Performance Prediction of SystemC TLM-2.0 Contention-Aware DNN Models

Maha Bhatti
Fowler School of Engineering
Chapman University
Orange, CA, USA
mbhatti@chapman.edu

Emad M. Arasteh
Fowler School of Engineering
Chapman University
Orange, CA, USA
arasteh@chapman.edu

***Abstract*—For effective embedded system design, system architects model, configure, and simulate transaction-level models to explore the design space to find the optimal design candidate for final implementation. Due to the exponential growth in the design complexity of embedded systems and the Internet of Things (IoT), running many TLM simulations has become time-consuming, inefficient, and power-demanding. In this work, we propose a machine learning model that captures and learns the complexities of the SystemC TLM-2.0 loosely-timed contention-aware (LT-CA) models, which consider the critical effect of memory and interconnect contention in system-level design and performance estimation. Our experimental results on the TLM-2.0 LT-CA models of two representative DNNs, GoogLeNet and ResNet, show high accuracy of our proposed model with a mean absolute percentage error (MAPE) of 3.12%. Using the enhanced predictive model, we effectively explore the design space to search and identify the optimal sets of design configurations derived through Pareto analysis supported by specialized performance-evaluating functions. Given the expeditious predictive model, the Pareto analysis shows 8 optimal design candidates out of 1,000 experimental candidates with 3 orders of magnitude speed-up. The proposed framework emphasizes the reduction of time and cost constraints by saving hundreds of hours of TLM simulation, enhancing the overall efficiency of system-level modeling and simulation.**

***Index Terms*—System modeling, SystemC, Internet-of-things, design space exploration, deep neural network, TLM-2.0, machine learning, simulation, predictive analysis**

I. Introduction

System-level modeling and simulation are essential techniques for designing and analyzing complex computer systems. System-level modeling involves creating abstract representations of system components and their detailed interactions, while simulation executes these models to verify functionality and evaluate performance. This effective approach enables embedded system designers to analyze and explore a set of design decisions early in the development process by reducing costs, shortening time to market, and improving efficiency prior to final implementation.

The growing complexity of system-level models of embedded systems and IoT devices has resulted in increasingly time-consuming simulations. Consequently, exploring diverse design configurations to identify the optimal candidate requires many simulations, substantial computational resources, and energy. To mitigate these challenges, this work proposes a machine learning-based approach to predict the estimated performance of IEEE SystemC simulation accurately. In particular, this work focuses on performance prediction of SystemC Transaction Level Modeling (TLM-2.0) loosely-timed contention-aware models [1] that consider the critical effect of memory and interconnect contention in performance estimation.

Fig. 1 illustrates the proposed machine-learning framework for efficient SystemC design, simulation, and exploration. By training on historical simulation data, the predictive model captures the complexities of TLM-2.0 LT-CA simulations, enabling accurate performance predictions for design space exploration and multi-objective optimization.

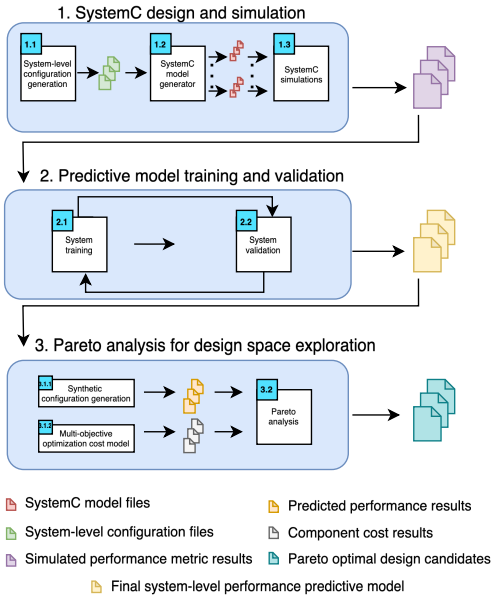


Fig. 1: Predictive modeling framework for efficient system-level design, simulation, and exploration

In summary, this paper aims to address the time-consuming SystemC simulations through the following contributions:

