RELAXR: Real-time Execution Framework for Latency-aware AI-native Extended Reality

Ziliang Zhang, Hyoseung Kim, Cong Liu
University of California, Riverside
{zzhan357, hyoseung, congl}@ucr.edu

***Abstract*—AI-native Extended Reality (AIXR) glasses are emerging as interactive real-time systems that combine motion-aligned rendering with scene-grounded AI feedback. Users can query about objects in view and receive visual grounding and language guidance directly on the rendered 3D frames. Existing XR research primarily optimizes XR-related latency to prevent motion sickness, but AIXR also requires timely AI feedback: late grounding or language responses may no longer match the user's view or speech context.**

To the best of our knowledge, our work is the first to define AI feedback latencies and analyze their soft real-time requirements for AIXR. We define camera-to-visual grounding (C2VG) and speech-to-LLM response (S2LLM) to quantify visual grounding and language response latencies. Our profiling shows that co-running XR and AI feedback pipelines on a shared GPU causes kernel interleaving, where runtime variation in either pipeline can amplify latencies in the other. Although server-based temporal isolation is a natural starting point, existing schemes are insufficient for AIXR because they suffer from static reservation failures, dynamic rendering workloads, and spatial or temporal waste.

We propose RELAXR, a runtime framework that manages the XR and AI feedback pipelines as jointly scheduled soft real-time workloads. RELAXR combines GPU-aware resource reservation, which provides temporal isolation and bounds AI-induced blocking of XR rendering; Renderer-Aware Budget-Period Adaptation, which handles runtime rendering changes; and pipeline reconstruction, which reduces resource waste during AI pipeline execution. Evaluations across PC-laptop and embedded platforms, controlled HOT3D scenes, and real-world experiments show that RELAXR reduces C2VG by up to 36.4% and S2LLM by up to 56.4%, while keeping XR-related latencies within 2.8% of the XR-only baseline. We open-source RELAXR to support the real-time systems community in building practical low-latency AIXR systems.

I. Introduction

AI-native Extended Reality (AIXR) glasses emerge as a new class of real-time wearable systems that bring together motion-aligned 3D rendering with scene-grounded AI perception and language interaction. Recent AIXR devices have been deployed across personal assistance, spatial computing, industrial design, driving and aviation guidance, and real-time translation and transcription [1–6].

Beyond rendering 3D virtual content at a target frame rate, AIXR devices interpret the user's egocentric environment, reason over speech, and return scene-grounded visual and language feedback to the user. To achieve this, AIXR devices generally run two pipelines. The *XR pipeline* continuously processes camera (CAM) and inertial measurement unit (IMU) inputs, estimates user motion, and renders 3D frames at the target frame rate. The *AI feedback pipeline* starts when the user speaks. It transcribes the speech, selects the latest CAM frame as the visual reference, extracts scene features to identify relevant objects for grounding, and combines this scene context with the transcript to generate an on-device LLM response. On-device execution is preferred for data privacy and remains feasible because the lightweight conversational tasks common on XR glasses can be served efficiently by on-device models [7–10]. Both pipelines impose user-visible timing requirements that must be met concurrently on a shared GPU, which is the central real-time challenge this paper addresses.

Existing research focuses on the responsiveness of the XR pipeline, typically through Motion-to-Display (M2D) and Camera-to-Display (C2D) latencies. These XR-related latencies measure how quickly IMU and CAM updates are reflected on the display, and they are critical for mitigating motion sickness [11–14]. However, AIXR introduces another user-visible soft real-time requirement: timely AI feedback. Unlike offline AI queries, AI feedback in AIXR is useful only if it remains relevant to the user's current view and speech. This time-dependent relevance means that AI feedback latencies should remain bounded relative to the resources reserved for the AI feedback pipeline, rather than grow arbitrarily under XR contention. Existing AIXR systems often treat AI feedback components as background execution [15–18]. This choice can preserve XR responsiveness, but it can also make AI feedback arrive after the interaction context has changed. For example, during the 2025 Meta Connect conference, an LLM response arrived late enough that the user repeated the same question, causing the model to skip the intended first step [19]. Therefore, AI feedback latencies are pivotal in AIXR systems because unbounded delay can break the interaction even when the XR pipeline remains responsive.

To quantify visual feedback latency, we define camera-to-visual grounding (C2VG) as the time from the sampling time of the CAM frame to displaying its grounded object mask on the head-mounted display (HMD). We define speech-to-LLM response (S2LLM) as the time from the end of user speech to displaying the LLM response generated from the speech and the corresponding CAM context. An ideal AIXR system should keep M2D and C2D close to conventional XR execution while minimizing C2VG and S2LLM during AI feedback. Meeting this goal is challenging because AIXR places two resource-demanding pipelines on the same GPU: periodic,

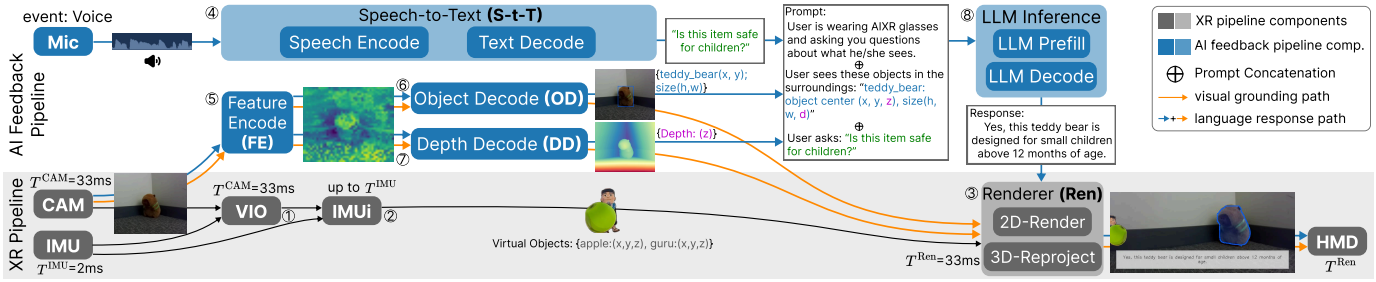


Fig. 1: A general AIXR system overview.

motion-driven XR rendering and aperiodic, speech-triggered AI feedback whose performance decays as the interaction context changes. When they run concurrently, kernels from separate streams interleave on one GPU timeline [20, 21], and runtime variation in either pipeline can interfere with the other. CPU-level priority is insufficient for this problem because contention occurs on the GPU after kernel launch, where preemption granularity is limited. Server-based isolation is a natural first step, but existing schemes still expose five AIXR-specific failures, including budget overuse (C1), budget drop and late replenishment (C2), starvation in low-priority servers (C3), rendering time surge (C4), and spatial or temporal waste (C5).

Contributions. We present **RELAXR**: a Real-time Execution framework for Latency-aware AI-native XR, which makes the following contributions.

- **A GPU-aware server design with bounded XR blocking.** RELAXR coordinates the XR and AI feedback pipelines through a GPU-aware server design that provides temporal isolation on the shared GPU timeline. It provides three server rules to address C1-C3 and bounds AI-induced blocking of an XR renderer execution.
- **A runtime budget-period adaptation mechanism.** Building on this server design, RELAXR addresses C4 by updating the servers' budget and period according to estimated renderer demand, so the reservation remains aligned with runtime rendering variation.
- **An AIXR-specific pipeline reconstruction.** Beyond server-level control, RELAXR addresses C5 by reconstructing the AI feedback pipeline to exploit the early availability of the reference CAM frame and heterogeneous GPU occupancy across AI layers, effectively reducing C2VG and S2LLM.
- **Open-source AIXR platform for follow-up research.** We release RELAXR as an open-source AIXR runtime that exposes its server design, adaptation, and pipeline reconstruction as modular components on top of a prototype AIXR system, enabling follow-up real-time scheduling and resource-management studies on a realistic AIXR workload.

We evaluate RELAXR on both PC-laptop and embedded-device platforms against five baselines and one ablation variant, using controlled HOT3D scene benchmarks [22] and a comprehensive real-world study spanning four daily-use scenarios across indoor/outdoor and stationary/mobile settings. In controlled scenes, RELAXR reduces all four user-visible latencies (M2D, C2D, C2VG, and S2LLM) by up to 56.4%, while incurring at most 0.12% runtime overhead. Crucially, RELAXR keeps XR-related latencies within 2.8% of the XR-only baseline, empirically consistent with our derived blocking bound and showing that AI feedback can be accelerated without sacrificing XR responsiveness. In real-world evaluation, where we measure whether each AI response is still contextually valid when it reaches the user, RELAXR maintains a 52.7% successful response rate at the highest prompt frequency in a stationary indoor scenario, while reducing average C2VG by 57.9% and average S2LLM by 30.1% over the existing baseline. These results show that RELAXR is effective across hardware platforms, controlled scene benchmarks, and real-world scenarios, and that lower AI feedback latencies help responses remain relevant to the user's current interaction context.

II. Background

A. AIXR System Pipelines

Fig. 1 abstracts a common framework in recent AIXR and AI-glasses systems [15, 16, 18, 23]. Unlike traditional XR systems that only render virtual content, AIXR systems interpret the user's egocentric scene and return scene-grounded visual and language feedback. In the Fig. 1 example workflow, the user looks at a teddy bear and asks, "Is this item safe for children?" The system anchors the query to the latest CAM frame, identifies the relevant objects, highlights them on the HMD, and generates an LLM response. Based on this workflow, we identify two pipelines across existing AIXR systems: the *XR pipeline* and the *AI feedback pipeline* [15, 16, 18, 23].

The *XR pipeline* maintains motion-aligned 3D frame outputs. As shown by the gray region in Fig. 1, CAM and IMU samples feed visual-inertial odometry (① VIO), which estimates the user's raw pose at the camera period T^{CAM} [24]. Between CAM updates, IMU integration (② IMUi) fuses high-frequency inertial samples at T^{IMU} to update the pose immediately before rendering [25]. The renderer (③) then runs every display period T^{Ren} with a 2D-render and a 3D-reprojection to generate the final 3D frames [26–28]. VIO and IMUi run on the CPU, while the renderer is GPU-dominant, making XR responsiveness sensitive to shared-GPU contention [14, 20].

