PredATW: Predicting the Asynchronous Time Warp Latency For VR Systems

AKANKSHA DIXIT, Electrical Engineering, IIT Delhi, India
SMRUTI R. SARANGI, Electrical Engineering, IIT Delhi, India

With the advent of low-power ultra-fast hardware and GPUs, virtual reality (VR) has gained a lot of prominence in the last few years and is being used in various areas such as education, entertainment, scientific visualization, and computer-aided design. VR-based applications are highly interactive, and one of the most important performance metrics for these applications is the motion-to-photon-delay (MPD). MPD is the delay from the user's head movement to the time at which the image gets updated on the VR screen. Since the human visual system can even detect an error of a few pixels (very spatially sensitive), the MPD should be as small as possible.

Popular VR vendors use the GPU-accelerated Asynchronous Time Warp (ATW) algorithm to reduce the MPD. ATW reduces the MPD if and only if the warping operation finishes just before the display refreshes. However, due to the competition between the different constituent applications for the single, shared GPU, the GPU-accelerated ATW algorithm suffers from an unpredictable ATW latency, making it challenging to find the ideal time instance for starting the time warp and ensuring that it completes with the least amount of lag relative to the screen refresh. Hence, the state-of-the-art is to use a separate hardware unit for the time warping operation. Our approach, *PredATW*, uses an ML-based hardware predictor to predict the ATW latency for a VR application, and then schedule it as late as possible while running the time warping operation on the GPU itself. This is the first work to do so. Our predictor achieves an error of only 0.22 ms across several popular VR applications for predicting the ATW latency. As compared to the baseline architecture, we reduce deadline misses by 80.6%.

ACM Reference Format:
Akanksha Dixit and Smruti R. Sarangi. 2024. PredATW: Predicting the Asynchronous Time Warp Latency For VR Systems. 1, 1 (July 2024), 39 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The[1] virtual reality (VR) industry is growing very rapidly particularly with the advent of the metaverse. The compound annual growth rate (CAGR) of the VR market space was 31% in 2023 and the total market size is expected to reach 165.91 billion dollars by 2025 [28]. Head mounted displays (HMDs) such as Meta's Oculus Quest 2 and Sony's PS VR2 are dominating this segment as of today [54, 62]. The key challenge is to compute a new image based on the head's instantaneous position, and make it appear that the movement is smooth. This technology is maturing. At the moment, rendering VR environments with

[1]This work received partial support from the Semiconductor Research Corporation (SRC).

Authors' addresses: Akanksha Dixit, Electrical Engineering, IIT Delhi, New Delhi, India, Akanksha.Dixit@ee.iitd.ac.in; Smruti R. Sarangi, Electrical Engineering, IIT Delhi, New Delhi, India, srsarangi@cse.iitd.ac.in.

XXXX-XXXX/2024/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

complex scenes is proving to be quite challenging. We need to ensure that the motion is smooth and there is no blurring. Hence, 90 Hz displays are typically used.

This means that the rendering task for every new frame needs to be less than 11 ms [32]. In the community, it is a known problem that the resources of VR devices are insufficient when it comes to rendering all the frames [14, 25]. There is a subtle point to be noted here. VR devices have two modes of operation: *standalone* mode and *tethered* mode. In the former mode, the VR device is a purely mobile device, whereas in the latter mode, the VR device is connected to a machine such as a desktop that handles all its rendering tasks. In both the modes, rendering at line-speed (i.e., one frame in < 11 ms) is considered to be very difficult for realistic scenes [5, 9]. Note that scenes are getting more and more complex with the need for better graphics and ubiquity of these devices.

Hence, in the vision literature, a classical technique known as asynchronous time warping (ATW) [27, 69] is used. The idea at its core is quite simple – if the rendering task does not finish on time, then display an approximation, which is computed using historical head motion vectors and the last rendered scene. As of today, ATW is an integral part of the VR rendering pipeline. It may not be extremely accurate all the time; however, it does provide a sense of continuity. Unless the user sees continuous motion that has minimal blurring, there is a threat of getting a feeling of visual disorientation [13, 53, 70], which can be quite disconcerting. The authors have themselves experienced it on their Oculus Quest 2 device, and that too for fairly simple scenes. Thus, the quality of the ATW algorithm's output is very important.

As of today, there is one dominant ATW algorithm based on the image reprojection technique [19, 69]. Most of the proposals including commercial designs either run ATW on a separate chip [72] or running it as an application on the same GPU that does the rendering (like Oculus Quest 2) [9]. We have an issue with both ends of the spectrum. Dedicated hardware for ATW has a significant cost in terms of silicon area (20-30 mm^2 in[73]). We basically need a small GPU to do this operation. On the other hand, running ATW along with the GPU is fraught with problems, which have been highlighted by a lot of researchers [73]. There is destructive interference due to resource contention between the two tasks: ATW and the GPU application. This degrades the quality, which we shall measure using the metric *motion-to-photon delay* or *MPD*. It is the duration between the start of ATW and the point at which the frame is displayed (next display refresh point). A high MPD means that our ATW algorithm uses stale data and is possibly slow. We thus **need to minimize the MPD**. This is a standard metric [56, 63, 76].

Xie et al. [73] show that applications can have a long MPD even with a fast implementation if the time warp is not initiated at the right time. They demonstrate that the MPD is high in two situations: the time warp is completed well before the display refreshes or if it misses the refresh deadline. Figure 1 explains this with three possible cases. In case A, after the frame F1 is rendered, A1 warps this frame based on the head pose at t1 and displays this warped frame at the refresh deadline RD1 (MPD1=RD1-t1). If A1 had been invoked closer to RD1, then it would have taken the latest head pose more than now, and MPD1 may have been lower. Unlike the previous case, for cases B and C, A2 and A3 preempt the rendering of frames F2 and F3, respectively, which get resumed when A2 and A3 are completed. Since the preemption latency is a highly variable quantity, two situations may arise: either the refresh deadline is missed due to large latency (Case C) or it is not missed (Case B). In case B, A2 warps frame F1 (F2 is not yet rendered completely) according to the head pose at t2 and displays this warped frame at refresh deadline RD2 (MPD2 = RD2 - t2). Since in case C, A3 misses the refresh deadline RD3, we see the same frame displayed at RD2,

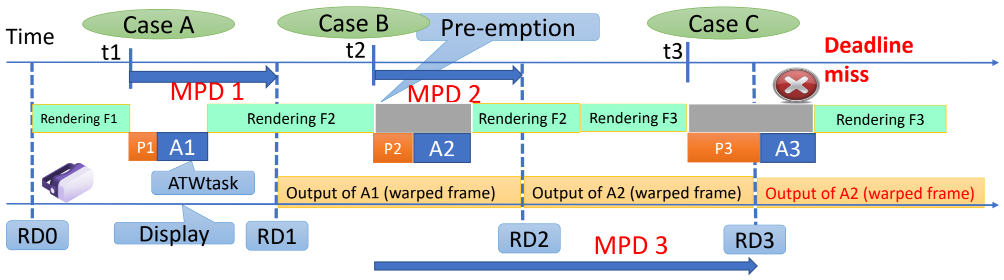


Fig. 1. Possible scenarios in the GPU-accelerated ATW algorithm. $F1$, $F2$, and $F3$ are frames. $A1$, $A2$, and $A3$ are ATW tasks. $P1$, $P2$, and $P3$ are preemption latencies. RD stands for the refresh deadline.

leading to a much larger MPD (MPD3= RD3 − t2). Hence, the main problem is finding the correct time instance to start the time warp operation. None of those mentioned above are ideal cases; they show the importance of precisely initiating the ATW operation, which is not straightforward because the ATW latency is variable.

Let us now look at some of the approaches that cannot be used. We will naturally be tempted to use the resources of the GPU more optimally such as partitioning SMs or using real-time scheduling algorithms to avoid the resource contention. Sadly, such software solutions have not proven to be effective because very little is known about the internals of the NVIDIA GPUs, which we extensively studied in the course of writing this paper. NVIDIA only supports the creation of virtual GPUs using its MIG technology [42]. However, Joshua et al. have recently published a paper [6] where they achieve a degree of partitioning using a hitherto unknown set of CUDA flags. This is not commonplace as of today and is not supported by popular graphics APIs like DirectX and Vulkan, which our workloads use. Like core partitioning, very little is known about the internal scheduling algorithms, and thus till date there are no proposals on making GPUs behave as competent real-time devices [4, 75]. Perhaps in the foreseeable future, we may see GPUs with the same kind of partitioning and real-time priority setting features as CPUs; however, that is clearly not the case today.

Given that the software option is not available to us, we propose a *very different idea*. We **predict** the latency of the ATW algorithm – mostly the time we lose due to contention – and launch the ATW algorithm at exactly the right moment, which is just before the refresh deadline. To the best of our knowledge, this has not been done before. To build an efficient predictor, we conduct an elaborate set of characterization experiments, pinpoint the features that determine the latency and build a decision tree predictor, which is both intelligent as well as explainable. We show that the proposed algorithm reduces the MPD significantly without degrading other performance metrics like frame quality, rendering speed, or resource utilization.

We shall revisit the question of what to run in software and what to run in hardware several times in the paper and conduct a few thought experiments, where we shall assume the existence of core partitioning and real-time features in GPUs. In spite of a thorough AI-based state space exploration, we shall show that the best software implementation is significantly inferior to a solution that runs ATW in software and the predictor in hardware (our chosen configuration). Even though our predictor itself runs very quickly, collecting and processing the features requires bespoke parallel hardware.

Our primary contributions are:

❶ We show that it is possible to predict the ATW latency on a GPU using simple metrics

that are collected per frame and the values of performance counters. We perform a thorough characterization to justify these metrics.
➋ We compare the performance of various ML-based prediction models and show that a simple decision tree along with a feedback component (a PID controller) provides accurate results as compared to other ML-based models with the added benefit of explainability.
➌ We were able to decrease the ATW refresh deadline miss rate by 80.6% as compared to a GPU-accelerated ATW architecture. Our prediction error is a modest 0.22 ms for a refresh interval of 11.1 ms.
➍ We design an AI-based Genetic Algorithm (GA) that explores the entire solution space for minimizing the MPD and shows that *PredATW* performs the best.

The paper is organized as follows. Section 2 provides the background of the VR architecture and ML-based models. Section 3 discusses the motivation for the work. The implementation details are given in Section 4. Section 5 shows the experimental results. We discuss related work in Section 6 and finally conclude in Section 7.

2 BACKGROUND

2.1 The VR-Architecture Loop

A brief overview of a typical VR-architecture loop is shown in Figure 2. In a VR system, the HMD tracker first gathers the user's position or motion data and then transmits that information to the application. The application copies the entire data to the GPU memory and sets up the rendering process. Finally, the rendering system generates two frames for the two eyes and displays those frames on the head-mounted displays (HMDs). In this process, a delay exists from the user's head's movement to the time at which the image gets updated on the VR screen; it is known as the motion-to-photon delay (MPD). The MPD should be as small as possible to provide a good user experience. Time warping is an image reprojection technique that is used to reduce this delay.

As discussed, time warping maps the previously rendered frame to the correct position based on the latest head-orientation information. Figure 2 depicts a high-level overview of this architecture. Note that ATW only applies to head rotational tracking, which means that the previously displayed frame is modified solely keeping in mind rotational changes of the head's position.

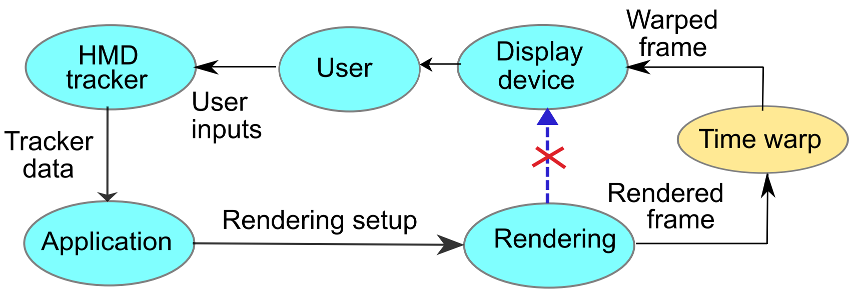


Fig. 2. Overview of the VR-architecture Loop

2.2 Inter-Frame Similarity

Zhao et al. show [77] that there exists reasonably high Inter-Frame similarity in VR applications. However, there are cases where genuinely similar frames may have significant MPD variations (as explained in Appendix A.1). We will utilize both aspects of VR applications to predict the ATW latency. Our study utilizes two methods to establish the existence of inter-frame similarity in a VR application. The first method is to just compare the contents (after pre-processing). Whereas, the second method is to compare a few frame-specific features (image features and runtime performance counter based). Let us discuss the first method here because that relies on prior work whereas the second method will be described in Section 3; it requires some bespoke mechanisms that we develop. The detailed discussion is given in Section 3.5.

2.2.1 Method 1: Macroblock Similarity. To calculate the similarity of frame contents, we use the DCT (Discrete Cosine Transform) based approach mentioned in [7, 20]. In this approach, the computation of frame similarity is solely based on the DC (zero frequency) coefficients of the DCT since they capture important features of a frame. We analyze the similarity at the level of macroblocks to efficiently capture frame differences. A *macroblock* is a 2×2 matrix that has traditionally been used in linear block transform-based image and video compression [71]. Let the two frames be A and B. We construct n macroblock matrices given as $M_1 ... M_n$ for each frame. Two frames are similar if at least a minimum number of macroblocks are similar.

2.3 ML-based Prediction Models

In machine learning, there are two types of prediction problems: classification and regression. In classification, we look for a model that can assist us with assigning a class to a datum. Regression, on the other hand, is a method for assigning continuous real values to data rather than classes or labels. Since we are predicting the ATW latency, our problem falls under the regression category. A regression model typically uses the Mean Square Error (MSE) as the loss function.

2.3.1 Linear Regression. Linear Regression is one of the simplest models used for regression. It tries to find a linear relationship between a dependent variable Y and a set of independent variables $X_1 \dots X_n$. In other words, it fits a straight line or a surface such that the errors between the actual values and the predicted values are minimized.

2.3.2 Decision Tree Regression. The decision tree is used for both classification and regression tasks. One of the most essential aspects of this model is its explainability and the fact that its performance does not get affected by the non-linearity of the data. The decision tree builds models in the form of a tree. It recursively splits the entire dataset into smaller subgroups in order to decrease the error. There are two types of nodes in the tree: decision (internal) nodes and leaf nodes. The root node is a decision node at the top of the tree and covers all the features. Each decision node has a condition that determines which path to take based o the values of its features. Whereas the leaf node contains the final predicted outcome.

To build the model, we start with the root node. For deciding on a feature at the root node, we iterate through all the features and find the optimal split of the dataset. The splitting of the dataset happens until the error is minimized or a certain threshold is reached such as a limit on the depth of the tree or the number of leaf nodes. The metric which is used to decide the splitting of the dataset is the sum of the squared error. This process is continued recursively. To predict the value for a given data point, we traverse through the entire tree following the conditions present at the decision nodes and reach a leaf node, which gives the final output.

2.3.3 Random Forest Regression. Random Forest, as the name implies, is a collection of decision trees. It combines the results of these decision trees to get the final result. This merging of the outcomes from various models or weak learners to produce the final result is known as *ensemble learning.* First, we randomly divide the dataset into k parts. Then, decision trees are built independently for all the parts. During the prediction process, outputs of all the individual decision trees are averaged out to yield the final outcome. The disadvantage of this model is that it needs rigorous training, hence is slower and more complex than a regular decision tree, even though may produce better results.

