# NetTLMSim: A Virtual Prototype Simulator for Large-Scale Accelerator Networks


Junsu Heo
Konkuk University
Seoul, South Korea
junsuheo@konkuk.ac.kr

Shinyoung Kim
Konkuk University
Seoul, South Korea
shinyoungkim@konkuk.ac.kr

Hyeseong Shin
Konkuk University
Seoul, South Korea
hyeseongshin@konkuk.ac.kr

Jaesuk Lee
Konkuk University
Seoul, South Korea
jaesuki98@konkuk.ac.kr

Sungkyung Park
Pusan National University
Busan, South Korea
fspark@pusan.ac.kr

Chester Sungchung Park
Konkuk University
Seoul, South Korea
chester@konkuk.ac.kr



## ABSTRACT

This paper presents NetTLMSim, a virtual prototype SystemC transaction-level modeling (TLM) simulator for accurate and efficient power-performance-area (PPA) prediction in large-scale accelerator networks. Conventional analytical models often fail to predict PPA accurately due to their lack of capability to capture dynamic network traffic, such as contention, and thus tend to lead to the sub-optimal design options when applied to design space exploration (DSE). NetTLMSim addresses this limitation through cycle-accurate modeling of dynamic traffic in accelerator networks, e.g., reducing delay prediction error by up to 52.8%. In this paper, NetTLMSim is integrated into an existing DSE framework for Transformer layer-pipeline spatial mapping (LP-SM), resulting in up to 39.3% lower network delay compared to the state-of-the-art framework. To further enhance the efficiency, we propose a set of simulation modeling strategies that combine computation skipping, module-specific transaction fidelity, and pass sampling. When applied to the DSE for Transformer LP-SM, it improves runtime by up to 497× over the conventional framework. *The NetTLMSim is open-sourced at https://github.com/SDL-KU/NetTLMSim*



## KEYWORDS

Deep Neural Network Accelerator, Simulation, Network-on-chip


## 1 INTRODUCTION

Deep neural networks (DNNs) have emerged as essential elements across diverse application domains, including image recognition [9, 18], object detection [26, 30], and natural language processing (NLP) [3, 10]. Driven by the need for adaptability to increasingly complex scenarios, modern DNNs continue to increase in depth, resulting in a growing number of layers. Advanced DNNs such as BERT for NLP and large language models (LLMs) clearly demonstrate this trend. For instance, BERT Large contains approximately 340 million parameters, a scale that typically exceeds available on-chip memory capacities. Such DNN models achieve unprecedented performance but require significant computational resources and efficient execution strategies due to their large depth and parameter count.

To meet the increasing computational demands of deeper neural networks, layer-pipeline spatial mapping (LP-SM) techniques have been proposed [11, 27]. The relevant inter-layer scheduling has been extensively studied [4], as it plays a critical role in achieving high resource utilization and data reuse when deploying DNN onto large-scale accelerator networks. The detailed LP-SM partitioning and pipelining have been presented in prior work, e.g., [5]. Nevertheless, to the best of the authors' knowledge, the optimization of LP-SM still remains to be an open problem since the corresponding design space is extremely large. For instance, the LP-SM design space for the DNN layers depicted in the left side of Fig. 1 turns out to contain approximately $36! \times 4^5$, thereby rendering the exhaustive search infeasible. Moreover, the optimization of mapping pipelined layers to computing cores is known to be an NP-hard problem [22].

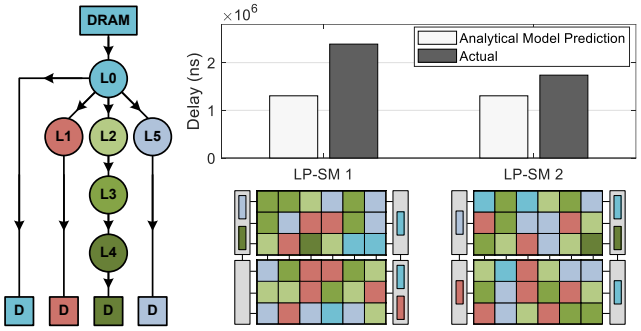



**Figure 1: Prediction inaccuracy in the analytical model. *Left*: Example layer dependency graph of a DNN workload. *Right*: Analytical model predicted delay and actual delay for different LP-SMs.**

Another challenge is to accurately predict the delay of large-scale accelerator networks. Most of the existing design space exploration (DSE) frameworks rely on the analytical models to quickly predict the network delay for a candidate spatial mapping [5, 17, 19, 20, 24]. However, these analytical models often make oversimplifying assumptions that limit their prediction accuracy. For instance, the analytical models typically abstract away critical dynamic traffic such as network contention. Consequently, as shown in the right side of Fig. 1, two different LP-SMs might appear to have identical delays under such coarse analytical modeling (i.e., modeling without dynamic traffic), while actual delays differ significantly. Such discrepancies can severely mislead the corresponding DSE, undermining the efficacy of DSE.