(1) Propose a data-driven statistical modeling framework to mitigate the numerous simulations of SystemC TLM-2.0 loosely-timed contention-aware models

979-8-3315-2037-3/25/$31.00

(2) Present machine learning (ML) architecture to optimally predict the performance of the SystemC LT-CA models of a deep neural network (DNN)

(3) Demonstrate the applicability of the proposed ML framework for effective DNN system-level design exploration using Pareto analysis

The remainder of the manuscript is structured as follows: Section II reviews performance modeling, IEEE SystemC transaction-level modeling, and application modeling of deep neural network in SystemC TLM-2.0. Section III details configuration data generation, SystemC model creation and simulation, and the development, training, and validation of the predictive neural network. Section IV presents Pareto analysis for optimal design selection. Section V reports the experimental results, and Section VI concludes the study.

II. Background

This section provides an overview of system-level modeling and the IEEE SystemC language that can model both hardware and software components. Additionally, it discusses related work on predictive modeling techniques for system-level analysis. Finally, it introduces the application driver for the proposed performance prediction framework.

A. Performance Estimation and IEEE SystemC Transaction Level Modeling (TLM)

The two common performance modeling approaches in system-level design are analytical and simulation-based. Analytical models use mathematical functions to represent computer system performance, but may oversimplify complex interactions between components due to reliance on workload and input parameters assumptions [2], [3]. Simulation-based models create computational representations that capture dynamic system interactions more accurately, but can be computationally intensive and time-consuming. Two commonly utilized system-level description languages for modeling, simulation, and validating complex system-on-chip models are SpecC [4] and SystemC [5]. The SystemC C++ class library, an IEEE standard, facilitates system and transaction-level modeling through discrete event simulation [6]. However, prolonged simulation run times often hinder simulation techniques when operating at lower abstraction levels. More recently, a loosely-timed contention-aware (LT-CA) modeling approach has been proposed, which offers high-speed simulation close to traditional loosely-timed models, yet shows the same accuracy for memory contention as low-level approximately-timed models [1]. Yet even LT-CA simulations show slow and long runtimes when exploring myriads of configurations to find the optimal design during the system-level design process.

B. Machine Learning for Performance Prediction of System-Level Models

The increasing complexity of embedded systems necessitates fast, accurate modeling techniques beyond traditional simulation and analytical methods. Machine learning-based system-level models, covering architectural prediction, cross-platform behavior, and runtime workload forecasting, offer promising solutions for efficient design and management [7]. In particular, predictive modeling employs machine learning techniques on historical data to identify patterns and predict system behavior efficiently, offering rapid and cost-effective insights, contingent on data quality.

Predictive models at lower abstraction levels have been recently explored. Zennaro et al. propose an accurate and rapid prediction of hardware component areas from specifications, improving hardware development efficiency [8]. Lopera et al. present a machine learning approach for accurate and fast delay estimation in RTL combinational circuits to facilitate early design decisions and reduce iteration time [9]. In this new work, we focus on a machine learning approach to predict the performance of high-level abstraction, i.e., the simulated time of SystemC TLM-2.0 models.

C. SystemC Application Driver: Deep Neural Network

As an application driver for our predictive modeling framework, we develop a SystemC TLM-2.0 model of a deep neural network (DNN) on a simplified many-core architecture with bus interconnect and shared memory similar to [1]. In particular, we simulate GoogLeNet [10], a deep convolutional neural network used in image detection on 142 cores using TLM-2.0 sockets, generic payloads with memory addressing and transaction timings. To confirm our findings, we also design and simulate a new SystemC TLM-2.0 LT-CA model of ResNet50 [11], another deep convolutional neural network on 176 cores. These DNN SystemC models can be easily integrated into SystemC TLM-2.0 virtual platform (VP) models with multi-level caches and shared memory, given the standardized TLM-2.0 programming interfaces and interoperable transactions.

III. Performance Prediction Framework for SystemC TLM-2.0 LT-CA modeling

Following Fig. 1, this section describes performance prediction framework including stage (1) SystemC design and simulation, and stage (2) predictive model training and validation.

