WORLDCAST: DISTRIBUTED MULTIPLAYER WORLD MODELS

Anonymous authors
Paper under double-blind review

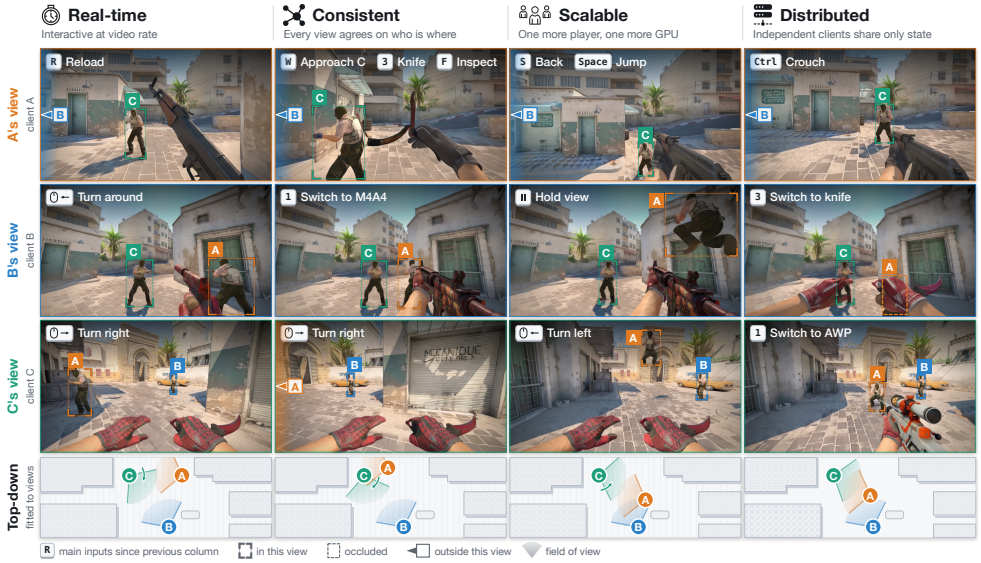

Figure 1: Three clients, each running on a separate machine, generate independently controlled views using shared world state. Only the initial frames are provided. Columns show successive moments, with controls annotated above each view; the bottom row shows a top-down layout fitted to the generated views. WorldCast supports real-time generation, cross-view consistency, scalability by adding clients, and distributed execution through state exchange.

ABSTRACT

Multiplayer world models must generate independently controlled views with consistent representations of both players and their shared environment. Most existing approaches coordinate multiple players through joint multi-view generation, whose cost grows with each additional player. We present **WorldCast**, a distributed multiplayer world model in which each player runs a local client comprising a video generator and a state model. Using recorded player positions and map geometry during training, the state model estimates the player's position from generated video and control inputs. Clients exchange player states and project them into camera-aligned player state fields that guide where and how other players are rendered. Shared scene state enables clients to reuse one another's generated observations to maintain consistent scene appearance across views. Experiments on Counter-Strike 2 demonstrate WorldCast's consistency, real-time performance, and distributed scalability. The camera-aligned player state field improves player rendering rates by over an order of magnitude over joint-generation methods, while shared scene state improves visual consistency over whole rounds. Each client runs in real time and exchanges only player and scene states, enabling scalable multiplayer generation without a centralized computational bottleneck. Image quality remains stable over hour-long rollouts.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Interactive video world models generate an agent's visual observations in response to its actions (Bruce et al., 2024; He et al., 2025b; Valevski et al., 2025; Huang et al., 2026). Multiplayer world modeling extends this setting to several independently controlled players in a shared environment (Cai et al., 2026; Liu et al., 2026; Savva et al., 2026). When players are visible, their locations should be consistent across viewpoints; previously observed parts of the environment should also retain their appearance across views and over time. Each view should remain responsive as players are added and, ideally, be generated on its own machine. Maintaining cross-view consistency without coupling the computation of different views is the central challenge.

Most prior methods generate players' views jointly through cross-view attention (Hu et al., 2026a;b; Liu et al., 2026; Mo et al., 2026; Savva et al., 2026; Sun et al., 2026a), increasing computation and inter-device communication as players are added. Generating views independently avoids this coupling, but each generator then has access only to its own observations and controls. It needs additional information to place other players correctly and reproduce scene content observed from other viewpoints. A distributed approach must therefore share both spatial information about players and generated scene appearance, without requiring the views to be generated together.

To provide this shared information while keeping generation independent, we propose **WorldCast**. Inspired by the organization of networked games (Funkhouser, 1995; Bernier, 2001), each player runs a client that maintains a local copy of the shared world state and generates its own view. We represent this state with two complementary components. **Player state** records players' positions and attributes, projected into a camera-aligned player state field that guides the rendering of other players. **Scene state** stores generated observations in a shared memory bank, supplying content missing from each client's recent context. Clients exchange these states once per video block while keeping intermediate features local. A fixed-resolution player state field and a bounded number of retrieved scene frames allow additional players without enlarging the generator's input.

Unlike in a networked game, WorldCast has no engine supplying positions and geometry at inference. Each client therefore includes a **state model**, trained with recorded positions and map geometry, that estimates its own position and scene depth from generated video and controls. Position estimates correct control-based extrapolation and update player state; depth supports player visibility checks and scene memory retrieval. After each block, clients publish these updates and share the generated observations, completing the loop between generation and state estimation.

