Memory Workload Synthesis Using Generative AI

Chengao SHI
Hong Kong University of Science and Technology
Hong Kong, China

Fan JIANG
Hong Kong University of Science and Technology
Hong Kong, China

Zhenguo LIU
Hong Kong University of Science and Technology (GZ)
Guangzhou, Guangdong, China

Chen DING
University of Rochester
Rochester, New York, USA

Jiang XU
Hong Kong University of Science and Technology (GZ)
Guangzhou, Guangdong, China

ABSTRACT

Generative AI, a novel general-purpose technology, has the capability to produce outputs closely resembling its training inputs. This paper presents the first use of such learning in memory workload synthesis. By devising three learning techniques and comparing them with two existing techniques, the paper shows that by carefully choosing the training input and processing the generated output, AI-based synthesis can produce generated workloads that have similar or better accuracy than the best of existing methods.

KEYWORDS

Computer systems organization → Memory system; **Neural networks**; **Statistical simulation**

1 INTRODUCTION

The recent release of ChatGPT and GPT-4 by OpenAI [19] has gained immense popularity since November 2022, amassing over 100 million users by January 2023, demonstrating the remarkable capabilities of GPT (*Generative Pre-trained Transformer*). The transformer architecture was introduced in 2017 [21].This architecture breaks away from previous neural networks in significant ways. Instead of convolution, it is based on a technique called multi-headed attention, which allows parallel encoding. Unlike the dense activation in convolution, the attention activation is sparse. Both parallel encoding and sparse activation provide greater speed, better scalability, and faster learning ability than convolutional or recurrent neural networks.

More importantly and surprisingly, it is capable of general-purpose learning and has been used for a myriad of purposes, including summarizing and graphing experimental results and writing a paper.

In this work, we study the use of generative AI in *memory workload synthesis*. Cycle-level simulators, such as Gem5 [5], are invaluable tools in computer architecture research. However, they suffer from prohibitively high computational costs for large-scale applications. A promising approach to reduce the simulation time is workload synthesis. By constructing shortened workloads, or microbenchmarks through statistical profiles, it captures key program behaviours including memory access patterns and data dependence of the profiled applications[1, 2, 12].

The nature of learning by generative AI is not well defined. One may say that it generates output *similar* to the input, but without a precise definition what exactly this similarity means. For workload synthesis, there are at least two types of similarity that are important. First, the latency of a memory access depends on nearby instructions. Second, the cache behaviour depends on data reuses that may span many instructions. The essence of this research hinges on two pivotal questions: (i) Can Transformer models effectively capture and reproduce the similar memory behaviors exhibited by program traces? (ii) How do these models compare against traditional methods in terms of synthesis quality and computational overhead?

The answers to these questions will help to understand how workload synthesis can benefit from generative AI to construct its representative microbenchmarks. In tandem with fast yet accurate simulators, we believe transformer-enabled synthesis promises to accelerate computer architecture research in the future.

2 BACKGROUND

2.1 Synthesis of Memory Workloads

Memory workload synthesis is a computational process that aims to generate synthetic traces that reproduces the memory behaviour of original workloads. Typically, statistical profiles are collected from the binary-instrumented workloads, without storing original traces, ensuring client confidentiality. These profiles later direct the workload generator, producing either synthetic memory traces or executables, to ensure a holistic capturing of memory behaviour. Various statistical models have been proposed [1, 3, 18], aiming at capturing the memory behaviour more accurately.

The pursuit of accurately capturing memory behaviour using locality features has led to the proposal of sophisticated methodologies and frameworks. *WEST* [3] builds its model around temporal locality by harnessing the LRU stack distance, or reuse distance[11] distribution on the cache hierarchy. Yet, the focus on temporal locality underscores its ability to model microarchitectural structures, such as prefetchers, that harness spatial locality. On the other hand, *STM* [1] emerges as a more promising solution. While it utilizes a similar framework to West, a pivotal distinction is introduced. STM's broadened approach integrates both temporal and spatial locality, facilitating a more in-depth understanding of memory behaviour and versatile explorations across memory hierarchies. Later, *HRD* [18] is proposed as an effective and efficient generalization of reuse distance to capture the spatial locality measured at multiple data-block granularities in addition to the temporal locality model of traditional reuse distance.

