# DeepSim: A Transformer Based Model For Fast Simulation And Exploring Computer System Design Space

Hamed Najafi
hnaja002@fiu.edu
Florida International University
Miami, USA

Xiaoyang Lu
xlu40@hawk.iit.edu
Illinois Institute of Technology
Chicago, USA

## 1 INTRODUCTION

Here in this abstract, I am going to point out two Computer system simulation issues and propose one solution for both. Computer system simulations are a vital part of numerous research on various topics regarding new ideas in this field. Using simulation gives us the opportunity to explore improvement ideas in this area and have a mindset of implementing them in system design. Gem5 [1] for instance is a mature simulator that has been used for years. However, running a simulation/emulation can be considerably time-consuming and that will hinder the research process especially if a huge number of simulations are required relative to the topic. Another issue that scholars in computer architecture design are facing is how can they find an optimized system configuration as the existing design space exploration methods are not satisfying. The second problem is even harder since we are not equipped with a strong tool to tell us what system configuration is the most optimized one since existing works are not measured with current intense workloads that affect the desired system configuration. A system configuration is just a trial backed up with research and even if the research behind it is strong, we may still be stuck in the local minimum in design space.

To address the first issue, the fact is, not in all the research topics do we need a full-fledged accurate report after running the simulation. In many examinations, we change a small portion of the input parameters in the simulation input such as cache size and run but no matter what the change is, the time required to get the result will be the same or worst. Also, we may just need to look into certain output parameters like IPC but we do not get to specify that in order to get a faster run unless we modify the simulation code each time. Here in DeepSim, we train a transformer network [10] with a possible combination of input parameters in the area of interest, in order to learn the correlation between design configuration and the result. Later on, we explain why we choose transformers and why will that work.

The second obstacle we have to obtain optimal configuration is the difficulty of the problem. In terms of simulation and emulation of a computer system, we can agree that the complexity of the system is above our general understanding since the effect of changing the system parameters will not be linear. Take a last-level cache size as an instance, if we only change this cache capacity, we will have a considerable number of changes happening in IPC, miss rate, C-AMAT [9], and many more. Another example is choosing cache size considering the Miss Rate Ratio (MRC). We see that increasing cache size cache does not necessarily reduce the miss rate at that point rather the function is like a stair function. Knowing the behavior of MRC can also be helpful for cache partitioning [8] and cache allocation in system design. The benefits of this trustworthy network can be used in abundant areas. Here at DeepSim, we offer a change in scenery that gives us power over the design space exploration problem. In the next paragraph, we will explain our reasoning for why transformers are a valuable tool for this task.

Even though transformers mainly are used as a language model, they are widely popular for their accuracy and robustness in other areas as well. When it comes to mimicking a simulator to predict its result or exploring the design space, we are talking about multiple chains of changes happening in parallel where a small change can cause a domino effect in the outcome. The self-attention layer looks for clues in each input and adds or changes the result considering it. It, by nature, will do the same. Transformers pay equal attention to each input and that adds a level of generality that is hard to gain with CNN or ANN. At the same time, there are input parameters that are more of a game changer like the number of cores in the CPU. The attention mechanism will enhance these important parts of the input, and as a result, transformers are appropriate tools for this topic.

In past recent years, it was always an idea to use Machine Learning approaches to guess what system configuration is a better choice as the design space has a gigantic number of possibilities, such as memory hierarchy parameters. In [5] they used ANN as a model which relatively generated acceptable results. However, after 17 years, due to the increasing trend of complex applications and traces, in addition to the nowadays complexity of system designs such as memory hierarchy and newer technology, we need a much more sophisticated model to gain insight into today's design space. Many other works have tried to explore design space utilizing analytical techniques or Machine Learning methods such as Deep Reinforcement Learning[2–4, 6], however, every one of them has its limitations. Having a transformer network to give us a fast reliable result for the configuration in mind can have many applications that can, now, be achieved. Transformers are the most robust network so far when we have enough data to train them.

## 2 MODEL DESIGN

Our input features of the transformer are either the parameters that we set when we execute the simulation or they are the result of the simulation after running it with those parameters. Similarly, the output prediction of the transformer can be the result of the simulation or can be the input parameters– declared as simulation prediction as is shown in figure 1. For instance, instead of predicting