The *AI feedback pipeline* is speech-triggered and scene-grounded. At the start of the k-th user speech, it selects the latest CAM frame as the visual reference. Speech-to-text (④ S-t-T) transcribes the audio [29], while the feature encoder

(⑤ FE) extracts visual features from the reference frame [30]. The object decoder (⑥ OD) extracts object semantics and 2D locations, and the depth decoder (⑦ DD) estimates depth for 3D-anchored grounding [8, 31, 32]. Once OD and DD finish, a renderer instance generates the visual grounding mask, allowing the user to verify the intended objects. Since the renderer maintains mask alignment using the latest pose fused from IMUi, FE, OD, and DD do not need to rerun for every frame output. Since the visual grounding mask is the first AI feedback displayed to the user, we define the k-th user speech triggered *visual grounding path* Γ_k^{visual}, shown by the orange arrows in Fig. 1, as follows:

$$\Gamma_k^{\text{visual}} = J_k^{\text{FE}} \rightarrow (J_k^{\text{OD}} \parallel J_k^{\text{DD}}) \rightarrow J_i^{\text{Ren}}$$

Here, J_k^{FE}, J_k^{OD}, and J_k^{DD} denote the FE, OD, and DD execution instances triggered by the k-th user speech, respectively. The operator $\parallel$ indicates that OD and DD can execute in parallel after FE. J_i^{Ren} denotes the i-th renderer instance, which generates the visual grounding mask using the OD and DD outputs started with the k-th speech event.

The transcript, object semantic labels, 2D locations, and depth information are then concatenated into the scene-grounded prompt for LLM inference (⑧). After the LLM response is generated, the next immediate renderer instance displays it as a dialogue box on the HMD. This defines the k-th *language response path* Γ_k^{lang}, shown by the blue arrows in Fig. 1:

$$\Gamma_k^{\text{lang}} = \left[J_k^{\text{STT}} \parallel \left(J_k^{\text{FE}} \rightarrow J_k^{\text{OD}} \parallel J_k^{\text{DD}}\right)\right] \rightarrow J_k^{\text{LLM}} \rightarrow J_i^{\text{Ren}}$$

J_k^{STT} and J_k^{LLM} are the S-t-T and LLM inference instances triggered by speech k, respectively. All other notation follows the same meaning as in Γ_k^{visual}. Because the AI feedback components execute deep neural models [8, 29–32], these components are GPU-dominant. Within each AI feedback component, layers are released sequentially when their preceding layer finishes. Although the $(k{+}1)$-th user speech may arrive before the k-th query completes, queries are never executed concurrently: the AI feedback pipeline delays the new query until the previous one finishes.

B. Latency Metrics

Conventional XR systems mainly evaluate whether displayed frames reflect the most recent user motion. We adopt two standard XR-related latency metrics from prior work [13, 14, 33].

Def. 1 (Motion-to-Display [14, 33])**.** The motion-to-display latency (M2D) is defined as the time from the latest motion sampled by the IMU to its display on the HMD. In Fig. 1, M2D is the time to complete IMUi→renderer sequence triggered by the latest IMU sample.

Def. 2 (Camera-to-Display [14])**.** The camera-to-display latency (C2D) is defined as the time between the latest motion captured by CAM and its corresponding display on the HMD. In Fig. 1, C2D is the time to complete VIO→IMUi→renderer sequence triggered by the latest CAM frame.

Together, M2D and C2D capture user-perceivable motion misalignment in XR sessions [12–14, 33, 34].

AIXR introduces another class of user-visible latency: whether AI-generated feedback still matches the user's current interaction context when it reaches the HMD. As discussed in §II-A, the AI feedback pipeline first displays 3D-anchored object masks generated from the reference CAM frame. If this visual grounding arrives late, the referenced object or scene may have changed, causing the mask to no longer indicate the intended object. Following reaction-time semantics for cause-effect chains [35, 36], we treat the selected CAM frame as the external update and the displayed grounding mask as the first corresponding system output. For the k-th user speech that triggers the AI feedback pipeline at t_k^{start} and the corresponding k-th FE instance that arrives at arrival time a_k^{FE}, we define the following visual feedback latency metric.

Def. 3 (CAM-to-Visual Grounding Latency)**.** The CAM-to-visual grounding latency (C2VG) is defined as the reaction time of the visual grounding path Γ_k^{visual}:

$$\text{C2VG}_k = \left(a_k^{\text{FE}} - t_k^{\text{start}}\right) + T^{\text{Ren}} + \sum_{j \in \Gamma_k^{\text{visual}}} \left(f_k^j - a_k^j\right)$$

where T_{ren} is the renderer period, and a_k^j and f_k^j are the arrival and finish times of the j-th instance of each component in Γ_k^{visual}. It measures the time from the start of the k-th user speech, when the AI feedback pipeline selects the latest available CAM frame sampled at t_k^{start} as the visual reference, to the displayed object mask on the HMD. Here, k indexes the k-th user speech that triggers the AI feedback pipeline and j indexes a released runtime instance on the visual grounding path (Γ_k^{visual}). For AI feedback components, an instance denotes a layer execution; for the XR pipeline, the relevant instance on Γ_k^{visual} denotes the renderer execution that displays the visual grounding mask. Within an AI feedback component, the next layer starts when the previous layer finishes, so $f_k^j = a_k^{j+1}$. Since the AI feedback pipeline is triggered by the user speech and the CAM frame is sampled periodically, the first term captures the delay from the CAM frame to the release of the first FE layer instance. The term T^{Ren} accounts for the display period necessary for a ready visual grounding mask to become visible through a renderer execution.

Lower C2VG allows the user to verify earlier whether the system has grounded the query to the intended object. Prior XR work shows that system delay can cause temporal misalignment between real and virtual content, and related visual-update studies report perceptual tolerance around the 50-70 ms range [37–40].

After the visual grounding mask is displayed, the user waits for the language response produced along Γ_k^{lang}. Because this path includes both upstream context generation and the renderer update that makes the response visible, LLM inference time alone does not capture user-perceived language-feedback latency. We therefore define S2LLM using the same reaction-time convention as C2VG.

TABLE I: Mean metrics for AIXR (max output 20 tokens).

Metrics (ms)	XR Only	AI Feedback Only	XR (No Priority) + AI (No Priority)	XR (High Priority) + AI (Low Priority)
M2D	17.91	N/A	**23.59**	22.33
C2D	36.53	N/A	45.33	**45.90**
C2VG	N/A	43.16	87.40	**87.60**
S2LLM	N/A	247.18	349.39	**352.58**

* Bold numbers indicate the worst value across all four configurations.

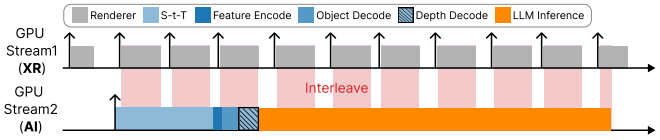


Fig. 2: GPU interleaved execution between XR and AI.

Def. 4 (Speech-to-LLM Response Latency)**.** The speech-to-LLM response latency (S2LLM) is defined as the reaction time of the language response path Γ_k^{lang}:

$$\text{S2LLM}_k = \left(a_k^{\text{STT}} - t_k^{\text{end}}\right) + T^{\text{Ren}} + \sum_{j \in \Gamma_k^{\text{lang}}} \left(f_k^j - a_k^j\right).$$

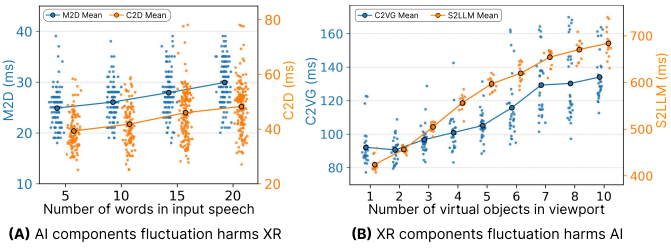
It measures the time from the end of the k-th user speech at t_k^{end} to the displayed LLM response on the HMD. Using the same symbols and indexing conventions as C2VG, a_k^j and f_k^j denote the arrival and finish times of runtime instance j on Γ_k^{lang}, which begins with the STT instance arriving at a_k^{STT}. Compared with C2VG, the first term captures the delay from speech completion to the release of the first S-t-T runtime instance. The term T^{Ren} accounts for the display-period wait needed for a ready LLM response to become visible through a renderer execution.

S2LLM captures the end-to-end reaction time from the completed user speech to the first completed language feedback. Because AIXR feedback is conversational and visually situated, S2LLM should remain at the sub-second level for the response to feel connected to the user's just-completed query [41, 42].

III. Motivation

A. Co-running XR and AI Feedback Pipelines

To understand the impact of co-running the XR pipeline and the AI feedback pipeline on the same XR device, we profile the AIXR system in Fig. 1 using the metrics in §II-B on a PC laptop with an NVIDIA RTX 3050 GPU, with more details provided in §V. We compare four configurations: XR-only, AI-feedback-only, co-running without priority assignment, and co-running with XR assigned a higher CPU priority using `renice` [43]. As shown in Tab. I, co-running degrades all four metrics, with C2D increasing by 25% and C2VG increasing by 103% over their single-pipeline baselines. Assigning higher CPU priority to XR provides little benefit: M2D decreases by only 5%, while C2D slightly increases by 1% compared with no-priority co-running. Such degradation happens because the XR and AI feedback pipelines run as separate



(A) AI components fluctuation harms XR **(B)** XR components fluctuation harms AI

Fig. 3: Effect of XR and AI feedback components.

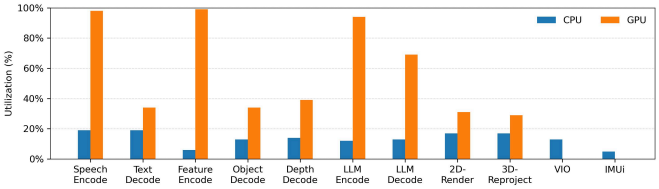


Fig. 4: Peak CPU and GPU utilization of each component.

processes, so their GPU kernels are submitted from separate CUDA contexts. At the GPU level, these CUDA contexts are time-sliced on the shared GPU timeline, which creates the interleaved execution shown in Fig. 2. Since this GPU scheduling does not inherit the CPU-side priority, `renice` cannot prevent the AI feedback components from time-sharing with the XR renderer [20, 21], resulting in significant GPU-level interference.