2.3.4 Gradient Boosting Regression. A gradient boosting regressor is also a variant of ensemble methods. It builds multiple decision trees and then combines their outcomes to produce final results. Unlike the random forest regressor, it creates regression trees from residuals—the difference between the actual and expected output. These residuals and features are used to train regression trees and the residual predicted by these models are incorporated into the input model. This process is repeated several times and the input model is pushed towards the correct prediction.

2.3.5 Convolutional Neural Network (CNN). A CNN is a type of neural network that is used for prediction purposes. We do not use CNNs in our case because of the following reasons.

- The structure of a CNN is very complex, each of its layers uses multiple multipliers and adders. Hence, it becomes slow and also requires a lot of storage. Since we want to predict the ATW latency so that the MPD is reduced, it is not suitable to have a predictor with a large delay.
- The second reason is that CNNs are not explainable. We cannot tell how each feature is contributing to the final prediction. As we want to analyze the application's and the system's behavior, we need an explainable model.

3 MOTIVATION

In this section, we shall first show the list of benchmarks that we use for experiments and then discuss the motivation for the methodology. Then, we evaluate the latency of GPU-accelerated ATW to identify the factors other than the time warping operation itself that are contributing to the latency overhead resulting in a larger MPD. After that, we show that in a VR application, there exists a substantial similarity between consecutive frames, which our predictor needs to exploit. Finally, we note that since the resolution of the foveal part of the human eye is very high, there is a very strong need to reduce the MPD as much as possible – this motivates our approach.

3.1 Overview of the Benchmarks

Table 1 shows the benchmarks used in this study. Orange Room is a benchmark test included in the VRMark benchmark suite [61]. The rest of the benchmarks are well-known VR applications and taken from NVIDIA VRwork [46] and Steam [64] similar to prior work [72, 73]. For a more accurate evaluation of the impact of the workload size on our solution, we render the applications at various resolutions (mentioned in Table 1).

3.2 Methodology

In this paper, we propose a method to predict the ATW latency before hand to reduce the MPD. The proposed predictor uses a few features (as shown later in Table 4). The features can be categorized into the following groups: frame-specific features (FS), parameters of the

Table 1. VR benchmarks

Abbr.	Name	Platform	API	Resolution
SPZ	Sponza	NVIDIA VRWorks [46]	Vulkan	1080p
IM	InMind VR [34]	Steam	DX11	1080p
IM2	InMind VR 2			720p
SH	Shooter	Custom	DX11	1080p
SH2	Shooter 2			720p
Sanc	Sanctuary VR [21]	Steam	DX11	720p
AS	Altspace VR [3]	Steam	DX11	1080p
OR	VRMark [61] Orange Room	Steam	DX11	1080p
SM	San Miguel	NVIDIA VRWorks [46]	DX11	1080p
DD	Dreamdeck [49]	Oculus store [50]	DX11	1080p
FH	Fun House [65]	SteamVR	DX11	1080p
LB	The Lab [68]	SteamVR	DX11	1080p

functions responsible for rendering frames (FP), and hardware performance counters. We use Meta's Oculus Quest 2 as our baseline VR device because it is the most popular VR headset currently due to its affordable price and ease of use; it accounts for almost 80% of the market share [17]. Oculus Quest 2 works in two modes: *Standalone* and *Tethered.* In the standalone mode, the headset does not need a PC or smartphone and rendering is performed on the headset itself. In the tethered mode, on the other hand, the headset only acts as a screen for a PC or smartphone to give a VR experience. To decide which mode we should use for this paper, we need to answer the following research questions.

RQ1: Can we use the standalone mode for this work?

To predict the ATW latency, the predictor needs the values for many features. Unfortunately, there is no profiler or software that can provide these values for the standalone mode of Oculus Quest 2. Hence, we cannot use the standalone mode, leaving just the tethered mode as an option. For the tethered mode, there is better profiling support available. The default choice for the profiler is the Oculus Debug Tool [36], which is a part of the official SDK (software development kit). Yet it only exposes highly abstracted performance parameters such as the frame rate and the total ATW latency, hence it is not suitable for this work and we thus needed to look for other profilers.

For the tethered mode, NVIDIA provides developer tools that enable the user to profile VR applications running on an NVIDIA GPU. These tools also provide the instruction traces for a running VR application. Hence, these tools can be used to collect the values for all our features. Details of this profiling tool, NVIDIA Nsight Graphics [45], are given in Section 3.3. Thus, our default configuration for experiments is the tethered mode of the Oculus device with a profiler from NVIDIA's tool suite. But before we move forward, we have two more questions to answer.

RQ2: Is the profiler enough or do we need the simulator to collect the features?

Since the implementation details of the time warp algorithm used by Oculus Quest 2 are not available publicly, we need to simulate it. We adopt the time warp algorithm employed in the popular toolkit ILLIXR [27] and implement it on our own. First, we capture the instruction traces of the VR application running our time warp algorithm by the profiler and then run those traces on the simulator to collect the values of features. One more thing to notice is that as discussed earlier, there are three categories of features. We can collect the features of the first two categories (frame-specific and functions called during rendering) directly with the help of the profiler. However, the third category which consists of the GPU's performance counters are quite challenging to collect [38], and thus a simulator is more useful.

RQ3: Can the proposed algorithm be implemented entirely in software?

A VR application or any other user program uses APIs to communicate with the GPU. APIs such as DirectX, OpenGL, and Vulkan translate high-level code to shader commands and directly communicate with the GPU's device driver. To run the predictor in software, we need to extract GPU-related performance counters and other features within the application itself. However, the latency of this feature collection process in software is very high (refer to Section 5.8).

Therefore, we propose using a hardware predictor. Here one may ask why we cannot offload the ATW algorithm to specialized hardware where we won't require a predictor. But in this case, an external component incurs a significant HW overhead [59]. Our proposed predictor is a simple decision tree with a few parameters; it has an insignificant hardware footprint. Another alternative to the HW predictor is to use spatial multiplexing, in which we dedicate a small number of GPU cores to ATW tasks so that there is no preemption and the ATW latency is fixed. But this approach results in inefficient utilization of GPU cores and does not provide satisfactory results (refer to Section 5.8).

3.3 Experimental Setup

As discussed in Section 3.2, we use one of the profilers from the NVIDIA tool suite. Since we need to debug the application at the frame-level, we use NVIDIA *Nsight™ Graphics* (NG) [45, 47], a profiler that helps us analyze every single frame in a stream of frames. This tool works seamlessly with various graphics APIs such as DirectX [37], Vulkan [23], and OpenGL [22]. In a graphics application to render a frame, the application issues several *draw calls* that are translated to PTX/SASS code. NG profiles each frame of an application at the granularity of *draw calls*. Using this profiler, we collect the following information: contents of the rendered frame, number of draw calls required for rendering the frame, the shader/kernel executed by the GPU (instruction traces), and the number of instances of each *draw call*, i.e., #software threads. We also capture the final rendered frame to calculate its perceived brightness [10].

The collected instruction traces are run on a simulator. We use the latest version of *GPUTejas* [35], a Java-based cycle-accurate GPGPU simulator, which simulates the architecture of GPU NVIDIA RTX™A4000. The configuration details of the simulator are given in Table 2. Here one can argue that if we are using the simulator-based approach, we can simulate the architecture of the GPU present in Oculus Quest 2 and run the instruction traces on that to imitate the standalone mode of the device. We simulate this architecture too on our simulator for the sake of completeness. For obvious reasons, we get much better results when we simulate the standalone mode as opposed to the tethered mode because the GPU on the HMD is far weaker (refer to Figure 4). However, the challenge is to show much better operation in the tethered mode because a desktop has a powerful multicore CPU and GPU; hence, our primary results are for the tethered mode because we wish to show a benefit for the case that is more disadvantageous to us, i.e., the tethered mode.

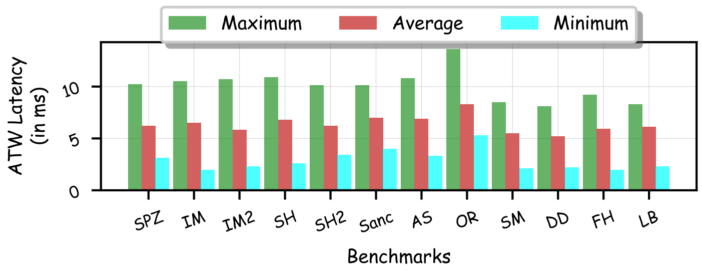
Table 2. Platform Configuration

Parameter	**Type/Value**	
Desktop Configuration		
CPU	Intel®Xeon®Gold 6226R @ 2.90GHz	
GPU	NVIDIA RTX™A4000	
GPU memory	16 GB	
VR Device	Oculus Quest 2	
Simulator Configuration		
	Tethered Mode	**Standalone Mode**
GPU Architecture	Ampere [41]	Kepler [40]
Clock Frequency	1 GHz	1 GHz
#TPCs, #SMs, #CUDA Cores	24, 48, 6144	8, 16, 128
Warp Size, #Warp schedulers per SM	32, 4	32, 2
Register file size per SM	256 KB	64 KB
I-cache size per SM	8 KB	4 KB
L1 data cache per SM	128 KB	64 KB
L2 cache size	4 MB	1.5 MB
Main memory	16 GB	2 GB

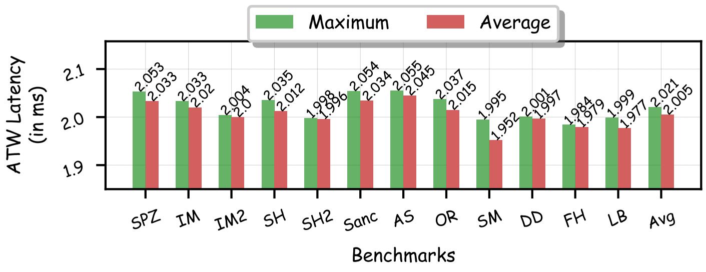
3.4 Latency of GPU-Accelerated Asynchronous Time Warp

In this section, we measure the latency of the GPU-accelerated ATW algorithm (ATW). As mentioned earlier, we adopt the time warping algorithm from *ILLIXR* [27], a well-known open-source XR system and testbed that implements the entire pipeline of a VR/AR application. The inputs to the time warp algorithm are the frame that needs to be warped and the HMD (head mounted display) matrices at the render and display times. These HMD matrices are 4×4 homogeneous matrices that contain the head's position at a given time. Frames and head poses are collected from the NG profiler and the VR headset, respectively. In this case, we do not take consecutive frames rendered by the application rather we select those frames that are graphically more complex so that we have more variation in the inputs because consecutive frames are generally similar in nature as discussed in Section 2. We shall delve into the inter-frame similarity in Section 3.5. To evaluate the latency, we simulate two types of real-life scenarios in GPUTejas: ❶ All the cores are utilized by a single VR application which uses GPU-accelerated ATW. ❷ The VR application shares the graphics resources with another application. We do not consider the case where more than two applications run on the same GPU because it is generally not the case. As discussed, the ATW kernel must preempt the already running rendering task. NVIDIA GPUs with the latest architecture, such as Ampere, support a fine-grained preemption model for graphics tasks that work at pixel-level [43]. We use the same model in our simulator by preempting at the thread-level because the pixel is merely another thread.

The ATW latency has three components: time warp latency, preemption latency, and context-switch time. To show that the preemption of the rendering process is the main reason for the increased and uncertain ATW latency in GPU-accelerated ATW, we first plot the ATW latency (refer to Figure 3a) and then we measure only the time warp latency ignoring the preemption latency and context-switch time (refer to Figure 3b). The large variation in the ATW latency and almost constant time warp latency for a particular benchmark and across the benchmarks show that the ATW latency depends upon the frame whose rendering is getting preempted and the graphics resources' availability.



(a)



(b)

Fig. 3. (a) Variation in the ATW latency; (b) Latency of the time warping operation.

We make the following observations from the figure:
❶ The maximum ATW latency is almost 15 ms which is much larger than the latency of the time warping kernel (≈ 2.02 ms).
❷ There is a lot of difference among the minimum, maximum and average values of the ATW latencies for a benchmark, which means that the ATW latency varies a lot across frames and it is highly unpredictable and uncertain. The results shown in Figures 3a and 3b confirm that the preemption of the rendering process results in a large ATW latency value (2 ms to 15 ms).
❸ For a particular benchmark, the maximum and average values of the time warping kernel are roughly the same, which means that it takes almost the same amount of time irrespective of the frame complexity and head poses. The maximum difference between the average and maximum value of the latency is 0.04 ms for all the benchmarks.

As mentioned in Section 3.3, we also simulate the architecture of the GPU present in Quest 2. Figure 4 shows the ATW latency of all the benchmarks for this case. We make the following observations from the figure:
❶ The ATW latency ranges from 3 ms to almost 22 ms across benchmarks; this was 2 ms to 15 ms for the previous case (refer to Figure 3a). This is because Adreno GPU is slower than NVIDIA RTX™A4000 in terms of the processing capabilities, memory size, and memory bandwidth.

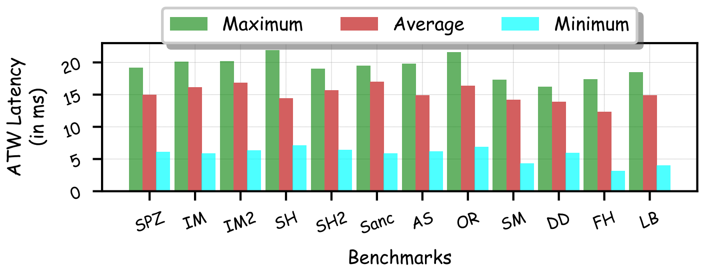


Fig. 4. Variation in the ATW latency (Standalone mode)

3.5 Inter-Frame Similarity

To show the similarity between frames, we extracted 100 consecutive frames from every application.

3.5.1 Frame-Level Similarity: As mentioned in Section 2.2.1, we use the macroblock similarity metric to find the similarity at the frame-level. Figure 5 shows the macroblock similarity percentage between two consecutive frames. It is evident from the figure that for all the benchmarks except for one benchmark (OR), all the frames are at least 70% similar to their neighboring frames. Even for OR, 90% of the frames have at least 70% inter-frame similarity.

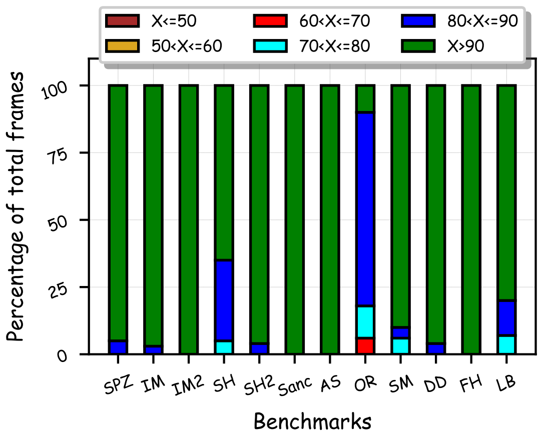


Fig. 5. Macroblock similarity percentage between two consecutive frames (X)

3.5.2 Feature-Level Similarity . To check the similarity, we collect a few frame-specific features/attributes (Table 3) from various sources and compare the corresponding values. For brightness, we take the rendered frame from NG and calculate the value using Equation 1 [10]; it is a function of the RGB values. For opacity, we read the frame's alpha channel and count pixels for which the normalized alpha value is >0.5 [1]. #Threads and #Pixels are collected directly from NG. GPUTime is obtained from the simulation results. Comparing the trends in the sub-figures (Figure 6) , we make the following observations:

❶ For 75% of the frames, variation in all the features' values from the previous frame's values is at most 25%.