In summary, this paper makes the following contributions:

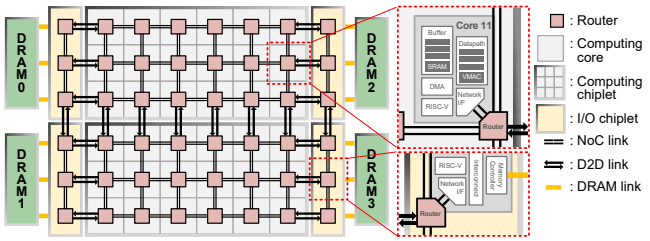



**Figure 2: Architecture of large-scale accelerator network.**

- We propose NetTLMSim, a virtual prototype full-system simulator that simulates the whole system of the large-scale accelerator network based on SystemC transaction level modeling (TLM) [23]. NetTLMSim achieves a substantial reduction in the prediction error of the state-of-the-art analytical delay models, by up to 52.8%, through its cycle-accurate modeling of dynamic traffic of an accelerator network.
- In order to evaluate the effectiveness of precise power-performance-area (PPA) prediction, NetTLMSim was integrated into a state-of-the-art DSE framework, resulting in up to a 37.8% reduction in network delay. Furthermore, the adoption of multi-fidelity modeling led to an exploration speedup of up to 200× compared to the baseline framework.

The remainder of this paper is organized as follows. Section 2 reviews background and related works. Section 3 presents NetTLMSim, a virtual prototype full-system simulator for large-scale accelerator networks. Section 4 provides the experimental results, highlighting the impact of the accurate PPA prediction. Finally, Section 5 concludes the paper.

# 2 BACKGROUND AND RELATED WORKS

## 2.1 Large-Scale Accelerator Network

Recent works such as Simba [27, 33], TPU [12], and Tianjic [7] have introduced large-scale accelerator network architectures to meet the increasing demand for high-throughput DNN inference. As described in Fig. 2, the state-of-the-art accelerator networks typically comprise one or more I/O chiplets and multiple computing chiplets that communicate through high-bandwidth die-to-die (D2D) links implemented on the package substrate [25]. Each I/O chiplet integrates a memory controller for DRAM access and connects to the computing chiplets through D2D interfaces. Each compute chiplet hosts an array of computing cores; every core integrates (i) a vector MAC (VMAC) datapath for pipelined layer execution, (ii) a local SRAM buffer for operand tensors, (iii) a DMA engine, and (iv) a network interface. The on-chip network (NoC) employs routers arranged in a 2-D mesh topology, offering bidirectional NoC links as well as D2D ports in all four cardinal directions. A network interface translates Advanced eXtensible Interface(AXI)-based intra-core transactions [1], adopted from Simba [33], into inter-core or core-to-DRAM messages. A dedicated RISC-V processor then schedules and orchestrates the resulting transfers.

## 2.2 Design Space Exploration

Design space exploration (DSE) plays a critical role in identifying efficient mappings for large-scale accelerator networks, where both hardware and dataflow parameters must be jointly optimized to maximize performance and resource utilization. Among the various scheduling strategies considered in DSE, LP-SM has become increasingly important with the growing scale of deep neural network (DNN) accelerators [4, 11, 27]. LP-SM extends beyond intra-layer scheduling [17, 19, 20, 24, 31] by orchestrating inter-layer execution across multiple processing elements in the network. In this work, LP-SM refers to the combination of the *core group* assigned to each pipelined operation at a specific core and the *flow of data* that determines where operand tensors are stored [5]. These *core group* and *flow of data* pairs form the fundamental source–destination relationships that guide both computation and communication within the accelerator network.

## 2.3 PPA Prediction

The accuracy and the PPA prediction runtime of PPA models widely vary depending on the modeling details [16].

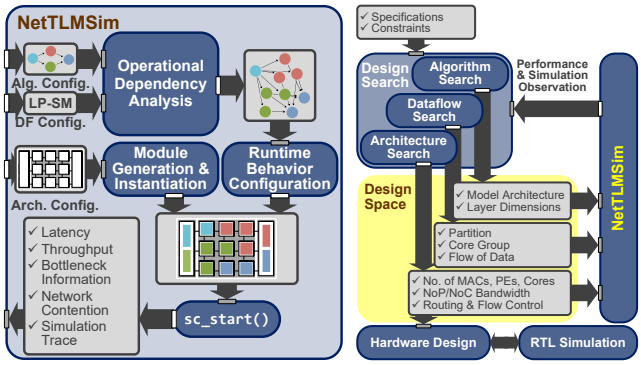
*2.3.1 Analytical Model.* The state-of-the-art PPA model for LP-SM in the Gemini framework [5] estimates delay as the maximum of computation and communication components. Computation delay is derived from the operation count of each pipelined layer divided by its throughput, while communication delay is computed from the data volume over each link or DRAM divided by the corresponding bandwidth. This average-rate approximation is computationally inexpensive but overlooks important effects such as network contention (e.g., head-of-line blocking and back-pressure) and per-hop pipeline/arbitration dely. These unmodeled effects accumulate under heavy load, leading to significant prediction errors and potentially steering the DSE toward suboptimal design choices.