Beyond co-running interference, runtime variation in either pipeline can further degrade the performance of the other, making static priority control even less reliable. In the AI feedback pipeline, longer user speech extends S-t-T and LLM inference. This keeps the AI feedback's CUDA context active for a longer interval and increases the amount of GPU work from AI feedback pipeline that competes with XR renderer execution. Hence, this contention increases renderer response time and therefore M2D/C2D, as shown in Fig. 3(A). In the XR pipeline, more virtual objects in the user's view increase rendering time. As rendering demand grows, the renderer consumes more GPU time in each display period, leaving less GPU time for the AI feedback pipeline and increasing C2VG/S2LLM, as shown in Fig. 3(B).

B. Challenges of Building a Server

The results above motivate separate servers for performance isolation between XR and AI feedback pipelines. However, a CPU-only server cannot isolate the dominant contention in AIXR. As shown in Fig. 4, subcomponent-level GPU utilization reaches 98%, whereas CPU demand remains at most 19% per component and 90% peak utilization even on one CPU core. Therefore, an AIXR server must coordinate CPU and GPU execution together.

Because commodity GPU drivers do not provide fine-grained control over preemption, we employ a *server-managed cooperative scheduling* approach. Each renderer instance or AI layer is treated as a non-preemptive GPU segment. Once started, a segment runs to completion, and the server can

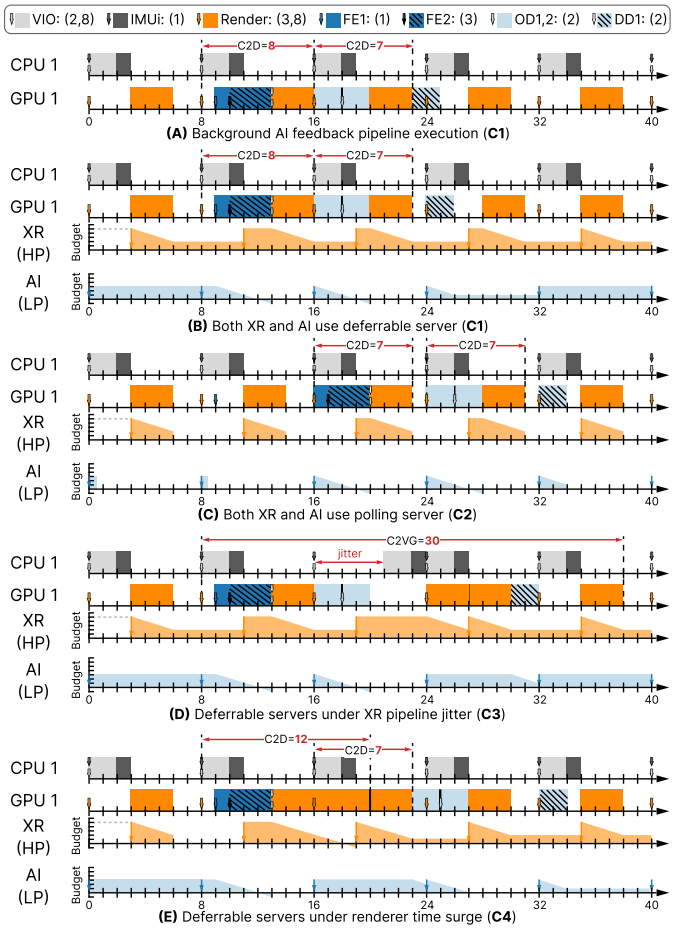


Fig. 5: Timelines under different execution and servers.

reschedule only at segment boundaries. This creates controllable preemption points without modifying the GPU driver. Under this abstraction, we examine three natural server choices: (i) *background AI execution* that allows AI segments to utilize all GPU time remaining after use by XR; (ii) the *polling server* policy that is the simplest form of reservation that does not introduce additional interference; and (iii) the *deferrable server* policy that can yield better responsiveness by preserving unused budget until the period ends.

To understand the challenges associated with the scheduling and server choices above, we instantiate them on the profiled AIXR workload in Fig. 5. Using the aforementioned cooperative scheduling approach, each renderer instance and each AI layer is treated as a GPU segment. The timelines show the nominal execution time of each XR component and the layer-wise nominal execution time of each AI feedback component (FE1-2, OD1-2, DD1). We assign XR segments higher priority than AI segments and follow the BOXR system architecture [14] in which the renderer starts strictly after IMUi completes.

C1: *Budget overuse*. A low-priority (LP) GPU segment can execute beyond its remaining budget and occupy GPU time reserved for high-priority (HP) segments. This happens because GPU segments are non-preemptive once dispatched: even if an HP XR renderer becomes ready, the runtime cannot interrupt an already-running LP AI segment until that segment completes. Fig. 5(A) shows the same failure under background AI execution, where LP FE1 blocks the HP XR renderer. Fig. 5(B) shows that a deferrable server does not eliminate this problem: if the AI feedback server dispatches a segment whose execution time exceeds the remaining AI budget, the segment can drive the AI budget negative and delay XR renderers in later periods, inflating M2D and C2D.

C2: *Budget drop and late replenishment*. An LP segment can be delayed when unused budget is discarded before the segment becomes ready, forcing it to wait for a later replenishment. Fig. 5(C) illustrates this issue with a polling server, where the AI budget is dropped in period two, so FE1 execution is delayed until the beginning of period three.

C3: *LP segment starvation*. Runtime jitter can shift HP segments closer to future arrivals, leaving no valid GPU time for ready LP segments to execute. As illustrated in Fig. 5(D), even when no HP segment is currently running, the available GPU time cannot be safely given to DD1 because it is held by the HP XR server for consecutive renderer segments, delaying DD1 execution and inflating C2VG. This issue arises because existing servers do not account for the next HP renderer release: using the available GPU time for DD1 may overlap with that release and delay the renderer, while reserving it for XR leaves DD1 waiting. Using a sporadic server for XR does not solve this problem, because delaying XR replenishment can also delay renderer dispatch and increase M2D/C2D.

C4: *Renderer-time surge*. Static budgets and periods become invalid when renderer demand increases at runtime. Fig. 5(E) shows this case, where more virtual objects in view increase renderer execution time, exhaust the fixed XR budget, and inflate C2D.

C5: *Spatial or temporal waste*. Beyond C1-C4, temporal isolation alone does not make AI feedback efficient. Unlike C4, which concerns adapting XR reservation to dynamic renderer demand, C5 concerns problems inside the AI feedback pipeline after resources are isolated. As shown in Fig. 6, temporal waste occurs because the reference CAM frame is available at the start of the speech, but existing sequential pipelines still wait for the user's speech to complete before starting visual grounding and scene-context preparation, inflating S2LLM [15, 16]. Spatial waste occurs because AI feedback components have different GPU occupancy, leaving unused thread-block capacity when low-occupancy components execute alone [44].

Overall, C1-C3 are fundamental server failures because existing policies cannot preserve HP reservations or provide safe LP execution under segment-level GPU control. C4-C5 are critical inefficiencies because static reservations and isolated AI execution cannot adapt to dynamic renderer demand or fully utilize temporal and spatial resources inside the AI feedback pipeline.

IV. RELAXR Server and Pipeline Reconstruction

A. Overview

We present **RELAXR**, a runtime framework for coordinat-

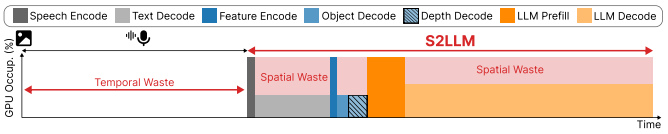


Fig. 6: Temporal and spatial waste in AI feedback pipeline.

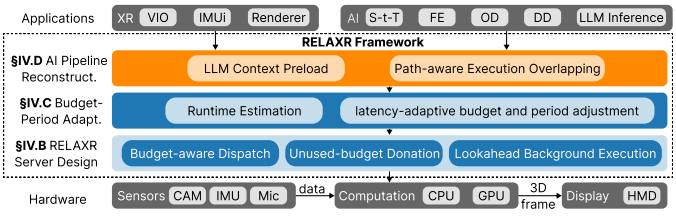


Fig. 7: RELAXR design overview.

ing the XR pipeline and the AI feedback pipeline on a shared GPU. RELAXR treats AIXR scheduling as a joint CPU-GPU reservation problem. To control the interference from §III-A, RELAXR uses a GPU-aware hierarchical server through CPU-side dispatch control. This is applicable because AIXR is a soft real-time scenario, and AI feedback pipeline execution can be regulated at component or layer boundaries without requiring fine-grained GPU control.

As shown in Fig. 7, RELAXR consists of three layers. The server design layer targets C1-C3 by enforcing temporal isolation on the shared GPU timeline. *Budget-aware Dispatch* addresses budget overuse (C1) by preventing LP GPU segments from starting under insufficient remaining budget. *Unused-budget Donation* addresses budget drop and late replenishment (C2) by allowing the HP server to donate unused budget to the LP server. *Lookahead Background Execution* addresses LP segment starvation (C3) by using background execution windows before a future XR renderer arrival to run selected AI feedback components without delaying XR components.

The budget-period adaptation layer targets renderer-time surge (C4). *Renderer-Aware Budget-Period Adaptation* estimates renderer demand online from viewport-derived signals and adjusts the server budget and period accordingly. This prevents a static reservation from becoming invalid when renderer time changes, while preserving execution opportunities for the AI feedback pipeline.

The AI feedback pipeline reconstruction layer targets spatial or temporal waste (C5). *Path-aware Execution Overlapping* reduces spatial waste by overlapping ready AI layers when the GPU has unused capacity and prioritizing layers on the visual grounding or language response paths. *LLM Context Preload* reduces temporal waste by preparing scene context during speech, allowing post-speech LLM inference to reuse cached context and reduce S2LLM. Together, these layers preserve XR responsiveness while bounding C2VG and S2LLM.