2.2 Transformer in Tabular Data Synthesis

Synthetic data, especially for image, video, and speech generation, is witnessing a surge in applications. Given the prevalence of tabular data in industries, there's a growing demand for tabular data synthesis to address challenges related to privacy and proprietary assets[6]. For instance, over 67% of datasets on the Google Dataset Search platform are categorized as structured or tabular data, often in formats such as CSV, XML, and JSON[4].

However, given the cost-intensive nature of gathering data and preprocessing, tabular datasets frequently exhibit several characteristics that hamper the utility of contemporary machine learning techniques as follows: 1.imbalanced label distribution, especially long-tailed patterns, making generalizations even harder[9];2.sensitive personal details, making data sharing untenable due to privacy and socio-ethical considerations[13]; 3.noisy data or missing values, which is usual in real-life scenarios, several data imputation methods have been proposed[16].

Traditionally, tabular data synthesis relies on techniques like *Variational Autoencoders(VAE)*[15] and *Generative Adversarial Networks(GAN)*[14]. These approaches often necessitate careful data preprocessing, such as encoding categorical columns or transforming numerical ones, impacting the generative model's performance. Conversely, transformer-based models like *GReaT*[7] utilize transformer-decoder networks and leverage auto-regressive generative large language models (LLMs) for the creation of highly authentic synthetic data. This approach offers more probabilistic control over sampling, eliminates the preprocessing bottleneck, and displays state-of-the-art performance over existing methods in various sizes.

3 TRANSFORMER BASED WORKLOAD SYNTHESIS

3.1 Why Using Tabular Transformer

From the perspective of the data itself, the memory behavior of a program is sequentially correlated, with data dependence between preceding and succeeding elements. The behavior of individual memory accesses is also defined by attributes such as read/write operations and addresses. Therefore, we need to include the bindings between different attributes of individual memory accesses in the input to allow the Transformer model to learn.

Unlike other forms of data such as time series or images, tabular data encapsulate a multitude of information in a structured manner, often representing multifaceted relationships. Each column in a table may represent distinct features or related attributes, and each row often represents a unique instance or record. As such, the synthesis of tabular data becomes critically essential, yet it's this very structure and complexity that makes the task particularly challenging.

A tabular generative model open-sourced by World Bank researchers is *Realistic Relational and Tabular Transformer (REaLTabFormer)* [20]. It focuses on generating single parent relational data and employs a GPT-2 encoder with a causal language model (LM) head to independently model the parent table. Once trained, this encoder remains static and serves to conditionally model the child tables. Notably, each child table necessitates a distinct conditional model, which is manifested as a sequence-to-sequence (Seq2Seq) transformer. A GPT-2 decoder equipped with a causal LM head is adeptly trained to produce observations from the child table, ensuring flexibility for variable lengths.

3.2 Trace-Driven Fine-Tuning

In transformer-based synthesis, the training process involves feeding a program trace to *REaLTabFormer* [20]. We call the process the *training period* and the cost the *training time*. Subsequent to the training, a new trace of the prescribed length is generated. We denote the process as the *generation period* and the cost the *generation time*. The cost of synthesis is the total cost of these two steps.

Central to our method is the concept of a program trace, which records a sequence of memory accesses. In this work, we use *Intel-Pin*[17] to profile the memory traces, where each access is a row containing the following five fields: running instruction count (`IC`), read or write (`Op`), physical address (`Addr`), data item size (`Size`), program counter (`PC`). We also calculate an extra row of reuse distance (`RD`) as a locality metric, based on the address of data items and a given cache-line size. As an example, Table 1 shows a trace with nine load and store instructions. It is a memory workload, where non-memory instructions, i.e. the missing instruction counts, are removed.

Table 1: An Example of Program Trace

Instruction Count	1	2	3	5	6	10	13	20	21
Read/Write	0	0	0	0	0	1	1	0	1
Physical Address	3672	2138	3712	3672	3888	168	3888	3712	3712
Data Size	8	1	8	8	8	8	8	4	4
Program Counter	1726	53	58	63	1734	3120	3122	3124	3126
Reuse Distance	∞	∞	∞	3	∞	∞	2	4	1

Given a program trace, we use three methods to synthesize a shorter trace. In these methods, a program trace is modified to have different information as discussed below. The generated output by the tabular transformer has the same format as the input trace it takes. We then convert the output trace to have the five fields as in the program trace.