*2.3.2 Simulator.* Simulation-based PPA predictions generally achieve smaller errors than analytical models [2, 6, 21]. However, as accelerator networks scale up, the runtime of virtual prototype simulators becomes prohibitively large for DSE [28]. To address this challenge, we adopt a virtual prototype full-system simulator based on SystemC/TLM 2.0, which is known to be 100–10,000× faster than RTL simulation [23]. Transaction-level modeling (TLM) has been successfully applied in prior work to efficiently estimate communication performance in accelerators [14, 15], demonstrating its capability to capture system-level behavior at high speed. Our proposed simulator follows this approach, enabling accurate yet fast delay estimation suitable for large-scale accelerator network exploration.

# 3 NETTLMSIM

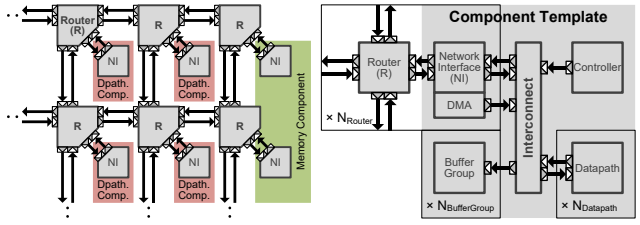
## 3.1 Simulator Overview

NetTLMSim is a virtual prototype simulation framework for large-scale accelerator networks, built on a modular architecture to flexibly model both hardware (HW) and software (SW) aspects of the system. As shown in Fig. 3(a), the simulation process (Fig. 3(a)) begins by parsing multiple configuration inputs, including algorithm, LP-SM, dataflow, and architecture settings. The architecture configuration defines the structural composition of the network, such as topology, link bandwidth, and component organization, while the other inputs specify execution behavior and dataflow. These parameters are passed through the operational dependency




(a) Simulation Process (b) Framework

**Figure 3: Simulator overview.**




(a) Testbench (b) Component Template

**Figure 4: Component-based modeling.**

analysis, module generation and instantiation, and runtime behavior configuration stages before launching the SystemC kernel (`sc_start()`).

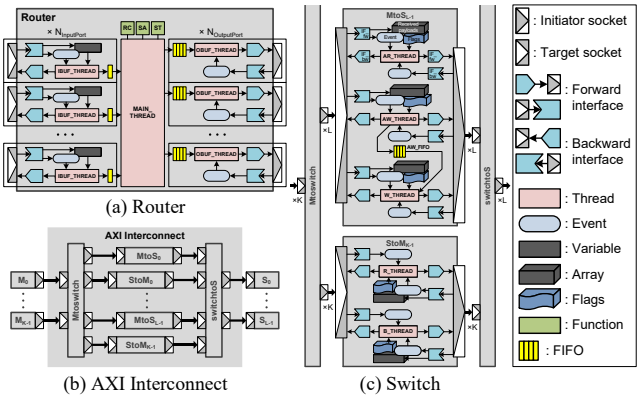
During execution, NetTLMSim collects detailed performance statistics—latency, throughput, bottleneck locations, network contention, and simulation traces—providing deep insight into system behavior.

NetTLMSim can also be integrated into a DSE environment as a performance evaluation back-end, as illustrated in Fig. 3(b). The DSE framework explores the design space across multiple levels—algorithm, dataflow, and architecture search—by generating candidate configurations and passing them to NetTLMSim, which returns precise performance and simulation observations. This integration allows the exploration process to converge toward optimal or near-optimal designs, while maintaining flexibility to architectural and control options without modifying the simulation infrastructure.

## 3.2 Component-Based Modeling for Flexible System Configuration

NetTLMSim adopts a component-based modeling approach to enable fast and efficient construction of diverse accelerator network configurations. Each hardware block—such as computing cores and memories—is modeled as an independent component with predefined interfaces. As illustrated in Fig. 4(a), these components are instantiated and interconnected within the testbench according to the target architecture.

Within each component, individual modules can be selectively activated, deactivated, or replicated according to the intended functionality. For example, when modeling a PE, computation datapaths




(a) Router (b) AXI Interconnect (c) Switch

**Figure 5: Network-on-Chip and interconnect modeling.**

and local buffers can be enabled while unused modules are disabled to reduce simulation complexity. Similarly, the same base component template can be repurposed as a GB by enabling only the required buffer modules and configuring the necessary number of network interfaces (NIs). As shown in Fig. 4(b), a GB may connect to multiple routers in the network, and thus requires multiple NIs to interface with each router.