B. RELAXR Server Design

System Model. We formalize the AIXR workload from §II-A under the cooperative scheduling abstraction in §III-B. The system runs on one CPU core and one on-device GPU that launches XR and AI feedback pipelines in separate CUDA

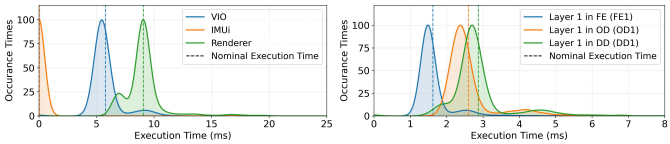


Fig. 8: Nominal execution times profiling.

contexts. The workload consists of a set of *GPU segments* $\mathcal{G} = \mathcal{G}_{\mathrm{XR}} \cup \mathcal{G}_{\mathrm{AI}}$, where a segment ℓ is the smallest non-preemptive dispatch unit: one renderer instance in $\mathcal{G}_{\mathrm{XR}}$ or one AI layer instance in $\mathcal{G}_{\mathrm{AI}}$. XR segments are released periodically every display period T^{Ren}, while AI segments are released aperiodically with the k-th user speech and follow the precedence of $\Gamma_k^{\mathrm{visual}}$ and Γ_k^{lang} defined in §II-B. Once dispatched, a segment runs to completion, so scheduling can occur only at segment boundaries.

Since exact worst-case execution times are difficult to obtain for all GPU segments and pessimistic bounds would waste execution opportunities, RELAXR profiles the average execution time of each segment ℓ as its *nominal execution time* $\hat{c}_\ell$ and uses it for server policy design, as shown in Fig. 8. Accordingly, RELAXR targets *soft real-time, profile-based* behavior rather than WCET-based hard guarantees: the analytical results in this section are stated with respect to the profiled bounds $\hat{c}_\ell$, and substituting high-percentile or safety-inflated profiles yields correspondingly conservative behavior. This positioning matches AIXR usage scenarios, where late feedback degrades user experience but does not constitute a hard-deadline violation. The server rules can be generalized to more than two servers, but our explanation focuses on the two servers corresponding to the XR and AI feedback pipelines defined in §II-A.

Based on C1-C3 in §III-B, RELAXR first builds a two-server hierarchy over CPU-dispatched GPU work for XR and AI feedback pipelines. We use a *GPU segment* to denote the smallest non-preemptive unit, such as one XR renderer instance or one AI layer instance. Each server maintains a period, budget, priority, and ready queue, and queries other servers' information through global APIs. The XR server has a higher priority, while the AI feedback server has a lower priority. The core idea is period-local dispatch and donation: RELAXR launches a GPU segment only when it fits the current period's remaining budget, donates unused budget from the HP server to an LP server before it is dropped, and enables background execution for a selected server only when doing so cannot delay any GPU segment in other servers.

Rule 1 (Budget-aware Dispatch)**.** A server dispatches a ready GPU segment only if the segment's execution time fits within the server's remaining budget in the current period. This check is triggered whenever the available budget or ready queue changes, including at a new server period, after a GPU segment finishes, and after budget donation. If the GPU segment cannot fit, it remains in the ready queue until the next dispatch point.

Rule 1 addresses C1 by preventing negative-budget execution. In Fig. 9(A), FE2 is blocked when the remaining AI

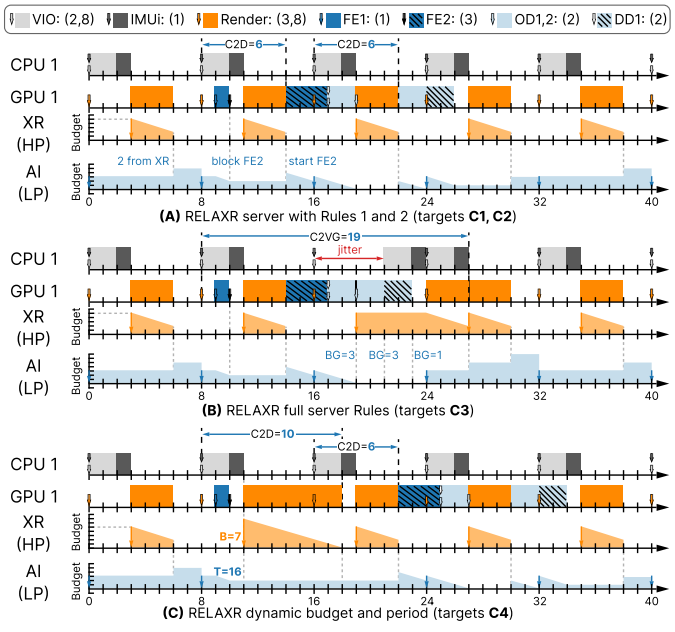


Fig. 9: Timelines of RELAXR server.

budget is insufficient, avoiding the budget overrun that caused priority inversion in Fig. 5(B).

Rule 2 (Unused-budget Donation)**.** RELAXR introduces an `all_finished` API to detect when an HP server has completed all GPU segments released in the current period. The API is invoked after the server's last ready GPU segment finishes. Once completion is confirmed, the HP server donates its remaining budget to the next highest-priority server, which becomes the recipient. The donated budget expires at the end of the recipient's current period. The recipient can use the donated budget only for GPU segments that pass Budget-aware Dispatch.

Rule 2 addresses C2 by donating budget to an LP server before it is dropped. Instead of waiting for a later replenishment, the AI feedback server can use unused XR budget within the same period. Fig. 9(A) shows this donation as the additional AI budget marked "2 from XR."

Rule 3 (Lookahead Background Execution)**.** After a GPU segment finishes, RELAXR computes a lookahead window before the worst-case earliest future GPU request from any higher-priority server. A lower-priority ready GPU segment may execute as background work only if its profiled execution time fits within this window. The window is recomputed after each GPU segment completion and cannot be carried across periods. To obtain this window across all servers, RELAXR introduces the `get_time_to_next` API call. This API returns the minimum jitter-adjusted time to the next GPU segment execution among all HP servers, which prevents background work from delaying any HP segments.

Rule 3 addresses C3 by using safe gaps created by jitter without delaying future XR work. In Fig. 9(B), the background windows marked BG allow selected AI segments to run between shifted XR releases. This also avoids the case where an AI segment fits the remaining AI budget but would still collide with an imminent XR renderer release.

The three rules have precedents, but each departs from them where AIXR requires. Rule 1 resembles TimeWall's forbidden zones and SIRAP's self-blocking [45, 46], but admits each GPU segment against a budget that may change through donation. Rule 2 echoes CASH/BASH-style capacity reclaiming [47–49], but makes donation priority-ordered, period-local, and subject to Budget-aware Dispatch. Rule 3 admits LP work only when it fits before the jitter-adjusted next HP release. This release-aware check is essential under non-preemption: static HP or unchecked reclaiming can still block the next renderer release (C1), while conservatively idling the GPU can starve ready AI segments (C3).

Exception Handling and Generalizability. During runtime, RELAXR handles three exceptions. First, if a GPU segment's actual execution time exceeds its server or component period, RELAXR skips that segment's next execution. Second, if VIO or IMUi runs longer than its nominal profile, RELAXR re-phases the XR server so the next server period starts at the next IMUi completion. Third, if `get_time_to_next` returns no positive jitter-adjusted window, background execution is disabled until the next segment completion or budget replenishment. The same rules can be generalized to more than two servers by ordering servers in rate-monotonic priority order, with shorter-period servers assigned higher priority. For any lower-priority server, donation is allowed only after all higher-priority servers have completed their execution of current-period GPU segments, and lookahead background execution is allowed only after checking ready queues from all servers. Under this generalized rate-monotonic server hierarchy, Lemma 1 shows that donated and lookahead LP execution does not introduce additional HP interference, and Corollary 1 further bounds the maximum AI-induced blocking time on XR.

Lemma 1 (No additional HP interference from LP execution)**.** *Under RELAXR server policies, donated or lookahead lower-priority execution does not introduce additional interference to higher-priority servers beyond the admitted non-preemptive GPU segment checked by Budget-aware Dispatch.*

Proof. Servers are ordered by rate-monotonic priority, where an HP server has a period no larger than an LP server. For example, in the two-server AIXR setting, the XR server is HP and the AI feedback server is LP; in the generalized multi-server case, the same ordering extends to any additional lower-priority servers [50]. RELAXR allows an LP GPU segment to execute only in two cases: all GPU segments in the HP server have finished their current-period execution, and the HP server donates unused budget, or Lookahead Background Execution finds a safe window before the next HP GPU segment starts. In both cases, the LP GPU segment must pass Budget-aware Dispatch, meaning its profiled execution time fits within the available donated budget or the lookahead window. Since the donated budget does not carry over to

the next period, LP execution cannot consume future HP budget. Therefore, LP execution uses only time that would otherwise be unused in the current period, and it does not create additional HP interference beyond the admitted non-preemptive GPU segment. □

Corollary 1 (Bounded AI blocking of XR release)**.** *Let $C_{\mathrm{AI}}^{\max} = \max_{\ell \in \mathcal{G}_{\mathrm{AI}}} \hat{c}_\ell$, where $\mathcal{G}_{\mathrm{AI}}$ is the set of AI feedback GPU segments and $\hat{c}_\ell$ is the execution-time bound used by Rule 1 (Budget-aware Dispatch) for segment ℓ. Under RELAXR server rules, any XR renderer release can be delayed by already-dispatched AI execution for at most $C_{\mathrm{AI}}^{\max}$.*

Proof. At an XR renderer release, either no AI segment is running, in which case there is no AI-induced blocking, or one AI segment is already running. Since GPU segments are non-preemptive, the renderer can be delayed only until that running AI segment reaches its next segment boundary, i.e., by at most its remaining execution time, which is bounded by $\hat{c}_\ell \le C_{\mathrm{AI}}^{\max}$ according to Rule 1. At that boundary, since the XR server has a higher priority, the ready renderer is dispatched before any further AI segments. Therefore, the maximum AI-induced blocking of any XR renderer release is bounded by $C_{\mathrm{AI}}^{\max}$. □

If $\hat{c}_\ell$ is set to the profiled nominal execution time, Corollary 1 gives the nominal blocking bound. Using safety-inflated or high-percentile profiles gives a correspondingly conservative bound.