We evaluate WorldCast on Counter-Strike 2 (Fig. 1). Given the same ground-truth (GT) player positions, our camera-aligned player field achieves a player rendering rate over ten times that of the joint-generation (Savva et al., 2026) and frame-concatenation baselines. This rate measures the proportion of visible players rendered at their expected positions. Predicted player positions retain much of this rendering rate over ten-second windows (Table 2). Shared scene state further improves visual consistency over whole rounds (Table 3). With scene state enabled, each client generates video at over 16 FPS across deployments of 2 to 16 clients (Fig. 6), including decoding but excluding synchronization waits. Image quality remains stable over hour-long rollouts.

We summarize our contributions as follows:

(1) We propose WorldCast, a distributed multiplayer world model whose clients each generate one player's view and share only a world state, which conditions each generator through a camera-aligned player state field and a scene memory bank.

(2) We develop a state model that estimates each player's position and scene depth from its generated video and controls, enabling clients to update the shared world state without a game engine supplying these quantities at inference.

(3) We demonstrate improved player rendering and whole-round visual consistency on Counter-Strike 2. Each client generates video in real time with 2 to 16 players, and image quality remains stable over hour-long rollouts.

2 RELATED WORK

Multiplayer World Models. Extending single-view world models (He et al., 2025b; Kanervisto et al., 2025), joint-generation methods denoise all views together (Hu et al., 2026a;b; Liu et al.,

2026; Mo et al., 2026; Savva et al., 2026; Sun et al., 2026a; Zhu et al., 2026), so views on different machines exchange attention features at every step (Appendix B); shared-screen methods bind each agent's actions to its character in one frame (Pondaven et al., 2026); shared-state methods condition views on a state advanced by actions, either centrally (Cai et al., 2026) or by per-player generators on a given map (Po et al., 2026), or encoded from all views' latest frames (Wu et al., 2026). WorldCast clients exchange only player state, estimated from generated video and controls, and scene state, a memory bank of generated blocks.

Controllable Video Generation. Structured conditioning provides spatial guidance for image and video generation. Object boxes can be encoded as additional tokens (Li et al., 2023; Gao et al., 2024; Russell et al., 2025), while image-aligned control maps convey edges or depth (Zhang et al., 2023), motion trajectories (Wang et al., 2025) or time-varying 3D states (Zheng et al., 2026). WorldCast applies such image-aligned conditioning to incorporate the player states exchanged between clients. Its state injector projects each visible player into the local camera and places both spatial and gameplay attributes at the corresponding generator tokens.

Autoregressive Video Diffusion. Autoregressive video diffusion models generate video incrementally with block-causal attention (Chen et al., 2024; Yin et al., 2025). Self Forcing (Huang et al., 2025b) trains on the model's own rollouts to reduce error accumulation, while distribution matching distillation (Yin et al., 2024a;b) enables few-step generators. To preserve scenes beyond a limited context window, memory-augmented models (Huang et al., 2025a; Li et al., 2025; Xiao et al., 2025; Yu et al., 2025; Nam et al., 2026; Sun et al., 2026b) retrieve their own past frames by field-of-view overlap, camera pose or 3D geometry. ShareVerse (Zhu et al., 2026) also retrieves other agents' frames by proximity within joint denoising. WorldCast combines four-step autoregressive generation with a memory bank shared across independent clients, allowing clients to reuse one another's generated content without jointly denoising their views.

3 METHOD

WorldCast lets multiple players inhabit one generated world, kept consistent through a shared world state. This state comprises player state, which describes players' positions and attributes, and scene state, which stores generated observations of the environment. We first introduce the distributed generation framework (Sec. 3.1), then describe how player state (Sec. 3.2) and scene state (Sec. 3.3) are maintained and incorporated into each generator. We next present the state model that estimates position and depth from generated video (Sec. 3.4), followed by training and inference (Sec. 3.5).

3.1 DISTRIBUTED GENERATION WITH SHARED WORLD STATE

To support scalable multiplayer video generation across machines, WorldCast separates shared world state from local rendering. As in Fig. 2(a), existing coupled designs coordinate players through joint multi-view generation that processes players' views together through cross-view attention or view concatenation. WorldCast instead assigns each of the P players an independent client that reads shared world state, generates its own view, and publishes updated state once per block (Fig. 2(b)). Each client runs on its own GPU and combines a video generator G_θ with a state model that estimates position and scene depth from generated video and controls. Only the shared world state is exchanged; activations, KV caches, and intermediate features remain local.

Consistent views require information beyond each client's recent context. The shared world state therefore comprises two complementary components: **player state**, a record of each player's position and attributes (Sec. 3.2), and **scene state**, a memory bank shared by all clients (Sec. 3.3). Player state tracks players outside the local view, while scene state preserves observations no longer in the recent context or available only from other clients. Together, these shared states help clients render players and scene appearance consistently across independently generated views.

Clients exchange state once per block and, in our evaluations, advance in lockstep (Appendix B). For generating block n, client i uses the shared states published after block $n-1$, extrapolating player positions and camera poses using the player controls. It then constructs a camera-aligned player state field and retrieves at most one memory entry from the memory bank. Specifically, for client i at block n, the generation is conditioned as

$$\mathbf{V}_n^i = G_\theta\big(\mathbf{a}_n^i,\ \mathbf{F}_n^i,\ \mathbf{M}_n^i,\ \mathbf{V}_{<n}^i\big), \tag{1}$$

