely incorporated directly into the prediction process, even though CLB and BRAM availability strongly influence the feasibility of a cache design.
- The interaction between quantized cache parameters, miss-rate estimation, and FPGA memory primitives is often insufficiently discussed.
- There is limited analysis of cache prediction under smart-vehicle-style resource constraints, where performance and efficiency must be balanced carefully.

To address these limitations, this paper investigates a learning-based approach for application-aware cache configuration in FPGA-based smart vehicle systems. The proposed method utilizes memory-access traces to capture workload behavior and combines this information with hardware resource constraints to estimate suitable cache parameters. A learning-based modeling strategy is employed to extract meaningful patterns from sequential access data, enabling the system to relate memory behavior to efficient cache configurations. By focusing on application-driven characteristics and hardware feasibility, the approach aims to reduce reliance on exhaustive design-space exploration while maintaining effective performance.

This paper makes the following contributions:

- It proposes a direct CNN–LSTM regression framework that predicts cache sets, associativity, and block size from memory-access traces and FPGA resource budgets.
- It formulates cache prediction in a continuous output space and then projects the outputs onto valid hardware-supported configurations through discretization.
- It evaluates the predicted configurations not only in terms of miss-rate behavior, but also in terms of CLB usage, BRAM usage, residual error patterns, and estimated on-chip power on a Xilinx Zynq-7020 target.
- It frames the study around embedded automotive design requirements, where resource-aware cache selection is important for latency-sensitive and energy-constrained FPGA deployment.

The remainder of this paper is organized as follows. Section 2 reviews related work on cache optimization and FPGA-based system design. Section 3 presents the proposed methodology, including dataset preparation and model formulation. Section 4 describes the experimental

[Downloaded from idea.iust.ac.ir on 2026-11-16]

setup and evaluation process. Section 5 discusses the results and their implications for smart vehicle applications. Finally, Section 6 concludes the paper and outlines directions for future research.

2. Related works

There has been considerable research on the use of FPGAs in smart vehicles and embedded intelligent systems [7]. Prior studies have explored FPGA-based implementations for a wide range of applications, including robotics, computer vision, and autonomous driving. Survey works such as [8] provide comprehensive overviews of FPGA-based accelerators for robotic systems, while [9] reviews the evolution of FPGA-supported computer vision techniques in automotive applications. Additional studies [10], [11] highlight the growing importance of FPGA platforms in next-generation vehicle systems, emphasizing their advantages in energy efficiency, parallel processing, and adaptability, as well as the associated design challenges.

Beyond application-level studies, a substantial body of work has focused on cache optimization and design-space exploration, which are critical for improving memory performance in embedded systems. Traditional approaches rely on trace-driven simulation techniques to evaluate cache configurations. Methods based on binomial trees exploit inclusion properties to analyze multiple configurations efficiently within a single pass [12], [13]. Subsequent improvements, such as CRCB-based methods and enhanced tag-matching strategies, have further reduced simulation overhead [14]. Other tools, including DEW and SCUD, refine replacement policies and memory-block handling to accelerate exploration compared to classical simulators such as Dinero IV [15], [16]. Although these approaches provide accurate evaluation, they often involve trade-offs between runtime efficiency, storage overhead, and implementation complexity.

More recently, machine learning techniques have been introduced to approximate cache configuration decisions, aiming to reduce the cost of exhaustive exploration. Several studies formulate the problem as a classification task, where predefined cache configurations are selected based on features extracted from execution traces. For example, the work in [17] employs Random Forest models trained on instruction-trace features to predict cache settings, achieving high classification accuracy of 96% after applying feature selection and dataset balancing techniques. However, such approaches are inherently limited by their dependence on discrete parameter classes, which restricts flexibility and may not generalize well to unseen configurations.

Other research efforts have explored hardware-assisted prediction mechanisms. In [18], cache configuration prediction is implemented directly on FPGA hardware using neural networks synthesized into logic, enabling runtime adaptation with reduced overhead. The approach achieved energy loss below 5% while using relatively limited hardware resources. While this approach demonstrates the feasibility of on-device prediction, it still relies on discretized representations and predefined configuration sets, which can limit adaptability in more complex design scenarios.

Data-driven approaches have also been applied to specific cache parameters. The study in [19] investigates the relationship between memory-access characteristics and optimal cache block size using data mining and neural networks. By analyzing spatial and temporal locality patterns in address traces, the authors demonstrate that features such as access distance and reuse behavior are strongly correlated with cache performance. A neural network with a 30-node input layer, two hidden layers, and a 3-node output layer achieved 95.24% accuracy. Although effective for parameter estimation, this line of work typically focuses on individual design variables rather than jointly optimizing multiple cache parameters under hardware constraints.

Despite these advances, several limitations remain in the context of FPGA-based embedded systems, particularly for resource-constrained applications. As summarized in Table 1, existing studies primarily formulate cache tuning as a classification problem and therefore restrict predictions to a finite set of predefined cache configurations. In addition, most approaches focus on individual cache parameters or rely on

[Downloaded from idea.iust.ac.ir on 2026-11-16]

Table 1 Comparison with prior machine-learning-based cache prediction studies

Feature	[17]	[18]	[19]	Proposed Work
Learning paradigm	Random Forest classification	ANN classification	Neural-network classification	CNN–LSTM regression
Primary objective	Cache configuration selection	Runtime cache adaptation	Cache block-size prediction	Joint cache-parameter prediction
Predicted parameters	Cache classes	Cache classes	Block size	Sets, ways, block size
Design-space representation	Discrete	Discrete	Discrete	Continuous-to-discrete
Hardware-resource constraints considered	No	Limited	No	Yes
Simultaneous optimization of multiple cache parameters	Limited	Limited	No	Yes
FPGA-aware optimization	No	Partial	No	Yes
Flexibility for unseen configurations	Limited	Limited	Limited	high

general-purpose workloads, limiting their ability to adapt to diverse memory-access behaviors. Hardware resource constraints, such as available BRAM and logic resources, are also rarely incorporated directly into the prediction process. These limitations motivate the need for a hardware-aware framework that can jointly optimize multiple cache parameters while adapting to both application-specific memory-access patterns and FPGA resource budgets. To address this gap, the present work formulates cache configuration as a regression problem and integrates memory-access information with hardware constraints to enable flexible and efficient cache selection across a broader design space.