C. Renderer-Aware Budget-Period Adaptation

Runtime Estimator. Static server budgets become stale when the renderer execution time changes at runtime. This is common in AIXR because the renderer workload changes with the visible objects, AI-generated grounding masks, and dialogue boxes. If renderer time exceeds its reserved budget (C4), later XR frames can be delayed, and the AI feedback window shrinks. Rather than reducing rendering quality, RELAXR adapts the XR reservation using an online renderer-time estimate. At the end of each period k, RELAXR estimates the next-period renderer time $\hat{c}_{\mathrm{ren},k+1}$ from the current viewport and the latest observed renderer time:

$$\min\left[c_{\mathrm{ren}}^{\max}, \max\left(c_{\mathrm{ren},k}^{\mathrm{obs}}, c_{\mathrm{ren}}^{0} + \alpha_n (n_k - n_0)^+ + \alpha_h h_k\right) + \sigma_r\right]$$

Here, c_{ren}^{0} is the renderer time measured under object count n_0, n_k is the number of visible virtual objects, h_k is the number of highlighted objects, and $(x)^+ = \max(x, 0)$. The coefficients α_n and α_h capture the marginal rendering cost of visible and highlighted objects. The inner estimate increases the baseline renderer time according to viewport complexity, while the max with $c_{\mathrm{ren},k}^{\mathrm{obs}}$ captures recent surges not explained by these features. Finally, σ_r adds a safety margin and $c_{\mathrm{ren}}^{\max}$ caps reservation growth.

Budget and Period Adjustment. RELAXR converts the renderer estimate into the next-period XR reservation R_{k+1} and defines the minimum AI budget $B_{\mathrm{AI}}^{\min}$:

$$R_{k+1} = \hat{c}_{\mathrm{ren},k+1} + \sigma_{\mathrm{XR}}, \qquad B_{\mathrm{AI}}^{\min} = C_{\mathrm{AI}}^{\max} + \sigma_{\mathrm{AI}}.$$

where R_{k+1} is the required XR budget for the next period T_{k+1} with an additional XR scheduling margin σ_{XR}. $C_{\mathrm{AI}}^{\max}$ denotes the maximum nominal execution time among AI segments with an additional AI scheduling margin σ_{AI}. RELAXR then updates the shared server period as:

$$T_{k+1} = \begin{cases} T_k, & R_{k+1} + B_{\mathrm{AI}}^{\min} \le T_k, \\ \left\lceil \frac{R_{k+1} + B_{\mathrm{AI}}^{\min}}{T_0} \right\rceil T_0, & \text{otherwise} \end{cases}$$

It then sets the next-period budgets as

$$B_{\mathrm{XR},k+1} = R_{k+1}, \quad B_{\mathrm{AI},k+1} = T_{k+1} - B_{\mathrm{XR},k+1}.$$

If the required XR reservation fits while preserving the $B_{\mathrm{AI}}^{\min}$, RELAXR keeps the current period and only updates the XR budget. Otherwise, RELAXR inflates the next period to the smallest integer multiple of the nominal period T_0 that can accommodate both the XR reservation and the minimum AI feedback window. The updated period T_{k+1} is applied to both the XR and AI feedback servers, keeping their periods aligned while redistributing their budgets. As illustrated in Fig. 9(C), this lets RELAXR absorb renderer-time growth while preserving a nonzero AI feedback budget.

Unlike prior adaptive server reservation schemes [51], RELAXR jointly adapts the shared period and redistributes budget between the coupled XR and AI feedback servers. The adjustment is constrained by both the estimated renderer demand and a minimum AI budget, so increased XR demand does not block AI feedback execution.

Lemma 2 (Policy preservation under dynamic budget and period)**.** *Renderer-Aware Budget-Period Adaptation preserves the validity of RELAXR server rules. After each applied update, the XR and AI feedback servers share the updated period T_{k+1}, the XR server receives budget $B_{\mathrm{XR},k+1} = R_{k+1}$ for the estimated renderer demand, the AI feedback server retains a budget of at least $B_{\mathrm{AI}}^{\min}$, and Rules 1–3 remain applicable.*

Proof. RELAXR updates server parameters only at period boundaries, so no in-period GPU segment is reclassified or carried across periods. Let $R_{k+1} = \hat{c}_{\mathrm{ren},k+1} + \sigma_{\mathrm{XR}}$ be the required XR reservation for the next period. If $R_{k+1} + B_{\mathrm{AI}}^{\min} \le T_k$, RELAXR keeps $T_{k+1} = T_k$ and sets $B_{\mathrm{XR},k+1} = R_{k+1}$, yielding $B_{\mathrm{AI},k+1} = T_{k+1} - B_{\mathrm{XR},k+1} \ge B_{\mathrm{AI}}^{\min}$. Otherwise, RELAXR sets $T_{k+1} = \lceil (R_{k+1} + B_{\mathrm{AI}}^{\min})/T_0 \rceil T_0$, so $T_{k+1} \ge R_{k+1} + B_{\mathrm{AI}}^{\min}$ and again $B_{\mathrm{AI},k+1} \ge B_{\mathrm{AI}}^{\min}$. Thus, each updated period starts with feasible aligned budgets. Budget-aware Dispatch uses the updated budget, Unused-budget Donation remains period-local, and Lookahead Background Execution uses the updated future-release times. Therefore, adaptation changes only feasible server parameters and does not invalidate the RELAXR server rules. □

D. AI Feedback Pipeline Reconstruction

AI feedback pipeline reconstruction addresses C5 by changing the internal execution order of the AI feedback pipeline. LLM Context Preload moves query-independent scene context

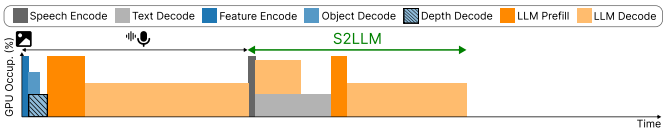


Fig. 10: LLM context preload and execution overlapping.

into the speech interval, reducing post-speech work on the language response path. Path-aware Execution Overlapping uses available GPU occupancy within the AI budget to overlap ready layers while prioritizing layers on either the visual grounding path or the language response path.

LLM Context Preload. The goal of LLM Context Preload is to reduce S2LLM by moving scene-context prefill out of the post-speech critical path. This is possible in AIXR because the reference CAM frame is selected at the beginning of speech, before the final transcript is available. The challenge is that RELAXR does not yet know the user's final query: preloading too little scene context may miss the queried object, while preloading too much increases prefill cost and can offset the latency benefit.

RELAXR addresses this by splitting the LLM input into a query-agnostic *scene preload* and a later query-specific *speech suffix*. During user speech, RELAXR runs the feature encoder, object decoder, and depth decoder to construct a bounded scene preload containing object labels, 2D locations, and depth. It then prefills this preload and caches the resulting KV states. After the speech ends, RELAXR appends the S-t-T transcript as the speech suffix and continues LLM inference from the cached preload. As shown in Fig. 10, this moves scene-context prefill into the speech interval, so the post-speech path only processes the user query and decodes the response. This preserves the same scene-grounded prompt semantics while shortening the S2LLM critical path. Unlike prior prefix-caching techniques that reuse cached prompt prefixes to reduce repeated prefill [52, 53], RELAXR exploits an AIXR-specific timing opportunity: the visual context is fixed before the user's speech completes, allowing scene-context preparation and prefill to overlap with live speech.

Path-aware Execution Overlapping. The goal of Path-aware Execution Overlapping is to reduce user-visible AI feedback latencies, including both C2VG and S2LLM, while avoiding spatial GPU waste caused by unused thread-block capacity. Temporal isolation may reserve time for the AI feedback server, but a low-occupancy AI layer running alone can still leave GPU spatially underutilized. At the same time, arbitrary overlap is not enough: launching a ready layer unrelated to the current user-visible feedback path may consume AI budget without reducing either C2VG or S2LLM.

RELAXR therefore makes overlap path-aware. Rather than introducing a separate dependency graph, RELAXR reuses the two AI feedback paths defined in §II-A: the visual grounding path Γ_k^{visual} for C2VG and the language response path Γ_k^{lang} for S2LLM. At each AI feedback server period start, RELAXR selects a target metric $\mu \in \{\text{C2VG}, \text{S2LLM}\}$ and assigns each ready layer ℓ_i a path-aware value $p_i^\mu = c_i + \max_{\ell_j \in \text{succ}(\ell_i)} p_j^\mu, \ell_i \in \Gamma_k^\mu$ where c_i is the layer's nominal execution time. A layer on the selected path is valued by its own execution time plus the longest downstream time to the next user-visible output, while an off-path layer ($p_i^\mu = 0$) is used only as fill work when no path-relevant layer can be launched. This prioritizes layers that directly advance the selected latency metric.

TABLE II: Scenes ordered by increasing difficulty.

	Topic	# Recording	# Object	# Input Tokens
S1	Pickup and recognition	130	3	10
S2	Object-related Q&A	175	6	20
S3	Single-object instructions	102	6	25
S4	Multi-object instructions	17	10	30

Given these values, RELAXR selects which ready layers to launch within the remaining AI budget and the available GPU occupancy using a knapsack-style dynamic program over the remaining AI budget and available GPU occupancy. At each decision point, let $R = \{\ell_1, \ldots, \ell_n\}$ denote the ready layers, B the remaining AI budget, and O the available GPU occupancy capacity. Each layer ℓ_i has nominal execution time c_i, occupancy footprint o_i, and path-aware value p_i^μ. RELAXR defines $D[i][b][o]$ as the maximum path-aware value obtainable from the first i ready layers under budget b and occupancy o, with $D[0][b][o] = 0$ and

$$D[i][b][o] = \begin{cases} \max(D[i-1][b][o], & c_i \le b \\ \quad D[i-1][b-c_i][o-o_i] + p_i^\mu) & o_i \le o \\ D[i-1][b][o] & \text{otherwise.} \end{cases}$$

Backtracking from $D[n][B][O]$ selects the launch plan, and each selected layer is dispatched at its earliest feasible time; layers whose combined occupancy fits within O are overlapped inside the AI feedback pipeline. As shown in Fig. 10, RELAXR overlaps Feature Encode with Object Decode to reduce C2VG, and LLM Decode with Text Decode to shorten S2LLM. Because the ready layers, budget, and target metric change online, RELAXR computes this plan at runtime rather than offline.

V. Evaluation

A. Setup

Hardware Setup. To evaluate across a spectrum of AIXR hardware, we conduct experiments on both a PC laptop equipped with an NVIDIA RTX 3050 GPU and an NVIDIA Jetson AGX Orin embedded device with an integrated GPU [54]. The laptop is powered by a 130W power source, and the AGX Orin is configured to run in the 60W MAXN mode. For the controlled scene evaluation in §V-B, both devices process egocentric video input from the HOT3D dataset [22] and pre-recorded speech generated using the Google Text-to-Speech API [55]. For the real-world evaluation in §V-C, a ZED Mini camera captures stereo video input, while a microphone records human speech in real time.