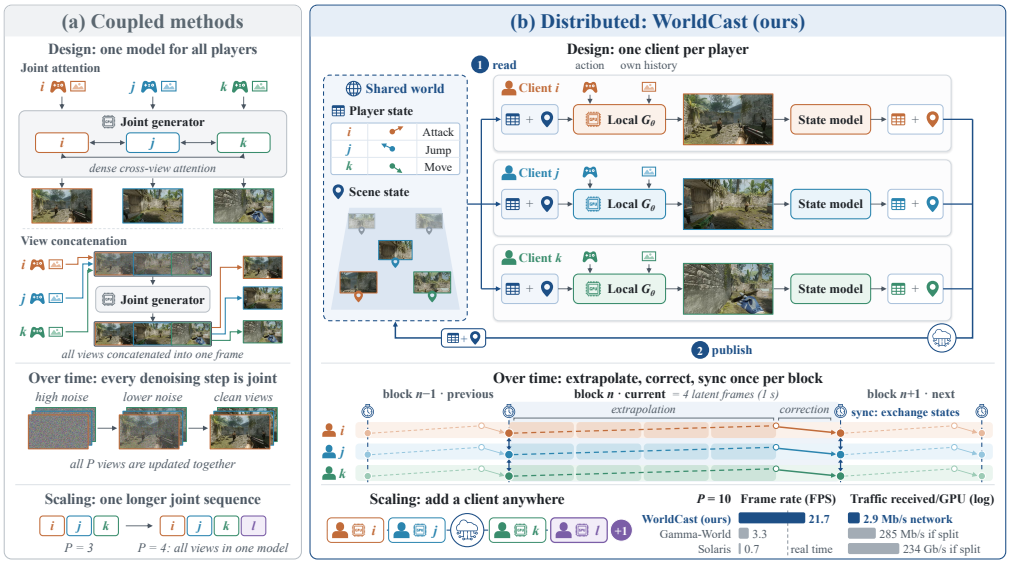

Figure 2: **(a)** Coupled designs generate all players' views in one model. **(b)** Each WorldCast player runs its own client, and clients exchange only the shared world state.

where $\mathbf{a}_n^i$ denotes the player's controls and attributes (e.g., weapon); $\mathbf{F}_n^i = \{\mathbf{F}_{n,f}^i\}_f$ is the player state field, formed by projecting other players' states onto client i's camera-aligned token grid for each latent frame f (Sec. 3.2); $\mathbf{M}_n^i$ contains the latent frames of the retrieved scene memory entry; and $\mathbf{V}_{<n}^i$ is the recent context. After generation, the state model (Sec. 3.4) estimates the player's position and scene depth from the generated latents and controls. The client publishes its updated player state and shares the generated block with its cameras and depth as a new memory entry.

The player state field has a fixed spatial resolution, and each generator reads at most one memory entry, regardless of the player count. Additional players can therefore be served by adding clients without enlarging each generator's attention sequence.

3.2 Shared Player State

Attributes and update. The shared player state contains one record $\mathbf{s}_n^i$ for each player i at block n, describing its position and other attributes, such as facing direction, controls, and weapon. Each record is updated and published only by that player's client. The position is estimated from generated video and controls by the state model (Sec. 3.4), while other attributes are obtained from the player's inputs. These records provide a common reference for rendering players across clients.

Camera-aligned player state field. Player states are expressed in world coordinates, whereas the generator operates on image tokens. The state injector bridges these representations by projecting other players into the observing client's camera and encoding their attributes at the corresponding token locations. Specifically, for each latent frame f, it constructs a player state field $\mathbf{F}_{n,f}^i$ on the generator's token grid. Each player within the view frustum and visible (Sec. 3.4) is projected into the observer's camera and splatted with a Gaussian kernel (Wang et al., 2025). The field encodes coverage, camera-relative depth and facing direction, and gameplay attributes such as team, identity, weapon, and controls. Contributions from overlapping players are combined according to their spatial weights and depths (Appendix F). This gives the generator explicit spatial guidance about where other players should appear and which attributes to render.

Injection into the generator. A convolutional stem Conv_0 with a zero-initialized output projection (Zhang et al., 2023) maps each $\mathbf{F}_{n,f}^i$ to the model width and adds it after an early DiT block:

$$\mathbf{h}_{f,u} \leftarrow \mathbf{h}_{f,u} + \mathrm{Conv}_0(\mathbf{F}_{n,f}^i)(u), \tag{2}$$

where $\mathbf{h}_{f,u}$ is the hidden state of token u in latent frame f.

3.3 SHARED SCENE STATE

Scene state composition. Shared scene state allows clients to reuse generated observations of the environment, helping maintain consistent scene appearance across views. It is represented as a memory bank, where each entry stores a block of latent frames generated by one client, together with their camera poses and coarse depth maps estimated by the state model (Sec. 3.4).

Scene memory update. Each client publishes every generated block as a new memory entry and maintains at most B of its own entries. When this capacity is exceeded, it withdraws the most redundant entry—the one whose content is best covered by the client's other entries under reprojection (Appendix E). Each entry therefore has a single owner responsible for its publication and removal, while remaining accessible to all clients.

Scene memory retrieval. Before generating block n, client i retrieves the memory entry $\mathbf{M}_n^i$ that best covers scene content missing from its recent context. It first back-projects the recent latent frames into 3D and projects these points into the cameras of the upcoming block. Pixels reached by none of these points are treated as missing. The client then projects the points of all memory entries into the same cameras. At each missing pixel, the nearest projected point to the camera defines the reference surface depth. An entry covers that pixel if one of its points lies within a depth tolerance of this surface, preventing occluded content from contributing to its coverage score. The client retrieves the entry covering the most missing pixels, or none if no entry covers any.