Routers, in contrast, are placed outside of the component templates to maximize flexibility in network topology design. This separation allows the simulator to independently configure and interconnect routers according to the desired NoC/NoP architecture, making it straightforward to construct a wide variety of topologies without altering the internal structure of other components.

By combining modular component templates with externally configurable routers, NetTLMSim supports rapid and highly customizable system composition, enabling efficient evaluation of diverse architectures.

## 3.3 Network-on-Chip and Interconnect Modeling

While many modules in NetTLMSim share a similar modeling structure, this section focuses on modules related to communication as representative examples—specifically the router and AXI interconnect. These modules are modeled at the transaction level to capture both functional correctness and performance characteristics such as contention, arbitration delay, and resource utilization.

The router model, illustrated in Fig. 5(a), is modeled with a parameterizable number of input and output ports, enabling it to support a wide range of network parameters. Each port is equipped with both a TLM *initiator socket* and a *target socket* to support bidirectional communication. The initiator socket is used to send transactions forward to the next module, while the target socket receives incoming transactions from the previous module. In this design, the forward path is responsible for delivering data (e.g., flits) toward their destination, and the backward path carries responses or acknowledgments back to the source. This separation of forward and backward paths allows NetTLMSim to accurately model the timing and ordering of both data delivery and response propagation. Incoming flits are received into per-port FIFOs, triggering SystemC events that invoke the main routing thread. This main

**Table 1: Configurable parameters of an individual router in NetTLMSim.**

| Parameters | Description |
|---|---|
| flow_cntl | Flow control |
| sw_arb | arbitration policy for switch allocation |
| len_pkt | Maximum length of a packet |
| bitw_flit | Bitwidth of a flit |
| bitw_port | Bitwidth of a port |
| clk_router | Clock period of router |
| clk_port | Clock period of a I/O port |
| delay_vc_alloc | Latency for virtual channel allocation |
| delay_sw_alloc | Latency for switch allocation |
| delay_credit | Latency for credit input to credit update |
| delay_hop | Latency per hop traversal |
| num_vc | Number of an virtual channel per port |
| depth_vc | FIFO depth of a single virtual channel |

thread performs three sequential stages: route computation (RC), virtual channel/switch allocation (SA), and flit switching/traversal (ST). Route computation determines the next hop based on the configured routing table and algorithm (e.g., DOR X–Y), while switch allocation resolves contention among incoming flits targeting the same output port. Traversal transfers flits across the internal crossbar to the designated output FIFO. All stages are synchronized with the simulation clock, ensuring accurate modeling of pipeline delays and contention effects. Parameters such as flow control, arbitration policy, virtual channel depth, and allocation delays—summarized in Table 3—are fully configurable, allowing exploration of different network microarchitectures.

The AXI interconnect model, illustrated in Fig. 5(b) and (c), supports multiple master and slave ports and implements the AXI protocol's separate channels for read/write address, write data, read data, and write response. Each channel is represented with dedicated threads and events to accurately capture handshake timing and transaction behavior. As shown in Fig. 5(b), the interconnect is organized into two main switch stages: *MtoS* (master-to-slave) and *StoM* (slave-to-master).

Fig. 5(c) provides a detailed view of an individual switch module used within these stages. Each switch contains separate submodules for each channel (e.g., AR, AW, W, R, B), each with its own thread to process transactions and FIFOs for buffering. Switch modules arbitrate between concurrent requests from multiple ports, and parameters such as arbitration policy, transaction latency, and buffer depth are fully configurable to match specific design requirements. This modular switch design allows the interconnect to scale efficiently while maintaining accurate per-channel modeling of AXI protocol behavior.

By combining these detailed communication module models, NetTLMSim provides a accurate representation of diverse on-chip network configurations. This enables thorough performance analysis under varying workloads and architectural parameters, while maintaining consistency with the configurable network parameter.

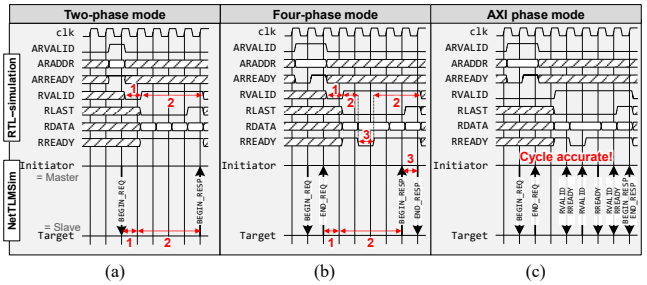



**Figure 6: Comparison of transaction-level modeling modes in NetTLMSim: (a) Two-phase, (b) Four-phase, and (c) AXI phase.**

## 3.4 Simulation Modeling Strategies

To flexibly balance simulation accuracy and runtime, NetTLMSim provides three complementary simulation modeling strategies. Each strategy offers distinct trade-offs between prediction accuracy and simulation speed, enabling users to select appropriate modes depending on the stage and purpose of DSE. The strategies are:

**(1) Computation skipping:** By default, NetTLMSim ensures functional correctness by performing actual computations between activations and weights in the neural network workload. In scenarios where functional verification is unnecessary, a computation-skipping mode can be enabled. In this mode, arithmetic operations are bypassed and replaced with estimated execution latencies, reducing simulation time significantly for computation-heavy workloads.