A. SystemC design and simulation

There is a wide range of system-level parameters that affect the system-level performance. This work focuses on 7 system-level parameters outlined in Table I. As shown in Fig. 1 (stage 1-1), the system-level configuration generator produces configuration files in a specific format suitable for input to the SystemC model generator.

Obtaining unbiased and representative data is critical for optimal predictive model performance. Fig. 2 presents the feature distribution, with density plots indicating no discernible patterns and normal distributions. High-quality, unbiased data collection is essential for reliable predictive modeling. Given the system-level parameters, the SystemC model generator generates corresponding SystemC model files for each configuration (stage 1-2). The system-level model specified in

Feature	Description
Computational capacity	Floating-point operations per second (FLOPS) available to each core
Shared memory latency	Wait time to transfer a word from memory to a core
Shared memory read delay	Access time for an individual read operation in memory
Shared memory write delay	Access time for an individual write operation in memory
Interconnect latency	Delay to route a transaction from a core to a memory
Interconnect arbitration policy	Transaction scheduling protocol (First-come-first-served (FCFS), Round-Robin (RR))
Transaction size	Transaction size initiated from a core ([32, 64, 128] bytes)

TABLE I: System-level parameters for many-core architecture with bus interconnect and shared memory

SystemC TLM-2.0 is simulated in batch mode to obtain results for performance metrics such as the simulated time (stage 1-3).

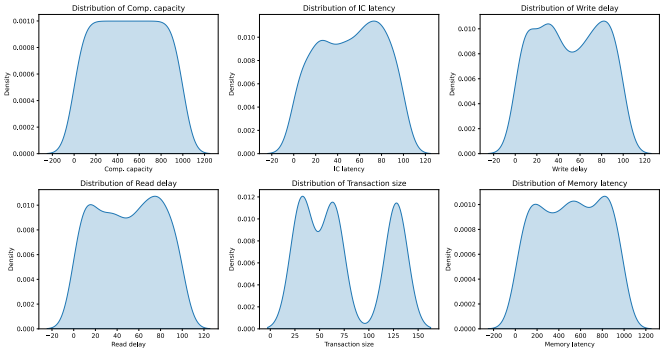


Fig. 2: System-level parameters distribution plot for a batch size of 500

B. Predictive model training and validation

Fig. 1 stage (2) previews an outline for the training and validation of the statistical model. This section discusses how the collected dataset is used in the process of training and validating a predictive model. It dives into the process of designing the architecture of a neural network, and findings later are presented through experimental measurements.

1) Baseline architecture: The baseline model is a feedforward neural network (Fig. 3) that processes all input variables simultaneously through multiple dense layers. Its simple, unidirectional architecture captures data complexity and feature interdependencies.

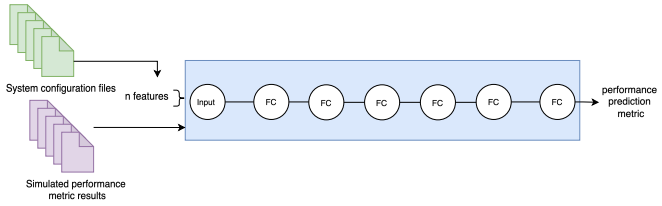


Fig. 3: Baseline performance prediction neural network architecture

2) Brute force architecture: The second model is an enhanced feedforward neural network, termed the brute force model (Fig. 4). It assigns separate processing pathways to each feature, capturing feature-specific relationships with the dependent variable. After individual processing, features are concatenated for joint analysis, enabling the model to learn distinct feature-to-output mappings. While this approach improves interpretability of feature relationships, it is computationally intensive, especially with increasing input dimensions.