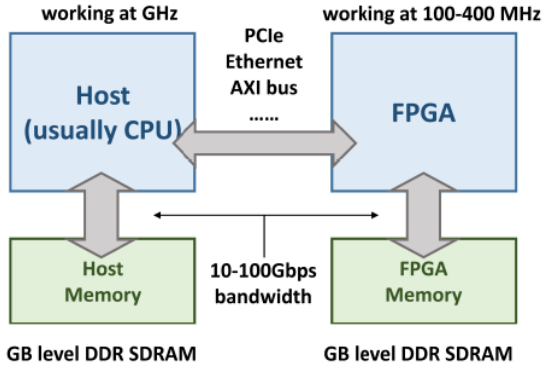


Figure 1: A typical structure of an FPGA accelerator

In contrast to prior work, this study focuses on application-aware cache configuration for FPGA-based smart vehicle systems, where both workload characteristics and hardware constraints play a central role. By leveraging memory-access traces and resource information, the proposed approach aims to provide a more flexible and scalable alternative to traditional exploration methods, supporting efficient cache design in embedded automotive platforms.

3. Methodology

This section presents the proposed framework for learning-based cache configuration in FPGA-based smart vehicle systems. The goal is to infer suitable cache parameters from memory-access behavior while accounting for hardware resource constraints and preserving feasibility on the target platform.

3.1. System overview

Figure 1 illustrates a representative FPGA-accelerated computing platform used in embedded and automotive applications. In this type of architecture, a host processor coordinates control and sequential tasks, while an FPGA performs hardware acceleration for compute-intensive operations. Depending on the deployment scenario, the FPGA may be implemented either as a discrete device connected through a high-speed interface such as PCIe or as part of a system-on-chip platform in which the processor and programmable logic share the same silicon substrate [7].

In discrete configurations, the host and accelerator may operate with separate memory spaces and exchange data through communication protocols and coherence mechanisms. In SoC-based designs, memory sharing is more direct, which can reduce communication overhead and simplify

[Downloaded from idea.iust.ac.ir on 2026-11-16]

integration. In both cases, performance is strongly influenced by memory-access efficiency, especially in workloads with frequent data movement and irregular access behavior.

For smart-vehicle workloads, this interaction between processing elements and the memory hierarchy is particularly important. Latency, bandwidth, and on-chip resource usage must be balanced carefully, making cache organization a central design concern. The proposed framework is therefore designed to support cache tuning under these hardware and workload constraints.

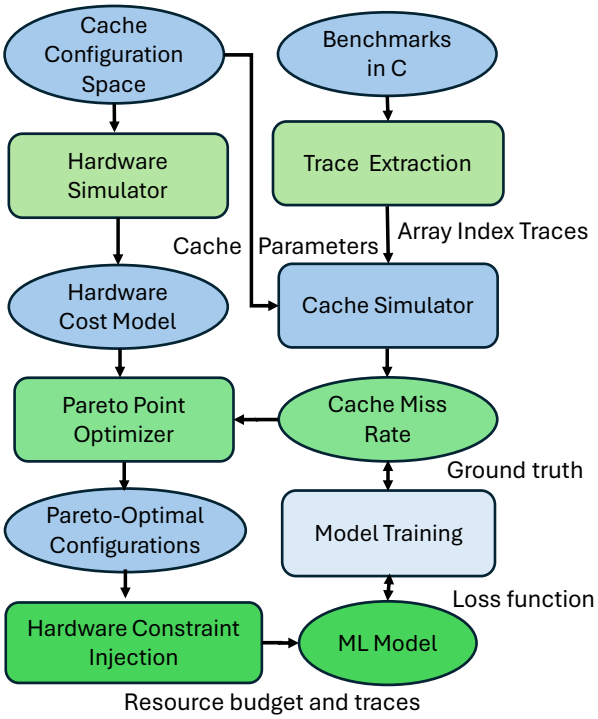


Figure 2: Proposed methodology flow

3.2. Proposed framework overview

The proposed methodology learns a mapping from application memory-access patterns and hardware constraints to feasible cache configurations. The overall workflow is summarized in Figure 2.

The process begins with the extraction of array-index traces from benchmark applications. These traces capture the access behavior that affects cache performance. Candidate cache configurations are then evaluated through simulation and hardware estimation to obtain performance and resource metrics. The resulting data are used to construct a training set in which each sample combines trace-derived behavioral information with FPGA resource constraints and the corresponding cache parameters.

A learning-based model is trained to approximate this mapping and generate cache configurations for unseen workloads. The purpose of this framework is to reduce dependence on exhaustive design-space exploration while maintaining compatibility with application-specific access behavior and hardware limitations.

3.3. Dataset generation

The effectiveness of the proposed framework depends on the quality and representativeness of the dataset used for training and evaluation. In this work, the dataset is constructed by combining memory-access traces extracted from benchmark applications with corresponding cache-performance and hardware-resource information. The objective is to establish a structured mapping between application behavior, expressed through memory-access patterns, and feasible cache configurations under FPGA resource constraints. This section describes the benchmark selection, trace extraction process, dataset organization, and filtering strategy used to generate the training and evaluation data.

3.3.1 Trace sources and benchmark selection

To construct a representative dataset for cache-configuration learning, we selected benchmark applications from three widely used suites: CHStone [20], Machsuite [21], and Polybench [22]. These benchmarks were chosen because they cover a broad range of computation patterns encountered in embedded and accelerator-oriented workloads, including signal processing, linear algebra, image processing, and control-style kernels. Using multiple benchmark families helps ensure that the training data are not biased toward a single access pattern or application structure, which is essential for evaluating cache behavior under diverse workload conditions.

The benchmark selection was motivated by two considerations. First, the chosen suites are commonly used in FPGA and high-level synthesis studies, which makes them suitable for studying cache behavior in resource-constrained

[Downloaded from idea.iust.ac.ir on 2026-11-16]

hardware platforms. Second, they provide sufficient variation in memory-access regularity, temporal locality, and access density to support learning a mapping between trace behavior and cache configuration. This diversity is particularly relevant for smart-vehicle-oriented systems, where memory access patterns may vary substantially across perception, signal-processing, and control workloads.

In this study, benchmark suites serve as trace sources rather than as end applications. The goal is not to model a single automotive workload directly, but to derive a structured set of memory-reference sequences that expose the cache predictor to a range of behaviors. This design supports cross-workload evaluation and allows the proposed method to learn from representative embedded computation patterns while remaining applicable to a broader class of FPGA-based smart-vehicle accelerators.

3.3.2 Access-pattern scope and trace extraction

To capture the memory behavior relevant to cache performance, array-index traces were collected during the execution of each benchmark application. The trace extraction process records the sequence of memory-access indices associated with array operations, providing a compact representation of access locality, striding patterns, and temporal dependencies. These characteristics are critical for determining cache efficiency, as they directly influence hit and miss behavior.

In principle, memory accesses can be categorized into read-only, write-only, and mixed read–write patterns, each affecting cache performance and resource utilization differently. In this study, we restrict the dataset to read-only access patterns in order to isolate the dominant factors influencing cache miss behavior and to simplify the modeling process. Read operations are particularly relevant in many data-intensive embedded workloads, where performance is often limited by data retrieval latency rather than write-back mechanisms. Focusing on read-only traces allows the model to learn clear relationships between access patterns and cache efficiency without introducing additional variability associated with write policies and buffering mechanisms.