In the basic method, the instruction count (`IC`) field is removed from the training input, which has only the remaining four fields. Removing the `IC` field reduces the training time by about 20% but has no significant effect on the generation time. Without `IC`, the column index is used by the transformer. The actual `IC` includes only memory-access instructions and is not continuous. In the generated trace, the column index is used as `IC`.

In the second method, a field is added to contain the reuse distance (`RD`) of data access. The reuse distance is for cache-line granularity and is measured by the number of cache lines. We still remove the `IC` field. The input has five fields for each row. During trace generation, we use the generated `RD` as follows. If `RD` is larger than 64, we use the generated physical address. If `RD` is less than or equal to 64, we generate the physical address that has this `RD`. The IC field is added by using the column index, as we do in the first method.

In the third method, we include the `IC` field but do not use any new field such as `RD` in the second method. The generated trace has the same format as the program trace.

4 EVALUATION

4.1 Experimental Setup

We use 12 different workloads in SPEC2017[8].[1] In order to capture the important phases of each workload, we fast-forward about 10 billion instructions before trace-collections and stop trace-collections once 10 million instructions of memory requests are recorded. All the traces are collected using a *Pin* based tool.

We use the three methods described in Section 3.2 to produce the synthesized trace using generative AI. In addition, we use two previous techniques, hierarchical reuse distance (HRD) and STM, as introduced in Section 2. In all five methods, we generate a trace of one million memory requests for each application, which is one-tenth of the profiled trace.

We set up the *Gem5*[5] simulator as a platform for validation. Given a trace of memory requests, we measure both the time and data movement. The timing is measured by the access latency for each request, and the data movement by the miss ratio at L1 and L2 caches. The L2 cache size is 256KB. For L1, we simulate eight configurations from 128 bytes, doubling at each step, to 16KB. We measure how accurate a synthesized trace is compared to the original trace in the timing and the miss ratios reported by Gem5.

4.2 Synthesis Costs

For the three REaLTabFormer based methods, we used *NVIDIA A100*[10] Tensor Core GPUs, founded on the Ampere Architecture, with 40GB high-bandwidth memory (HBM2) whose memory bandwidth surpasses 2TB/s. The GPUs are rented from *Google Colab*. A100 has 19.5 TFLOPS F32 and 156 TFLOPS Tensor Float 32. Each run of REaLTabFormer was on a single A100 GPU.

For HRD and STM, we employed an Intel 12th Generation (Alder Lake) multicore processor 12900, featuring 16 Cores (24 Threads) with base and turbo frequencies of 2.4 GHz and 5.1 GHz, respectively. The system incorporated a 30MB Cache and Dual-Channel DDR4-3600 MHz Memory. Both HRD and STM were executed in a single-threaded mode.

Table 2 compares the time taken by the five methods. The three AI based methods use over two hours, while HRD takes less than a minute, and STM slightly over 4 minutes. The table shows the breakdown of the cost. The most timing consuming step is training in AI based methods. Although Tab-RD requires post processing in the generation step, it still takes less time than IC.

4.3 Synthesis Quality

The quality is measured by the similarity between the results of the original and the synthesized workload. We evaluate two measures: the timing shown by the read latency and the data movement shown by the miss ratio at the two cache levels.

[1]The selected programs from SPEC2017 are 600.perlbench_s, 602.gcc_s, 603.bwaves_s, 605.mcf_s, 607.cactuBSSN_s, 619.lbm_s, 620.omnetpp_s, 623.xalancbmk_s, 631.deepsjeng_s, 638.imagick_s, 641.leela_s, 644.nab_s, all with the reference input.

4.3.1 Read Latency. We first evaluate the timing accuracy. Table 3 shows the mean memory read latency in μs[2]. For each configuration (L1 size) and each method, the table shows the geomean of the average read latency of the 12 workloads. The second column named *Reference* shows the ground truth, which is the read latency for the trace before synthesis. The remaining columns compare the latency result from five synthesis techniques: three based on AI and two previous analytical methods. In addition to the mean latency, each table cell shows the range, which is the geomean of the highest and the lowest latency values of 12 workloads.