Table 3. Frame-specific attributes

Parameter	Abbr.	Description	Collected From
Brightness	Br	Perceived brightness of the frame [10]	NG
#Threads	Th	Number of software threads spawned for drawing the frame	NG
#Pixels	Px	Number of pixels rendered in a frame	NG
Opacity	Op	% of opaque pixels	NG
GPUTime	G_t	GPU execution time or rendering time (in ms)	GPUTejas
ATW Latency	A_t	ATW Latency (in ms)	GPUTejas

❷ Features that show almost constant behavior across the frames are brightness, #threads, and #pixels.

$$Brightness = 0.299 \times R + 0.587 \times G + 0.114 \times B \tag{1}$$

where R, G, and B represent the average pixel intensities in each channel.

3.6 Resolution of the Human Eye and Low MPD

A human with 6/6 vision has a resolution of 0.59 arc minute per line pair [16] (resolution: $\approx$ 0.3 arc-minutes). Now assume that the head rotates at the rate of 200 degrees per second [66] (in a fast VR game). The minimum eye resolution thus corresponds to 0.02 ms. If the MPD is more than that we will perceive a lag, which may lead to discomfort and visual disorientation. We have sadly experienced this on Oculus Quest 2 when we were playing a Harry Potter style game with a broom and an orb. This calculation is however theoretical and the vision system can compensate to some degree. Thus, the concept of JND (just-noticeable difference) has been proposed, which represents practical limits (measured in lab settings). The JND for regular yaw-based head motion has been measured to be as low as 1 ms [30] when there are other stimuli that last for more than 60 ms, and roughly 3.2 ms without other stimuli [29]. For haptics-VR applications for robotic arms, the JND limit is 2 ms [2]. We conducted various in-vivo experiments and found that the JND for HMDs is even lower than 1 ms (refer to Appendix A.4). Hence, this means that we cannot miss frame deadlines (one frame appears every 11 ms) or delay ATW by more than 0.5 ms. The differences will be perceived by at least some users with good visual acuity. **This explains why in prior works [73], the authors have tried to reduce the MPD by 4-5 ms and why the gaming industry is moving towards 144 to 360 Hz displays.**

Insights:
❶ The ATW latency varies from 2 ms to 15 ms across our benchmarks. Inter-process resource contention on a GPU accounts for this.
❷ There exists high inter-frame similarity in VR applications ($\approx$ 80% of the time).
❸ The human vision system is very sensitive to deadline misses and latencies of even 1 ms.

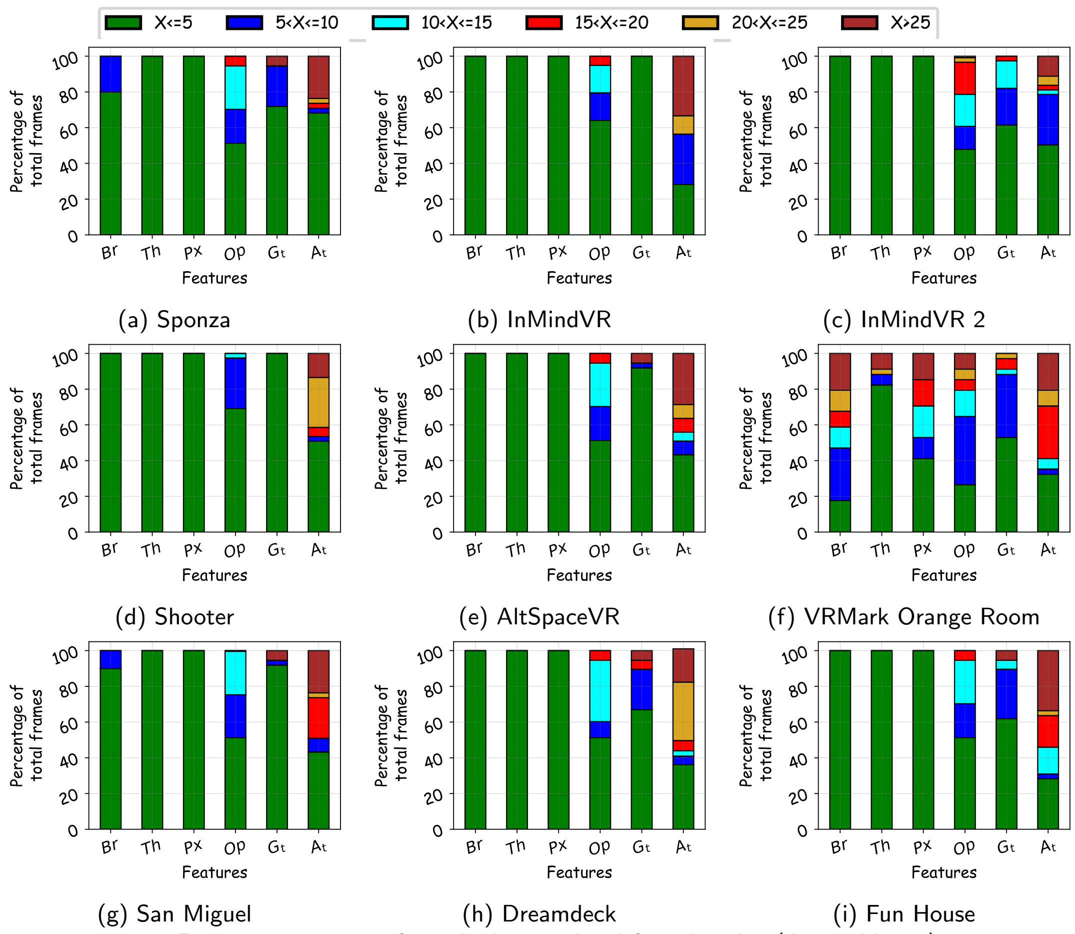


Fig. 6. Percentage variation from the last rendered frame's value (denoted by X)

4 IMPLEMENTATION

As mentioned in Section 2, our aim is to predict the ATW latency for VR applications. We broadly divide our implementation into various sub-components as follows: ❶ We first create an environment that simulates the GPU-accelerated ATW algorithm. ❷ We then identify the input features for the regression model. ❸ We run the simulations to collect the sample data and divide the collected data into a training and testing set. ❹ We then train multiple predictors and compare their performance for the test data. ❺ After selecting the best performing predictor, at run time we check its performance in terms of the refresh deadline miss rate and how close the completion of the time warping is to the display refresh point. ❻ We finally design a hardware for our predictor.

4.1 Simulation Environment

As mentioned in Section 3.5, we use the *GPUTejas* [35] simulator for our experiments. Table 2 shows the simulator configuration. We implemented two types of scenarios: ❶ Only one benchmark runs with GPU-accelerated ATW on the simulator. Since, the display refresh rate is 90 Hz, in the period of every 11.11 ms, we need to preempt the frame's simulation and initiate the ATW operation. ❷ The VR benchmark shares cores with a desktop game and a CUDA application. For this, we divide the cores equally among the applications and the cores are assigned to the other running application if one application finishes.

4.2 Identifying the Features

The important task in designing any prediction model to predict the ATW latency is to define the set of input features. Good feature selection is critical to the success of the model. In our case, the selected features must be correlated with the frames' characteristics and the amount of resource contention. The features should also have a correlation with the ATW latency that we want to predict.

To achieve this, we must first examine the GPU activities in operation at the time of ATW initiation. To capture this information, we plot various micro-architectural parameters and performance counters for the GPU as a time-series graph. A smaller portion of the time-series plot is displayed in Figure 7, which shows the MPDs for three different refresh intervals. In the figure, we have four vertical lines, each representing a specific type of event. The red line shows when ATW is invoked, the blue line represents when preemption completes and the time warp operation starts, the green line shows when the time warp ends, and the grey line indicates when the display is refreshed. Ideally, we would want the red and green lines to be as close as possible. The time lost in clearing the active queue (preemption latency) is the time interval between the red and the blue lines.

In Figure 7, we can see that the interval between the red and blue lines varies across all the three cases, while the blue and green lines remain equally spaced. This indicates that the time warp latency remains almost *constant* across frames while the preemption latency varies. Moreover, the rate at which the active queue empties differs. The reason is as follows. When ATW is invoked, the GPU treats it as a high-priority thread and starts preempting all currently running threads present in the active queue. These threads could belong to the rendering task of the primary VR application or could belong to any other applications that were also running on the GPU. We have incorporated a thread-level preemption model similar to the latest NVIDIA GPUs (refer to [43]), which means that every warp must be fully executed and the active queue needs to be emptied before starting the time warp operation. Since, GPUs typically execute thousands of threads concurrently to achieve high throughput, and each thread may have a different execution time, the preemption latency becomes large and unpredictable. The most important variable here is the number of semi-executed warps that are waiting to finish. They may be delayed for a lack of access to GPU's computational resources or memory values. Examples of such threads that pose such problems are as follows (as seen in our experiments): threads performing vertex transformations, computation of lighting effects (ambient, diffuse, specular, and other forms of lighting), etc. Recall that we can run the time-warping kernel only after the warps have fully finished executing because the intra-warp context switching is still not there in NVIDIA GPUs. To see such effects in real-world devices, please refer to Appendix A.1.3.

The summary of this discussion is that the variation in preemption latency is correlated with the workload on the GPU at the time of ATW initiation. For instance, Case 3 had fewer active threads, which completed faster. However, in the first two cases, the total number of active warps was large, resulting in a more extended preemption latency as compared to Case 3. But, there is a difference in the two latencies because it depended upon which threads were getting executed and the nature of the work they were doing. All of this needs to be captured by our decision tree. Hence, we select these hardware performance counters as features because these are highly correlated with the MPD (refer to Appendix A.2).

As mentioned earlier, the ATW latency depends upon the frame currently getting rendered and our model should be able to learn the frame's behavior. Hence, we consider a few frame-specific metrics as input features. These features are #vertices,#pixels rendered, #draw calls, and #threads for those draw calls. We used these features to calculate the inter-frame similarity too in Section 3. Similarly, the level of opacity in a frame is an important characteristic that we include as one of the input features. Any opaque regions in the frame have an impact on the complexity of the rendering process. This is because transparent fragments are not subjected to the pixel shading process. Therefore, the complexity of the shader varies depending on the opacity of the frame, which in turn affects the preemption latency [1]. This information can be retrieved from the alpha channel which is associated with the frame buffer and stores a related opacity value for each pixel. We define this feature as *opacity*. Similarly, the GPU execution time or rendering time of a frame depends upon the resources' availability and the frame's complexity (refer to Section 3.4). Hence, we make the GPU execution time an input feature. Since, the frame which the time warping preempts is not rendered completely, we do not have its execution time at the time of prediction. So, we take the execution time for the last rendered frame. In Section 3.4, we observed that the ATW latency varies across frames due to the activities associated with the preemption of the current frame's rendering tasks, in other words, on the current resource usage. This means that it incorporates the effect of resource contention. As a result, we use the ATW latency of the previous refresh interval as one of the input features.

We only take into account those parameters that model the characteristics of the frames or in any way influence the resource contention because these are the primary factors affecting the prediction. The head pose, for instance, is one possible feature because the time warp algorithm takes the head pose as an input. However, our simulations show that the computational part of the ATW procedure remains unchanged regardless of the features' values (Refer to Figure 3b). Thus, we do not list it among our features. Table 4 shows all the input features. Note that we have a column that says whether the data is for the current frame or the previous frame. Many features such as the rendering time can only be collected for the previous frame; they are in the process of getting computed for the current thread.

It is worth noting that we have carefully selected the features for prediction in our approach to make it insensitive to the number of other applications (noise) running within the GPU. To clarify, let us take the example of the *number of active warps* feature. The predictor is only concerned with the total number of active warps at a particular moment, irrespective of which applications generated those warps or how many of them did so. The same principle applies to the other features as well.

4.3 Prediction Model

We implement four regression models, as explained in Section 2.3. We use scikit-learn v1.0.1 [57], an open-source machine learning library, for implementing these regression models. The entire dataset is first divided into disjoint training and test sets. The splitting of the dataset is done randomly. Our results show that the Decision Tree Regressor gives the least prediction error. Moreover, as mentioned in Section 3, the decision tree provides an explainable model and we require explainability in our case. We check the importance of all the features in the prediction process and the sensitivity of the model with respect to some of these features. Here, one concern is the generalizability of the predictor for the different types of graphics applications and ATW algorithms. Section 5.1 shows that the predictor works well across benchmarks. Regarding ATW algorithms, as of today, all the time-warping algorithms use the same core computations: matrix (re)projections, and for

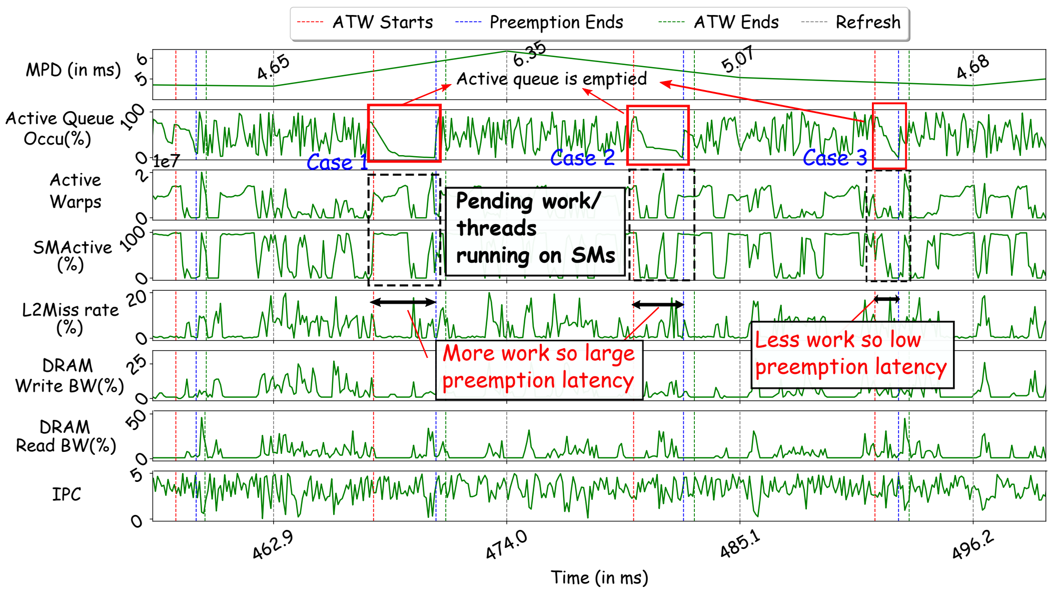


Fig. 7. Time-series plot of various hardware performance counters

Table 4. List of features

Feature	Abbr.	Description	Type	Source	Frame
GPUTime	G_t	Frame rendering time	HPC	GPUTejas	prev
Prev ATW Latency	A_t	The ATW latency	HPC	GPUTejas	prev
*L2 Miss**	L_m	L2 cache miss rate	HPC	GPUTejas	prev
*SM Active**	SMA	(Total cycles when SMs were active/ total elapsed cycles)*100	HPC	GPUTejas	prev
*Active Warps**	AW	# active warps	HPC	GPUTejas	prev
*Active Instructions**	AI	# executed instructions	HPC	GPUTejas	prev
*DRAM Read Bandwidth**	DR	(DRAM read cycles / total elapsed cycles) * 100	HPC	GPUTejas	prev
*DRAM Write Bandwidth**	DW	(DRAM write cycles / total elapsed cycles) * 100	HPC	GPUTejas	prev
Opacity	Op	% of opaque pixels	FS	NG	prev
#Pixels	Px	# rendered pixels	FS	NG	prev
#Vertices	Vr	# vertices in the frame	FS	NG	curr
#Threads	Th	# software threads	FP	NG	curr
#DrawCalls	D_{cl}	#draw calls	FP	NG	curr

NG: Nsight Graphics, **HPC:** HW performance counter, **FS:** Frame-specific, **FP:** Function parameter
*: These values are for the duration in which the preemption is happening.

them, our predictor works nicely. We don't currently have enough information to establish whether our prediction will work for a drastically different method that hasn't yet been developed.