**Table 1: Modifiable parameters in ChampSim**

| Parameter Name | Info |
|---|---|
| Number of cores | number of simulated cores |
| L1I Cache | size, associativity, block size |
| L1D Cache | size, associativity, block size |
| L2 Cache | size, associativity, block size |
| Last Level Cache | size, associativity, block size |
| Prefetcher | prefetching methods |
| Branch predictor | bimodal, gshare, tournament, TAGE |
| Replacement policy | LRU, LFU, or RRIP |
| Warmup instructions | # of instructions to warm up |
| Simulation instructions | # of instructions to simulate |
| Trace file | Input trace file for the simulation |
| Total points in design space ≈ | $2.41 * 10^{12}$ |

the simulation result, if we want to set a goal for a parameter such as miss rate and want to explore the design space, we can flip the input by output and train the network backward to gain power over the certain desired result such as IPC and ask the network for its prediction about the ideal system that can gain that IPC. by following each color in figure 1, we can see the process. previous studies like [5] had a similar idea of using an artificial neural network (ANN) as a model, which is a lot simpler and relatively weaker model, and gained promising results as they were able to predict IPC with a 1-2% error.

## 2.1 Dataset

For this work, we consider ChampSim as a dataset generator source. we run ChampSim with all the combinations possible at the input to generate all the report output at once to generate data. ChampSim simulator has certain execution configuration parameters as shown in table1 that we can modify to generate a variety of output results to make up the dataset. An example of the number of possible points in the design space is shown as well. For instance, we can consider 4 different cache capacities for these 3 level caches. This alone gives us 12 different configurations. If we go through all these 11 parameters, we make billions of configurations that give us a rich unbiased dataset to feed the network. In addition to the aforementioned parameters, many other parameters can be customized inside the simulator code such as DRAM, memory controller, latency, etc. As a result, we have access to a tremendous amount of training data.

# 3 ROAD MAP

Two outcomes of this research are:

- We introduced a model that is trained with simulation input parameters as features that can accurately generate simulation output as it is shown in figure 1 so we do not have to run a whole simulation for every insignificant change, instead, we do an inference from the network. This can save a lot of time during research.
- Exploring the system design space is a complex problem that DeepSim has solved due to the magnitude of the available training data and the availability of state-of-the-art Transformers networks. Since the relationship between different outcome performance and input futures are not clear to us, we can train the network with outputs that we have from our simulator and train it. In the prediction part, we can ask for a miss rate and see how the network informs us about cache configuration and etc to gain that miss ratio as it is shown in figure 1. The same technique–CBOW and Skip-gram– is used in NLP studies[7].

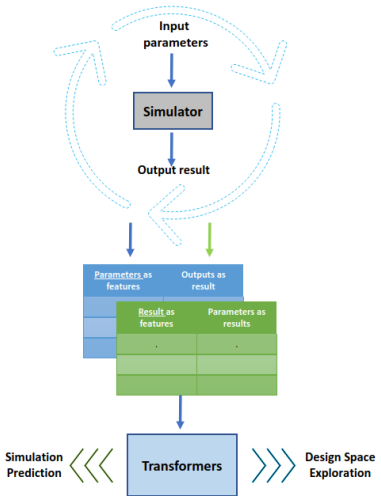


**Figure 1: DeepSim block diagram**

In the end, we argue that a learning model like a transformer can be used for two purposes in the most optimized way. First, it can be trained to be used instead of a simulator when there is no need for a full-fledged result. Second, by exchanging the input for output, we can efficiently explore the design space regardless of the starting point problem. In this way, we need to generate the dataset using our simulator and train the network.

# REFERENCES

[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.

[2] Tianshi Chen, Yunji Chen, Qi Guo, Zhi-Hua Zhou, Ling Li, and Zhiwei Xu. 2014. Effective and efficient microprocessor design space exploration using unlabeled design configurations. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 1 (2014), 1–18.

[3] Christophe Dubach, Timothy Jones, and Michael O'Boyle. 2007. Microarchitectural design space exploration using an architecture-centric approach. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 262–271.

[4] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. 2020. DRiLLS: Deep reinforcement learning for logic synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 581–586.

[5] Engin Ïpek, Sally A McKee, Rich Caruana, Bronis R de Supinski, and Martin Schulz. 2006. Efficiently exploring architectural design spaces via predictive modeling. *ACM SIGOPS Operating Systems Review* 40, 5 (2006), 195–206.

[6] Rik Jongerius, Andreea Anghel, Gero Dittmann, Giovanni Mariani, Erik Vermij, and Henk Corporaal. 2017. Analytic multi-core processor model for fast design-space exploration. *IEEE Trans. Comput.* 67, 6 (2017), 755–770.

[7] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168* (2013).

[8] G Edward Suh, Srinivas Devadas, and Larry Rudolph. 2001. Analytical cache models with applications to cache partitioning. In *ACM International Conference on Supercomputing 25th Anniversary Volume*. 323–334.

[9] Xian-He Sun and Dawei Wang. 2013. Concurrent average memory access time. *Computer* 47, 5 (2013), 74–80.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).