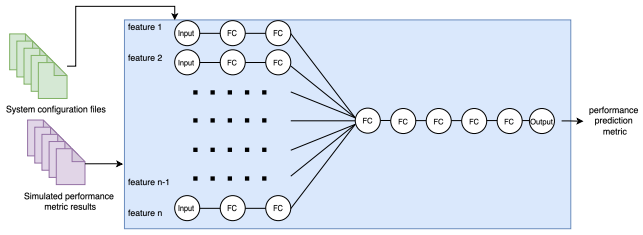


Fig. 4: Brute force performance prediction neural network architecture

3) Optimized architecture: Looking closer at the data and its relationship with the output variable, we can cluster specific features with similar relational characteristics. Namely, memory latency has a strictly linear relationship with the output variable, simulated time. Other features, including computational capacity, read and write latencies, and interconnect delay, show more scattered nonlinear relationships with the dependent feature. Categorical variables such as arbitration policies also have a unique relationship.

Given these three types of relationships, we create an optimized version of the statistical model illustrated in Fig. 5. The proposed neural network uses feature splitting and creates three different processing paths in the neural network. One path processes only the memory latency and has a linear activation. This allows us to learn the complexities of the memory latency feature, focusing heavily on linearity. The second path in the neural network processes the remaining features with a nonlinear activation function, Rectified Linear Unit (ReLU). This allows the model to learn each feature's nonlinear relationship with the dependent variable. Lastly, the third path processes categorical variables, such as arbitration policies, with a nonlinear activation function, ReLU.

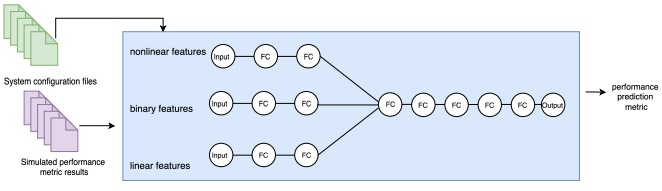


Fig. 5: Optimized performance prediction neural network architecture

The three separate paths process different inputs independently. Their outputs are combined through fully connected layers, which integrate all features to produce a single predicted simulated time.

After acquiring high-quality data, models are trained using batches of 100–500 samples. Proposed neural networks employ gradient descent and backpropagation to optimize weights and biases, enhancing predictive accuracy. Three distinct architectures are developed and trained with an 80/20 train-validation split.

IV. Performance Prediction for Effective Design Space Exploration

Evaluating designs involves optimization across multiple criteria, usually with conflicting objectives. Multi-objective optimization techniques provide the designer with a selection of designs from which they can choose the most suitable option [12]. This selection should consist only of "reasonable" designs. The goal of multi-objective optimization techniques is to identify these sets of designs. In general, we want to minimize the following cost function J:

$$J(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})) \tag{1}$$

where $\mathbf{x}$ is the vector of decision variables, $f_i(\mathbf{x})$ are the individual cost components. The goal is to solve the following optimization problem:

$$\min_{\mathbf{x}} \; J(\mathbf{x}) \tag{2}$$

In multi-objective optimization, it is generally infeasible to identify a solution that simultaneously minimizes all objective functions. Consequently, the system designer focuses on Pareto optimal solutions, solutions for which no objective can be improved without causing a degradation in at least one other objective. Therefore, design space exploration (DSE) based on Pareto points involves identifying a set of Pareto-optimal solutions to enable the designer to select the most suitable implementation.

Utilizing the performance prediction model, we can efficiently estimate the performance outcomes associated with a broad spectrum of design configurations. This capability enables systematic exploration of the design space, facilitating the identification of optimal design candidates for subsequent refinement using detailed models, such as SystemC TLM-2.0 approximately-timed models or Register-Transfer Level (RTL) implementations (stage (3) in Fig. 1). In this study, we concentrate on a bi-objective optimization framework aimed at minimizing both execution time and component costs. To this end, we propose the following cost functions:

$$f_1(\mathbf{x}) = execution\ time\ cost\ (\mathbf{x}) \tag{3}$$

$$f_2(\mathbf{x}) = component\ cost\ (\mathbf{x}) \tag{4}$$

Execution time costs can be effectively estimated using a performance prediction model. To demonstrate the practicality of our approach, we introduce an analytical component cost model that assigns uniform weights to processors, main memory, and interconnects, as detailed in Equation 5. Based on the design metrics of each subsystem—processors, memory, and interconnects—we develop specialized cost functions tailored to each component.

$$Component\ cost = w_p C(p) + w_m C(m) + w_{ic} C(ic) \tag{5}$$

As illustrated in Fig. 1, a diverse array of design decisions can be input into the predictive model to accurately estimate execution times. By integrating these component cost estimations, the Pareto analysis leverages both execution time and component costs to identify and generate optimal configuration sets, facilitating informed design space exploration.