While this restriction provides a controlled setting for analyzing cache behavior, it also defines the scope of applicability of the proposed framework. The current model is most suitable for read-dominated automotive workloads, such as inference-oriented perception pipelines, including object detection, lane detection, traffic-sign recognition, and other CNN-based accelerators, where memory accesses are largely driven by repeated reads of weights and feature maps. In contrast, workloads with frequent state updates or write-heavy memory activity, such as sensor-fusion buffers, state-estimation modules, SLAM pipelines, and control-loop updates, may exhibit cache behavior that is not fully captured by the present read-only traces. In these cases, write-back traffic, coherence effects, and replacement dynamics can influence miss rate and resource utilization more strongly than the current model accounts for [23]. Therefore, the proposed framework should be interpreted as a predictor for read-dominated cache workloads, while mixed read–write scenarios remain an important direction for future extension.

From a generalization perspective, extending the framework to mixed access patterns would require incorporating additional trace annotations to distinguish read and write operations, as well as modeling the impact of write policies (e.g., write-through and write-back) on cache behavior and resource usage. A possible extension involves augmenting the input representation with access-type encoding and retraining the predictor to jointly model read–write interactions. This direction would enable a more comprehensive evaluation of cache configurations in heterogeneous and dynamic memory-access scenarios typical of advanced automotive applications.

The trace extraction was performed through program instrumentation and execution profiling, ensuring that the collected sequences reflect realistic runtime behavior rather than static code structure. Each trace is associated with a specific kernel or application instance and preserves the

[Downloaded from idea.iust.ac.ir on 2026-11-16]

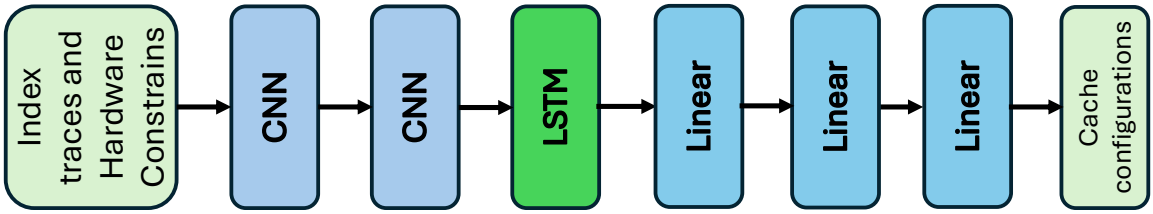


Figure 3: Model architecture

ordering of memory references, enabling the subsequent learning model to capture both local access regularity and longer-range dependencies.

3.3.3 Dataset schema and labeling

The dataset construction process follows the pipeline illustrated in Figure 2 and consists of three main stages: (i) extraction and preprocessing of memory-access traces, (ii) generation of performance and hardware-cost labels through cache-configuration exploration, and (iii) selection and augmentation of Pareto-optimal design points.

In the first stage, memory-access traces are extracted from benchmark applications and segmented into fixed-length samples. Each sample is represented as a sequence of 1000 delta values, where each element corresponds to the difference between consecutive memory addresses. If a trace is shorter than the required length, it is repeated until the sequence reaches the target size. This delta-based representation captures spatial locality and stride patterns while reducing dependence on absolute address values. All trace sequences are normalized using min–max scaling to the range [-1, 1] to ensure numerical stability during training.

In the second stage, a reference dataset of cache performance and hardware cost is generated through systematic exploration of the cache design space. The cache parameters are swept across a wide range, including number of sets (1–128), associativity (1–32), and block size (2–128). For each configuration, the cache miss rate is computed using a cycle-accurate simulator with an LRU replacement policy [17], while the corresponding FPGA resource usage (CLB and BRAM) is estimated using a validated hardware cost model derived from synthesis-based analysis [24] using AMD Vivado for the Zedboard [25] with an AMD Zynq™ 7000 FPGA. These measurements form the basis for labeling each trace–configuration pair.

To ensure that the dataset reflects only practically relevant design choices, a Pareto-based filtering step is applied. For each memory-access pattern, all evaluated configurations are grouped and compared across three objectives: miss rate, CLB usage, and BRAM usage. Only configurations that are not dominated in all three metrics are retained as Pareto-optimal points, representing efficient trade-offs between performance and hardware cost.

To improve the diversity and generalization capability of the dataset, several augmentation strategies are employed. First, trace-based samples are generated using the delta representation and fixed-length windowing described above. Second, for each Pareto-optimal configuration, additional samples are created by injecting variability into hardware constraints. Specifically, the available CLB and BRAM budgets are scaled within the range [50%, 150%] using geometric sampling, generating multiple hardware-constrained instances per configuration. In the reported implementation, 16 additional samples are generated for each Pareto point. This augmentation enables the model to learn how cache configurations adapt to varying resource budgets.

Each final dataset sample is therefore composed of: (i) a normalized delta-based memory-access sequence of length 1000, (ii) two hardware constraint features representing CLB and BRAM budgets, and (iii) the corresponding cache parameters (sets, ways, and block size) derived from Pareto-optimal configurations.

To facilitate efficient model training, the miss-rate values associated with each configuration are stored in a precomputed lookup table, allowing fast retrieval during optimization and enabling

[Downloaded from idea.iust.ac.ir on 2026-11-16]

the use of composite loss functions that incorporate both prediction error and hardware-constraint penalties.

The dataset is constructed using MachSuite and PolyBench benchmarks for training and validation, and CHStone benchmarks for testing, ensuring a program-level separation between seen and unseen workloads. In total, more than 20,000 samples are generated for training and validation, which are randomly split in an 80:20 ratio, while maintaining kernel-level separation to prevent data leakage. The test set consists of approximately 5,000 samples derived exclusively from CHStone applications, enabling a robust evaluation of generalization to unseen workloads.

This structured dataset provides a reproducible and application-aware mapping between memory-access behavior, hardware constraints, and cache configurations, forming the foundation for training the proposed learning-based predictor.

3.4. Model architecture

As shown in Figure 3, the proposed predictor adopts a hybrid neural architecture that combines convolutional and recurrent components to map memory-access behavior and hardware constraints to cache configurations. The input to the model consists of normalized memory-access traces together with FPGA resource budgets, specifically the available BRAM and CLB values. This combined representation allows the network to learn not only the locality structure of the trace but also the hardware feasibility context in which the cache must be implemented.

Memory-access traces can be viewed as structured sequential signals in which both short-range locality and long-range reuse behavior influence cache performance. Consequently, the architecture combines convolutional and recurrent modules to capture these complementary characteristics. The first part of the model consists of two one-dimensional convolutional layers designed to extract local access patterns, such as stride regularity, short-range locality, and repeated memory-reference motifs. In particular, the convolutional filters capture recurring structures associated with spatial locality and cache-line utilization. Batch normalization and nonlinear activation are applied after each convolution to stabilize training and improve feature extraction.