Conditioning the generator. As shown in Fig. 3 right, the latent frames of the retrieved memory entry $\mathbf{M}_n^i$ are prepended to the recent context frames from $\mathbf{V}_{<n}^i$ and the target frames $\mathbf{V}_n^i$. Memory frames attend only to memory frames, while recent and target frames can attend to them under the generator's block-causal attention mask. Besides, to express the geometry of observations from different viewpoints, we encode each token's viewing ray using Plücker coordinates relative to the first target frame's camera:

$$\mathbf{r}_{f,u} = \big(\mathbf{d}_{f,u},\ (\mathbf{o}_f/\ell) \times \mathbf{d}_{f,u}\big) \in \mathbb{R}^6, \qquad \mathbf{h}_{f,u} \leftarrow \mathbf{h}_{f,u} + \phi(\mathbf{r}_{f,u}), \tag{3}$$

where $\mathbf{d}_{f,u}$ is the unit viewing direction of token u in latent frame f, $\mathbf{o}_f$ is the camera center, and ℓ is a fixed length scale. The MLP ϕ maps these coordinates to an embedding added before the first DiT block. This enables the generator to use shared scene observations together with their relative camera geometry through its existing attention layers.

3.4 THE STATE MODEL

The shared player and scene states require positions and geometry that correspond to the generated world. At inference, these quantities are not supplied by a game engine. Each client therefore uses a state model to estimate its own position and scene depth from generated latents and controls (Fig. 3, right). Position estimates update the shared player state, while depth estimates support scene memory retrieval and player visibility.

State model architecture. As shown in Fig. 3 right, a convolutional encoder maps each latent frame to a token, an action encoder adds the corresponding control embedding, and a causal transformer processes the resulting sequence. A motion head predicts the player's displacement between consecutive latent frames. A place head provides an absolute position estimate on the current map by classifying the frame into a map cell and predicting its offset within the cell (Appendix A). A separate depth head predicts a coarse depth map from the generated latents.

Position estimation. The motion head captures displacement from changes in the generated frames, including stops at obstacles, but integrating its displacements accumulates error. The place head provides a map-based reference, including height, but can occasionally mis-localize a frame. We combine their estimates with a complementary filter:

$$\mathbf{p}_f = \tfrac{1}{2}\left(\mathbf{p}_{f-1} + \mathbf{\Delta}_f\right) + \tfrac{1}{2}\,\mathbf{A}_f, \tag{4}$$

where $\mathbf{p}_f$ is the position estimate at latent frame f, $\mathbf{\Delta}_f$ is the motion-head displacement, and $\mathbf{A}_f$ is the place-head estimate, which is the offset to start at the given initial position $\mathbf{p}_0$. After each completed block, the client publishes the position estimates, together with its other attributes, as the updated player state. Before generating the next block, each client extrapolates its camera poses and other players' positions from their latest published player states by integrating movement and view

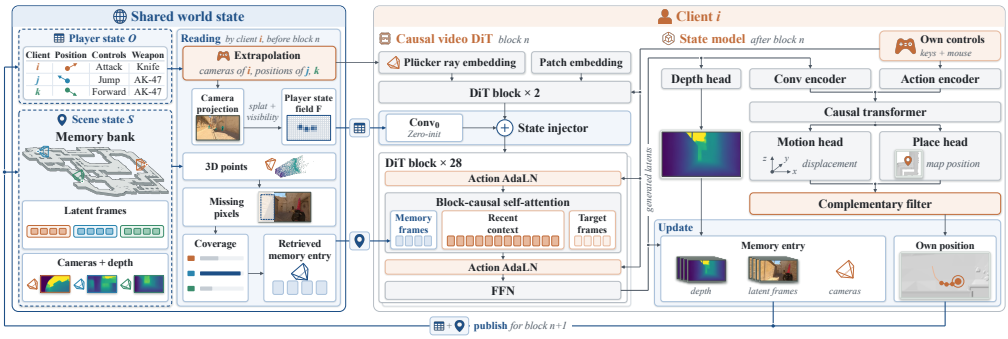

Figure 3: A client and the shared world state. **Left:** player states are projected into the camera-aligned player state field, and each generated block is added to the memory bank of the scene state, from which the next block retrieves one memory entry. **Right:** the causal video DiT of client i, where the player state field enters after the second DiT block, the memory frames join the recent and target frames, and every frame carries Plücker rays; after block n, the state model estimates the player's position and the depth of each generated frame, which are published for block $n+1$.

controls, as networked games do between server updates (Bernier, 2001). These predictions guide player state field construction and scene memory retrieval. Since extrapolation ignores collisions and terrain height, it may diverge from the motion depicted in the generated video. Therefore, once the block is complete, the state model uses the new frames and Eq. (4) to correct the client's own position for the next state update.

Depth and visibility. The predicted depth maps and camera poses back-project generated observations into 3D points in a common map coordinate system. These points support both scene memory retrieval (Sec. 3.3) and visibility gating for the player state field (Sec. 3.2). Before generation, the client projects points from its recent context into the cameras of the upcoming block. Their depths determine whether other players are occluded: a player is considered visible when context points reach its projection and their depth lies no more than a fixed margin in front of it (Appendix F). Since the context may not cover newly exposed regions, the depth head processes the clean estimate $\hat{\mathbf{x}}_0$ after the first denoising step. This depth updates visibility for the remaining steps.

State model training. The motion and place heads are supervised with recorded player positions, and the depth head is supervised with depth maps rendered from the game's public 3D map assets. These supervision targets are used only in training (Appendix A).