V. Experimental Measurements and Results

We utilize state-of-the-art deep neural networks such as GoogLeNet and ResNet for SystemC TLM-2.0 modeling and simulation, evaluating performance of the predictive models using metrics such as mean absolute error (MAE) and mean absolute percentage error (MAPE). The analysis focuses on training and validation loss, as well as error distributions in predictions. Models are trained with batch sizes of 100, 200, 300, 400, and 500 to assess the impact of data volume on model performance. Due to the stochasticity inherent in the training process, validation results can exhibit variability across different runs. We report metrics from a single training run to exemplify typical predictive model performance. SystemC TLM simulations are conducted on a workstation with an Intel® Xeon® W-2255 CPU at 3.70 GHz, running Ubuntu Linux 22.04.

Spearman correlation heatmap visualization shows an overview of the relationships between features in the dataset. As shown in Fig. 6, there is little to no monotonic relationship between the pairs of variables, and the variables are essentially independent. This implies that statistical models such as linear regression are not suitable for capturing complex patterns in features.

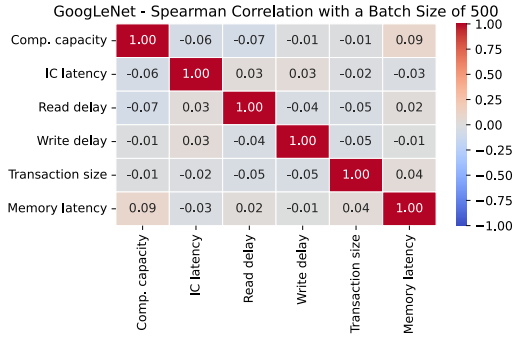
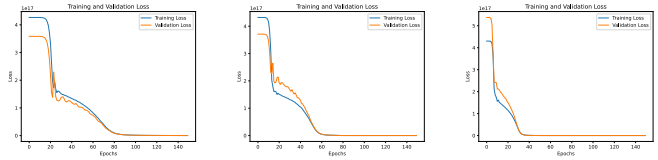


Fig. 6: Spearman correlation heatmap of the features with batch size as 500

Fig. 7 illustrates the training and validation loss of the optimized neural network architecture. With 100 configurations, validation loss fluctuates and remains unstable. Increasing to 200 configurations reduces fluctuation and training epochs.

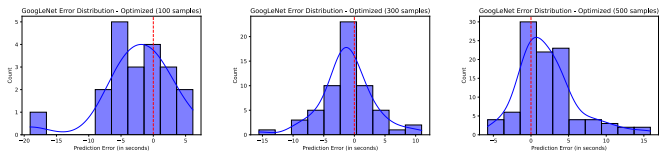
At 300 configurations, validation loss becomes more stable and converges in fewer epochs, indicating optimal performance. Overall, higher configuration counts improve convergence stability and reduce training time, with upwards of 300 configurations yielding more consistent and efficient training outcomes.



(a) (b) (c)

Fig. 7: Training and validation loss for GoogLeNet with batch sizes (a) 100 (b) 200 (c) 300

Fig. 8 demonstrates the residual error distributions of the optimized neural network designed for GoogLeNet. As the number of training samples increases from 100 to 300 and then 500, the optimized model's prediction errors become more tightly clustered and then begin to skew in the positive direction. With 100 samples, the errors have a wider range and the density curve is skewed to the left, showing a strong tendency to overestimate predictions. At 300 samples, the errors have a tighter range and a more normal distribution. Increasing the sample size to 500, the error also has a narrower range and the density curve is slightly skewed to the left, and the model is making slight over-predictions. Overall, 500 samples achieve the best results because a slight over-predictions are preferable to under-predictions.