The second column shows that for these workloads, the read latency starts high, drops precipitously around 2KB, and then becomes negligible. It is higher than 33 μs for cache sizes 128B to 1KB, 25 μs at 2KB, and stays at 2.1 μs at 4KB and greater sizes.

Among the five synthesis techniques, Tab-RD is the most accurate for cache sizes smaller than 2KB. The other four methods are less accurate but similar to each other except for Tab-IC, whose latency is consistently lower than the ground truth. For cache sizes larger than 2KB, HRD is the most accurate, followed by STM and Tab-IC. Tab-Base and Tab-RD are the least accurate, although the absolute difference is small (less than 3 μs). None of the methods is accurate for the 2KB threshold cache size. HRD is the closest, over 6 μs higher, and Tab-Base is the farthest, over 14 μs higher. For L1 cache sizes from 4KB to 16KB, The read latency across all methods becomes significantly small, with REaLTabFormer's Tab-Base and Tab-RD showing much higher latency compared to other methods.

Table 4 shows the comparison for 12 workloads. Averaged over 8 hardware configurations (varied L1 cache size), the table shows the geomean read latency of each workload and those from the five synthesis techniques. The ground truth given by the reference trace is shown in the second column. The remaining columns show first three AI-based techniques and then two conventional techniques.

The reference read latency varies between 24 and 38 μs across 12 workloads, with the geomean at 32.5μs. By the geomean, HRD is the most accurate at 33.8 μs, followed by Tab-RD at 36.3 μs and STM at 37.2 μs, while the result is too high by Tab-Base and too low by Tab-IC. Tab-Base, 39.0 μs, is close to Tab-RD and STM. Looking at each workload, HRD is the most accurate in all individual cases. Tab-RD is more accurate than STM in more than half of the workloads, 605, 607, 619, 623, 631, 638, 644.

The main difference from AI-based techniques and traditional techniques comes from the latency results for workloads 631, 638 and 641, which have the largest relative error in both Tab-Base and Tab-RD, while HRD maintains good accuracy in these cases. Overall, from the perspective of workloads, HRD is the most accurate in reproducing the reference read latency.

4.3.2 Miss Ratios. Tables 5 to 8 compare the L1 and L2 miss ratios respectively, first for different cache configurations and then for different workloads. The L1 miss ratio, as reported by Gem5, is the number of L1 misses divided by the number of L1 accesses, including both reads and writes. The L2 miss ratio is the number of

[2]When asked to describe Table 3, GPT-4 replied "The table represents latency measurements for two main categories: REaLTabFormer (with its three methods: Tab-Base, Tab-RD, and Tab-IC) and Traditional (with its two methods: HRD and STM). The measurements are taken across different L1 cache sizes."

Table 2: Comparison of Synthesis Time (h:m:s)

Methods		Training/Profiling	Generation	Total Cost
REaLTab	Base	1:39:43±0:06:46	0:20:27±0:00:90	2:00:10±0:06:51
	RD	2:13:42±0:04:90	0:29:60±0:01:47	2:43:42±0:05:12
	IC	2:37:18±0:09:94	0:34:20±0:01:03	3:11:38±0:09:99
Trad	HRD	0:00:04:49±0:00:00:36	0:00:50:14±0:00:05:67	0:00:54.63±0:00:05.73
	STM	0:02:50:12±0:00:00:85	0:01:13:20±0:00:00:82	0:04:03.31±0:00:01.09

Table 3: Read Latency (μs) of System Configurations

L1 Size	Reference	REaLTabFormer			Traditional	
		Tab-Base	Tab-RD	Tab-IC	HRD	STM
128B	37.4137±9.92497	38.7896±9.3345	38.9558±9.7325	29.6694±9.5026	34.4064±10.2590	41.8590±7.0406
256B	36.8903±10.2554	39.0276±9.4335	37.6864±10.3045	29.7062±9.4897	34.0346±10.3563	40.6587±9.5634
512B	36.0381±10.4466	39.4092±9.5608	36.6487±10.5516	29.2366±9.4188	33.6215±10.3833	39.3389±11.5229
1KB	33.4471±10.9816	39.8508±9.6357	35.9896±10.5616	25.9639±10.0382	33.2077±10.3090	37.6201±13.8018
2KB	25.2486±12.4658	39.3350±9.8288	33.3499±11.3098	17.3430±11.0772	31.5611±10.6096	33.6720±16.4813
4KB	2.1419±2.6475	4.8042±11.4474	4.8199±11.5312	3.6307±3.4630	2.4147±3.6816	2.4979±3.8404
8KB	2.1415±2.6474	4.7707±11.4095	4.8080±11.5185	3.6285±3.4569	2.1323±2.5162	2.4979±3.8404
16KB	2.1415±2.6474	4.7707±11.4095	4.8080±11.5185	3.6285±3.4569	2.1323±2.5162	2.4979±3.8404
Geomean	11.9387±6.36714	17.8324±10.2179	17.0661±10.8593	12.3880±6.66805	12.0785±6.39936	13.8086±7.49812