AIXR Software Setup. We implement a complete AIXR system following the architecture in Fig. 1. The XR pipeline includes the components described in §II-A: OpenVINS [24]

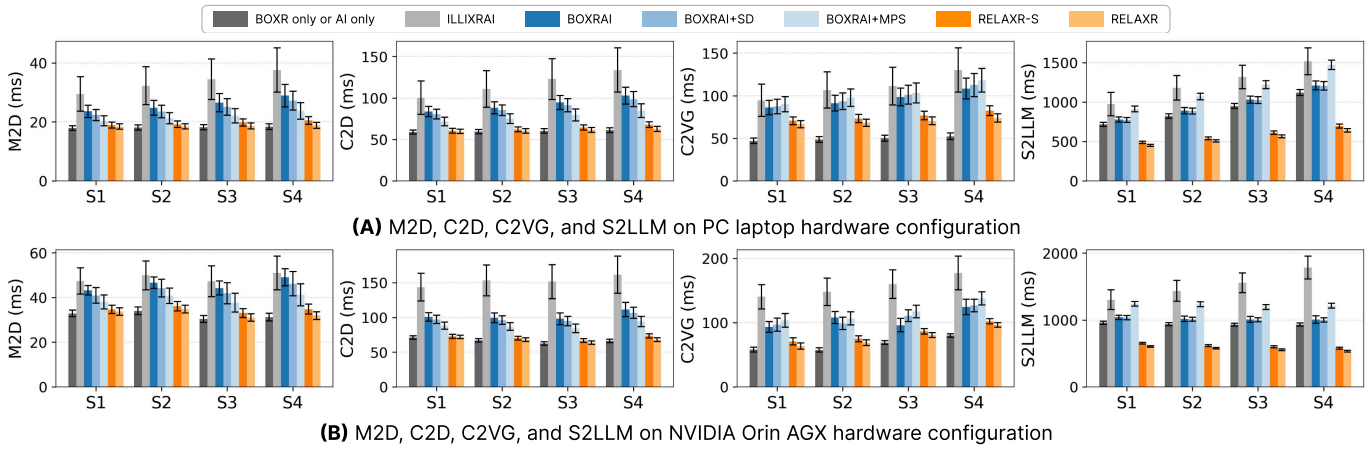


Fig. 11: Comparison of XR-related latencies (M2D, C2D) and AI feedback latencies (C2VG, S2LLM) across all scenes.

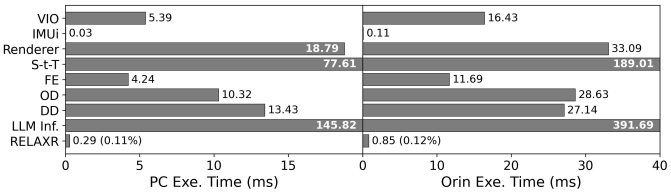


Fig. 12: RELAXR overhead.

for VIO, GTSAM [25] for IMUi, and OpenCV/Godot [26, 56] for rendering. The AI feedback pipeline uses Whisper [29] for S-t-T, DINOv2 [30] for FE, FastSAM [32] for OD, and Depth Anything V2 [31] for DD. We use TinyLlama-1.1B [57] as the default LLM because of its low inference latency and on-device efficiency. We evaluate alternative LLM architectures in §V-B.

Scene Benchmarks. To evaluate RELAXR under AIXR tasks with increasing visual and language complexity, we construct a scene benchmark from the HOT3D dataset [22]. We group the video recordings into four difficulty levels, S1-S4, as shown in Tab. II. The levels increase in semantic complexity, number of objects, and maximum input-speech length. For each level, we generate speech questions with bounded word counts, ranging from 10 words in S1 to 30 words in S4, so that both scene complexity and speech complexity increase across the benchmark.

Baseline Configurations. We compare seven configurations, including four baselines and two RELAXR variants. *BOXR* [14] is the standalone XR pipeline without AI feedback. *BOXRAI* co-runs BOXR with the AI feedback pipeline under default OS scheduling. *ILLIXRAI* co-runs the ILLIXR XR pipeline with the same AI feedback pipeline under default OS scheduling, providing a cross-runtime baseline based on ILLIXR's publisher-subscriber execution architecture [33]. Since ILLIXRAI uses a different execution architecture from BOXRAI and RELAXR, it allows us to evaluate how the underlying runtime architecture affects AIXR system performance. *BOXRAI+SD* adds the CPU-side `SCHED_DEADLINE` scheduler [58] to BOXRAI, representing deadline-based CPU scheduling and resource reservation without GPU-aware admission. *BOXRAI+MPS* uses NVIDIA Multi-Process Service (MPS) to allocate 50% of the GPU compute capacity to each of the XR and AI feedback pipelines, providing a fair spatial isolation baseline by using the MPS's Active Thread Percentage mechanism. *RELAXR-S* includes our GPU-aware server policies and workload optimizations, but excludes runtime budget-period adaptation. *RELAXR* is the full system with GPU-aware server policies, *Renderer-Aware Budget-Period Adaptation*, and workload optimizations.

B. Controlled Scene Evaluation

RELAXR Overhead. RELAXR introduces low runtime overhead compared with the AIXR pipeline components. As shown in Fig. 12, on the PC platform, RELAXR takes only 0.29 ms, accounting for 0.11% of the profiled execution time. On the embedded Orin platform, RELAXR incurs only 0.85 ms overhead, accounting for 0.12% of the profiled execution time. This low overhead comes from the bounded dynamic program used in Path-aware Execution Overlapping, which selects ready GPU segments over discretized budget and occupancy instead of an expensive online exhaustive search.

Overall Effectiveness. We measure all four latency metrics across S1-S4 and both hardware platforms. For consistency in S2LLM, we cap LLM generation at 100 output tokens in all runs. As shown in Fig. 11, BOXRAI first reduces latencies by up to 48.1% compared with ILLIXRAI. Since both run the same AI feedback pipeline under default OS scheduling, their comparison captures the effect of the underlying XR execution architecture. Therefore, BOXRAI and its variants provide the controlled basis for comparing against RELAXR-S and attributing the additional gains to RELAXR's components. Accordingly, RELAXR-S further reduces latencies over the BOXR-based baselines, including up to 44.7% compared with BOXRAI and up to 39.7% compared with BOXRAI+SD. Furthermore, BOXRAI+MPS exposes the limits of static spatial isolation: reserving half of the GPU for the XR pipeline trims M2D and C2D slightly below BOXRAI+SD (by up to 14.1%), but S2LLM inflates by 22.0% because computation-heavy LLM decoding cannot borrow from XR compute even when it sits idle. By reclaiming such idle capacity at runtime,

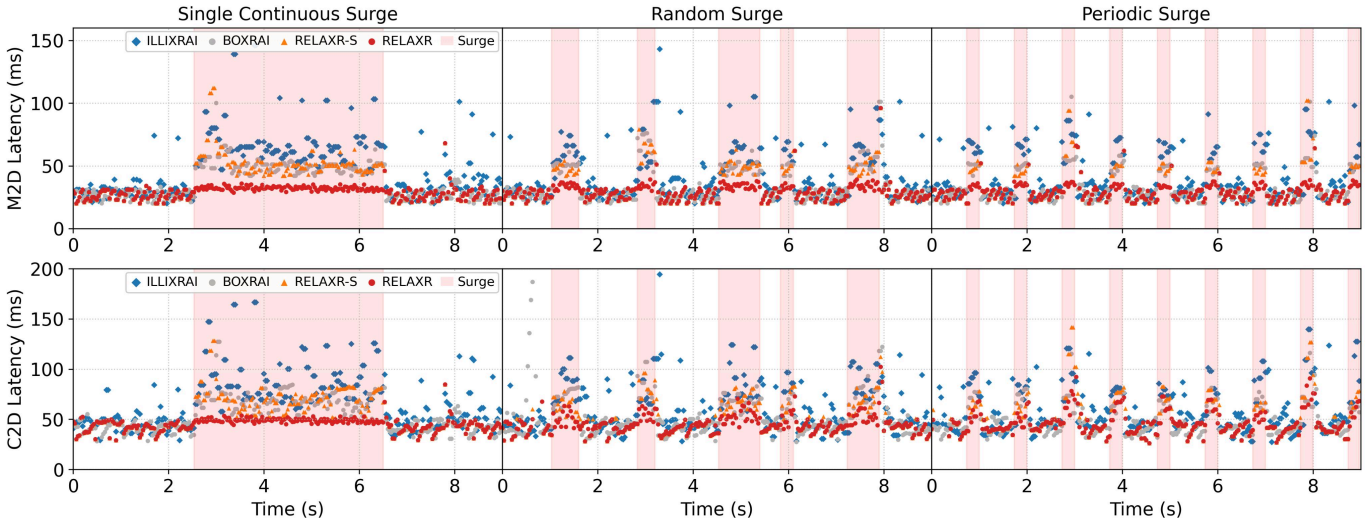


Fig. 13: M2D and C2D under rendering surge.

TABLE III: S2LLM (ms) under different LLM inferences.

Baseline	TinyLlama-1.1B	Orca-Mini-3B	Llama3.2-1B	Qwen3-1.7B
BOXRAI	777.64	1373.32	1504.73	9388.05
RELAXR	487.47↓	929.19↓	837.62↓	4259.74↓

RELAXR-S outperforms BOXRAI+MPS on all four metrics, with up to 52.7% lower S2LLM. The XR-related latency gains mainly come from the RELAXR server rules, while the AI-feedback latency gains come from Path-aware Execution Overlapping and LLM Context Preload. RELAXR further improves over RELAXR-S by up to 13.0% due to Renderer-Aware Budget-Period Adaptation. Overall, RELAXR reduces M2D by up to 44.0%, C2D by up to 47.3%, C2VG by up to 36.4%, and S2LLM by up to 56.4% compared with ILLIXRAI. The ablation between RELAXR and RELAXR-S shows that static servers alone cannot adapt to renderer-demand changes, and runtime budget-period adaptation is necessary to keep XR reservations aligned with dynamic workloads.