**(2) Module-specific transaction fidelity:** The transaction-level modeling fidelity can be independently configured for each module in the accelerator network. Modules with low performance sensitivity can use coarser-grained abstraction, while timing-critical modules employ cycle-accurate modeling. As shown in Fig. **??**, NetTLMSim supports two-phase, four-phase, and AXI phase modes. The AXI phase mode augments the standard four-phase handshake with additional protocol phases to accurately capture beat-level handshake delays in interfaces such as DRAM controllers.

**(3) Pass sampling:** In NetTLMSim-Full, all pipeline passes are simulated with full detail. For example, in a workload with 70 passes, passes 7–63 may share identical execution behavior; all are modeled in NetTLMSim-Full, though redundant passes reuse the first simulation's timing. As illustrated in Fig. 7, each pass consists of a sequence of load, compute, and store operations across multiple cores and DRAM modules, with many passes exhibiting identical timing patterns. In contrast, NetTLMSim-Fast selects a single representative pass from the entire workload and simulates only that pass in detail, skipping the rest. Because the execution patterns of the skipped passes are equivalent to the simulated one, this approach significantly accelerates simulation while maintaining close correlation with NetTLMSim-Full results.

As summarized in Table 2, each strategy can be tuned to achieve different points in the accuracy–speed space. In practice, combining these strategies allows tailoring the simulator to specific DSE phases—for instance, using faster, lower-fidelity settings during early exploration, and switching to slower, high-fidelity modes for final design validation.

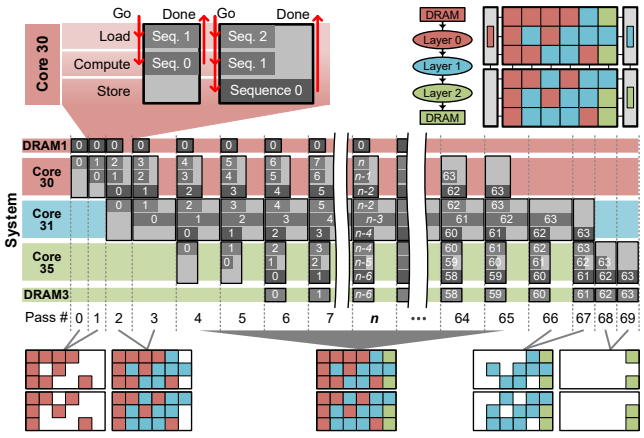



**Figure 7: Timing diagram of large-scale accelerator network under an example LP-SM.**

**Table 2: Trade-offs between accuracy and speed for NetTLMSim strategies and modes.**

| Strategy | Mode | Accuracy | Speed |
|---|---|---|---|
| Computation skipping | Full computation | High | Low |
| | Skip computation | Medium | High |
| Module-specific fidelity | AXI phase | High | Low |
| | Four-phase | High–Med | Med |
| | Two-phase | Medium | High |
| Pass sampling | NetTLMSim-Full | High | Low |
| | NetTLMSim-Fast | High–Med | High |

## 3.5 Simulator Extensibility

NetTLMSim offers high extensibility, enabling seamless integration with both commercial tools and academic research frameworks. First of all, it can be easily plugged into the commercial simulation libraries such as Synopsys's Platform Architect[29] through module-level plug-in, allowing interoperability with existing simulation IPs without modification. This compatibility ensures that NetTLMSim can be readily incorporated into established virtual prototyping workflows alongside other performance models and architectural components. For academic research frameworks, NetTLMSim can be readily applied to the existing DSE frameworks such as Gemini[5]. A compact interface script enables input and output log exchange via files, requiring no invasive changes to the DSE infrastructure. This flexibility allows rapid deployment of NetTLMSim in various exploration pipelines while preserving accuracy and minimizing integration effort.