The decision tree is an open-loop system and does not capture the short-term history. Hence, we propose a solution based on cascaded learning, which uses a simple PID controller to learn from previous prediction errors in the recent past – this enhances the accuracy of the predictor. At any given time t, the controller takes the previous prediction error e_{t-1}, computes its PID component E_t, and feeds it to the decision tree along with the features $\langle f_t \rangle$.

This allows for a more accurate prediction and better performance overall. Equation 2 shows that E_t consists of three elements: the scaled version of the most recent prediction error (P), the integration of past errors over time (I), and the rate of change in the prediction error (D). The complete design of the proposed predictor is shown in Figure 8.

To check the robustness of our prediction model, we rely on the leave-one-out cross-validation (LOOCV) [12] a method to avoid overfitting. In this approach, we leave a data point out and train the model on the rest of the date points. This process is repeated for every single data point in the dataset (basically rotate the train-test set). The final prediction accuracy is the mean accuracy computed across all train-test pairs.

$$\begin{aligned}
&E_t = P + I + D\\
&where,\\
&P(ProportionalControl) = k_p * e_{t-1}\\
&I(IntegralControl) = I_{t-1} + k_i * e_{t-1} * \Delta t\\
&D(DerivativeControl) = k_d * (e_{t-1} - e_{t-2})/\Delta t\\
&\Delta t = 0.011s, \text{ since the refresh interval is 11.11 ms.}
\end{aligned} \tag{2}$$

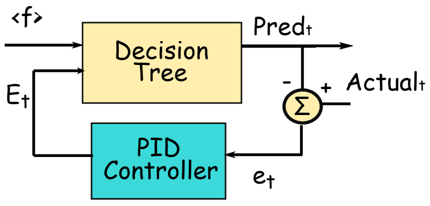


Fig. 8. Design of the proposed predictor. $\langle f \rangle$, $E(t)$, and e_t are features, the controlled error, and the prediction error at time t, respectively.

4.4 Workflow of PredATW

After identifying a suitable ML model and features for predicting the MPD, we will explain the workflow of the *PredATW* method in detail in this section. The proposed method can be divided into two stages: feature collection and prediction using a decision tree-based predictor. All the features required for the prediction, except *Opacity*, are either available from the application at runtime or are hardware performance counters that can be directly captured from the GPU. Therefore, we propose a hardware module to calculate *opacity*. For the second stage i.e. prediction, we propose a hardware structure too. Hence, the proposed hardware accelerator has two components: ❶ opacity calculator and ❷ a decision tree-based predictor. As mentioned in Section 1, the ATW runs once in each refresh interval, and we need to predict its latency for that interval. Therefore, *PredATW* runs once in each refresh interval, just after the display refreshes. To delay the launch of the ATW kernel based on the predicted latency, we envisage a high-resolution timer that raises an interrupt when ATW needs to be started. Then, the OS sends a signal to the application, which invokes the ATW function registered as a callback. The final architecture for the proposed design is shown in Figure 9. The details of the hardware accelerator are given in Section 4.5.

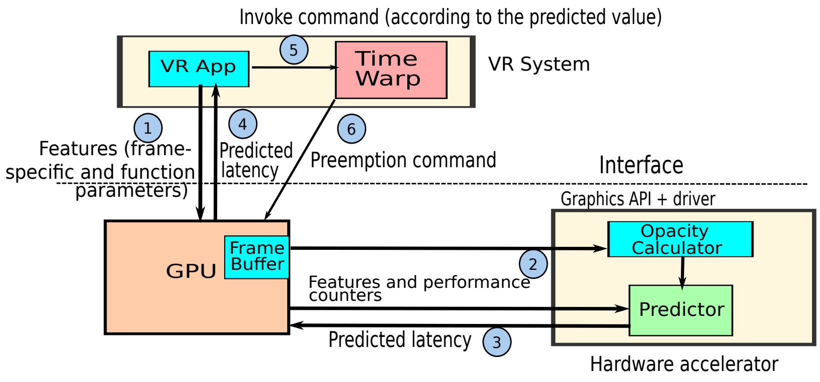


Fig. 9. System Architecture

4.5 Hardware Accelerator

As mentioned in Section 4.4, the proposed HW accelerator consists of two components. One calculates opacity, and the other is the predictor (decision tree). We design the hardware using Verilog. The opacity computation simply takes values from the frame buffer's alpha channel and count pixels with an alpha value greater than 0.5 (8-bit binary representation: 10000000) [1]. To determine if a pixel's alpha value exceeds the threshold, we check the most significant bit (MSB) of each pixel. If the MSB is 1, then the value exceeds the threshold. We do not need a comparator to do this; we can simply count the number of pixels with MSB as 1 by adding the MSB of all the pixels. To compute the opacity, we need to transmit the alpha channel of the frame from the GPU memory to the accelerator through the PCIe bus. Since we only require the most significant bits (MSBs), we transmit only those. The data is transferred using the GPU's PCIe Gen4 bus with a bandwidth of 64 GBps. The memory interface bandwidth of the NVIDIA RTX A4000 is 256 bits, so we cannot access all the data at once. Therefore, we need to add these 256 bits before the next set of bits arrives. To achieve this, we use a binary adder tree of depth $ceil(log_2(256))$ which enables us to add 256 bits in parallel. We do not compute the opacity in percentage because we only need relative values. Furthermore, we implement the decision tree as a tree of multiplexers. Our decision tree has 103 leaf nodes and a depth of 10. There are 102 decision nodes, so we have 102 multiplexers. The select signal of each multiplexer is the threshold value of that decision node. We store the values of decision nodes and leaf nodes in registers. For leaf nodes, we use registers of 8 bits. Due to a wide range in threshold values, decision nodes' register sizes are not uniform across the tree. The minimum and maximum sizes of the registers are 5 and 32, respectively. The overheads of the hardware accelerator are the overheads of *PredATW*.

5 RESULTS AND ANALYSIS

As discussed earlier, we use a simulator-based approach to evaluate our proposed design, *PredATW*. The configuration details of the physical system and the simulator are given in Table 2.

5.1 Performance of Various Predictors

As mentioned in the previous section, we implement four prediction models: random forest, linear regression, gradient boosting, and a decision tree. We use the same set of features for all the models. We compare the performance of these models in terms of the Mean Absolute

Error (MAE). The formula for MAE is shown in Equation 3.

$$MAE = \frac{\sum_{i=1}^{N} |Y_i - \hat{Y}_i|}{N} \tag{3}$$

5.1.1 Performance without the PID Controller. The results are shown in Figure 10. We observe that the decision tree-based regressor performs the best among all the models. It achieves an error of 0.33 ms on an average for all the benchmarks which is very modest. The prediction error of the decision tree is less than 0.38 ms for all the benchmarks.

Consider the case of the random forest predictor. The average error in this case is 0.55 ms, which is 83.3% greater than the prediction error in the case of the decision tree. In spite of having 100 randomly created decision trees, we were not able to observe a higher accuracy. There is no benefit gained by the randomness. On the other hand, each decision tree was trained with fewer samples, which reduced the accuracy of all the constituent trees. Consider the case of linear regression, the average error we get for this is 0.41 ms, 37% greater than that of the decision tree. We can infer that the relationship is far from linear. Hence, such simple models will not work. Similar to random forest, the complex gradient boosting model has a prediction error of 0.42 ms that is 40% more than the prediction error of the decision tree.

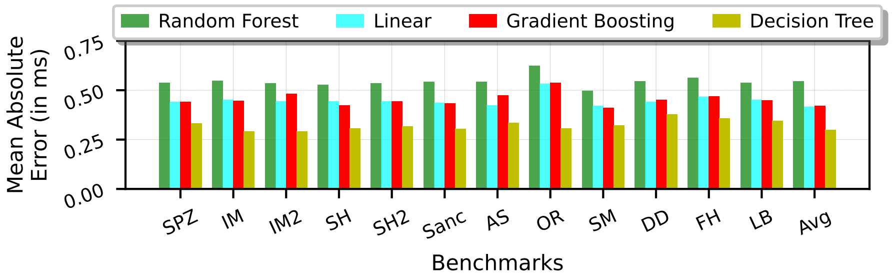


Fig. 10. MAE of the different prediction models

5.1.2 Performance with the PID Controller. As mentioned in Section 4.3, we include a feedback component in the predictor to improve the performance of the predictor further. Figure 11 shows the MAEs after using the PID component of prediction error as an input feature to the predictor. Similar to the previous case, the least prediction error is for the decision tree i.e. 0.22 ms. Figure 12 shows that the average error of the decision tree is reduced by 31.37%. To understand the impact of the PID controller on the prediction error better, please refer to Appendix A.3.

5.2 Effect of the Predictor on the MPD

To see the effect of our proposed approach on the MPD, we record the deadline miss rate with and without the predictor. We also see how close the completion of the time warp operation is to the *display refresh point*. As mentioned in Section 4, the display refresh rate is 90 Hz, which means that the display is refreshed every 11.11 ms. We analyze the refresh deadline miss rate for the following three cases:

❶ GPU-accelerated ATW, which is the *Baseline* architecture.

❷ ATW is invoked just after the frame is rendered (no preemption) or before 2.02 ms of the screen refresh operation, whichever occurs first. Recall that 2.02 ms is the average latency of

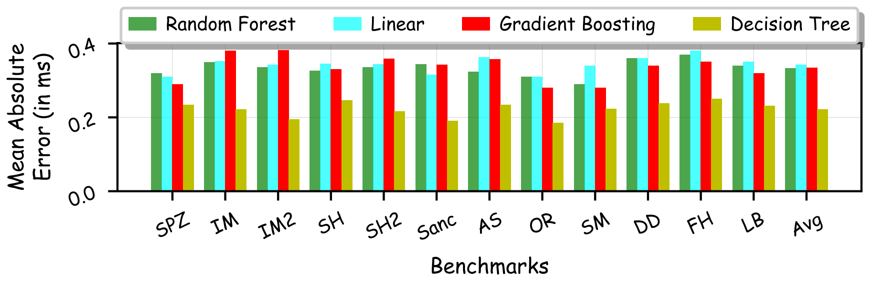


Fig. 11. MAE of the different prediction models

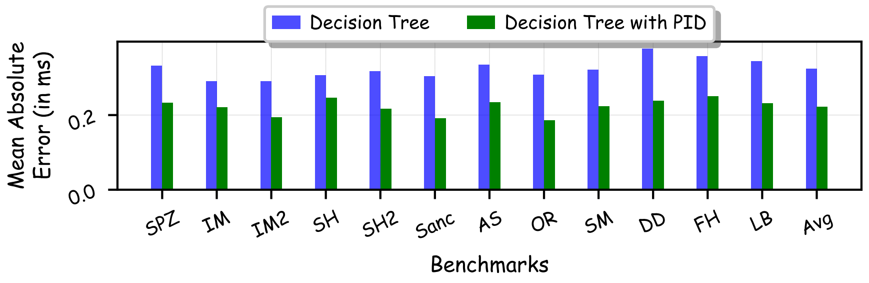


Fig. 12. Improvement in the MAE of the decision tree after applying the PID control

the time warping kernel. (Refer to Figure 3b) (we refer to this as the *Eager* approach). Note that simply starting it before 2.05 ms (maximum latency) will not work because in this case there will always be a deadline miss unless the preemption latency is very close to zero.
❸ ATW is invoked on the basis of the predicted ATW latency, which is our proposed approach. We call it *PredATW*.

5.2.1 Refresh Deadline Miss Rate. As mentioned in Section 1, when the ATW fails to get completed before the display refreshes, the MPD is increased and the user sees an anomalous or blurry view of the world. Figure 13 shows the refresh deadline miss rates for the three cases. The figure shows that for all benchmarks, our predictor decreases the deadline miss rate. For GPU-accelerated ATW (the baseline), the deadline miss rate starts at 5% and goes up to almost 27% for all the benchmarks. The average miss-rate is around 9.8%, which is significant.

For the *Eager* case where the ATW is invoked just after the frame is rendered, the average deadline misses are less than 1% . However, this will ensure a large MPD and the reason for this is clear: we have a large gap between the time that ATW finishes and the display refresh point. The result shows that *PredATW* (our scheme) reduces the number of deadline misses substantially: from 9.8% to 1.9%. In this case, the miss rates are in the range of 1.2 to 4.1 % for all the benchmarks. The relative reduction in the deadline misses is substantial. The overall reduction in the miss rate of *PredATW* is 80.6%.

5.2.2 ATW Completion → Display Refresh Point. In Section 1, we discussed that the MPD is increased if ATW is finished too early before the refresh deadline. Table 5 shows that *PredATW* brings the ATW completion much closer to the refresh deadline and reduces the

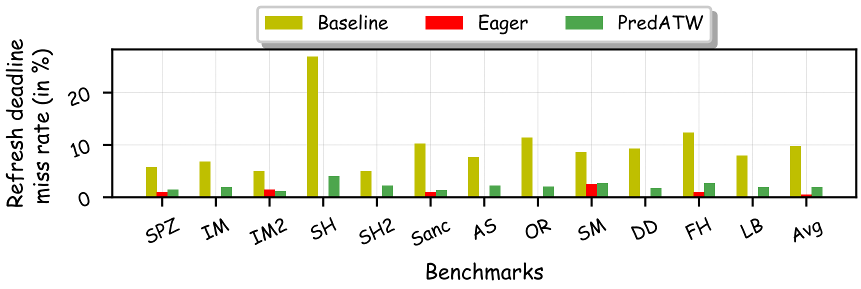


Fig. 13. Effect of the predictor on the refresh deadline miss-rate

Table 5. The average duration between ATW completion and the display refresh point (milliseconds) when a deadline is not missed.

Benchmarks	**Baseline**	**Eager**	**PredATW**
SPZ	3.17	6.31	0.26
IM	3.28	8.78	0.34
IM2	2.34	6.96	0.19
SH	3.58	7.3	0.42
SH2	3.25	5.63	0.36
Sanc	2.67	6.87	0.21
AS	2.87	7.12	0.35
OR	2.66	7.01	0.23
SM	3.75	8.10	0.24
DD	3.25	7.65	0.36
FH	3.34	6.87	0.32
LB	3.15	5.78	0.29
Mean	3.10	7.03	0.29

MPD. We observe that as compared to *Eager*, *PredATW* reduces the duration by 92.4%. It is also 90.6% lower than *Baseline*. The important point to note is that since the prediction error is very less, for *PredATW* this duration is on an average less than 0.42 ms all the time, which will most likely not be perceived (see Section 3.6). For the other two schemes, we saw a lot of high values in our results which will be perceived by the users.

5.3 Sensitivity Analysis

As mentioned in Section 3.3, we identify a list of hardware performance counters that show the contention effect on the ATW latency. To show how these features affect the prediction error, we train and test the model with and without these features.

5.3.1 Effect of Different Feature Combinations on the Prediction Error (Ablation Study). The features used for the proposed predictor are classified into three categories: HPCs, frame-specific features (FSs), and function parameters(Fps). We try different feature combinations to show the importance of each category (refer to Figure 14). The average prediction error when considering all the features is 0.22 ms, which is increased by 59%, 850%, and 459% when considering only HPCs, frame-specific features, and function parameters, respectively.