The resulting feature maps are then processed by an LSTM layer to capture longer-range dependencies within the access sequence. This recurrent component is important because cache behavior depends not only on immediate access patterns but also on the temporal ordering and reuse relationships across a larger execution window. In this sense, the CNN layers extract localized structural information, while the LSTM summarizes its temporal progression and preserves sequential context. The combination therefore provides a suitable representation for cache-configuration prediction, where both local reuse and long-horizon access dependencies affect miss-rate behavior.

After sequential feature extraction, the learned representation is passed to a set of fully connected layers that regress the three cache parameters: number of sets, associativity (ways), and block size. These layers produce continuous-valued parameter estimates, which are subsequently projected onto valid hardware-supported configurations during post-processing.

Because the proposed framework relies on precomputed lookup tables for miss-rate and hardware-cost estimation, cache configurations must correspond to synthesizable and previously evaluated design points. In practice, these design points are generated using discrete cache organizations, typically based on power-of-two geometries, which are widely adopted in FPGA cache implementations due to their compatibility with indexing logic, memory packing, and synthesis-tool behavior [26]. Consequently, discretization is not treated as an arbitrary post-processing operation, but rather as a necessary projection from the continuous regression space to the feasible hardware design space.

The CNN–LSTM model produces continuous-valued estimates for the cache parameters. To achieve a hardware-realizable mapping, the predicted values are converted to the nearest valid configurations using operations such as $\log_2 \rightarrow \text{round} \rightarrow 2^{(\cdot)}$ for parameters constrained to power-of-two representations, while other parameters are rounded to the nearest synthesis-

[Downloaded from idea.iust.ac.ir on 2026-11-16]

supported integer values. The discretized configurations are then used for all hardware-resource, miss-rate, and power-consumption lookups, ensuring that the reported system-level evaluation metrics inherently include the effects of quantization and hardware realizability. The reference cache configurations are generated from valid hardware-supported designs and therefore require no additional quantization.

Although modern FPGA memory primitives, such as Intel M20K/M9K memories [26] and Xilinx BRAM generators [27], support flexible width and depth configurations, practical synthesis flows often allocate complete BRAM or URAM primitives and impose packing constraints that approximate power-of-two organizations. Prior FPGA cache implementations have therefore employed both power-of-two and non-power-of-two configurations depending on design goals and synthesis behavior [28]. In the present work, power-of-two organizations are adopted for consistency with the lookup-table generation process, hardware evaluation flow, and prior cache-design practices.

In addition, hardware-aware penalty terms are incorporated into the training loss to discourage predictions that would lead to large mismatches after discretization. This strategy encourages the learned continuous representations to remain close to feasible hardware configurations and improves stability after projection into the discrete design space. As a result, the proposed architecture supports both accurate prediction and hardware-aware cache selection for FPGA-based systems.

3.5. Training procedure and evaluation protocol

To ensure that the predicted cache configurations are both accurate and hardware feasible, the model is trained using a composite loss function that integrates prediction accuracy with performance and hardware-constraint penalties.

The primary prediction error is measured using the mean absolute error (MAE) across the three output parameters (sets, ways, and block size). For each output dimension m, the MAE is defined as:

$$MAE_m = \frac{1}{n} \sum_{i=1}^{n} \left| Y_i^{(m)} - \hat{Y}_i^{(m)} \right| \quad (1)$$

where $Y^{(m)}$ and $\hat{Y}^{(m)}$ denote the ground-truth and predicted values for each of the three outputs. The overall error is computed as:

$$MAE = \frac{1}{3} \left(MAE_1 + MAE_2 + MAE_3 \right) \quad (2)$$

To incorporate hardware and performance considerations, the total loss function is defined as:

$$\mathcal{L} = \alpha \cdot \ell_{MR} + \beta \cdot \ell_{BRAM} + \gamma \cdot \ell_{CLB} + MAE \quad (3)$$

Where ℓ_{MR} represents the estimated cache miss rate, and l_{BRAM} and l_{CLB} penalize violations of FPGA resource constraints. The weighting coefficients α, β, and γ are selected empirically to balance prediction accuracy and hardware feasibility; moderate variations in these parameters were observed to have limited impact on convergence behavior.

The hardware-violation penalties are defined as:

$$\ell_{CLB} = \begin{cases} \left| \widehat{CLB} - CLB_{avail} \right|, & if\ \widehat{CLB} - CLB_{avail} \\ 0, & otherwise, \end{cases} \quad (4)$$

$$\ell_{BRAM} = \begin{cases} \left| \widehat{BRAM} - BRAM_{avail} \right|, & if\ \widehat{BRAM} - BRAM_{avail} \\ 0, & otherwise, \end{cases} \quad (5)$$

The miss-rate and hardware-usage values are obtained from precomputed lookup tables, which map cache configurations to performance and resource metrics derived from simulation and synthesis. This approach avoids repeated simulation during training and enables efficient optimization while preserving consistency with hardware-aware evaluation.

The dataset used for training and validation is constructed from MachSuite and PolyBench, while CHStone is reserved as a held-out test set. This program-level separation ensures that the evaluation reflects the model's ability to generalize to unseen workloads. Data augmentation is applied by generating additional samples for each Pareto-optimal configuration through controlled perturbation of hardware budgets. Specifically, CLB and BRAM constraints are scaled within the range $[50\%, 150\%]$ using geometric sampling, and

[Downloaded from idea.iust.ac.ir on 2026-11-16]

only configurations that introduce at least one tighter constraint are retained. The resulting values are cast to integers and clipped to device-supported ranges (e.g., BRAM: 2–240, CLB: 237–6500), ensuring that all generated samples correspond to realistic hardware scenarios.

Each training sample consists of a normalized delta-based memory-access trace of length 1000 and the associated hardware constraints. All input features are scaled to the range $[-1,1]$ using min–max normalization to ensure consistent numerical ranges and stable training behavior.

During training, regularization techniques such as dropout and early stopping are employed to mitigate overfitting and improve generalization. Model performance is monitored on the validation set, and hyperparameters such as learning rate and batch size are adjusted accordingly.

Overall, this training formulation enables the model to learn cache configurations that jointly satisfy prediction accuracy, performance objectives, and hardware constraints, while maintaining robustness across varying workload characteristics and resource conditions.

4. Results

4.1 Hyperparameter tuning

The proposed framework contains several hyperparameters related to the neural-network architecture, optimization procedure, and hardware-aware loss formulation. To determine an effective configuration while avoiding excessive optimization complexity, a staged tuning strategy was adopted in which groups of hyperparameters were adjusted sequentially rather than simultaneously.