Crucially, **RELAXR keeps XR-related latencies (M2D and C2D) within 2.8% of the XR-only baseline**, showing that timely AI feedback can be achieved without sacrificing motion responsiveness. The remaining gap mainly comes from segment-boundary blocking analyzed in §IV-B and conservative XR reservations. Runtime budget and period adjustment reduce shared-GPU interference but cannot eliminate it completely, because GPU segments remain non-preemptive after launch and budget-period updates take effect only at period boundaries.

Latency Across LLM Models. We evaluate S2LLM across four lightweight LLMs on the PC laptop with a maximum of 100 output tokens: TinyLlama-1.1B [57], Orca-Mini-3B [59], Llama3.2-1B [60], and Qwen3-1.7B [61]. As shown in Tab. III, RELAXR reduces S2LLM across all models, with up to a 54.6% reduction on Qwen3-1.7B. For the three lightweight mobile models, RELAXR keeps S2LLM below one second. The improvement comes from LLM context preload, which moves scene-context prefill into the speech interval and reuses the cached context after speech ends, reducing post-speech LLM inference time.

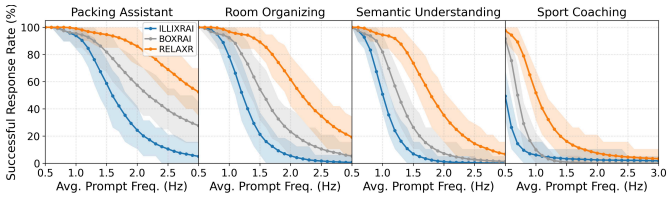


Fig. 14: Success response with increasing prompt frequency. Lines show the average and shaded areas indicate error range.

Renderer Surge. To evaluate robustness under C4, we stress the PC-laptop renderer with continuous, random, and periodic surge profiles by adding 10-15 high-resolution virtual objects and visual-grounding masks while keeping the AI feedback workload unchanged. As shown in Fig. 13, BOXRAI and ILLIXRAI show large M2D/C2D spikes during surge windows, and RELAXR-S also degrades when its fixed XR reservation is exceeded. In contrast, RELAXR expands the XR reservation through Renderer-Aware Budget-Period Adaptation, keeping both M2D and C2D lower across all surge profiles. Compared with ILLIXRAI, RELAXR reduces mean M2D/C2D by up to 24.2%/20.8% and worst-case M2D by up to 67.9% during surges. These results show that static GPU-aware reservation is insufficient under renderer-time fluctuations, while adaptive budget-period control better preserves XR responsiveness.

C. Real-world Evaluation

Scenario Setup. The controlled benchmarks in Sec. V-B establish RELAXR's behavior under reproducible workloads. We now validate that these gains hold in real-world deployment on the NVIDIA Jetson AGX Orin platform [54], using live stereo video from a ZED Mini camera [62] and synchronized speech input from a microphone. We evaluate four daily-use AIXR scenarios: Packing Assistant evaluates indoor stationary object selection, Room Organizing evaluates indoor mobile spatial reasoning, Semantic Understanding evaluates

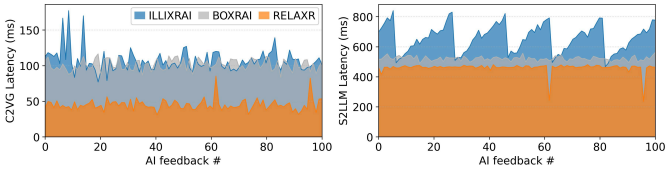


Fig. 15: AI feedback latencies under Room Organizing.

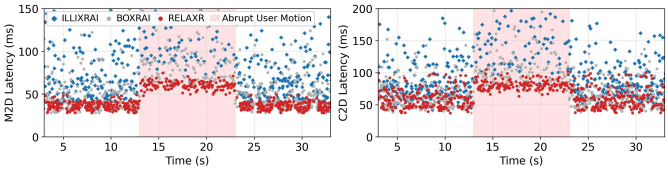


Fig. 16: M2D/C2D under abrupt motion in Sport Coaching.

outdoor stationary scene understanding, and Sport Coaching evaluates outdoor mobile guidance. For each scenario and prompt frequency, we wear the complete AIXR setup and run each framework for five 60-second trials.

Success Response Rate. We count a live AI feedback response as successful if, when displayed, it still matches the user's current speech and view context. We use 0.5 Hz and 1 Hz to represent realistic prompt frequencies, while frequencies above 1 Hz serve only as stress-test conditions. As shown in Fig. 14, all systems perform well at low prompt frequency, but BOXRAI and ILLIXRAI drop sharply as prompts become more frequent, indicating that delayed AI feedback often becomes stale. In contrast, RELAXR consistently maintains the highest success rate across all four scenarios. At 3 Hz in Packing Assistant, RELAXR achieves a 52.7% successful response rate, compared with 28.5% for BOXRAI and 5.3% for ILLIXRAI. This improvement comes from reducing C2VG and S2LLM through GPU-aware reservation, LLM Context Preload, and Path-aware Execution Overlapping, which help AI feedback arrive before the interaction context changes.

AI Feedback Latencies. We profile one hundred consecutive live AI feedback events in the Room Organizing scenario, as shown in Fig. 15. RELAXR consistently keeps both C2VG and S2LLM below the two baselines. Compared with ILLIXRAI, RELAXR reduces average C2VG/S2LLM by 57.9%/30.1%. Compared with BOXRAI, RELAXR still reduces C2VG/S2LLM by 55.7%/11.8%. RELAXR also improves tail latency, reducing maximum C2VG/S2LLM by up to 51.9%/42.8%. These results show that RELAXR reduces both typical and worst-case AI feedback latencies in live deployment.

Robustness Analysis. We stress RELAXR under real-world motion disturbance by asking the user to perform rapid head and body motion during a sustained 10-second burst window in the Sport Coaching scenario. To minimize bias, we randomly alternate between ILLIXRAI, BOXRAI, and RELAXR without disclosing which framework is active during each trial. As shown in Fig. 16, RELAXR maintains lower M2D and C2D than ILLIXRAI throughout. Outside the burst, RELAXR reduces M2D by 29.1% and C2D by 29.5%; during the burst, the reductions grow to 58.1% and 53.2%, respectively. This widening gain occurs because Renderer-Aware Budget-Period Adaptation expands the XR reservation as rendering demand rises, while the GPU-aware server rules prevent AI segments from delaying renderer execution.

VI. Related Work

AIXR systems. XR systems are undergoing a transformative shift from passive virtual-scene rendering to AI-driven interaction with the physical world. By integrating lightweight language, vision, and sensing models [7, 9, 63–65], AIXR systems can provide real-time on-display assistance for spatial reasoning, object-aware interaction, and context-aware user guidance [8, 10, 15, 16, 18, 23, 66, 67]. Existing work demonstrates the feasibility of AI-driven XR interaction, but it does not characterize how the system affects XR-related latencies and AI feedback latencies, which RELAXR addresses with a latency-aware runtime framework.

Resource reservation and real-time isolation. Server-based reservation has been widely studied in real-time systems, including hierarchical scheduling, resource sharing, adaptive and multi-resource servers, and time partitioning for multi-core and accelerator platforms [45, 46, 51, 68–70]. Specifically, TimeWall's forbidden zones decline jobs that cannot finish before a static partition boundary [45], and CASH/BASH-style servers reclaim residual capacity among preemptive EDF reservations through deadline-tagged global queues [47–49]. These provide foundations for temporal isolation but are not directly applicable to AIXR, where the two pipelines run as separate processes, launch GPU work through different streams, and have highly dynamic execution times driven by user speech and scene complexity. RELAXR instead targets a structurally different setting: two coupled GPU-dominant pipelines co-scheduled as non-preemptive segments, with demand varying on both sides at runtime and context freshness (C2VG/S2LLM) rather than per-job response time as the objective. Its rules therefore admit each dispatch against period-local budgets that change mid-period through donation and against jitter-adjusted future releases, rather than against static partitions or preemptive EDF reservations. GPU partitioning mechanisms such as MPS and Fractional GPUs [71, 72] manage spatial resources but cannot adapt to AIXR's changing temporal and spatial demand. RELAXR closes this gap with a pipeline-aware design that adapts GPU reservations to preserve XR responsiveness while minimizing AI feedback latencies.

VII. Conclusion

We presented RELAXR, a runtime framework for co-running XR and AI pipelines on a shared GPU. RELAXR defines C2VG and S2LLM to capture user-visible AI feedback latencies, and proposes three mechanisms in §IV to reduce AI feedback latencies while preserving XR responsiveness. Evaluations show that RELAXR can meet XR and AI feedback timing requirements simultaneously. RELAXR currently assumes no strict runtime dependency between XR and AI feedback pipelines; extending the design to dependency-constrained pipelines and heterogeneous accelerators remains future work.

Acknowledgment

This work was supported by the National Science Foundation (NSF) grants 2312395, 2300525, 2312397, and 2343653.