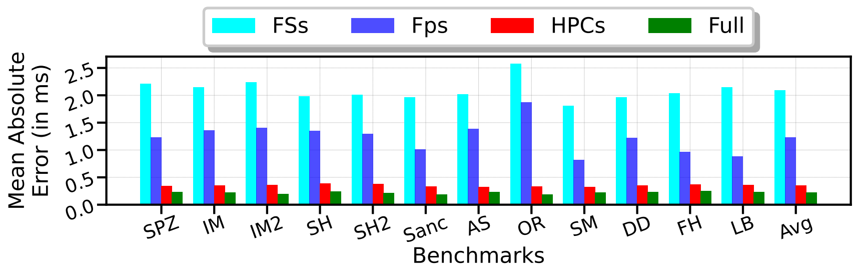


Fig. 14. Effect of various feature combinations on the prediction error

5.4 Explanability of the Predictor

A common concern with machine learning-based predictors is that they are opaque and difficult to understand. However, we delve into the internals of the prediction process and investigate the decision making process. We examine the features utilized in the decision paths in great detail and obtain a bunch of insights that are *intuitively explainable*. We have used abbreviations for features instead of their full names. Table 4 provides the list of features with their abbreviations.

5.4.1 Analyses of the Decision Paths. We have conducted an extensive analysis to gain a deeper understanding of which features are used in the decision-making process. To achieve this, we thoroughly reviewed the decision paths of all test data points and kept track of how many times each feature was used at least once in the decision making process. We have presented our findings in Figure 15. We make the following observations from the figure:
❶ We note that AI and SMA are used in 100% of the data points. This was expected because we have already established that these features have a strong correlation factor with the MPD.
❷ G_t, DR, and E_t are used in almost 50% of the data points. This indicates that these features are also essential in the decision-making process.
❸ The remaining features are also utilized in the decision-making process to a decent extent.

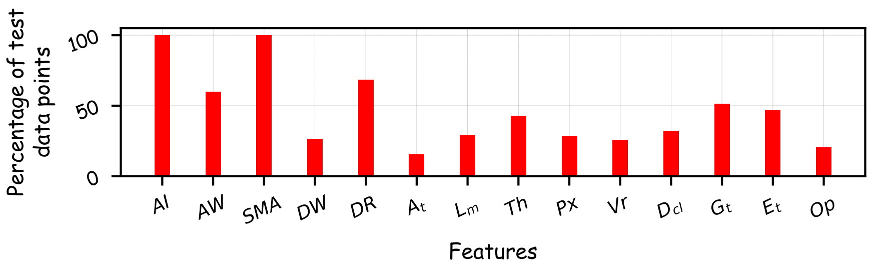


Fig. 15. Percentage of the test data points containing a feature in their decision path

5.4.2 Importance of each feature. To show the importance of each feature in the prediction process, we find the frequency of each feature in the decision path for 100 representative test samples and plot them using the radar plot in Figure 16. We make the following observations from the figure:
❶ The radar plot in the figure shows that three features – AI, AW, and SMA – are used the maximum number of times in the decision paths.

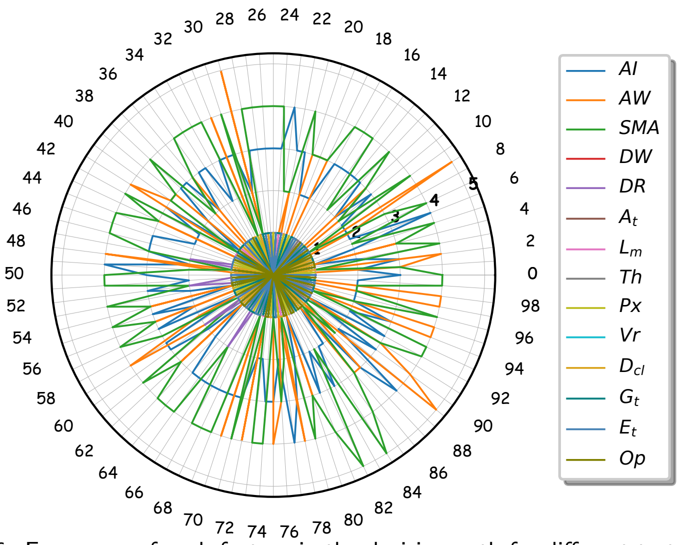


Fig. 16. Frequency of each feature in the decision path for different test points

❷ G_t and E_t play a major role in the sense that they are close to the root and are used to make the first few decisions. This basically means that the predicted value is very sensitive to this parameter.
❸ DR is at the other end of the spectrum – it is used 0-2 times.
❹ The rest of the features have a moderate frequency and thus have a moderate amount of sensitivity.

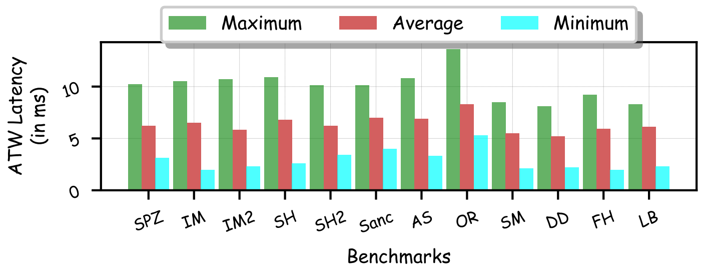
Table 6. Platform Configuration

Parameters	GPU 1	GPU 2	GPU 3
Mode	Tethered	Tethered	Standalone
GPU Architecture	Ampere [41]	Ada Lovelace [44]	Kepler [40]
Clock Frequency	1 GHz	1 GHz	1 GHz
#TPCs, #SMs, #CUDA Cores	24, 48, 6144	38, 76, 9728	8, 16, 128
Warp Size, #Warp schedulers per SM	32, 4	32, 4	32, 2
Register file size per SM	256 KB	256 KB	64 KB
I-cache size per SM	8 KB	8 KB	4 KB
L1 data cache per SM	128 KB	192 KB	64 KB
L2 cache size	4 MB	6 MB	1.5 MB
Main memory	16 GB	16 GB	2 GB

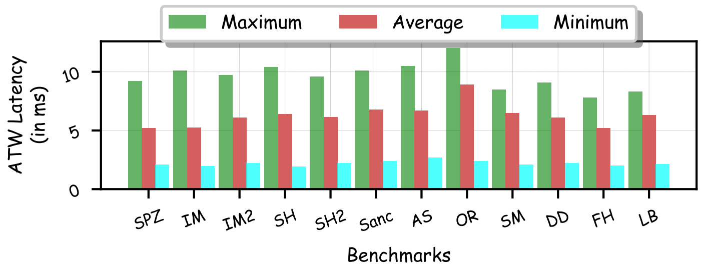
5.5 Effect of Underlying Hardware on the Predictor Performance

To demonstrate how the underlying hardware influences the performance of our proposed method, we conducted simulations on three different GPU architectures. For standalone mode, we simulated the architecture of the Adreno GPU that is present in Oculus Quest 2. For tethered mode, we simulated two NVIDIA GPUs: RTX™A4000 and RTX™4080, based on the Ampere [41] and Ada Lovelace [44] architectures, respectively. Table 6 provides configuration details for the simulated architectures.

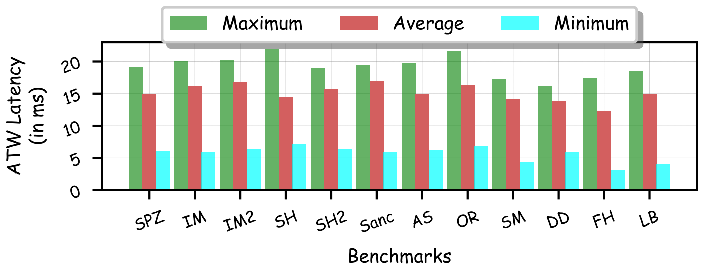
5.5.1 ATW Latency for Simulated Architectures. Figures 17a, 17b, and 17c show the ATW latency for the three scenarios (the standalone mode and the tethered mode with two



(a)



(b)



(c)

Fig. 17. (a) Variation in the ATW latency (Tethered mode with NVIDIA RTX™A4000); (b) Variation in the ATW latency (Tethered mode with NVIDIA RTX™4080); (c) Variation in the ATW latency (Standalone mode).

different GPUs). We make the following observations from the figures:
❶ The ATW latency ranges from 4 ms to almost 22 ms across benchmarks for the standalone mode; this is 2 ms to 15 ms for the tethered mode. This is because the Adreno GPU is slower than NVIDIA RTX™A4000 and RTX™4080 in terms of its processing capabilities, memory size, and memory bandwidth.
❷ Both the NVIDIA GPUs have nearly an equal ATW latency range due to their higher computational power.

Conclusion drawn from these results: The summary of these results is that the broad trends across different architectures are the same. Given that our design is meant to capture those trends, it tends to work well.

5.5.2 Predictor Performance. Figure 18 shows the prediction accuracy for the three simulated scenarios. We make the following observations from the figure:
❶ The range of the prediction error is very little, i.e., from 0.19 ms to 0.24 ms.
❷The average prediction errors for all the three simulation setups are nearly the same and hover around 0.22 ms. Given these results, we believe that our solution will work for other setups too (in the future as well).

It is worth noting that despite the differences in the range of the ATW latency across the three simulation setups and the varying microarchitectural resources, the prediction errors were nearly identical. This indicates that the prediction accuracy is not influenced by the underlying hardware and remains more or less consistent across GPU architectures.

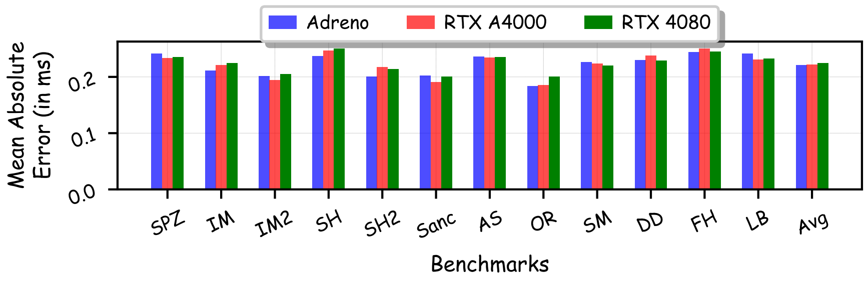


Fig. 18. Performance of PredATW under different design scenarios

5.6 Overheads of PredATW

The overheads of the proposed hardware accelerator are the overheads of *PredATW*. As mentioned in Section 4.4, the hardware accelerator has two modules: the opacity calculator and the predictor. We implemented the HW design in Verilog and synthesized it. Specifically, we synthesized, placed, and routed the hardware using the Cadence Genus Tool (TSMC 7 nm technology) and obtained the power, area, and timing numbers. Table 7 shows the latency, area, and power overheads of the hardware accelerator and its individual modules. The results are for the 1080p image resolution. We make the following observations from the results:
❶ The latency of opacity calculation is 16.86 μs, which includes a 15.6 μs data transfer latency.
❷ The latency of the decision tree is 2.626 μs which is insignificant.
❸ The total area is 0.27 mm^2, and the total latency of the hardware accelerator is 19.49 μs. Both values are negligible.

Table 7. Overheads of the hardware accelerator

Step/ Module	**Area**	**Power**	**Latency**
Opacity calculator	0.09 mm^2	11.42 μW	16.86 μs
Decision Tree	0.17 mm^2	31.96 μW	2.626 μs
Total	**0.27** mm^2	**43.38** μW	**19.49** μs

5.7 Comparison with the State-of-the-Art

We compared the performance of our proposed method with two state-of-the-art methods [59, 73] that accelerate the time warp operation by offloading it to a separate hardware unit. Both schemes propose preemption-free ATW designs that asynchronously execute on separate hardware without interrupting the rendering tasks on the host GPU. The difference between the two is that Smit et al. [59] propose a dual-GPU architecture in which the scene is rendered on the host GPU and warping is done on a separate GPU. Both the GPUs share data using a producer-consumer buffer, which is implemented using interprocess shared memory. On the other hand, PIM-VR is a processing-in-memory-based design that executes the ATW within a 3D-stacked memory. It connects the host GPU directly with an off-chip Hybrid Memory Cube (HMC). Figure 19 shows the MPDs for these two solutions along with that of *PredATW*, which clearly uses far less hardware. It is important to note that the additional GPU that they use is as powerful as the primary GPU that runs the VR application. One more point to note here is that since both of these solutions are preemption-free, there are no deadline misses. Hence, the MPD for *PredATW* is also for the cases with no deadline misses.

We make the following observations from the results:

❶ The ATW latency is almost constant across benchmarks for the state-of-art solutions because of no preemptions.

❷ The proposed solution, *PredATW*, has an average ATW latency of 2.87 ms, which is 47.91% and 53.78% less than that of PIM-VR and the dual-GPU architecture, respectively. This is a massive advantage of our design. The reasons are mentioned in point ❸.

❸ The high ATW latency values for the two related works are due to the fact that the data transmission latency tends to play a very major role. The entire frame including all the metadata needs to be transferred to the secondary GPU, and then transferred back. In the case of PIM-VR, this overhead is somewhat lower because it uses a high-bandwidth off-chip memory; however, it is still present and is sizeable.

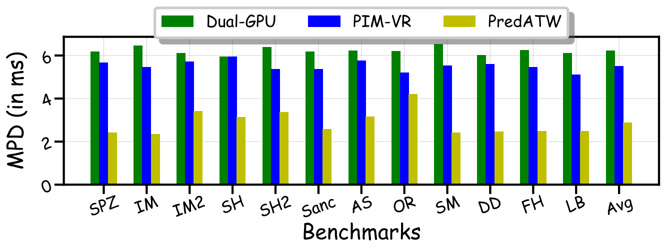


Fig. 19. MPD of various state-of-the-art solutions

5.8 Solution Space Exploration (CPU, GPU, Accelerator)

An important question that needs to be answered is bespoke hardware really required? Why not run all of it in software (on the CPU or GPU)? Section 3.2 briefly explained that feature collection in software is a high-latency task and thus CPUs are not idea for running the predictor. Let us investigate this issue in depth in this section. We have three types of tasks, rendering (R), ATW (A) and prediction (P), and three possible choices to run these tasks, CPU, GPU and a HW accelerator. The solution space is shown in Table 8. Furthermore, we can *spatially partition* the CPU or GPU cores among the tasks [59, 60]. Note that at the moment NVIDIA GPUs do not provide support for spatial partitioning ; however, there are reports that some undocumented features can possibly be used [6]. It would thus not be unwise to speculate that such support may become commonplace in the future. Hence, we assume the existence of fine-grained core partitioning in the GPU as well. Our solution space is thus very large and finding an optimal solution necessitates an AI algorithm that minimizes the MPD and at the same time minimizes *frame staleness* (relying too much on ATW because of slow frame rendering). We experimentally found that the refresh deadline miss rate is roughly proportional (directly) to the MPD. Second, the frame staleness, defined as the number of times ATW uses an old frame for warping is inversely correlated with the output frame's quality [8]. This increases the judder perceived by the user, which also induces motion sickness. Because ATW alters the frame only for the head and eye movements and does not handle any other temporal changes; hence it works efficiently and reduces the MPD only when the most recently rendered frame is used as input [58, 69, 74]. Table 8 shows the best performance in a software-only scheme with and without spatial partitioning. We needed to use the simulator for the spatial partitioning experiment. It was calibrated with our hardware ($\pm 10\%$). We prioritize MPD over staleness unless mentioned otherwise.