Table 4: Read Latency (μs) of Workloads

App	Reference	REaLTabFormer			Traditional	
		Tab-Base	Tab-RD	Tab-IC	HRD	STM
600.perlbench_s	32.719±11.470	41.003±7.601	38.191±9.735	23.930±10.785	35.368±10.871	35.518±14.207
602.gcc_s	30.471±10.880	38.397±9.425	36.676±10.478	35.379±10.779	32.058±10.552	34.639±15.205
603.bwaves_s	38.542±8.805	39.268±9.413	35.046±10.791	40.726±7.485	37.341±9.908	41.671±7.409
605.mcf_s	32.243±10.614	40.575±8.230	37.814±10.103	24.486±10.600	34.100±10.402	38.610±11.674
607.cactuBSSN_s	35.725±10.352	38.915±9.290	35.400±10.619	25.582±8.855	36.990±9.909	41.321±7.573
619.lbm_s	33.532±10.775	38.682±9.449	33.481±10.733	23.934±8.747	34.508±10.545	38.967±11.962
620.omnetpp_s	34.243±10.813	40.215±8.521	39.230±9.503	24.160±9.877	35.414±10.582	37.248±13.192
623.xalancbmk_s	33.886±10.549	39.056±9.334	37.747±10.300	24.912±10.065	31.507±10.374	38.673±12.401
631.deepsjeng_s	24.824±1.262	38.274±9.187	38.279±9.184	23.474±1.969	24.824±1.263	7.200±8.843
638.imagick_s	26.634±10.512	33.727±11.026	33.211±11.187	21.440±7.670	29.282±9.564	39.482±10.202
641.leela_s	30.311±12.583	40.422±8.669	38.685±9.946	22.193±11.637	33.937±10.927	38.221±12.697
644.nab_s	33.610±10.812	40.219±8.283	34.301±10.674	22.452±9.068	32.888±10.576	37.651±12.262
Geomean	32.466 ± 10.612	38.998 ±9.068	36.364 ±10.372	25.051 ± 9.953	33.838 ±10.338	37.173 ± 12.022

L2 misses divided by the number of L2 accesses. L2 miss ratios are trivial, either 0% or close to 100%. The accuracy is best measured by the L1 miss ratio.

Table 5 shows that miss ratios are positive for cache sizes from 128B to 2KB. The last row shows that when measured by the geomean, HRD and Tab-RD are more accurate in reproducing the reference trace result. This is consistent with the timing results in Table 3, where HRD and Tab-RD are the most accurate for cache sizes up to 2KB.

Table 6 compares L1 cache miss ratios from the perspective of different workloads. After taking the geomean of all hardware configurations, the reference L1 miss ratios vary from 0.198 to 0.063. In general, HRD and Tab-RD are the most accurate among all 12 workloads. While HRD performs much better in 631 and 619, Tab-RD excels in 605 and 638. Tab-Base has the largest relative error across all workloads. Notably, all AI-based techniques seem not to fit well into the simplest trace pattern from workload 631, which is the reason Tab-RD falls behind HRD slightly after taking the geomean. We will discuss workload 631 in Section 4.5.

4.4 The Reuse Distance (RD) Threshold

Tab-RD uses the threshold of RD=64 (Section 3.2). We have tested different thresholds above and below but found that this particular threshold gave the best overall result. Upon closer inspection, we see that the threshold includes most of the data reuses.

Table 9 shows the Cumulative Distribution Function (CDF) of the RD histograms of the original 10M traces for each workloads.