The training setup was fixed during the initial exploration stage. Memory-access traces were represented using fixed-length sequences of 1000 normalized delta values, and model training was performed using the Adam optimizer with a batch size of 128. The training and validation datasets were constructed from augmented samples derived from MachSuite and PolyBench benchmarks, resulting in more than 20,000 training and validation samples after augmentation. These samples were divided using an 80:20 split while maintaining strict kernel-level separation to prevent data leakage. Consequently, traces originating from a given benchmark kernel appeared exclusively in either the training or validation subset, ensuring that validation was performed on unseen workload patterns.

The architecture search focused on identifying a suitable balance between feature-extraction capacity and model complexity. Several combinations of convolutional and recurrent dimensions were evaluated, including CNN channel sizes of 32 and 64 and LSTM hidden dimensions of 4, 8, and 16. Larger convolutional widths improved the extraction of local access patterns and stride-related features, while excessively large recurrent dimensions increased model complexity without producing consistent improvements in prediction accuracy. The best trade-off between accuracy and stability was obtained using 64 CNN channels and an LSTM hidden size of 8.

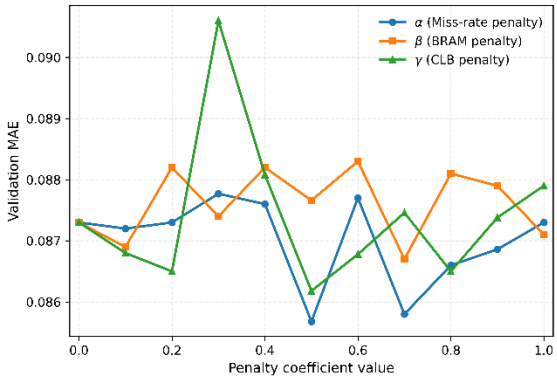


Figure 4 validation MAE obtained during first-stage one-dimensional coefficient exploration.

In addition to architectural tuning, the weighting coefficients of the composite loss function in Eq. (3) were explored through a coarse-to-fine search procedure. The coefficients α, β, and γ, corresponding respectively to the miss-rate, BRAM, and CLB penalty terms, were varied over the interval [0,1] with a step size of 0.1. During the initial exploration stage, each coefficient was evaluated independently to identify stable operating ranges, after which combinations of the most promising values were jointly assessed.

[Downloaded from idea.iust.ac.ir on 2026-11-16]

Figure 4 presents the validation MAE obtained during the first-stage one-dimensional coefficient exploration. The results show that the validation error varies only slightly across neighboring coefficient values, indicating that the proposed optimization framework is relatively robust to moderate variations of the penalty weights and does not rely on a narrowly tuned parameter setting.

Based on this analysis, the following candidate coefficient sets were selected for the second-stage search:

$$\alpha \in \{0.5,0.7\}, \beta \in \{0.1,0.7\}, \gamma \in \{0.2,0.5,0.8\}$$

The candidate combinations were subsequently evaluated jointly, and the final configuration ($\alpha = 0.5$, $\beta = 0.7$, $\gamma = 0.8$) was selected because it achieved the best overall balance between prediction accuracy, miss-rate preservation, and hardware-resource feasibility.

Overall, the hyperparameter tuning process demonstrated that the proposed CNN–LSTM architecture can effectively capture memory-access behavior while maintaining compatibility with hardware-aware optimization objectives. The selected configuration was subsequently used in all experiments reported in the following sections.

4.2 Training and validation results

The proposed CNN–LSTM model was trained using normalized memory-access traces together with BRAM and CLB hardware constraints to predict cache parameters while maintaining hardware feasibility and minimizing cache miss behavior. The input representation combined fixed-length delta-based memory-access sequences with normalized FPGA resource budgets, enabling the model to jointly learn workload-dependent locality patterns and resource-aware cache configurations.

Figure 5 and Figure 6 illustrate the evolution of the training and validation metrics throughout the optimization process. As shown in Figure 5(a), both the training and validation prediction errors decreased rapidly during the early epochs, indicating effective feature learning from the memory-access traces. After approximately 150–200 epochs, the curves stabilized with only minor fluctuations, suggesting convergence of the optimization process. The lowest validation loss was achieved at epoch 426 with a learning rate of 0.01, corresponding to a validation MAE of approximately 0.085. Beyond this point, only marginal improvements were observed, indicating that the selected configuration provided a suitable balance between convergence and generalization performance.

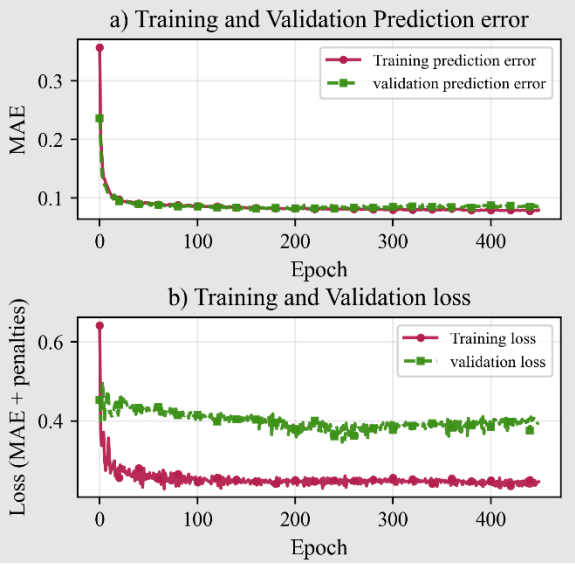


Figure 5: Transition of prediction error, loss throughout the training process

The gap between training and validation prediction error remained consistently small throughout training. The average generalization gap was approximately 0.002, with a maximum observed value below 0.011, indicating limited overfitting despite the complexity of the regression task. Furthermore, the validation error exhibited low variance during the final training stages, with a standard deviation of approximately 0.0011 over the last 50 epochs, demonstrating stable convergence behavior. Because the validation set consisted of benchmark kernels excluded from the training subset, this result also suggests that the model generalizes effectively to previously unseen memory-access patterns.

The loss curves in Figure 5(b) further demonstrate stable optimization behavior. Both training and validation losses decreased substantially during the initial epochs before gradually approaching a steady operating region.

[Downloaded from idea.iust.ac.ir on 2026-11-16]

The absence of abrupt divergence or oscillatory instability indicates that the adopted optimization and regularization strategy successfully stabilized the learning process.