3.5 Training and Inference

Training stages. We train G_θ in four stages (Appendix Fig. 8). Stage 1 fine-tunes a pretrained video diffusion model on Counter-Strike 2 with bidirectional attention on short clips. Stage 2 conditions G_θ on the shared world state, adding the player state field, then scene state. Player states are GT states taken from recorded matches, and visibility labels are computed from the game's public 3D map assets. Memory frames are GT frames of one block from another player's view in the same round, selected to cover what the recent context misses (Appendix E). Stages 3 and 4 make generation real-time: stage 3 converts G_θ to block-causal attention (Chen et al., 2024; Yin et al., 2025) through teacher forcing, and stage 4 distills it to a four-step generator by distribution matching (Yin et al., 2024a;b), trained on its own rollouts to reduce accumulated error (Huang et al., 2025b).

Objective. Stages 1 to 3 minimize the weighted flow-matching loss:

$$\mathcal{L} = \mathbb{E}_{t,\boldsymbol{\epsilon}}\Big[\sum_k \alpha_k\, c_k\, \big\|\mathbf{v}_\theta(\mathbf{x}_t, t)_k - \mathbf{v}_k\big\|_2^2\Big], \tag{5}$$

where $\mathbf{x}_t$ is the latent noised to level t with noise $\boldsymbol{\epsilon}$, k indexes latent positions, and $\mathbf{v}_\theta$ predicts the target velocity $\mathbf{v}$ using the conditioning in Eq. (1). Two weights emphasize regions that a uniform loss would underweight. From stage 2 onward, the foreground weight α_k increases the loss at positions occupied by visible players, with larger weights for smaller projected players; elsewhere, $\alpha_k = 1$. For memory-conditioned samples, only target frames are supervised. The memory weight c_k emphasizes target regions covered by the memory frames but absent from the recent context, and is normalized to have unit mean within each target frame. Without memory frames, $c_k = 1$.

Inference. At inference, each client executes the generation cycle of Sec. 3.1 with the four-step generator. The completed block is decoded, and the client updates and publishes its player state and a new memory entry for subsequent generation.

4 EXPERIMENTS

Datasets. On four Counter-Strike 2 maps of OpenCS2 (Blanchon, 2026), split by match into 303 training and 40 held-out matches, we evaluate on 256 ten-second windows and on 32 whole rounds generated concurrently by three clients of one team (five in Table 4a), the other players following the recording. Unless marked as predicted, methods receive GT player positions from the recording and visibility from map geometry, so that errors of the state model do not confound the comparison.

Metrics. We report *FVD*, *PSNR*, *SSIM*, and *LPIPS* for video quality. Using YOLO11 (Jocher & Qiu, 2024), *Rendered* (the rendering rate) measures the fraction of visible players detected in the recording that are rendered within one body width of their state-specified positions. *Unmatched* measures the fraction of generated detections without a nearby match to a visible player or a recorded detection. Both are medians over views of ten-second windows (Appendix D). Whole-round rollouts drift from the recording, so we measure consistency between two clients with overlapping views at the same time (Appendix E). *Mat. Pix.* (Bai et al., 2025) counts pixels matched by GIM (Shen et al., 2024). *Inliers* counts LoFTR (Sun et al., 2021) matches consistent with one relative pose, and *RotErr* (He et al., 2025a) is its rotation error. *Depth* is the fraction of pixels where both clients' predicted depths agree. *VBench* measures visual quality over hour-long rollouts.

Implementation. G_θ is initialized from Wan2.2-TI2V-5B (Wan et al., 2025); a block is 4 latent frames (1 s at 16 fps), and the recent context holds 12 frames.

4.1 MAIN RESULTS

Bidirectional Model Results and Comparisons. We compare against Solaris (Savva et al., 2026), which uses joint cross-view attention, and frame concatenation. For a controlled comparison, we retrain both with the same backbone, training data, budget, and GT player positions. Our bidirectional model uses only shared player state in this comparison. As shown in Table 1, WorldCast outperforms both baselines across all reported metrics. Its player rendering rate reaches 0.817, compared with 0.056 for Solaris and 0.074 for frame concatenation, while its unmatched rate falls to 0.039. These gains demonstrate more accurate player placement alongside improved video quality (also shown in Fig. 4).

AR Model Results and State Model in the Loop. To test whether the state model can replace GT player positions, every client renders the other players at the positions and visibility predicted by the state models (Table 2). The render rate stays close to that with GT player positions, with most visible players still drawn as expected, although the unmatched rate rises from 0.031 to 0.093.

Scene State over Whole Rounds. WorldCast improves PSNR, SSIM and LPIPS after 30 s and over whole rounds (Table 3) over every design choice except GT frames in the memory bank, which no client has; before 30 s, the model trained without scene state is on par (Table 3).

Table 1: Bidirectional multiplayer generation on ten-second windows. All methods use the same backbone, training data, budget, and GT player positions.

	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Rendered ↑	Unmatched ↓
Frame concatenation	48.2	15.37	0.493	0.460	0.074	0.771
Solaris	49.0	15.66	0.501	0.449	0.056	0.739
WorldCast	**39.7**	**16.17**	**0.507**	**0.418**	**0.817**	**0.039**

Table 2: Four-step AR generation results on ten-second windows and effect of predicted states. Both settings use shared player state and scene state. Joint-generation methods have no AR version.