Table 5: L1 Cache Miss Ratio of System Configurations

L1 Size	Reference	REaLTabFormer			Traditional	
		Tab-Base	Tab-RD	Tab-IC	HRD	STM
128B	0.479	0.905	0.460	0.738	0.561	0.648
256B	0.387	0.825	0.355	0.665	0.502	0.575
512B	0.284	0.691	0.269	0.553	0.43	0.445
1KB	0.179	0.502	0.191	0.406	0.335	0.287
2KB	0.084	0.267	0.104	0.236	0.198	0.137
4KB	0.0	0.0	0.0	0.001	0.001	0.0
8KB	0.0	0.0	0.0	0.001	0.0	0.0
16KB	0.0	0.0	0.0	0.001	0.0	0.0
Geomean	0.227	0.566	0.261	0.490	0.402	0.391

Table 7: L2 Cache Miss Ratio of System Configurations

L1 Size	Reference	REaLTabFormer			Traditional	
		Tab-Base	Tab-RD	Tab-IC	HRD	STM
128B	0.0	0.0	0.001	0.0	0.0	0.0
256B	0.0	0.0	0.001	0.0	0.0	0.0
512B	0.0	0.0	0.001	0.0	0.0	0.0
1KB	0.0	0.001	0.001	0.0	0.0	0.0
2KB	0.0	0.001	0.002	0.001	0.0	0.0
4KB	0.998	0.891	0.949	0.174	0.033	1.0
8KB	1.0	1.0	1.0	0.19	0.999	1.0
16KB	1.0	1.0	1.0	0.19	1.0	1.0
Geomean	0.999	0.297	0.434	0.072	0.182	1.0

Table 6: L1 Cache Miss Ratio of Workloads

App	Reference	REaLTabFormer			Traditional	
		Tab-Base	Tab-RD	Tab-IC	HRD	STM
600	0.383	0.622	0.331	0.620	0.442	0.565
602	0.265	0.831	0.538	0.457	0.462	0.555
603	0.400	0.677	0.715	0.630	0.727	0.617
605	0.283	0.756	0.297	0.450	0.385	0.580
607	0.652	0.764	0.803	0.762	0.804	0.749
619	0.576	0.706	0.754	0.699	0.665	0.666
620	0.363	0.682	0.343	0.675	0.446	0.598
623	0.336	0.710	0.453	0.570	0.488	0.511
631	0.063	0.878	0.879	0.498	0.063	0.061
638	0.198	0.582	0.116	0.475	0.530	0.287
641	0.276	0.634	0.310	0.505	0.365	0.561
644	0.464	0.777	0.788	0.684	0.830	0.640
Geomean	0.351	0.706	0.487	0.577	0.464	0.533

Table 8: L2 Cache Miss Ratio of Workloads

App	Reference	REaLTabFormer			Traditional	
		Tab-Base	Tab-RD	Tab-IC	HRD	STM
600	0.128	0.213	0.209	0.215	0.052	0.116
602	0.139	0.158	0.169	0.000	0.054	0.108
603	0.137	0.170	0.165	0.000	0.036	0.113
605	0.131	0.167	0.170	0.194	0.065	0.113
607	0.112	0.161	0.156	0.159	0.032	0.101
619	0.104	0.165	0.160	0.173	0.040	0.100
620	0.123	0.174	0.181	0.181	0.055	0.111
623	0.127	0.171	0.167	0.180	0.042	0.112
631	0.142	0.144	0.099	0.151	0.061	0.152
638	0.196	0.184	0.181	0.215	0.045	0.155
641	0.140	0.188	0.187	0.206	0.041	0.115
644	0.125	0.171	0.162	0.181	0.036	0.110
Geomean	0.138	0.167	0.163	0.072	0.045	0.113

It shows that for most workloads, 95% of their RD are below this threshold. A high CDF value at RD=64 means that the RD beyond this threshold are relatively infrequent. By focusing on the RD values below this threshold, the fine-tuning by Tab-RD captures most of the data reuses for most of the workloads.

4.5 A Case of Poor AI Learning

The cumulative RD distributions in Table 9 show a special case which is 631.*deepsjeng_s*. In this workload, virtually all reuses have the reuse distance 0, which means that they are consecutive uses of the same cache block. Checking the 10M profiled trace, we see that the trace only captures a data initialization loop. The entire trace consists of only *memory writes* to consecutive word addresses, resulting in a long sequence of RD of 0[3], with a relative small number of cold misses[4].