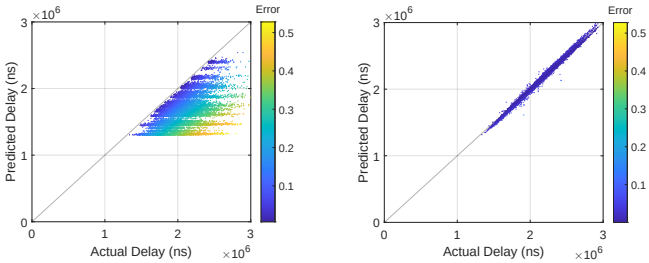
# 4 SIMULATION RESULTS

## 4.1 Experimental Setup

The DSE parameters considered in this study and their corresponding values are summarized in Table 3. To focus on analyzing the impact of LP-SM on network delay, all hardware and other design parameters except LP-SM were fixed to a single configuration. The workload used is the BERT-Large[8] model adopted from Gemini, with dimensions and layer configurations unchanged. Additionally, the layer grouping, partitioning, and hardware parameters from the optimal design point identified by the Gemini DSE framework were directly adopted. Most parameters not listed in the table were taken from Gemini[5] and SIMBA[27], while the core configuration follows MAGNet[31] and the NoC router is based on the assumptions from Matchlib[13].

**Table 3: DSE parameters**

| | | |
|---|---|---|
| **System** | No. Chiplets in the X-direction | 1 |
| | No. Chiplets in the Y-direction | 2 |
| | No. Cores in the X-direction | 6 |
| | No. Cores in the Y-direction | 6 |
| | No. DRAM Devices | 4 |
| | DRAM Bandwidth per Device | 32 GB/s |
| | Clock Frequency | 1 GHz |
| **Network** | Topology | Mesh |
| | Routing Algorithm | X-Y DOR |
| | Flow Control | Wormhole (unicast), Cut-through (multicast) |
| | Multicast Routing | Tree-based |
| | Bandwidth | 32 GB/s (NoC link), 16 GB/s (D2D link) |
| **Core** | No. MAC Units | 1024 |
| | SRAM Capacity | 2 MB |




**(a) Analytical Model Pearson Corr.=0.564** **(b) NetTLMSim-Fast Pearson Corr.=0.997**

**Figure 8: Delay prediction error for 30,000 LP-SMs obtained from 10 DSE runs.**

## 4.2 Accuracy–Speed Trade-offs in Delay Prediction

Fig. 8(a) shows that the analytical model exhibits low correlation with the actual delay (Pearson Corr. = 0.564) and incurs a prediction error of up to 52.8%. Such inaccuracy can lead to erroneous delay predictions for design options in DSE, thereby reducing the likelihood of identifying optimal or near-optimal configurations. In contrast, Fig. 8(b) shows that delays predicted by NetTLMSim-Fast have a high correlation with the actual delay (Pearson Corr.=0.997), demonstrating its capability to closely approximate real delay values. NetTLMSim-Full, which models execution in full cycle-accurate detail, produces delays identical to the actual delay, ensuring maximum prediction accuracy but at a higher simulation cost (13–20 s). In comparison, NetTLMSim-Fast achieves a similar level of accuracy with significantly lower prediction time (1.5–3 s), while the analytical model offers the fastest prediction (150–200 ms) but with low accuracy. As summarized in Table 4, these results highlight the clear trade-off between accuracy and performance, with NetTLMSim-Fast providing a balanced option for use in DSE.

**Table 4: Prediction speed of different delay predictions.**

| Delay Prediction | Prediction Speed |
|---|---|
| Analytical model | 150-200 ms |
| NetTLMSim-Fast | 1.5-3 s |
| NetTLMSim-Full | 13-20 s |

## 4.3 Impact of Delay Prediction Accuracy on DSE Efficiency

Fig. 9 shows the delay distributions of the best LP-SMs identified using each delay prediction method at three different exploration times: 10, 30, and 90 minutes. When the analytical model is used, the distribution remains broad throughout the exploration process, indicating that the predicted performance of the selected LP-SMs varies widely from the actual performance. This wide spread is a direct consequence of the prediction errors inherent to the analytical model, which cause the DSE to misjudge the relative quality of candidate designs. As a result, many designs that appear promising in the prediction phase turn out to be far from the true optimum when evaluated accurately. In contrast, NetTLMSim based predictions exhibit a much narrower distribution, with most selected LP-SMs yielding performance near the theoretical limit given by the Roofline model [32].

To fairly compare prediction methods under such variability, we adopt the 99th percentile delay as a robust performance metric. Unlike the minimum or average delay, which may be skewed by outliers or overly optimistic predictions, the 99th percentile delay provides a conservative and reliable estimate of the best achievable performance that the DSE method can consistently guarantee. This is especially important in cases like the analytical model, where poor prediction accuracy can result in wide and misleading distributions.