Because the search space does not have a predefined structure, we used Genetic Algorithms (GA), which have previously worked effectively for similar unstructured problems [31, 39]. The key concept of a GA is crossover and mutation, which implies that we start with a set of good candidate solutions and then prune them to build the next set of solutions until we get the desired solution or the algorithm converges. In the crossover technique, we choose a point and swap configuration parameters beyond that point between the two parent solutions. For example, in our case, after one-point crossover, the two parent solutions– R(CPU), A(GPU), P(CPU) and R(GPU), A(CPU), P(CPU) yield two new solutions– R(CPU), A(CPU), P(CPU) and R(GPU), A(GPU), P(CPU). Our algorithm uses one-point and two-point crossover techniques. Then we mutate the new solutions by introducing changes in the partitioning percentages by adding or subtracting a random number (in %) from the original values. We experimentally measured the performance of all the solutions generated by this algorithm and arrived at a couple of solutions that are at the Pareto-optimal front.

Without spatial partitioning, the best solution corresponds to the case – CPU (P) and GPU (R, A) (real hardware). The refresh deadline miss rate we get is 42%. The staleness is the same as PredATW. In comparison, for PredATW, it is 1.99%. The reason for such a high deadline miss rate is the latency of the feature collection process. For our best SW case (CPU-based predictor), we need to rely on the NVIDIA NSIGHT Perf-SDK APIs [48] and application-specific annotations to collect our features, which itself takes 2.1 ms. This results in a very high latency of the predictor.

With partitioning, the two extreme ends at the Pareto-optimal front are as follows. The solution at one end minimizes the deadline miss rate (less than 1%) when we reserve 50% of GPU cores for ATW and rendering (resp.) and do not use the predictor at all; however, this results in very low utilization of the GPU, only 59.6% which was almost 91% for *PredATW*. This increased the frame staleness by 1.67×. Other solutions where we reserve fewer cores for ATW provide a slight improvement in frame staleness and utilization values but considerably increase the deadline miss rate. The other solution that optimizes the staleness is when the entire GPU is reserved for rendering, and ATW runs on the CPU. In this case, the deadline miss rate is almost 48% due to slow ATW; however, the frame staleness is decreased only by 8.9% compared to *PredATW*. All these SW solutions are significantly inferior as compared to PredATW.

With a HW accelerator, we have two choices: offload either the predictor (*PredATW*) or ATW to the accelerator (no predictor needed). The area overhead for the latter choice is 37× of *PredATW*.

Table 8. All possible solutions for designing an ATW latency predictor. R, A, and P stand for rendering, ATW and prediction, respectively.

Solution	**CPU**	**GPU**	**HWA**	**Best case performance**	
				Without partitioning	**With partitioning**
Pure SW	-	R, A, P	-	49% refresh deadline miss rate, 21× of *PredATW*	<1% deadline miss rate, only 59.6% GPU utilization, rendering time and frame staleness 1.8× and 1.67× of *PredATW*
	R, A, P	-	-		
	R, A	P	-		
	R, P	A	-		
	R	A, P	-		
	A, P	R	-		
	A	R, P	-		
	P	R, A	-		
SW + HW	-	R	A	37× area of *PredATW* [73]	-
	-	R, A	P	*PredATW*	-

6 RELATED WORK

Since the MPD in VR systems adversely affects the user's experience, over the last few years, multiple approaches have been explored to reduce this latency for interactive VR systems. Recent works focus on ❶ predictive tracking [26, 33]; ❷ motion prediction + a programmable display layer (PDL) [59, 60]; ❸ accelerating the time warp operation by offloading it to a separate hardware unit [72, 73]; and ❹ optimizing the rendering process [51, 52]. We present a brief comparison of related work in Table 9.

1. Recent works [26, 33] focus on machine learning-based approaches to anticipate viewing directions and positions to satisfy the ultra-low latency requirement of VR systems. Xueshi et al. [26] propose to do predictive pre-rendering on the edge device. Kunda et al. [33] investigate a few common prediction algorithms, dead reckoning and Kalman filtering to predict future viewing directions and locations based on past behavior. FlashBack [11] predicts all the possible positions as well as orientations and pre-renders all views corresponding to these predictions. But pre-computing all the possible views and storing them in the memory increases the compute and storage overhead substantially.

2. In this work, we use the latest pose obtained from the tracker device to warp the frames instead of predicting the head motion, which can be reasonably inaccurate. The only thing that we predict is the latency of ATW so that we can invoke the time warping kernel on time such that it finishes just before the display refresh point. Furthermore, our predictor is a simple decision tree-based predictor that does not require as much computing power as other CNN-based predictors and is very fast also.

3. Some works [59, 60] implement a dual-GPU architecture to avoid preemption. In this architecture, the first GPU, *client*, is responsible for rendering application frames as well as generating motion information and the colour, depth, and intensity attributes of the pixels. The data is transferred via the PCIe bus to a shared system memory, and the second GPU, *server*, uses this data to bring the display device up to date with a new frame. The second

GPU, server runs a custom program that takes the most recent application frame and data saved in shared memory and warps the frame based on the data and the current viewpoint. The second GPU, *server* runs a custom program that takes the most recent application frame and data saved in shared memory and warps the frame based on the data and the current viewpoint. However, these solutions [59, 60] have high hardware overheads and also high power overheads because of the additional memory accesses and writes to the PCIe bus.
4. One way to reduce the MPD significantly is to make the time warp operation fast. Waveren et al. [69] investigate various hardware platforms for implementing a low-latency time warp algorithm. Xie et al. [73] propose PIM-VR, a Processing-In-Memory based ATW design that asynchronously executes ATW within a 3D-stacked memory without interrupting the rendering activities being performed on the host GPU. They also identify a mechanism for reducing redundancy, which further simplifies and speeds up the ATW operation. Some works [51, 52] optimize the rendering process itself to reduce the MPD. Toth et al. [67] show that the number of rendered pixels can be reduced by up to 36% by rendering multiple optimized sub-projections without compromising on the visual quality. *Foveation* is the quality degradation that increases with distance from the fovea (center of the retina) [24]. Foveated rendering algorithms exploit this phenomenon to improve performance [51, 52]. These algorithms reduce the rendering complexity and quality in the periphery while keeping the image quality in the foveal area of the display at a high level. Q-VR [72] merges foveated rendering with the time warping operation to further improve the rendering performance. Note that these ideas are orthogonal to our approach and don't target the main culprit: variable context switch times.

Table 9. A comparison of related work

Work	**ML-based**	**Predicted Data**	**ATW**	**In-situ ATW**	**Optimizing Time Warp**	**Optimizing Rendering**	**Overhead**
Prediction-based Tracking [26, 33]	✓	Head and body motions	×	-	-	×	High latency
FlashBack[11]	×	All possible viewing positions	×	-	-	×	High latency
Motion Predictions + PDL [59, 60]	✓	Head and body motions	✓	×	✓	×	Additional GPU
PIM-VR [73]	×	×	✓	×	✓	✓	ATW accelerator
Q-VR [72]	×	×	✓	×	×	✓	ATW accelerator
OpenUVR [55]	×	×	✓	✓	×	✓	External HW
PredATW	✓	**ATW latency**	✓	✓	×	×	

7 CONCLUSION

In the light of the pantheon of work in this field, we adopt a very different approach. Instead of focusing on new rendering algorithms or predicting the head motion or using separate GPU hardware, we propose a relatively low-budget solution that requires only a small hardware unit. We show that we can predict the ATW latency reasonably accurately in the presence of noise, even when other GPU applications are running. The average prediction error is limited to 0.22 ms, and the deadline miss rate is minimal ($\approx 2\%$). Hence, our submission is that sophisticated solutions that require extra hardware support are not required. Even our software solution is reasonably potent and outperforms prior work that use dedicated GPUs (for MPD). We conducted a wide range of experiments with a large variety of benchmarks, GPU architectures and prediction models. Our results are robust and tend to generalize.

ACKNOWLEDGMENTS

This work was partially supported by the Semiconductor Research Corporation (SRC) via grant number 2021-IR-3053.

REFERENCES

[1] Tomas Akenine-Mo, Eric Haines, Naty Hoffman, et al. 2018. *Real-time rendering.* AK Peters/CRC Press, Florida, USA.

[2] Mohammad Alja'Afreh. 2021. *A QoE Model for Digital Twin Systems in the Era of the Tactile Internet.* Ph. D. Dissertation. Université d'Ottawa/University of Ottawa.

[3] Inc. AltspaceVR. 2016. Altspace VR. https://store.steampowered.com/app/407060/AltspaceVR/. [Online; accessed 2022-10-20].

[4] Tanya Amert, Nathan Otterness, Ming Yang, James H Anderson, and F Donelson Smith. 2017. GPU scheduling on the NVIDIA TX2: Hidden details revealed. In *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, IEEE, Paris, France, 104–115.

[5] Michael Antonov. 2016. Asynchronous Timewarp Examined. https://developer.oculus.com/blog/asynchronous-timewarp-examined/. Accessed: 2022-05-10.

[6] Joshua Bakita and James H. Anderson. 2023. Hardware Compute Partitioning on NVIDIA GPUs*. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, San Antonio, Texas, 54–66. https://doi.org/10.1109/RTAS58335.2023.00012

[7] R. Balsree, Amit Thawani, Srividya Gopalan, and V. Sridhar. 2005. Inter-Frame Similarity based Video Transcoding. In *Seventh IEEE International Symposium on Multimedia (ISM 2005), 12-14 December 2005, Irvine, CA, USA*. IEEE Computer Society, Irvine, CA, USA, 618–623. https://doi.org/10.1109/ISM.2005.71

[8] Russell M Barnes. 2017. *A positional timewarp accelerator for mobile virtual reality devices.* University of California, Santa Barbara, California, USA.

[9] Dean Beeler and Anuj Gosalia. 2016. Asynchronous Time Warp. https://developer.oculus.com/blog/asynchronous-timewarp-on-oculus-rift/.

[10] Sergey Bezryadin, Pavel Bourov, and Dmitry Ilinih. 2007. Brightness calculation in digital image processing. In *International symposium on technologies for digital photo fulfillment*, Vol. 1. Society for Imaging Science and Technology, IEEE, Las Vegas, Nevada, USA, 10–15.

[11] Kevin Boos, David Chu, and Eduardo Cuervo. 2016. FlashBack: Immersive Virtual Reality on Mobile Devices via Rendering Memoization. *GetMobile Mob. Comput. Commun.* 20, 4 (2016), 23–27. https://doi.org/10.1145/3081016.3081026

[12] Jason Brownlee. 2020. LOOCV for Evaluating Machine Learning Algorithms. https://machinelearningmastery.com/loocv-for-evaluating-machine-learning-algorithms/. [Online; accessed 2023-03-07].

[13] Umer Asghar Chattha, Uzair Iqbal Janjua, Fozia Anwar, Tahir Mustafa Madni, Muhammad Faisal Cheema, and Sana Iqbal Janjua. 2020. Motion Sickness in Virtual Reality: An Empirical Evaluation. *IEEE Access* 8 (2020), 130486–130499. https://doi.org/10.1109/ACCESS.2020.3007076

[14] Ruizhi Cheng, Nan Wu, Matteo Varvello, Songqing Chen, and Bo Han. 2022. Are we ready for metaverse? A measurement study of social virtual reality platforms. In *Proceedings of the 22nd ACM Internet Measurement Conference.* ACM, Nice, France, 504–518.

[15] Lih-Yih Chiou, Tsung-Han Yang, Jian-Tang Syu, Che-Pin Chang, and Yeong-Jar Chang. 2019. Intelligent policy selection for GPU warp scheduler. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, IEEE, Hsinchu, Taiwan, 302–303.

[16] ClarkVision. 2018. Notes on the Resolution and Other Details of the Human Eye. https://clarkvision.com/imagedetail/eye-resolution.html. [Online; accessed 2022-10-14].

[17] Counterpoint. 2023. Global XR (AR and VR Headsets) Shipments Market Share: By Quarter. https://www.counterpointresearch.com/global-xr-ar-vr-headsets-market-share/. [Online; accessed 2023-03-28].

[18] Akanksha Dixit and Smruti R. Sarangi. 2024. Effects of the MPD on user experience. https://csciitd-my.sharepoint.com/:f:/g/personal/eez208413_iitd_ac_in/Eu6R1YCl5ChElXj5Gl00-c4BPBq1c826OrHW17EFVHwVBg?e=9Nunp9. [Online; accessed 2024-03-04].

[19] Linus Franke, Laura Fink, Jana Martschinke, Kai Selgrad, and Marc Stamminger. 2021. Time-Warped Foveated Rendering for Virtual Reality Headsets. *Comput. Graph. Forum* 40, 1 (2021), 110–123. https://doi.org/10.1111/cgf.14176

[20] Borko Furht and Pornvit Saksobhavivat. 1998. Fast content-based multimedia retrieval technique using compressed data. *MSAS* 3527 (1998), 561 – 571. https://doi.org/10.1117/12.325851

[21] Newgrange Game. 2022. Sanctuary VR. https://store.steampowered.com/app/565730/Sanctuary_{}VR_{}Also_{}contains_{}nonVR_{}version/. [Online; accessed 2022-10-20].

[22] KHRONOS Group. 2022. OpenGL. https://www.opengl.org/. [Online; accessed 2023-03-03].

[23] KHRONOS Group. 2023. Vulkan. https://www.vulkan.org/. [Online; accessed 2023-03-03].

[24] Brian K. Guenter, Mark Finch, Steven Mark Drucker, Desney S. Tan, and John M. Snyder. 2012. Foveated 3D graphics. *ACM Trans. Graph.* 31, 6 (2012), 164:1–164:10. https://doi.org/10.1145/2366145.2366183

[25] David M Hoffman and Grace Lee. 2019. Temporal requirements for VR displays to create a more comfortable and immersive visual experience. *Information Display* 35, 2 (2019), 9–39.

[26] Xueshi Hou, Jianzhong Zhang, Madhukar Budagavi, and Sujit Dey. 2019. Head and Body Motion Prediction to Enable Mobile VR Experiences with Low Latency. In *2019 IEEE Global Communications Conference, GLOBECOM 2019, Waikoloa, HI, USA, December 9-13, 2019*. IEEE, Waikoloa, HI, USA, 1–7. https://doi.org/10.1109/GLOBECOM38437.2019.9014097

[27] Muhammad Huzaifa, Rishi Desai, Samuel Grayson, Xutao Jiang, Ying Jing, Jae Lee, Fang Lu, Yihan Pang, Joseph Ravichandran, Finn Sinclair, Boyuan Tian, Hengzhi Yuan, Jeffrey Zhang, and Sarita V. Adve. 2021. ILLIXR: Enabling End-to-End Extended Reality Research. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, Virtual, 24–38. https://doi.org/10.1109/IISWC53511.2021.00014

[28] Fortune Business Insights. 2023. Virtual Reaity Market. https://www.fortunebusinessinsights.com/industry-reports/virtual-reality-market-101378. [Online; accessed 2023-07-28].

[29] Jason Jerald and Mary Whitton. 2009. Relating scene-motion thresholds to latency thresholds for head-mounted displays. In *2009 IEEE virtual reality conference*. IEEE, IEEE, Lafayette, USA, 211–218.

[30] Jason J Jerald. 2009. *Scene-motion-and latency-perception thresholds for head-mounted displays*. Ph. D. Dissertation. The University of North Carolina at Chapel Hill.

[31] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. 2021. A review on genetic algorithm: past, present, and future. *Multimedia tools and applications* 80 (2021), 8091–8126.