While 631.*deepsjeng_s* has the trivial memory access pattern, AI synthesis performs unusually poorly. The geomean L1 miss ratio is 0.063 for the reference workload, but it is 0.5 by Tab-IC and over 0.8 by Tab-Base and Tab-RD. The simplest pattern is not well reproduced in AI-based synthesis, leading to large relative errors in terms of both latency and miss ratio results. If the results from app 631 are excluded, Tab-RD outperforms HRD.

[3]During the experiments, we measure the RD based on the physical address of each memory request. The calculation of RD is in terms of cache block size that presumes a cacheline size of 64 bytes. A RD of 0 means that the two consecutive addresses are in the same cache block.

[4]In a contiguous word-by-word traversal, there is a cache miss every 16 accesses.

4.6 Main Findings

From the results, we make the following observations:

(1) REaLTabFormer, a most recent AI technique based on large language models (LLM), when used out of the box, is not as good as (less accurate than) existing analytical techniques.
(2) By augmenting the training input with a domain specific measure, the AI technique produces results as accurate as or slightly better than the best of existing techniques. For example, Tab-IC offers much better read latency results and Tab-RD achieves nearly or slightly better miss ratio accuracy compared to HRD.
(3) Not only is the accuracy of AI learning highly sensitive to the fields supplied in the input, but also the cost of AI training. However, the longest learning time, 2.5 hours in Tab-IC, actually produces less accurate results than Tab-RD, whose training is half an hour faster.
(4) AI techniques can learn to reproduce similar instruction sequences and similar reuse patterns. The most accurate

Table 9: Cumulative Reuse Distance (RD) Distributions

Applications	RD=64	RD=32	RD=16	RD=8	RD=4	RD=2	RD=1	RD=0
600.perlbench_s	0.953	0.884	0.794	0.677	0.563	0.477	0.414	0.334
602.gcc_s	0.970	0.908	0.845	0.775	0.705	0.636	0.590	0.508
603.bwaves_s	0.873	0.790	0.718	0.634	0.480	0.178	0.096	0.048
605.mcf_s	0.983	0.954	0.904	0.705	0.556	0.447	0.384	0.282
607.cactuBSSN_s	0.916	0.770	0.577	0.375	0.213	0.098	0.060	0.030
619.lbm_s	0.950	0.874	0.773	0.597	0.384	0.176	0.086	0.041
620.omnetpp_s	0.961	0.889	0.824	0.730	0.614	0.532	0.482	0.353
623.xalancbmk_s	0.969	0.855	0.803	0.734	0.698	0.641	0.615	0.459
631.deepsjeng_s	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
638.imagick_s	0.886	0.809	0.777	0.643	0.627	0.519	0.506	0.032
641.leela_s	0.958	0.899	0.797	0.738	0.660	0.556	0.505	0.354
644.nab_s	0.942	0.872	0.773	0.605	0.474	0.129	0.050	0.023

learning by Tab-RD depends on the selection of the fields in the input and on the post-processing of the output.

(5) Tab-RD shows that RD is better than AI for synthesizing short distance reuses, while AI is better at long distance reuses.

(6) Learning using one particular information, the instruction count, requires the highest training and generation time but produces the lowest accuracy among all methods.

5 SUMMARY

In this paper, we have developed a new technique for memory workload synthesis based on generative AI that performs slightly better than the best of existing techniques for smaller cache sizes and similar for other cache sizes. It uses the reuse distance as the additional information in training. The output is modified based on the generated reuse distance. The combination takes less time than the method using unmodified input and output, yet the synthesis accuracy is significantly higher.

As with any early-stage research, there are opportunities for improvement and further investigation. Here are some highlights that need to be improved in future work. One observation through our simulation results on Gem5 is that these workloads seem to fit neatly into 4KB of L1 cache, which is rather small. This can be solved by profiling diverse traces of these workloads that include memory-intensive regions. Also, we collected traces purely by Intel-PIN. A natural progression will be to integrate mechanisms for trace gathering from a weakly ordered memory infrastructure like ARM, RISC- V, and MIPS. Also, the model can be extended to include atomics, transactional requests and flushes. Encapsulating these requests types could further improve the accuracy of the models with respect to actual memory subsystems.