(a) (b) (c)

Fig. 8: GoogLeNet error distribution plots for different batch sizes (a) 100 (b) 300 (c) 500

Fig. 9 shows the MAPE distribution for the three statistical models designed for GoogLeNet. Following similar pattern in error distribution plots, performance of three architectures improve as the available training data increases. The optimized model performs the best and shows significant results even when trained with a limited dataset. With maximum dataset of 500 samples, the resulting MAPE is around 3.12% allowing the model to perform with 96.88% accuracy. The baseline model achieves 65.08% accuracy, while the brute-force model improves to 87.18% accuracy. Overall, the optimized model performs the best with each set of configurations and is able to adapt and generalize to the data more efficiently and accurately. The MAPE distribution for ResNet shows a similar pattern where the optimized model outperforms the remaining two architectures and is further enhanced when trained with more data.

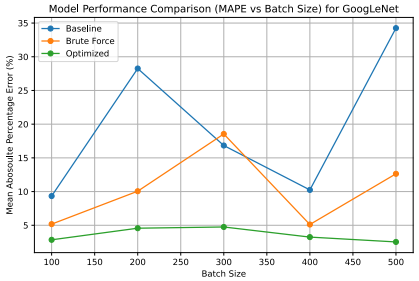


Fig. 9: GoogLeNet Mean Absolute Percentage Error (MAPE) distribution

Fig. 10 focuses on the MAE for the three different architecures of the statistical model designed for GoogLeNet. The optimized version of the model retains the lowest MAE, and on average the models tend to perform better when trained with more data because the MAE shows a decreasing trend. The optimized model when trained with 500 configurations has an MAE of 0.008 seconds. Training with 500 configurations, the baseline model results in an MAE of 0.029 seconds, and the brute force model results in an MAE of 0.012 seconds. Overall, the optimized model outperforms the other architectures due to consistently having a lower MAE compared to the other models. The same trend is observed with the MAE distribution for ResNet except that the MAE range is larger due to the nature of the DNN and has longer ranges for simulated time.

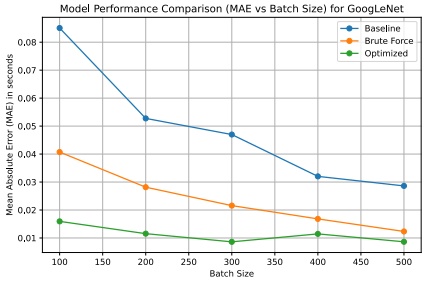


Fig. 10: GoogLeNet Mean Absolute Error (MAE) distribution

Table II shows the representative Pareto optimal set of system-level configurations for the GoogLeNet DNN. These Pareto optimal design candidates allow system-level designers to optimize multi-objective cost tradeoffs without running hundreds of time-consuming simulations.

Finally, as depicted in Fig. 11, the red trendline visually represents the Pareto front. The individual data points along this frontier correspond to design candidates identified as Pareto-optimal; they are non-dominated for the set of objective functions. Being non-dominated implies that no other candidate in the solution space performs better in all objectives simultaneously, thereby establishing these points as optimal tradeoff solutions. Given a rapid performance prediction model, the Pareto front can be instantly obtained and serve as a critical reference in future design stages, enabling informed selection of design candidates that maximize overall performance while adhering to multi-objective considerations.

Execution time cost	Component cost	Memory latency (ns)	Comp. capacity (GFLOPS)	IC latency (ns)	Read delay (ns)	Write delay (ns)
4.86	43.82	1	329	70	66	62
15.65	39.55	13	386	98	99	22
26.29	32.62	28	83	65	60	61
36.92	27.21	39	20	79	33	27
123.86	25.89	132	74	91	90	68
129.83	24.91	139	40	81	97	38
144.75	17.05	155	11	93	74	60
893.34	14.50	959	9	81	55	70

TABLE II: GoogLeNet Pareto-optimal points and their corresponding system-level configurations