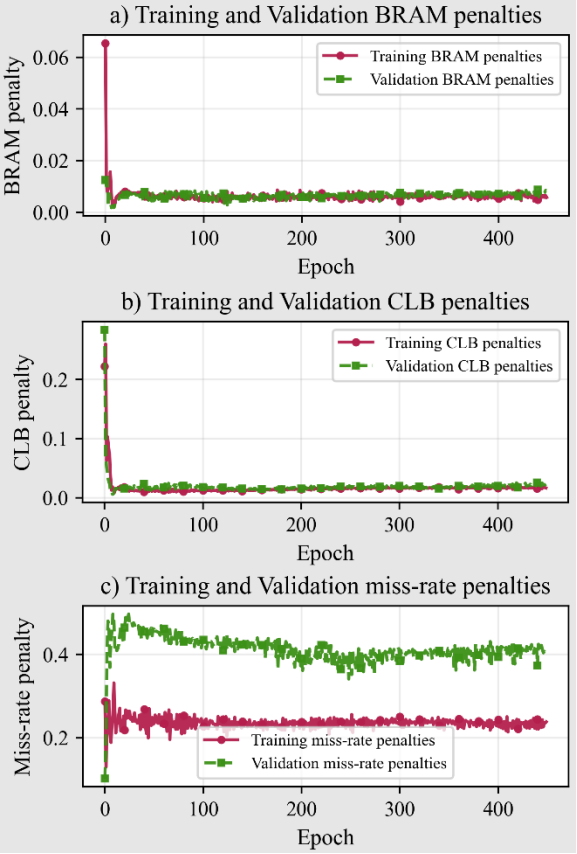


Figure 6: Transition of hardware cost and miss rate penalties throughout the training process

Figure 6 presents the evolution of the hardware-aware penalty terms included in the composite loss function. The BRAM and CLB penalties rapidly converged toward small values during the early stages of training, indicating that the model quickly learned to avoid hardware-infeasible cache configurations. In contrast, the miss-rate penalty decreased more gradually and remained comparatively larger throughout training. This behavior reflects the greater complexity of learning workload-dependent cache-performance characteristics compared with satisfying hardware-resource constraints. The observed training dynamics therefore suggest a hierarchical learning behavior in which the model first adapts to feasible hardware regions and subsequently refines cache-performance optimization.

Overall, the training and validation results demonstrate that the proposed framework achieves stable convergence, limited overfitting, and effective integration of hardware-awareness into the learning process, supporting its suitability for cache-configuration prediction in FPGA-based embedded systems.

4.3 Final testing and prediction accuracy

The final model was evaluated on a held-out CHStone-derived test set containing around 5,000 samples, none of which were used during training or validation. This workload-level separation provides a strict generalization test because the model is required to predict cache configurations for application kernels that were excluded from model development. In contrast to ordinary continuous regression, the proposed task is constrained by a sparse, synthesis-supported design space in which valid cache parameters are restricted to discrete power-of-two values. For this reason, both standard regression metrics and quantization-aware measures are reported below.

Table 2 summarizes the prediction performance of the discretized cache configurations on the unseen test set. The CNN–LSTM model first produces continuous-valued estimates for the cache parameters, and these outputs are then projected to the nearest valid hardware-supported configurations before evaluation. Accordingly, the MAE, RMSE, and R^2 values reported in Table 2 are computed on the post-processed cache parameters against the corresponding ground-truth configurations, which are already valid discrete hardware-supported designs generated by the exploration process and therefore require no additional quantization. The model achieves a miss-rate MAE of 0.064, with an RMSE of 0.155, indicating that the quantized predictions preserve the reference miss-rate behavior reasonably well under unseen workloads. For cache parameters, exact numerical matching is less informative than in ordinary regression because small shifts between neighboring power-of-two configurations can produce large numerical residuals even when the resulting cache behavior remains similar. Accordingly, the reported R^2 values should be interpreted cautiously, especially for set and associativity prediction.

[Downloaded from idea.iust.ac.ir on 2026-11-16]

Table 2 Regression performance on the test set

Parameter	MAE	RMSE	R^2
Sets	6.941	13.622	-5.477
Ways	3.075	3.713	-0.218
Words	19.132	30.265	0.285
miss rate	0.064	0.155	-0.247

Because the outputs are constrained to synthesis-supported cache levels, a more practical measure is the ±1 quantization-level accuracy, which indicates whether the predicted cache configuration lies within one neighboring power-of-two level of the reference configuration. Table 3 shows that the proposed model remains close to the target configuration in a substantial fraction of test cases, especially for block size.

Table 3 Quantization-aware prediction accuracy

Parameter	±1 Quantization-Level Accuracy
Sets	59.70%
Ways	57.62%
Words	75.09%

To assess deployment cost, the predicted and reference cache configurations were also projected onto a Xilinx Zynq-7020 (XC7Z020-1CLG484) [25] target operating at 100 MHz. For this device, the available resources are 6,650 CLBs (CLB_{Baseline}) and 280 BRAM blocks ($BRAM_{\text{Baseline}}$). Table 4 reports the average resource consumption of the Pareto-selected reference configuration and the proposed predicted configuration as percentages of the available device resources. Hardware usage calculated by $(\text{CLB}/\text{CLB}_{\text{Baseline}} + \text{BRAM}/\text{BRAM}_{\text{Baseline}})/2$.

The reference configuration occupies 19.13% of CLBs and 59.86% of BRAMs, whereas the predicted configuration uses 15.38% of CLBs and 70.00% of BRAMs. Thus, the learned model shifts resource pressure from logic toward memory while remaining within the same embedded FPGA budget. Using AMD Xilinx Power Estimator [29] under default preimplementation assumptions, the reference configuration is estimated at 0.201 W, while the predicted configuration is estimated at 0.196 W, corresponding to an approximate 2.5% reduction in estimated on-chip power.

Table 4 Resource consumption and estimated power on Xilinx Zynq-7020

Metric (Average)	Reference / Pareto	Predicted	Relative Error
CLBs usage	19.13%	15.38%	-3.75%
BRAMs usage	59.86%	70.00%	+10.14%
hardware usage	39.46%	42.69%	+3.23%
miss rate	39.67%	39.23%	-0.44%
Estimated on-chip power	0.201W	0.196W	-0.005W

To further evaluate model robustness and identify potential systematic prediction errors, residual analyses were performed for the cache-parameter predictions. Since the number of sets exhibited the largest prediction error among the three cache parameters and therefore represents the most challenging prediction task, the corresponding residual results are presented in Figure 7 and Figure 8. Figure 7 shows the residual histogram, while Figure 8 presents the residual scatter plot as a function of the predicted number of sets.

The histogram indicates that the majority of residuals are concentrated near zero, demonstrating that most predictions remain close to the reference cache configurations after discretization. The scatter plot does not exhibit a pronounced monotonic trend across the prediction range, suggesting the absence of a strong systematic bias. Although a small negative mean residual (−4.82) is observed, the residuals remain distributed around the zero-error line for most configurations. The visible clustering of residuals into discrete horizontal bands is expected because cache parameters are constrained to synthesis-supported power-of-two values. Consequently, prediction errors occur in quantized steps between neighboring valid cache configurations rather than as continuous deviations.