In conclusion, while our current research offers foundational insights, we recognize the potential avenues for improvement and re-evaluation. We are optimistic that addressing these will not only fortify our methodology but also offer valuable insights.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of MEMSYS for their constructive feedback and Nathan Ganger, Dylan McKellips, and Yifan Zhu for their help with the presentation of the material.

This work was supported by the Guangzhou-HKUST(GZ) Joint Funding Program (No. 2023A03J0013). This work was also supported by the National Science Foundation (Contract No. CCF-2217395, CCF-2114319). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding organizations.

REFERENCES

[1] Amro Awad and Yan Solihin. 2014. STM: Cloning the spatial and temporal memory access behavior. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. https://doi.org/10.1109/hpca.2014.6835935

[2] Mario Badr, Carlo Delconte, Isak Edo, Radhika Jagtap, Matteo Andreozzi, and Natalie Enright Jerger. 2020. Mocktails: Capturing the Memory Behaviour of Proprietary Mobile Architectures. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 460–472. https://doi.org/10.1109/ISCA45697.2020.00046

[3] Ganesh Balakrishnan and Yan Solihin. 2012. WEST: Cloning data cache behavior using Stochastic Traces. In *IEEE International Symposium on High-Performance Comp Architecture*. 1–12. https://doi.org/10.1109/HPCA.2012.6169042

[4] Omar Benjelloun, Shiyu Chen, and Natasha Noy. 2020. Google Dataset Search by the Numbers. arXiv:2006.06894 [cs.IR]

[5] 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

[6] Vadim Borisov, Tobias Leemann, Kathrin Sessler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. 2022. Deep Neural Networks and Tabular Data: A Survey. *IEEE Transactions on Neural Networks and Learning Systems* (Jan 2022), 1–21. https://doi.org/10.1109/tnnls.2022.3229161

[7] Vadim Borisov, Kathrin Seßler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. 2023. Language Models are Realistic Tabular Data Generators. arXiv:2210.06280 [cs.LG]

[8] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering* (Berlin, Germany) *(ICPE '18)*. 41–42.

[9] Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. 2019. Learning Imbalanced Datasets with Label-Distribution-Aware Margin Loss. arXiv:1906.07413 [cs.LG]

[10] Jack Choquette and Wish Gandhi. 2020. NVIDIA A100 GPU: Performance & Innovation for GPU Computing. In *2020 IEEE Hot Chips 32 Symposium (HCS)*. 1–43. https://doi.org/10.1109/HCS49909.2020.9220622

[11] C. Ding and Y. Zhong. 2001. *Reuse Distance Analysis*. Technical Report UR-CS-TR-741. University of Rochester.

[12] L. Eeckhout, J. Sampson, and B. Calder. 2006. Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.* https://doi.org/10.1109/iiswc.2005.1525996

[13] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. 2016. Privacy-Preserving Distributed Linear Regression on High-Dimensional Data. Cryptology ePrint Archive, Paper 2016/892. https://doi.org/10.1515/popets-2017-0053 https://eprint.iacr.org/2016/892.

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:1406.2661 [stat.ML]

[15] Diederik P Kingma and Max Welling. 2022. Auto-Encoding Variational Bayes. arXiv:1312.6114 [stat.ML]

[16] Wei-Chao Lin and Chih-Fong Tsai. 2020. Missing Value Imputation: A Review and Analysis of the Literature (2006–2017). *Artif. Intell. Rev.* 53, 2 (feb 2020), 1487–1509. https://doi.org/10.1007/s10462-019-09709-4

[17] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: building customized program analysis tools with dynamic instrumentation. *ACM SIGPLAN Notices* (Jun 2005), 190–200. https://doi.org/10.1145/1064978.1065034

[18] Rafael K. V. Maeda, Qiong Cai, Jiang Xu, Zhe Wang, and Zhongyuan Tian. 2017. Fast and Accurate Exploration of Multi-level Caches Using Hierarchical Reuse Distance. In *Proceedings of the International Symposium on High-Performance Computer Architecture*. 145–156. https://doi.org/10.1109/HPCA.2017.11

[19] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[20] Aivin V. Solatorio and Olivier Dupriez. 2023. REaLTabFormer: Generating Realistic Relational and Tabular Data using Transformers. arXiv:2302.02041 [cs.LG]

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017).