	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Rendered ↑	Unmatched ↓
WorldCast (GT positions)	47.5	16.03	0.497	0.423	0.897	0.031
WorldCast (Predicted positions)	53.0	15.65	0.490	0.443	0.830	0.093

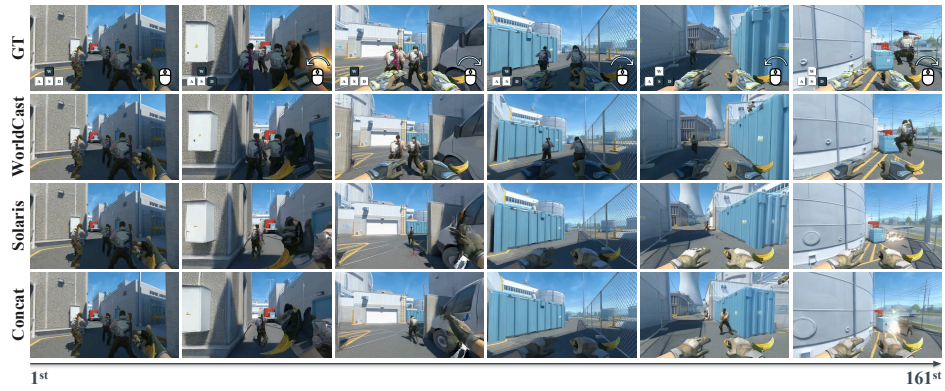

Figure 4: Ten seconds of a held-out round, generated from the first frame and GT player positions (GT: recording with controls; Concat: frame concatenation).

Table 3: Scene state over whole rounds. We use GT player positions to exclude the impact of imprecise player position in rendering performance.

	First 30 s			After 30 s		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Trained without scene state	**14.93**	**0.439**	**0.507**	12.96	0.387	0.603
Scene state off at inference	14.37	0.424	0.545	12.93	0.391	0.615
Only its own earlier blocks	14.56	0.425	0.539	13.53	0.399	0.591
Least-covering memory	14.27	0.421	0.550	12.70	0.380	0.628
WorldCast	14.88	0.431	0.521	**14.16**	**0.409**	**0.556**
GT frames in the memory bank	15.17	0.442	0.504	14.91	0.428	0.517

State Estimation. On rollout frames, the state model estimates positions more accurately than integrating the controls and keeps its error low over a minute (Appendix Table 6; Fig. 5).

Closed-Loop Deployment. Closed-loop deployment gives each client only its recorded first frame and position (Fig. 7). After each block, it publishes its estimated position, renders other players at their published positions, and adds the block to the scene state. The scene state keeps clients more consistent with each other than with it off at inference: all four agreement measures improve (Table 4a). In one-hour rollouts, VBench scores change little (Table 4b). With one GPU per player, each client exceeds 16 FPS for 2 to 16 players and receives only the others' player states and memory entries (a few Mb/s), while Gamma-World (Liu et al., 2026) and Solaris, smaller and hosting all players on one GPU as released, fall below 16 FPS from two players (Fig. 6).

4.2 Ablation Studies

In the player state field (Table 5), camera alignment matters most: without it, the render rate falls to that of joint generation, and every other retrained variant also lowers it; disabling the visibility test at inference draws occluded players and raises the unmatched rate. For the scene state, retrieving any client's best-covering entry (our method) achieves the best results after 30 s (Table 3).

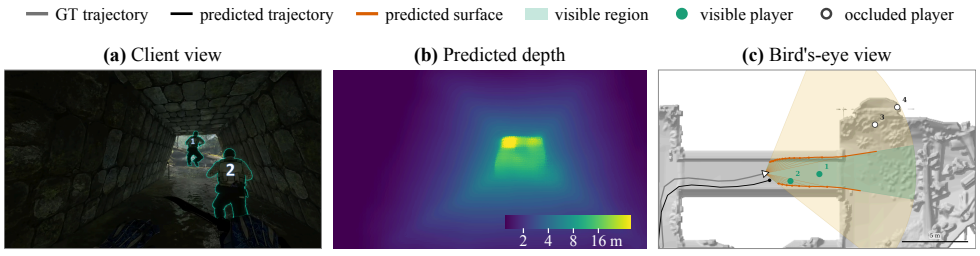

Figure 5: State model on a recorded frame: **(a)** frame, **(b)** predicted depth, **(c)** predicted and GT trajectories and visibility over the map.

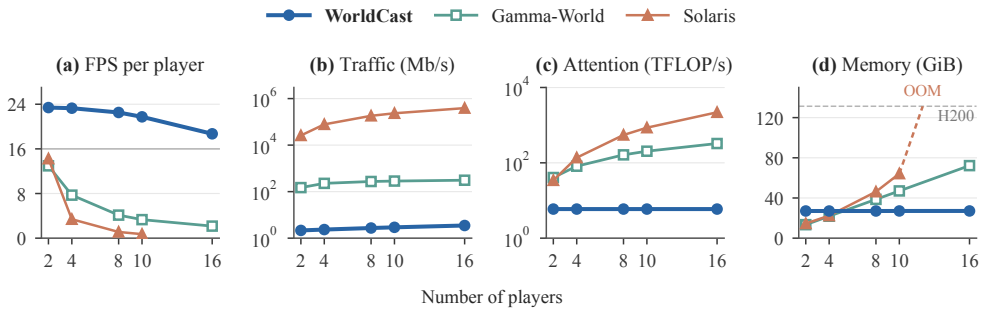

Figure 6: Serving P players on H200 GPUs: **(a)** FPS per player with decoding, **(b)** traffic received, **(c)** attention compute, **(d)** memory per GPU; Gamma-World and Solaris run on one GPU as released ((b): one player per GPU).

Table 4: **(a)** Closed-loop deployment over whole rounds, scene state off at inference and on: agreement between clients. **(b)** VBench (Huang et al., 2024) over one-hour rollouts.