[Downloaded from idea.iust.ac.ir on 2026-11-16]

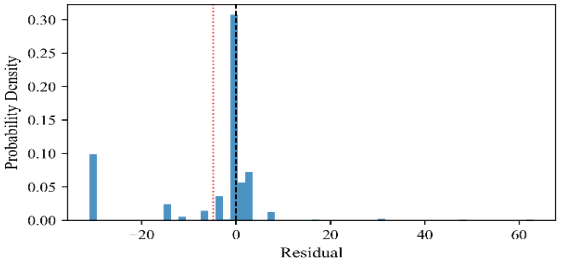


Figure 7 Distribution of residual errors for set prediction on the CHStone test set.

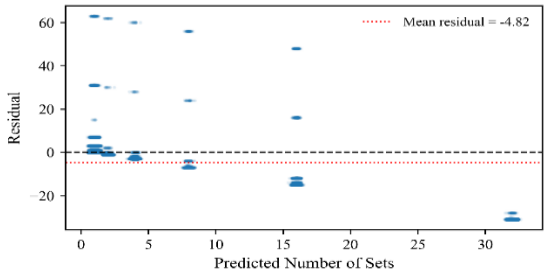


Figure 8 Residual scatter plot of set prediction errors versus predicted set values

This behavior reflects the discrete nature of the cache-design space rather than a deficiency of the regression model. Overall, the residual analysis indicates that the proposed predictor maintains stable generalization behavior on unseen workloads while producing cache configurations that remain close to the reference Pareto-optimal solutions across the evaluated design space.

Overall, the final test results show that the proposed model generalizes to unseen CHStone workloads, preserves the miss-rate behavior of reference cache configurations, and provides hardware-feasible cache designs with moderate resource use on a resource-constrained embedded FPGA platform.

5. Discussion

The study introduces a machine-learning model that predicts cache configuration parameters—such as sets, ways, and block size—based on application-specific array-index traces and available hardware budgets (CLBs and BRAMs). Unlike traditional methods, the model generates cache parameters directly, with hardware utilization and performance metrics like miss rate derived from precomputed lookup tables for predicted configurations.

The model's key outcome shows that, on average, the predicted cache configurations reduced CLB utilization by 3.75% while increasing BRAM utilization by 10.14% relative to the reference Pareto-optimized solutions. As a result, the overall hardware utilization increased by 3.23%. Despite this resource redistribution, the average miss rate remained nearly unchanged, with a difference of only 0.44%, indicating that the proposed predictor preserves application-level memory performance while exploring alternative hardware trade-offs. Power estimation on the target Zynq-7020 platform further showed a slight reduction in average on-chip power consumption, from 0.201 W to 0.196 W. These results suggest that the predicted cache configurations achieve comparable performance while maintaining efficient hardware utilization and energy characteristics. To encourage feasible solutions during training, penalty terms associated with CLB utilization, BRAM utilization, and miss rate were incorporated into the loss function, guiding the model toward balanced trade-offs between performance and hardware-resource consumption.

In conclusion, the predictor does not replace the need for detailed synthesis or simulation; instead, it efficiently narrows down the design space by suggesting cache configurations that achieve near-Pareto performance while respecting hardware constraints. This capability can significantly reduce the evaluation cost of cache design exploration and provide practical starting points for more detailed, tool-level optimization. In the context of smart vehicle applications, where real-time processing and efficient resource utilization are critical, this approach enables faster development cycles and more reliable system designs. By optimizing cache configurations for embedded systems in vehicles, such as autonomous driving modules or infotainment systems, the predictor supports the creation of energy-efficient and high-performance solutions that meet the stringent requirements of automotive environments.

Our ML-based approach offers a distinct advantage over related works by enabling the

[Downloaded from idea.iust.ac.ir on 2026-11-16]

simultaneous optimization of multiple cache parameters—sets, ways, and block size—across a broader and continuous design space, unlike most prior studies that rely on classification-based methods limited to predefined parameter classes or specific aspects like cache size or line size. While methods like [30] and [17] achieve high prediction accuracies (96–99%), their reliance on discrete classes restricts design flexibility and requires extensive preprocessing, whereas ANNs [18] face issues like misclassification or energy degradation under suboptimal configurations.

Our regression framework integrates memory-access trace information with hardware-resource constraints to predict cache configurations that balance performance and implementation cost. Experimental results demonstrate that the proposed approach maintains a comparable miss rate while exploring alternative resource-utilization trade-offs and reducing estimated power consumption on the target platform. These findings indicate that machine-learning-based cache prediction can significantly reduce the cost of design-space exploration by providing high-quality cache configurations without exhaustive simulation. Future work will extend the proposed framework to mixed read–write memory-access workloads by incorporating access-type information and write-policy effects into the learning process. In addition, future research will investigate transferability across different FPGA families and evaluate deployment on embedded platforms under real-time automotive constraints, including inference latency, hardware footprint, and hardware/software co-design considerations.

6. Conclusions

This study introduced a hardware-aware regression framework for cache-configuration prediction that combines trace-based learning with FPGA resource constraints. The evaluation showed that the model generalizes to unseen benchmark workloads and produces cache configurations that remain close to the reference Pareto-optimal solutions while preserving performance and hardware feasibility. Residual analysis further indicated limited systematic bias, supporting the stability of the predictor across the evaluated design space. The results also show that the learned configurations can shift the resource trade-off in a useful way, reducing logic pressure while maintaining acceptable memory usage and slightly lowering estimated power on the target embedded FPGA platform. Overall, the proposed method offers a practical alternative to exhaustive cache exploration for resource-constrained embedded systems. Future work will extend the framework to mixed read-write workloads, investigate transferability across additional FPGA families, and explore tighter integration with synthesis-driven optimization flows.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Acknowledgment:

This work was supported by Shahid Rajaee Teacher Training University under grant number 23905/395017.

References

[1] A. Bosaghzadeh and M. NASIRI MANJILI, "Inverse perspective mapping for real-time Lane Detection in City Streets," (in eng), Automotive Science and Engineering, Research vol. 10, no. 3, pp. 3311–3323, 2020, doi: 10.22068/ase.2020.540.

[2] C. Urmson and W. Whittaker, "Self-driving cars and the urban challenge," IEEE Intell. Syst, vol. 23, no. 2, pp. 66–68, 2008.

[3] S. D. U. Montanaro, S. Fallah, M. Dianati, A. Stevens, D. Oxtoby, and A. Mouzakitis, "Towards connected autonomous driving: Review of use-cases," Vehicle Syst. Dyn, vol. 59, no. 6, pp. 779–814, 2019.

[4] H. Chehardoli, A. Ghasemi, and M. Daneshyian, "An optimal consensus to guarantee the stability and crash avoidance of large-scale traffic flow in presence of time delay," (in eng), Automotive Science and Engineering, Research vol. 10, no. 4, pp. 3434–3445, 2020, doi: 10.22068/ase.2020.546.

[5] J. C. W. Payre, and P. Delhomme, "Intention to use a fully automated car: Attitudes and a priori acceptability," Transp. Res. F, Traffic Psychol. Behav., vol. 27, pp. 252–263, 2014.