References

[1] Apple, "Apple Vision Pro," 2024.
[2] Meta, "Meta Quest 3," 2023.
[3] E. Realities, "Even G2," 2026.
[4] C. Kong, J. Fort, A. Kang, J. Wittmer, S. Green, T. Shen, Y. Zhao, C. Peng, G. Solaira, A. Berkovich *et al.*, "Aria gen 2 pilot dataset," *arXiv preprint arXiv:2510.16134*, 2025.
[5] J. Engel, K. Somasundaram, M. Goesele, A. Sun, A. Gamino, A. Turner, A. Talattof, A. Yuan, B. Souti, B. Meredith *et al.*, "Project aria: A new tool for egocentric multi-modal ai research," *arXiv preprint arXiv:2308.13561*, 2023.
[6] L. M. Feraru, D. C. Klonoff, and D. G. Armstrong, "Ray-ban meta: A ray of hope in the operating room (and beyond)," *Surgical Innovation*, p. 15533506251383696, 2025.
[7] Z. Liu, C. Zhao, F. Iandola, C. Lai, Y. Tian, I. Fedorov, Y. Xiong, E. Chang, Y. Shi, R. Krishnamoorthi *et al.*, "Mobilellm: Optimizing sub-billion parameter language models for on-device use cases," in *Forty-first International Conference on Machine Learning*, 2024.
[8] C. Jung, J. Lee, G. Kim, J. Kim, S. Park, and H. Cha, "Aria: Optimizing vision foundation model inference on heterogeneous mobile processors for augmented reality," in *Proceedings of the 23rd ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2025.
[9] J. Xu, Z. Li, W. Chen, Q. Wang, X. Gao, Q. Cai, and Z. Ling, "On-device language models: A comprehensive review," *arXiv preprint arXiv:2409.00088*, 2024.
[10] X. Li, Z. Lu, D. Cai, X. Ma, and M. Xu, "Large language models on mobile devices: Measurements, analysis, and insights," in *Proceedings of the Workshop on Edge and Mobile Foundation Models*, 2024, pp. 1–6.
[11] Centers for Disease Control and Prevention, "Motion Sickness," 2024.
[12] Optofidelity, "Apple Vision Pro Benchmark Test 1: See-Through Latency, Photon-to-Photon," 2024.
[13] A. Dixit and S. R. Sarangi, "Minimizing the motion-to-photon-delay (mpd) in virtual reality systems," *arXiv preprint arXiv:2301.10408*, 2023.
[14] Z. Zhang, Z. Li, H. Kim, and C. Liu, "Boxr: Body and head motion optimization framework for extended reality," in *2024 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2024.
[15] S. Srinidhi, E. Lu, and A. Rowe, "Xair: An xr platform that integrates large language models with the physical world," in *2024 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2024, pp. 759–767.
[16] D. Li, N. Numan, X. Qian, Y. Chen, Z. Zhou, E. Alekseev, G. Lee, A. Cooper, M. Xia, S. Chung *et al.*, "Xr blocks: Accelerating human-centered ai+ xr innovation," *arXiv preprint arXiv:2509.25504*, 2025.
[17] R. Suzuki, M. Gonzalez-Franco, M. Sra, and D. Lindlbauer, "Everyday ar through ai-in-the-loop," in *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 2025, pp. 1–5.
[18] M. D. Dogan, E. J. Gonzalez, K. Ahuja, R. Du, A. Colaço, J. Lee, M. Gonzalez-Franco, and D. Kim, "Augmented object intelligence with xr-objects," in *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, 2024, pp. 1–15.
[19] S. Hayden, "Why Ray-Ban Meta Glasses Failed on Stage at Connect," 2025.
[20] Y. Wang, C. Liu, D. Wong, and H. Kim, "Gcaps: Gpu context-aware preemptive priority-based scheduling for real-time tasks," *arXiv preprint arXiv:2406.05221*, 2024.
[21] J. Bakita and J. H. Anderson, "Demystifying nvidia gpu internals to enable reliable gpu management," in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2024, pp. 294–305.
[22] P. Banerjee, S. Shkodrani, P. Moulon, S. Hampali, S. Han, F. Zhang, L. Zhang, J. Fountain, E. Miller, S. Basol, R. Newcombe, R. Wang, J. J. Engel, and T. Hodan, "HOT3D: Hand and object tracking in 3D from egocentric multi-view videos," *CVPR*, 2025.
[23] D. Khan, X. Liu, O. Mena, D. Jia, A. A. Kouyoumdjian, and I. Viola, "Loxr: Performance evaluation of locally executing llms on xr devices," *arXiv preprint arXiv:2502.15761*, 2025.
[24] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.
[25] F. Dellaert and G. Contributors, "borglab/gtsam," May 2022.
[26] Godot, "Godot engine," Aug. 2017.
[27] J. Jerald, P. Giokaris, D. Woodall, A. Hartholt, A. Chandak, and S. Kuntz, "Developing virtual reality applications with unity," in *2014 IEEE Virtual Reality (VR)*. IEEE, 2014, pp. 1–3.
[28] G. Sellers and J. Kessenich, *Vulkan programming guide: The official guide to learning vulkan*. Addison-Wesley Professional, 2016.
[29] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *International conference on machine learning*. PMLR, 2023, pp. 28 492–28 518.
[30] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby *et al.*, "Dinov2: Learning robust visual features without supervision," *arXiv preprint arXiv:2304.07193*, 2023.
[31] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, "Depth anything v2," *Advances in Neural Information Processing Systems*, vol. 37, pp. 21 875–21 911, 2024.
[32] X. Zhao, W. Ding, Y. An, Y. Du, T. Yu, M. Li, M. Tang, and J. Wang, "Fast segment anything," *arXiv preprint arXiv:2306.12156*, 2023.
[33] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, B. Tian, H. Yuan, J. Zhang, and S. V. Adve, "Illixr: Enabling end-to-end extended reality research," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021.
[34] Optofidelity, "Apple Vision Pro benchmark test 2: Angular Motion-to-Photon Latency in VR," 2024.
[35] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–24, 2019.
[36] H. Sobhani and H. Kim, "Modeling and scheduling of fusion patterns in autonomous driving systems (extended version)," *arXiv preprint arXiv:2510.23895*, 2025.
[37] M. C. Jacobs, M. A. Livingston, and A. State, "Managing latency in complex augmented reality systems," in *Proceedings of the 1997 symposium on Interactive 3D graphics*, 1997, pp. 49–ff.
[38] R. Azuma and G. Bishop, "Improving static and dynamic registration in an optical see-through hmd," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, pp. 197–204.
[39] L. C. Loschky and G. S. Wolverton, "How late can you update gaze-contingent multiresolutional displays without detection?" *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 3, no. 4, pp. 1–10, 2007.
[40] A. Ishihara, H. Aga, Y. Ishihara, H. Ichikawa, H. Kaji, K. Kawasaki, D. Kobayashi, T. Kobayashi, K. Nishida, T. Hamasaki *et al.*, "Integrating both parallax and latency compensation into video see-through head-mounted display," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 5, pp. 2826–2836, 2023.
[41] J. Nielsen, *Usability engineering*. Morgan Kaufmann, 1994.
[42] S. C. Levinson and F. Torreira, "Timing in turn-taking and its implications for processing models of language," *Frontiers in psychology*, vol. 6, p. 136034, 2015.
[43] L. system programming training, "renice(1) — linux manual page," 2026.
[44] S. K. Saha, Y. Xiang, and H. Kim, "Stgm: Spatio-temporal gpu management for real-time tasks," in *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2019, pp. 1–6.
[45] T. Amert, Z. Tong, S. Voronov, J. Bakita, F. D. Smith, and J. H. Anderson, "Timewall: Enabling time partitioning for real-time multicore+ accelerator platforms," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 455–468.
[46] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "Sirap: a synchronization protocol for hierarchical resource sharingin real-time open systems," in *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, 2007, pp. 279–288.

[47] M. Caccamo, G. Buttazzo, and L. Sha, "Capacity sharing for overrun control," in *Proceedings 21st IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2000, pp. 295–304.

[48] M. Caccamo, G. C. Buttazzo, and D. C. Thomas, "Efficient reclaiming in reservation-based real-time systems with variable execution times," *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 198–213, 2005.

[49] G. Lipari and S. Baruah, "Greedy reclamation of unused bandwidth in constant-bandwidth servers," in *Proceedings 12th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2000, pp. 193–200.

[50] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[51] N. Stoimenov, L. Thiele, L. Santinelli, and G. Buttazzo, "Resource adaptations with servers for hard real-time systems," in *Proceedings of the tenth ACM international conference on Embedded software*, 2010, pp. 269–278.

[52] I. Gim, G. Chen, S.-s. Lee, N. Sarda, A. Khandelwal, and L. Zhong, "Prompt cache: Modular attention reuse for low-latency inference," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 325–338, 2024.

[53] L. Ye, Z. Tao, Y. Huang, and Y. Li, "Chunkattention: Efficient self-attention with prefix-aware kv cache and two-phase partition," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 11 608–11 620.

[54] Nvidia, "Jetson AGX Orin," 2023.

[55] G. Cloud, "Text-to-speech ai," 2023.

[56] G. Bradski, A. Kaehler *et al.*, "Opencv," *Dr. Dobb's journal of software tools*, vol. 3, no. 2, pp. 1–81, 2000.

[57] P. Zhang, G. Zeng, T. Wang, and W. Lu, "Tinyllama: An open-source small language model," *arXiv preprint arXiv:2401.02385*, 2024.

[58] L. Kernel, "The linux kernel 7.1.0-rc4," 2026.

[59] S. Mukherjee, A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and A. Awadallah, "Orca: Progressive learning from complex explanation traces of gpt-4," 2023.

[60] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[61] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[62] StereoLabs, "ZED Mini Stereo Camera," 2021.

[63] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti *et al.*, "Smolvla: A vision-language-action model for affordable and efficient robotics," *arXiv preprint arXiv:2506.01844*, 2025.

[64] Y. Li, H. Wang, Q. Jin, J. Hu, P. Chemerys, Y. Fu, Y. Wang, S. Tulyakov, and J. Ren, "Snapfusion: Text-to-image diffusion model on mobile devices within two seconds," *Advances in Neural Information Processing Systems*, vol. 36, pp. 20 662–20 678, 2023.

[65] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th symposium on operating systems principles*, 2023, pp. 611–626.

[66] E. Bozkir, S. Özdel, K. H. C. Lau, M. Wang, H. Gao, and E. Kasneci, "Embedding large language models into extended reality: Opportunities and challenges for inclusion, engagement, and privacy," in *Proceedings of the 6th ACM Conference on Conversational User Interfaces*, 2024, pp. 1–7.

[67] S. Kim, H. Kwon, J. Song, J. Jo, Y.-H. Chen, L. Lai, and V. Chandra, "Dream: A dynamic scheduler for dynamic real-time multi-model ml workloads," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, 2023, pp. 73–86.

[68] R. I. Davis and A. Burns, "Resource sharing in hierarchical fixed priority pre-emptive systems," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*. IEEE, 2006, pp. 257–270.

[69] H. Kim, S. Wang, and R. Rajkumar, "vmpcp: A synchronization framework for multi-core virtual machines," in *2014 IEEE Real-Time Systems Symposium*. IEEE, 2014, pp. 86–95.

[70] R. Inam, N. Mahmud, M. Behnam, T. Nolte, and M. Sjödin, "The multi-resource server for predictable execution on multi-core platforms," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2014, pp. 1–12.

[71] NVIDIA, "Multi-Process Service," 2026.

[72] S. Jain, I. Baek, S. Wang, and R. Rajkumar, "Fractional gpus: Software-based compute and memory bandwidth reservation for gpus," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 29–41.