	First 30 s		After 30 s	
(a) Scene state	Off	On	Off	On
Mat. Pix. (K) ↑	72.0	**92.1**	31.1	**44.3**
Inliers ↑	46.2	**51.5**	18.8	**27.7**
RotErr (°) ↓	31.8	**29.0**	59.1	**55.6**
Depth ↑	0.243	**0.247**	0.092	**0.120**

	Minutes into the hour		
(b) VBench	0–20	20–40	40–60
Subject consistency ↑	0.803	0.797	0.797
Background consistency ↑	0.910	0.912	0.910
Imaging quality ↑	70.3	70.3	69.2
Aesthetic quality ↑	0.486	0.489	0.486

Table 5: Ablations of the player state field (stage 2, ten-second windows, 6,000 training steps each).

	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Rendered ↑	Unmatched ↓
WorldCast	45.7	15.54	0.495	0.455	**0.763**	0.076
Trained without camera alignment	72.2	15.17	0.490	0.479	0.093	0.826
Trained without visibility	45.8	15.67	0.498	0.449	0.667	0.088
Trained without foreground weight	43.9	15.71	0.500	0.446	0.691	**0.060**
Trained with late injection	47.2	15.52	0.499	0.456	0.549	0.203
Trained with a coarse field	41.9	15.69	0.497	0.445	0.708	0.083
Field off at inference	73.2	15.34	0.491	0.467	0.000	0.333
Visibility off at inference	60.8	15.15	0.486	0.475	0.748	0.415



Figure 7: Closed-loop deployment: a whole round generated concurrently, each client on its own GPU and starting from its first recorded frame.

5 Conclusion and Limitations

WorldCast explores a world model as the engine of a networked game, in which each client generates one player's view on its own GPU and clients exchange only the shared world state. Our method attains a player rendering rate over an order of magnitude higher than joint generation, and scene state improves whole-round consistency. The main limitation is the state model's position error on generated video, which grows with time (Appendix G).

REPRODUCIBILITY STATEMENT

Rendered and Unmatched are specified in Sec. 4, with their eligibility and matching conventions in Appendix D; Appendix F gives the field's channels, splatting, composition and injection with numerical settings. The state model's inputs and supervision are in Sec. 3.4, and its training order and evaluation in Appendix A; the scene state's storage and retrieval are in Sec. 3.3 and Appendix E. The deployment protocol, hardware, and per-client measurements are in Sec. 4.1 and Appendix B, with the released baselines timed in Appendix Table 7. Training stages and the amount of training data for every reported model are in Sec. 3.5 and Appendix C, with the GPU-hours of the three models of Table 1. Tables 10, 11, 12, 18, 19 and 20 report paired differences and whether their 95% bootstrap intervals (10,000 draws over windows or rounds, which resample data, not training runs) exclude zero; the other tables report point estimates. Appendix G states their conditions and limits.

AI USE STATEMENT

We used a large language model to aid and polish the writing of this manuscript: revising prose, condensing sections to length, checking citations against their sources, and formatting tables and figures from result files. Every sentence, number and citation was checked by the authors against the underlying experiments and sources, and the authors take full responsibility for the content of this work.

REFERENCES

Jianhong Bai, Menghan Xia, Xintao Wang, Ziyang Yuan, Zuozhu Liu, Haoji Hu, Pengfei Wan, and Di Zhang. SynCamMaster: Synchronizing multi-camera video generation from diverse viewpoints. In *ICLR*, 2025.

Daniel Barath, Jana Noskova, Maksym Ivashechkin, and Jiri Matas. MAGSAC++, a fast, reliable and accurate robust estimator. In *CVPR*, 2020.

Yahn W. Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *GDC*, 2001.

Julien Blanchon. OpenCS2 dataset, 2026. URL `https://huggingface.co/datasets/blanchon/opencs2_dataset`.

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *ICML*, 2024.

Ziqi Cai, Siqi Yang, Yimu Wang, Zixian Gao, Yunheng Liu, Shuchen Weng, Erwin Wu, Kaipeng Zhang, and Boxin Shi. MASS: Multiplayer world models with authoritative shared state. arXiv preprint arXiv:2608.06257, 2026.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion Forcing: Next-token prediction meets full-sequence diffusion. In *NeurIPS*, 2024.

Thomas A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *I3D*, 1995.

Ruiyuan Gao, Kai Chen, Enze Xie, Lanqing Hong, Zhenguo Li, Dit-Yan Yeung, and Qiang Xu. MagicDrive: Street view generation with diverse 3D geometry control. In *ICLR*, 2024.

Hao He, Yinghao Xu, Yuwei Guo, Gordon Wetzstein, Bo Dai, Hongsheng Li, and Ceyuan Yang. CameraCtrl: Enabling camera control for video diffusion models. In *ICLR*, 2025a.

Xianglong He, Chunli Peng, Zexiang Liu, Boyang Wang, Yifan Zhang, Qi Cui, Fei Kang, Biao Jiang, Mengyin An, Yangyang Ren, et al. Matrix-Game 2.0: An open-source real-time and streaming interactive world model. arXiv preprint arXiv:2508.13009, 2025b.

Anthony Hu, Václav Volhejn, Adrien Ramanana Rahary, Chris Mulder, Aditya Makkar, Alyx Liao, Amélie Royer, Manu Orsini, Adam Jelley, Eloi Alonso, et al. Multiplayer interactive world models with representation autoencoders. arXiv preprint arXiv:2607.05352, 2026a.