[32] Viktor Kelkkanen, Markus Fiedler, and David Lindero. 2021. Synchronous remote rendering for VR. *International Journal of Computer Games Technology* 2021 (2021), 1–16.

[33] Ripan Kumar Kundu, Akhlaqur Rahman, and Shuva Paul. 2021. A Study on Sensor System Latency in VR Motion Sickness. *J. Sens. Actuator Networks* 10, 3 (2021), 53. https://doi.org/10.3390/jsan10030053

[34] Luden.io. 2015. InMind VR. https://store.steampowered.com/app/343740/InMind_VR/. [Online; accessed 2022-10-20].

[35] Geetika Malhotra, Seep Goel, and Smruti R. Sarangi. 2014. GpuTejas: A parallel simulator for GPU architectures. In *2014 21st International Conference on High Performance Computing (HiPC)*. IEEE, Goa, India, 1–10. https://doi.org/10.1109/HiPC.2014.7116897

[36] Meta. 2022. Oculus Debug Tool. https://developer.oculus.com/documentation/native/pc/dg-debug-tool/. [Online; accessed 2023-03-06].

[37] Microsoft. 2022. DirectX graphics and gaming. https://learn.microsoft.com/en-us/windows/win32/directx. [Online; accessed 2023-03-03].

[38] Diksha Moolchandani, Anshul Kumar, and Smruti R. Sarangi. 2022. Performance and Power Prediction for Concurrent Execution on GPUs. *ACM Trans. Archit. Code Optim.* 19, 3, Article 35 (may 2022), 27 pages. https://doi.org/10.1145/3522712

[39] Patrick Ngatchou, Anahita Zarei, and A El-Sharkawi. 2005. Pareto multi objective optimization. In *Proceedings of the 13th International Conference on Intelligent Systems Application to Power Systems*. IEEE, IEEE, Seoul, Korea, 84–91.

[40] NVIDIA. 2014. NVIDIA's Next Generation CUDA Compute Architecture. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf. [Online; accessed 2023-01-08].

[41] NVIDIA. 2020. NVIDIA Ampere Architecture. https://www.nvidia.com/en-in/data-center/ampere-architecture/. [Online; accessed 2023-03-03].

[42] NVIDIA. 2021. NVIDIA Multi-Instance GPU. https://www.nvidia.com/en-in/technologies/multi-instance-gpu/. [Online; accessed 2023-07-28].

[43] NVIDIA. 2021. NVIDIA RTX A4000 FEATURES & BENEFITS. https://download.boston.co.uk/downloads/b/4/d/b4d996a2-fc90-45c7-9fc8-2ab2e2a8a12a/Boston-NVIDIARTXA4000-Features-Benefits.pdf. [Online; accessed 2023-08-03].

[44] NVIDIA. 2022. NVIDIA Ada Lovelace Architecture. https://www.nvidia.com/en-in/geforce/ada-lovelace-architecture/. [Online; accessed 2024-03-03].
[45] NVIDIA. 2022. NVIDIA Nsight Graphics. https://developer.nvidia.com/nsight-graphics. [Online; accessed 2023-01-08].
[46] NVIDIA. 2022. NVIDIA VRWorks Graphics. https://developer.nvidia.com/vrworks. [Online; accessed 2022-10-05].
[47] NVIDIA. 2023. Nsight Graphics User Guide. https://docs.nvidia.com/nsight-graphics/UserGuide/index.html. [Online; accessed 2023-08-03].
[48] NVIDIA. 2023. NVIDIA Nsight Perf SDK. https://developer.nvidia.com/nsight-perf-sdk. [Online; accessed 2023-08-03].
[49] Oculus. 2016. Oculus Dreamdeck. https://www.oculus.com/deeplink/?action=view&path=app/941682542593981&ref=oculus_desktop/. [Online; accessed 2024-02-01].
[50] Oculus. 2016. Oculus PC App. https://www.oculus.com/. [Online; accessed 2024-02-01].
[51] Anjul Patney, Joohwan Kim, Marco Salvi, Anton Kaplanyan, Chris Wyman, Nir Benty, Aaron E. Lefohn, and David Luebke. 2016. Perceptually-based foveated virtual reality. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference, SIGGRAPH '16, Anaheim, CA, USA, July 24-28, 2016, Emerging Technologies*. ACM, CA, USA, 17:1–17:2. https://doi.org/10.1145/2929464.2929472
[52] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron E. Lefohn. 2016. Towards foveated rendering for gaze-tracked virtual reality. *ACM Trans. Graph.* 35, 6 (2016), 179:1–179:12. https://doi.org/10.1145/2980179.2980246
[53] Bruno Patrão, Samuel Pedro, and Paulo Menezes. 2020. How to Deal with Motion Sickness in Virtual Reality. In *22nd Portuguese Meeting on Computer Graphics and Interaction 2015*, Paulo Dias and Paulo Menezes (Eds.). The Eurographics Association, Coimbra, Portugal, 34–36. https://doi.org/10.2312/pt.20151201
[54] PS VR2. 2022. PlayStation VR2. https://www.playstation.com/en-us/ps-vr2/. [Online; accessed 2023-03-07].
[55] Alec Rohloff, Zackary Allen, Kung-Min Lin, Joshua Okrend, Chengyi Nie, Yu-Chia Liu, and Hung-Wei Tseng. 2021. OpenUVR: An open-source system framework for untethered virtual reality applications. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, IEEE, Virtual, 223–236.
[56] Yeongil Ryu and Eun-Seok Ryu. 2021. Overview of motion-to-photon latency reduction for mitigating VR sickness. *KSII Transactions on Internet and Information Systems (TIIS)* 15, 7 (2021), 2531–2546.
[57] Scikit. 2022. scikit-learn. https://scikit-learn.org/stable/. [Online; accessed 2022-10-20].
[58] Sensics. 2016. Time-warp Explained. https://medium.com/insights-on-virtual-reality/time-warp-explained-febea194ca87. [Online; accessed 2023-08-04].
[59] Ferdi A. Smit, Robert van Liere, Stephan Beck, and Bernd Fröhlich. 2009. An Image-Warping Architecture for VR: Low Latency versus Image Quality. In *IEEE Virtual Reality Conference 2009 (VR 2009), 14-18 March 2009, Lafayette, Louisiana, USA, Proceedings*. IEEE Computer Society, Louisiana, USA, 27–34. https://doi.org/10.1109/VR.2009.4810995
[60] Ferdi Alexander Smit, Robert van Liere, and Bernd Froehlich. 2010. A Programmable Display Layer for Virtual Reality System Architectures. *IEEE Trans. Vis. Comput. Graph.* 16, 1 (2010), 28–42. https://doi.org/10.1109/TVCG.2009.75
[61] UL Solutions. 2016. VRMark. https://store.steampowered.com/app/464170/VRMark/. Accessed: 2022-10-06.
[62] UL Solutions. 2020. Oculus Quest 2. https://www.meta.com/quest/products/quest-2/. [Online; accessed 2022-01-08].
[63] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. 2020. Simultaneous run-time measurement of motion-to-photon latency and latency jitter. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, IEEE, Virtual, 636–644.
[64] Steam. 2023. Steam. https://store.steampowered.com/. [Online; accessed 2023-03-03].
[65] Lightspeed Studios. 2016. NVIDIA®VR Funhouse. https://store.steampowered.com/app/468700/NVIDIA_VR_Funhouse/. [Online; accessed 2024-02-01].
[66] Sora Thompson. 2016. VR and The Resolution of The Human Eye. https://medium.com/@FreneticPony/virtual-reality-and-the-resolution-of-the-human-eye-5e601b0ef030. [Online; accessed 2022-10-14].

[67] Robert Toth, Jim Nilsson, and Tomas Akenine-Möller. 2016. Comparison of projection methods for rendering virtual reality. In *Proceedings of High Performance Graphics, HPG 2016, Dublin, Ireland, June 20-22, 2016*, David Luebke and Steven Molnar (Eds.). Eurographics Association, Dublin, Ireland, 163–171. https://doi.org/10.2312/hpg.20161202

[68] Valve. 2016. The Lab. https://store.steampowered.com/app/450390/The_Lab/. [Online; accessed 2024-02-01].

[69] J. M. P. van Waveren. 2016. The asynchronous time warp for virtual reality on consumer hardware. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology, VRST 2016, Munich, Germany, 2-4 November, 2016*, Dieter Kranzlmüller and Gudrun Klinker (Eds.). ACM, Munich , Germany, 37–46. https://doi.org/10.1145/2993369.2993375

[70] Corey Whelan. 2021. What Causes Virtual Reality (VR) Motion Sickness? https://www.healthline.com/health/vr-motion-sickness. [Online; accessed 2022-10-14].

[71] Wikipedia. 2022. Macroblock. https://en.wikipedia.org/wiki/Macroblock.

[72] Chenhao Xie, Xie Li, Yang Hu, Huwan Peng, Michael Taylor, and Shuaiwen Leon Song. 2021. Q-vr: system-level design for future mobile collaborative virtual reality. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, USA, 587–599.

[73] Chenhao Xie, Xingyao Zhang, Ang Li, Xin Fu, and Shuaiwen Song. 2019. PIM-VR: Erasing Motion Anomalies In Highly-Interactive Virtual Reality World with Customized Memory Cube. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Washington DC, USA, 609–622. https://doi.org/10.1109/HPCA.2019.00013

[74] Xinreality. 2020. Timewarp. https://xinreality.com/wiki/Timewarp. [Online; accessed 2023-08-04].

[75] Ming Yang. 2018. Avoiding pitfalls when using NVIDIA GPUs for real-time tasks in autonomous systems. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems*. IEEE, Barcelona, Spain, 20:1–20:21.

[76] Minxia Yang, Jiaqi Zhang, and Lu Yu. 2019. Perceptual tolerance to motion-to-photon latency with head movement in virtual reality. In *2019 Picture Coding Symposium (PCS)*. IEEE, IEEE, Ningbo, China, 1–5.

[77] Shulin Zhao, Haibo Zhang, Sandeepa Bhuyan, Cyan Subhra Mishra, Ziyu Ying, Mahmut T. Kandemir, Anand Sivasubramaniam, and Chita R. Das. 2020. Déjà View: Spatio-Temporal Compute Reuse for' Energy-Efficient 360 degree VR Video Streaming. In *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Valencia, Spain, May 30 - June 3, 2020*. IEEE, Valencia, Spain, 241–253. https://doi.org/10.1109/ISCA45697.2020.00030

A APPENDIX

A.1 MPD Variation in Similar Frames

Frames that are genuinely similar may experience sizeable MPD variations. There are two main reasons for this: extraneous factors like other tasks executing concurrently and a small but important "butterfly effect". Let us show a bunch of simulation results and correlated real-world data for a representative benchmark: Sponza.

A.1.1 Impact of Extraneous Factors: In this section, we shall discuss how the GPU workloads affect the MPD. Figure 20 shows the MPDs for three different refresh intervals. In the figure, we have four vertical lines, each representing a specific type of event. The red line shows when ATW is invoked, the blue line represents when preemption completes and the time warp operation starts, the green line shows when the time warp ends, and the grey line indicates when the display is refreshed. Ideally, we would want the red and green lines to be as close as the possible. To time lost in clearing the active queue (preemption latency) is the time interval between the red and the blue lines.

When ATW is invoked, the GPU treats it as a high-priority thread and starts preempting all currently running threads present in the active queue. These threads could belong to the rendering task of the primary VR application or could belong to any other applications that were also running on the GPU. We have incorporated a thread-level preemption model similar to the latest NVIDIA GPUs (refer to [43]), which means that every warp must be fully executed and the active queue needs to be emptied before starting the time warp operation. Since, GPUs typically execute thousands of threads concurrently to achieve high throughput, and each thread may have a different execution time, the preemption latency becomes large and unpredictable. The most important variable here is the number of semi-executed warps that are waiting to finish. They may be delayed for a lack of access to GPU's computational resources or memory values. Examples of such threads that pose such problems are as follows (as seen in our experiments): threads performing vertex transformations, computation of lighting effects (ambient, diffuse, specular, and other forms of lighting), etc. Recall that we can run the time-warping kernel only after the warps have fully finished executing because of intra-warp context switching is still not there in NVIDIA GPUs.

In Figure 20, we can see that the interval between the red and blue lines varies across all the three cases, while the blue and green lines remain equally spaced. This indicates that the time warp latency remains almost *constant* across frames while the preemption latency varies. Moreover, the rate at which the active queue empties differs. This variation is correlated with the workload on the GPU at the time of ATW initiation. For instance, Case 3 had fewer active threads, which completed faster. However, in the first two cases, the total number of active warps was large, resulting in a more extended preemption latency as compared to Case 3. But, there is a difference in the two latencies because it depended upon which threads were getting executed and the nature of work they were doing. A lot of this is captured by our decision tree.

A.1.2 Butterfly Effect. Next, we show the butterfly effect and illustrate how small changes in the scheduling of threads can lead to very different ATW latencies (time from the red line (ATW start) to the green line).

The warp scheduler works as follows (reference:[15, 35]). If threads get free SMs; they are moved to the active queue and start getting executed while the rest of the threads wait in the scheduling queue. We deliberately shuffled the order of threads in the scheduling queue 1000 clock cycles before invoking the ATW kernel for Case 1. This had the effect of altering

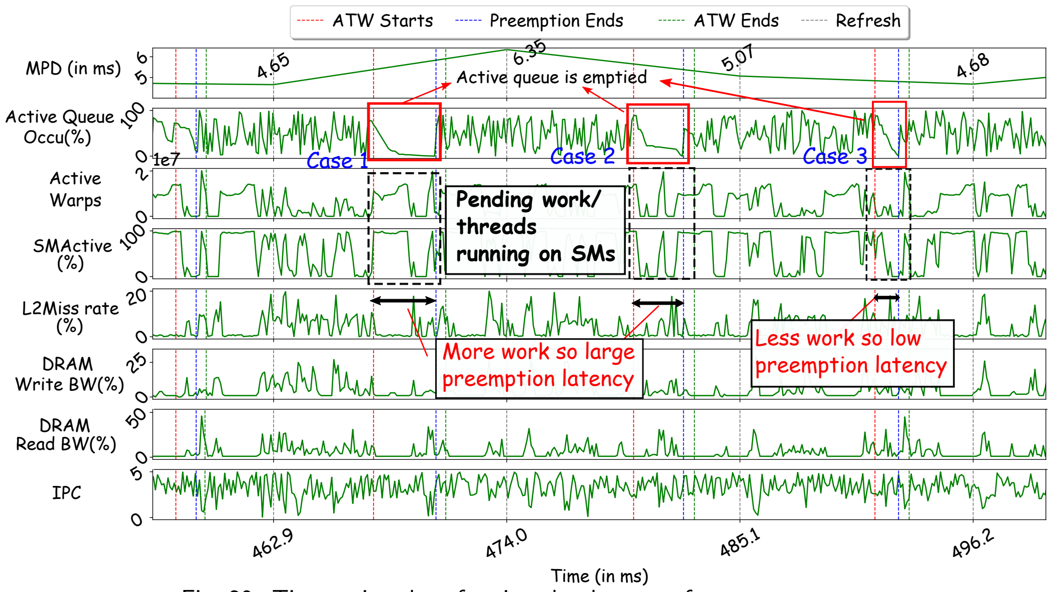


Fig. 20. Time-series plot of various hardware performance counters

their order within the active queue. In Figure 21, we can see that this change resulted in a significant reduction in the preemption latency, which decreased from 2.54 ms to a mere 1.1 ms. We repeated the process of changing the order of threads again, and in Figure 22, we can see that this time, the preemption latency increased significantly to 4.1 ms.