[Downloaded from idea.iust.ac.ir on 2026-11-16]

[6] A. Khosravian, M. Masih-Tehrani, A. Amirkhani, and S. Ebrahimi-Nejad, "Robust autonomous vehicle control by leveraging multi-stage MPC and quantized CNN in HIL Framework," Applied Soft Computing, vol. 162, p. 111802, 2024/09/01/ 2024, doi: https://doi.org/10.1016/j.asoc.2024.111802.

[7] S. Z. K. Guo, J. Yu, Y. Wang, and H. Yang, "[DL] a survey of FPGA-based neural network inference accelerators," ACM Trans. Reconfigurable Technol. Syst., vol. 12, no. 1, pp. 1–26, 2019.

[8] B. Y. Z. Wan, T. Y. Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury, and S. Liu, "A survey of FPGA-based robotic computing," IEEE Circuits Syst. Mag, vol. 21, no. 2, pp. 48–74, 2021.

[9] V. N. D. Castells-Rufas, J. Borrego-Carazo, M. Codina, C. Sanchez, D. Gil, and J. Carrabina, "A survey of FPGA-based vision systems for autonomous cars," IEEE Access, vol. 10, pp. 132525–132563, 2022.

[10] B. P. G. Aishwarya, P. V. Joshi, K. M. Sudarsham, K. Vaidyanathan, P. Parandkar, and A. Dsouza, "‘‘A survey on use of FPGA in automotive system," in Int. Conf. Distrib. Comput., VLSI, Electr. Circuits Robot. (DISCOVER), 2022, pp. 51–56.

[11] A. J. Mohammad Shihabul Haque, and Sri Parameswaran, "SuSeSim: a fast simulation strategy to find optimal L1 cache configuration for embedded systems," in In Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis, Grenoble, France, 2009.

[12] A. a. I. Janapsatya, A. and Parameswaran, S., "Finding optimal L1 cache configuration for embedded systems," in Asia and South Pacific Conference on Design Automation, 2006, p. 6, doi: 10.1109/ASPDAC.2006.1594783.

[13] M. D. H. a. A. J. Smith., "Evaluating associativity in CPU caches," IEEE Trans. Comput, vol. 38, no. 12, pp. 1612–1630, 1989, doi: 10.1109/12.40842.

[14] A. I. A. Janapsatya, and S. Parameswaran, "Finding optimal L1 cache configuration for embedded systems," presented at the In Asia and South Pacific Conference on Design Automation,, 2006.

[15] J. P. Mohammad Shihabul Haque, Andhi Janapsatya, and Sri Parameswaran, "DEW: A Fast Level 1 Cache Simulation Approach for Embedded Processors with FIFO Replacement Policy," in In Proceedings of the 2010 Design, Automation and Test in Europe Conference, 2010, vol. 1: IEEE,.

[16] N. T. Nobuaki Tojo, Masao Yanagisawa, and Tatsuo Ohtsuki, "Exact and fast L1 cache simulation for embedded systems," in In 2009 Asia and South Pacific Design Automation Conference, 2009, pp. 817–822.

[17] L. a. J. Ribeiro, Ricardo and Júnior, Francisco and da Silva, Jones Yudi and Silva, Ivan Saraiva, "Evaluating a Machine Learning-based Approach for Cache Configuration," in IEEE 13th Latin America Symposium on Circuits and System, 2022, pp. 1–4, doi: 10.1109/LASCAS53948.2022.9789040.

[18] A. G.-R. Ruben Vazquez, and Greg Stitt, "Work-in-Progress: Offloading Cache Configuration Prediction to an FPGA for Hardware Speedup and Overhead Reduction," in In 2019 International Conference on Hardware/Software Codesign and System Synthesis, 2019, pp. 1–2.

[19] S. K. a. C. Chantrapornchai, "On the finding proper cache prediction model using neural network," presented at the 2016 8th International Conference on Knowledge and Smart Technology, 2016.

[20] Y. H. a. H. T. a. S. H. a. H. T. a. K. Ishii, "CHStone: A benchmark program suite for practical C-based high-level synthesis," 2008, pp. 1192–1195, doi: 10.1109/ISCAS.2008.4541637.

[21] B. a. A. Reagen, Robert and Shao, Yakun Sophia and Wei, Gu-Yeon and Brooks, David, "MachSuite: Benchmarks for accelerator design and customized architectures," IEEE International Symposium on Workload Characterization, pp. 110–119, 2014, doi: 10.1109/IISWC.2014.6983050.

[22] M. Reisinger. "PolyBench/C 4.2.1." https://github.com/MatthiasJReisinger/PolyBenchC-4.2.1 (accessed.

[23] A. Khosravian, A. Amirkhani, M. Masih-Tehrani, and A. Yazdanijoo, "Multi-domain autonomous driving dataset: Towards enhancing the generalization of the convolutional neural networks in new environments," IET Image Processing, vol. 17,

[Downloaded from idea.iust.ac.ir on 2026-11-16]

no. 4, pp. 1253–1266, 2023, doi: https://doi.org/10.1049/ipr2.12710.
[24] G. a. U. J. Brignone, M. and Lazarescu, Mihai T. and Lavagno, Luciano, "Array-Specific Dataflow Caches for High-Level Synthesis of Memory-Intensive Algorithms on FPGAs," IEEE Access, vol. 10, pp. 118858–118877, 2022.
[25] Avnet. ZedBoard Hardware User's Guide [Online] Available: https://digilent.com/reference/_media/reference/programmable-logic/zedboard/zedboard_ug.pdf
[26] I. Corporation. M20K Blocks Mixed-Width Configurations, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683213/current/m20k-blocks-mixed-width-configurations.html.
[27] X. (AMD). 7 Series FPGAs Memory Resources User Guide (UG473), 2019.. Available: https://docs.amd.com/v/u/en-US/ug473_7Series_Memory_Resources.
[28] C. a. K. Barone, Rishika and Limaye, Ankur and Castellana, Vito Giovanni and Gozzi, Giovanni and Fiorito, Michele and Ferrandi, Fabrizio and Tumeo, Antonino, "To Cache or not to Cache? Exploring the Design Space of Tunable, HLS-generated Accelerators," New York, NY, USA, 2024: Association for Computing Machinery, pp. 210–218, doi: 10.1145/3695794.3695815.
[29] X. AMD. Power Design Manager User Guide (UG1556), 2025. [Online]. Available: https://docs.amd.com/r/en-US/am007-versal-memory.
[30] O. a. Y. Navarro, Jones and Hoffmann, Javier and Hernandez, Hector Gerardo Munoz and Hubner, Michael, "A Machine Learning Methodology for Cache Memory Design Based on Dynamic Instructions," Association for Computing Machinery, vol. 19, no. 2, 2020.

[Downloaded from idea.iust.ac.ir on 2026-11-16]