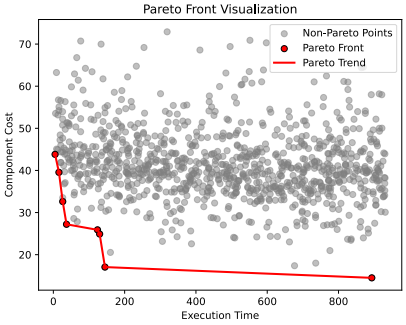


Fig. 11: Pareto set visualization of 1000 design points for GoogLeNet

Fig. 12 compares the runtime for running TLM simulations of GoogLeNet and ResNet, along with using the predictive model. Running more simulations results in a linear increase in runtime for both DNNs. ResNet has a longer runtime and sometimes takes days, depending on the number of simulations. Using the predictive model, the runtime shows no significant increase and can save hundreds of hours of simulations. As a result, the predictive model offers a substantial reduction in computational overhead, potentially saving hundreds of hours compared to traditional simulation methods. Of course, creating a suitable dataset for training the model requires computational resources; however, the benefits of the predictive model amortize these initial training costs.

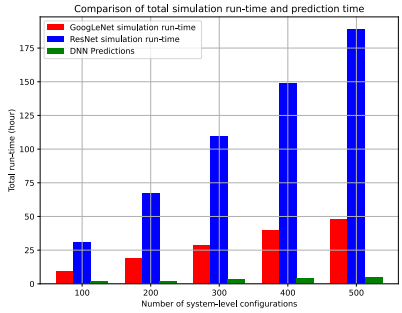


Fig. 12: Comparison of SystemC TLM simulation run-time versus prediction inference time across different system-level configurations for design space exploration

VI. Conclusion

In this work, we proposed a machine learning-based framework to accelerate design space exploration for embedded systems modeled using SystemC TLM-2.0 loosely-timed contention-aware (LT-CA) models. Our approach captured and learned the complex performance behaviors of LT-CA models, including critical memory and interconnect contention effects, enabling fast and accurate performance prediction. Experimental results on two representative deep neural networks, GoogLeNet and ResNet, demonstrated the high accuracy of our model with an MAPE of 3.12%, and an MAE of 0.008 seconds. Leveraging the predictive model, we efficiently explored the design space. We identified 8 optimal design candidates through Pareto analysis, achieving three orders of magnitude speed-up compared to traditional simulation-based approaches. Overall, the proposed framework significantly reduced simulation time and computational cost, providing an effective solution for accelerating system-level modeling and design space exploration of complex embedded systems. In the future, we plan to expand this predictive model approach to apply to other state-of-the-art DNNs, enhance the complexity of our model by implementing new system-level parameters, and add further performance metrics for cost analysis, such as energy, area, and quality.

References

[1] E. M. Arasteh and R. Dömer, "Fast loosely-timed deep neural network models with accurate memory contention," *ACM Trans. Embed. Comput. Syst.*, Aug. 2024.

[2] M. I. Frank, A. Agarwal, and M. K. Vernon, "Lopc: Modeling contention in parallel algorithms," p. 276–287, 1997.

[3] C. Chen and F. Lin, "An easy-to-use approach for practical bus-based system design," *IEEE Trans. Computers*, vol. 48, no. 8, pp. 780–793, 1999.

[4] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.

[5] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*, 2002.

[6] IEEE Computer Society, *IEEE Standard 1666-2011 for Standard SystemC Language Reference Manual*. IEEE, New York, USA, 2011.

[7] E. S. Alcorta, P. Brisk, and A. Gerstlauer, "Ml for system-level modeling," pp. 545–579, 2022.

[8] E. Zennaro, L. Servadei, K. Devarajegowda, and W. Ecker, "A machine learning approach for area prediction of hardware designs from abstract specifications," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 413–420.

[9] D. S. Lopera, L. Servadei, V. P. Kasi, S. Prebeck, and W. Ecker, "Rtl delay prediction using neural networks," in *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*, 2021, pp. 1–7.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[12] P. Marwedel, *Embedded Systems Design*, 4th ed. Springer International Publishing, 2021.