Fig. 10 compares the 99th percentile delays obtained by DSEs using the analytical model and NetTLMSim. Fig. 10(a) summarizes the results of 90-minute DSE runs for each layer group in the Transformer encoder block. NetTLMSim yields a substantial reduction for layer groups 0 and 1, because communication delay dominates computation delay. A more detailed view of layer group 0 is shown in Fig. 10(b), where each curve reports the 99th-percentile delay over 100 independent DSE runs. With the analytical model, the 99th-percentile remains essentially flat, indicating limited ability to surface better designs. In contrast, with NetTLMSim-Fast the 99th-percentile decreases consistently and, after 90 minutes, delivers an LP-SM with 39.3% lower delay than the analytical model. Although NetTLMSim-Fast offers lower per-prediction speed than the analytical model (Table 4), its higher sample efficiency yields faster convergence.

Fig. 11 extends this comparison by measuring the runtime required for each method to achieve a target delay with 99% probability. This metric captures the combined effects of prediction accuracy and search efficiency. The analytical model's inaccuracies cause the required DSE runtime to grow rapidly as the target delay approaches the theoretical limit, leading to an extreme disparity—**497**× increase for the DSE runtime required to reach a target delay within **5%** of the theoretical limit—compared to the NetTLMSim-Fast. In contrast, NetTLMSim-Fast maintains a much lower runtime

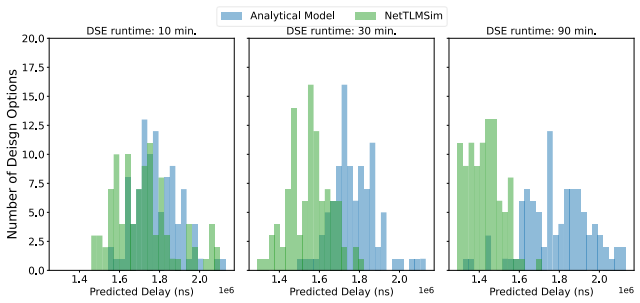



**Figure 9: Delay distributions of the best LP-SMs found using each delay prediction**

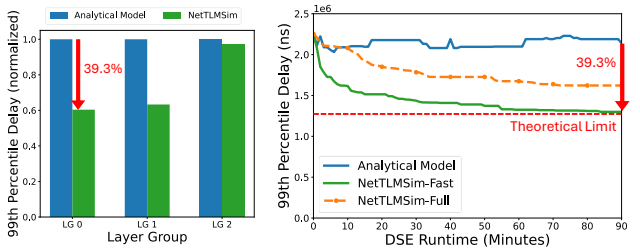



**Figure 10: Comparison between DSEs with analytical model and NetTLMSim. (a) DSE results after 90 minutes for each layer group in Transformer. (b) DSE results for layer group 0.**

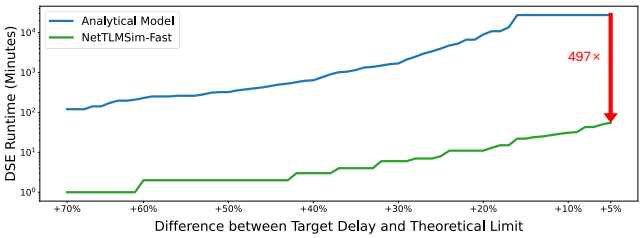



**Figure 11: DSE runtime required to meet the target delay with 99% probability.**

across all target delay thresholds, as its accurate predictions consistently guide the DSE toward high-quality designs without wasted exploration on misleading candidates. This result reinforces the central finding: improving delay prediction accuracy translates directly into faster, more reliable attainment of near-optimal performance in DSE.

## 5 CONCLUSION

In this work, NetTLMSim is introduced as a virtual prototype simulator designed to improve PPA prediction accuracy in large-scale accelerator networks. By explicitly modeling dynamic network traffic, including contention effects, it reduces delay prediction error by up to 52.8% compared to conventional analytical models. Integration with a Transformer LP-SM DSE framework demonstrates that accurate PPA prediction enables more effective exploration, achieving up to 39.3% lower network delay over a state-of-the-art

framework. A set of simulation modeling strategies, which combine computation skipping, module-specific transaction fidelity, and pass sampling, is also proposed, accelerating exploration by up to 497× over the conventional DSE framework. These results demonstrate that accurate and efficient simulation can significantly enhance the effectiveness of DSE in large-scale accelerator networks. As future work, the simulation logs generated by NetTLMSim will be leveraged to train AI/ML-based models, enabling log-driven DSE that can further improve exploration speed and quality.

## ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2023-00222085, Development of memory module and memory compiler for non-volatile PIM optimized for data characteristics and data access characteristics of AI processor.)

## REFERENCES

[1] ARM. 2017. AMBA AXI and ACE Protocol Specification AXI3, AXI4, AXI5, ACE and ACE5. https://arm.com

[2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. https://doi.org/10.1145/2024716.2024718

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Jingwei Cai, Yuchen Wei, Zuotong Wu, Sen Peng, and Kaisheng Ma. 2023. Inter-layer scheduling space definition and exploration for tiled accelerators. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–17.