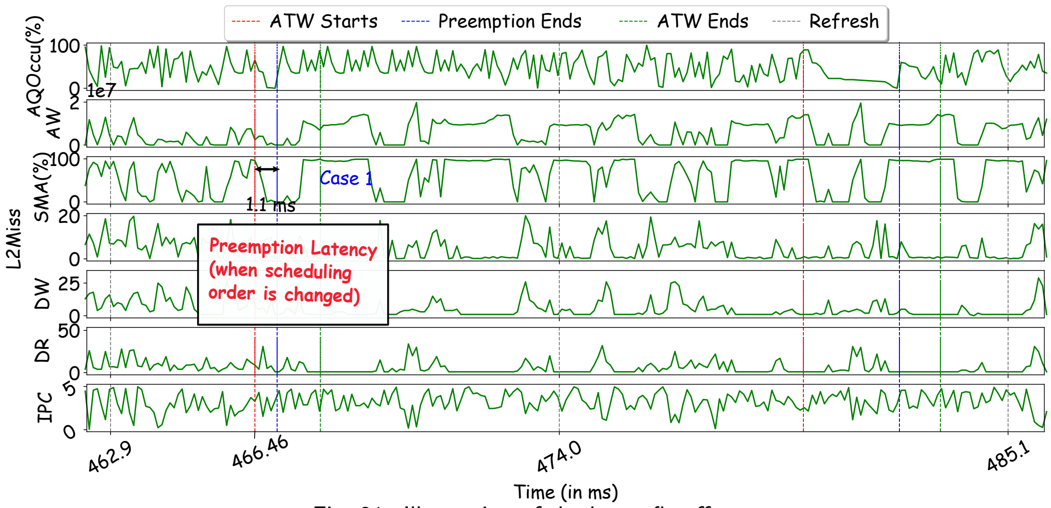


Fig. 21. Illustration of the butterfly effect

A.1.3 Comparison with Real-world Data. We were interested to investigate whether such effects are seen in real-world devices as well. A skeptical reader can always point out that this is an artefact of our simulation. The variance in the ATW latency is evident across benchmarks in real-world data too. We captured this information using the Oculus Debug

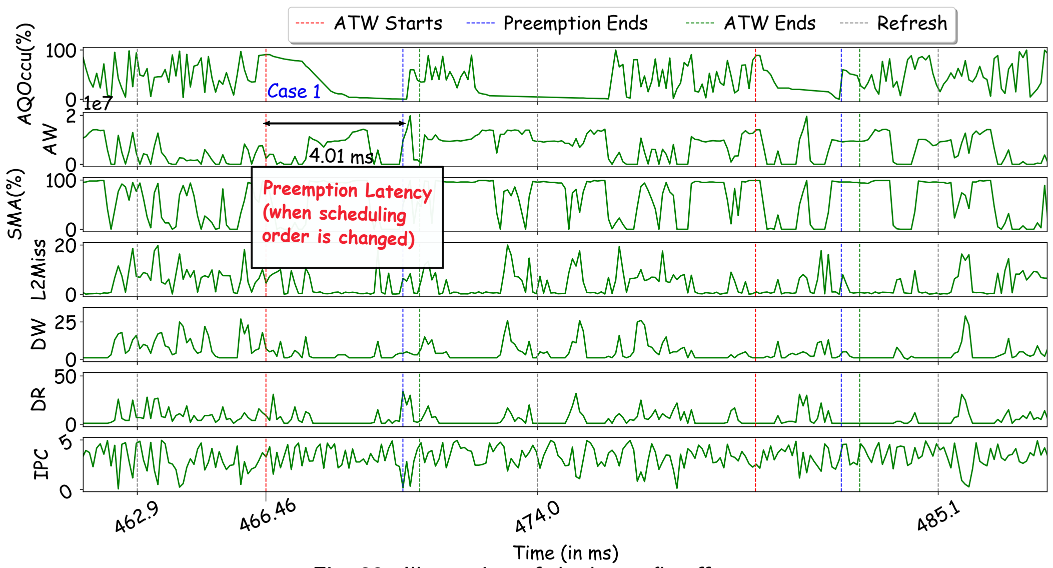


Fig. 22. Illustration of the butterfly effect

Tool. We measured this variance by calculating the standard deviation of the latency for 1000 consecutive frames – that were similar – and then normalizing it to the average latency (refer to Figure 23). We plotted the ATW latency for 200 consecutive refresh intervals for one of the benchmarks (refer to Figure 24). The results reconfirm that *even when* the consecutive frames are similar, the latency fluctuates significantly.

A similar time series plot is shown for Oculus 2 data in Figure 25. We can clearly see the same trends. Note the distance between the red (ATW start) and the blue (preemption end) lines. They show a degree of variance that we also saw in our simulated data. This clearly shows the effect of contention. Given that this data was collected at a different time instant, the absolute figures will not match; however, we should focus on the trends.

This proves that our simulation results mimic real-world effects.

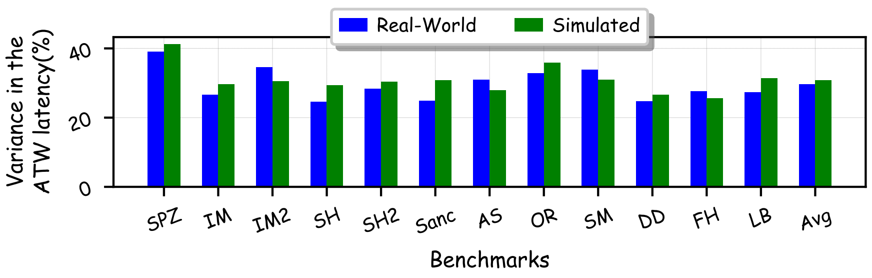


Fig. 23. Normalized variance in the ATW latency (note the similarity between simulated and real-world data)

A.2 Correlation of HPCs with the MPD

Since we have a time series plot for the MPD and all the activities happening on the GPU, we can easily find the correlation coefficient between the MPD and all the microarchitectural

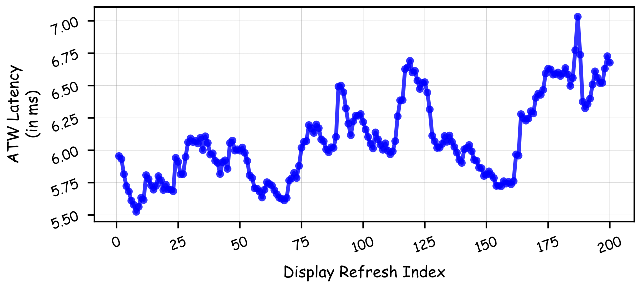


Fig. 24. The atw latency across similar frames captured from Oculus Quest 2

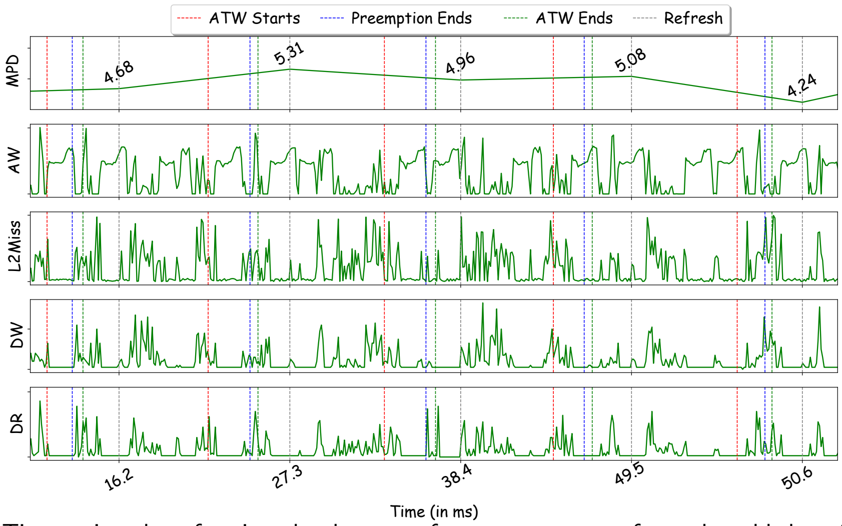


Fig. 25. Time-series plot of various hardware performance counters for real-world data (normalized values)

parameters. Figure 26 shows that active SMs, warps, and instructions are the top three highly correlated features with the MPD across benchmarks. Hence, these features are the ideal choice for predicting the MPD. However, other features also have a decent correlation coefficient, and they can also be considered for the prediction.

A.3 Effect of the PID Controller on Prediction Errors

To investigate the cause of the reduction, we analyzed the histograms of the prediction errors both before and after applying the PID controller for each benchmark. We can find some of these histograms in Figures 27 and 28. One point that is evident from these figures is that the feedback component reduces the tails of the histograms, which means the range of the prediction error is minimized. Also, it makes the prediction error symmetrical around zero. For instance, in the case of the *Sponza* benchmark, the prediction error initially ranged from -1.622 to 1.730 ms, but after applying the PID controller, it was reduced to -0.695 to 0.583 ms.

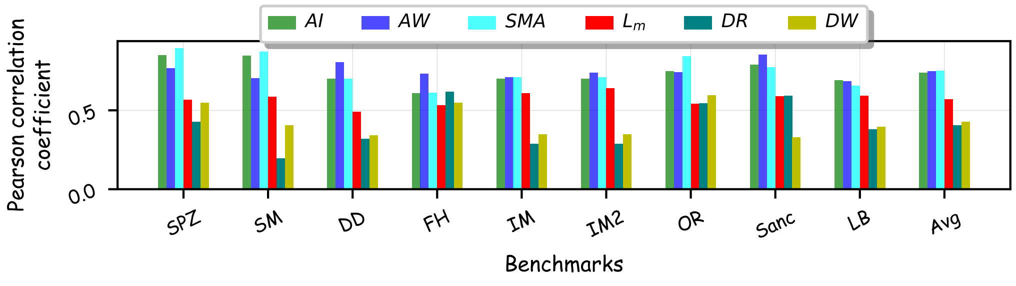
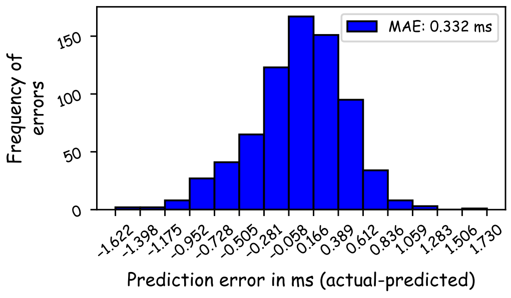
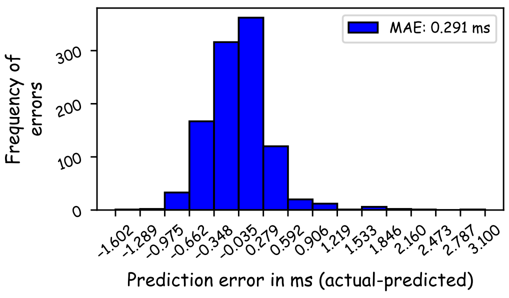


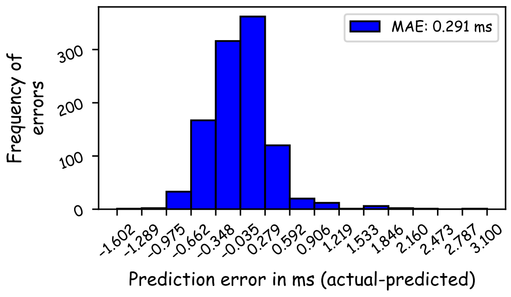
Fig. 26. Correlation of the identified micro-architectural parameters with the MPD



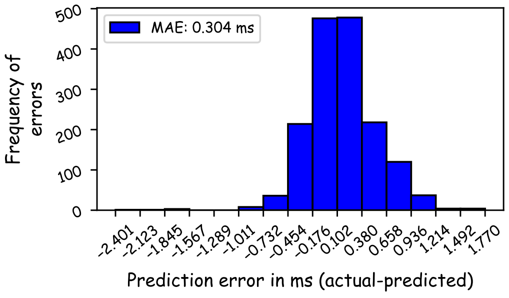
(a) Sponza



(b) InMindVR



(c) InMindVR 2

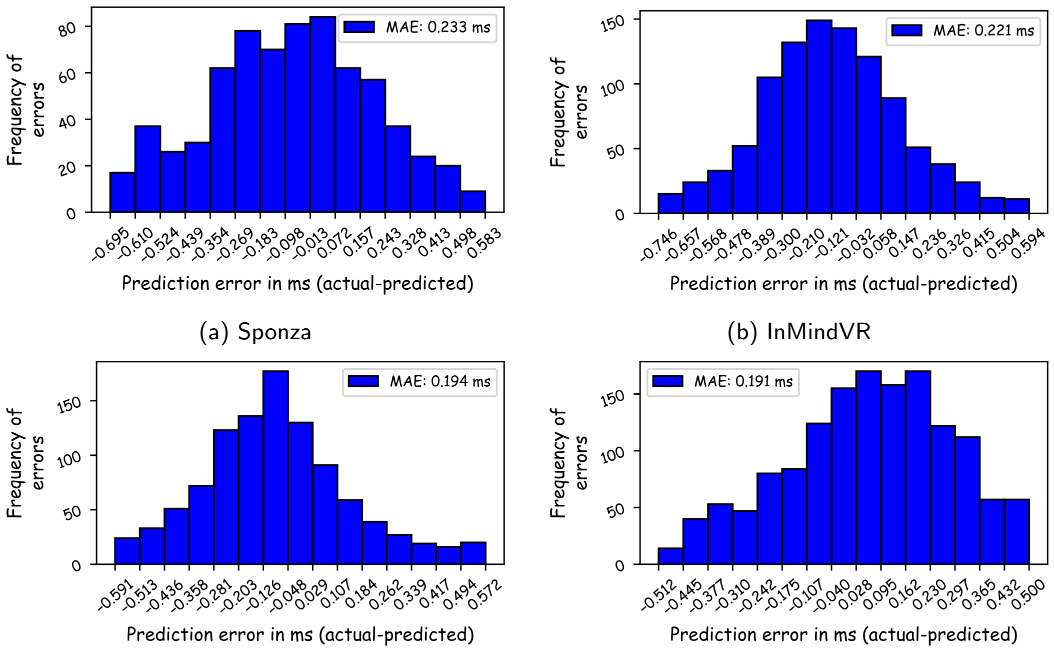


(d) Sanctuary VR

Fig. 27. Histogram of the prediction errors without the PID controller

A.4 Importance of MPD

We have invested a fair amount of effort in showing that even small changes to the MPD have a substantial effect on the quality of experience. Please refer to the videos at the following link [18]. The baseline video has zero additional MPD and there is a sequence of videos with an additional MPD of 1 ms, 2ms, 4ms and 8 ms, respectively. Even though these videos were originally made for HMDs, we can still see that on a monitor the quality difference is quite perceptible. The effect is far more *pronounced* on HMDs. In fact, even for the 1 and 2 ms cases, it is so bad that it induced a significant amount of nausea to one of our older team members. The JND (just noticeable delay) in the case of HMDs is much lower than



(a) Sponza

(b) InMindVR

(c) InMindVR 2

(d) Sanctuary VR

Fig. 28. Histogram of the prediction errors with the PID controller

1 ms as we found out in our in-vivo experiments and thus the need to reduce the MPD is very important. The effect of the MPD on the visual quality of experience and the overall well-being has been documented in the following references [13, 53, 56, 70].