[5] Jingwei Cai, Zuotong Wu, Sen Peng, Yuchen Wei, Zhanhong Tan, Guiming Shi, Mingyu Gao, and Kaisheng Ma. 2024. Gemini: Mapping and Architecture Co-exploration for Large-scale DNN Chiplet Accelerators. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 156–171. https://doi.org/10.1109/HPCA57654.2024.00022

[6] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[7] Lei Deng, Guanrui Wang, Guoqi Li, Shuangchen Li, Ling Liang, Maohua Zhu, Yujie Wu, Zheyu Yang, Zhe Zou, Jing Pei, et al. 2020. Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation. *IEEE Journal of Solid-State Circuits* 55, 8 (2020), 2228–2246.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

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

[10] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39.

[11] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. Tangram: Optimized coarse-grained dataflow for scalable nn accelerators. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 807–820.

[12] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[13] Brucek Khailany, Evgeni Khmer, Rangharajan Venkatesan, Jason Clemons, Joel S Emer, Matthew Fojtik, Alicia Klinefelter, Michael Pellauer, Nathaniel Pinckney, Yakun Sophia Shao, et al. 2018. A modular digital VLSI flow for high-productivity SoC design. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.

[14] Sunwoo Kim, Sungkyung Park, and Chester Sungchung Park. 2021. System-level communication performance estimation for DMA-controlled accelerators. *IEEE Access* 9 (2021), 141389–141402.

[15] Sunwoo Kim, Jooho Wang, Youngho Seo, Sanghun Lee, Yeji Park, Sungkyung Park, and Chester Sungchung Park. 2020. Transaction-level model simulator for communication-limited accelerators. *arXiv preprint arXiv:2007.14897* (2020).

[16] Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Jason Jabbour, Ikechukwu Uchendu, Susobhan Ghosh, Behzad Boroujerdian, Daniel Richins, Devashree Tripathy, Aleksandra Faust, et al. 2023. Archgym: An open-source gymnasium for machine learning assisted architecture design. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–16.

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

[18] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*. 10012–10022.

[19] Liqiang Lu, Naiqing Guan, Yuyue Wang, Liancheng Jia, Zizhang Luo, Jieming Yin, Jason Cong, and Yun Liang. 2021. Tenet: A framework for modeling tensor dataflow based on relation-centric notation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 720–733.

[20] Liqiang Lu, Zizhang Luo, Size Zheng, Jieming Yin, Jason Cong, Yun Liang, and Jianwei Yin. 2023. Rubick: A Unified Infrastructure for Analyzing, Exploring, and Implementing Spatial Architectures via Dataflow Decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 4 (2023), 1177–1190.

[21] Francisco Muñoz-Martínez, José L Abellán, Manuel E Acacio, and Tushar Krishna. 2021. Stonne: Enabling cycle-level microarchitectural simulation for dnn inference accelerators. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 201–213.

[22] Umit Y Ogras, Jingcao Hu, and Radu Marculescu. 2005. Key research problems in NoC design: a holistic perspective. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 69–74.

[23] OSCI, Open SystemC Initiative (OSCI). 2008. *TLM–2.0 User Manual*.

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

[25] John W Poulton, John M Wilson, Walker J Turner, Brian Zimmer, Xi Chen, Sudhir S Kudva, Sanquan Song, Stephen G Tell, Nikola Nedovic, Wenxu Zhao, et al. 2018. A 1.17-pJ/b, 25-Gb/s/pin ground-referenced single-ended serial link for off-and on-package communication using a process-and temperature-adaptive voltage regulator. *IEEE Journal of Solid-State Circuits* 54, 1 (2018), 43–54.

[26] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[27] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 14–27. https://doi.org/10.1145/3352460.3358302

[28] Lukas Steiner, Matthias Jung, Felipe S Prado, Kirill Bykov, and Norbert Wehn. 2022. Dramsys4. 0: An open-source simulation framework for in-depth dram analyses. *International Journal of Parallel Programming* 50, 2 (2022), 217–242.

[29] Synopsys, Inc. [n. d.]. Synopsys Platform Architect Ultra. https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html. Accessed: 2025-08-06.

[30] Mingxing Tan, Ruoming Pang, and Quoc V Le. 2020. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10781–10790.

[31] Rangharajan Venkatesan, Yakun Sophia Shao, Miaorong Wang, Jason Clemons, Steve Dai, Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. Magnet: A modular accelerator generator for neural networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[32] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (April 2009), 65–76. https://doi.org/10.1145/1498765.1498785

[33] Brian Zimmer, Rangharajan Venkatesan, Yakun Sophia Shao, Jason Clemons, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2020. A 0.32–128 TOPS, scalable multi-chip-module-based

deep neural network inference accelerator with ground-referenced signaling in 16 nm. *IEEE Journal of Solid-State Circuits* 55, 4 (2020), 920–932.