Teng Hu, Mingchun Lu, Yating Wang, Jiangning Zhang, Jinkun Hao, Ye Pan, Ran Yi, Lizhuang Ma, and Dacheng Tao. MetaWorld: Scaling multi-agent video world model from single-view video data. arXiv preprint arXiv:2606.02753, 2026b.

Junchao Huang, Xinting Hu, Boyao Han, Shaoshuai Shi, Zhuotao Tian, Tianyu He, and Li Jiang. Memory Forcing: Spatio-temporal memory for consistent scene generation on Minecraft. arXiv preprint arXiv:2510.03198, 2025a.

Junchao Huang, Guian Fang, Shengju Qian, Xianghao Kong, Zhuoran Zhao, Wei Huang, Yihua Du, Zixin Zhang, Justin Cui, Yuchao Gu, et al. SolarWM: Open data and scalable training for long-horizon video world models. arXiv preprint arXiv:2609.02886, 2026.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self Forcing: Bridging the train-test gap in autoregressive video diffusion. In *NeurIPS*, 2025b.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. VBench: Comprehensive benchmark suite for video generative models. In *CVPR*, 2024.

Glenn Jocher and Jing Qiu. Ultralytics YOLO11, 2024. URL `https://github.com/ultralytics/ultralytics`.

Anssi Kanervisto, Dave Bignell, Linda Yilin Wen, Martin Grayson, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Tabish Rashid, Tim Pearce, Yuhan Cao, et al. World and human action models towards gameplay ideation. *Nature*, 2025.

Runjia Li, Philip Torr, Andrea Vedaldi, and Tomas Jakab. VMem: Consistent interactive video scene generation with surfel-indexed view memory. In *ICCV*, 2025.

Yuheng Li, Haotian Liu, Qingyang Wu, Fangzhou Mu, Jianwei Yang, Jianfeng Gao, Chunyuan Li, and Yong Jae Lee. GLIGEN: Open-set grounded text-to-image generation. In *CVPR*, 2023.

Fangfu Liu, Kai He, Tianchang Shen, Tianshi Cao, Sanja Fidler, Yueqi Duan, Jun Gao, Igor Gilitschenski, Zian Wang, and Xuanchi Ren. Gamma-World: Generative multi-agent world modeling beyond two players. arXiv preprint arXiv:2605.28816, 2026.

Sicheng Mo, Yuheng Li, Ziyang Leng, Krishna Kumar Singh, and Bolei Zhou. Streaming multi-agent autoregressive diffusion model with world state registers. arXiv preprint arXiv:2607.21594, 2026.

Jisu Nam, Yicong Hong, Chun-Hao Paul Huang, Feng Liu, JoungBin Lee, Jiyoung Kim, Siyoon Jin, Yunsung Lee, Jaeyoon Jung, Suhwan Choi, et al. WorldCam: Interactive autoregressive 3D gaming worlds with camera pose as a unifying geometric representation. arXiv preprint arXiv:2603.16871, 2026.

Ryan Po, David Junhao Zhang, Amir Hertz, Gordon Wetzstein, Neal Wadhwa, and Nataniel Ruiz. MultiGen: Level-design for editable multiplayer worlds in diffusion game engines. arXiv preprint arXiv:2603.06679, 2026.

Alexander Pondaven, Ziyi Wu, Igor Gilitschenski, Philip Torr, Sergey Tulyakov, Fabio Pizzati, and Aliaksandr Siarohin. ActionParty: Multi-subject action binding in generative video games. In *ECCV*, 2026.

Lloyd Russell, Anthony Hu, Lorenzo Bertoni, George Fedoseev, Jamie Shotton, Elahe Arani, and Gianluca Corrado. GAIA-2: A controllable multi-view generative world model for autonomous driving. arXiv preprint arXiv:2503.20523, 2025.

Georgy Savva, Oscar Michel, Daohan Lu, Suppakit Waiwitlikhit, Timothy Meehan, Dhairya Mishra, Srivats Poddar, Jack Lu, and Saining Xie. Solaris: Building a multiplayer video world model in Minecraft. arXiv preprint arXiv:2602.22208, 2026.

Xuelun Shen, Zhipeng Cai, Wei Yin, Matthias Müller, Zijun Li, Kaixuan Wang, Xiaozhi Chen, and Cheng Wang. GIM: Learning generalizable image matcher from internet videos. In *ICLR*, 2024.

Huiqiang Sun, Zhan Peng, Size Wu, Kun Wang, Kang Liao, Dianyi Wang, Xingyu Zeng, Sheng Jin, Yangguang Li, Zhiguo Cao, et al. Prisma-World: Camera-controllable multi-agent video world model. arXiv preprint arXiv:2606.09507, 2026a.

Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. LoFTR: Detector-free local feature matching with transformers. In *CVPR*, 2021.

Wenqiang Sun, Haiyu Zhang, Haoyuan Wang, Junta Wu, Zehan Wang, Zhenwei Wang, Yunhong Wang, Jun Zhang, Tengfei Wang, and Chunchao Guo. WorldPlay: Towards long-term geometric consistency for real-time interactive world modeling. In *ICML*, 2026b.

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines. In *ICLR*, 2025.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. arXiv preprint arXiv:2503.20314, 2025.

Angtian Wang, Haibin Huang, Jacob Zhiyuan Fang, Yiding Yang, and Chongyang Ma. ATI: Any trajectory instruction for controllable video generation. arXiv preprint arXiv:2505.22944, 2025.

Haoyu Wu, Jiwen Yu, Yingtian Zou, and Xihui Liu. MultiWorld: Scalable multi-agent multi-view video world models. arXiv preprint arXiv:2604.18564, 2026.

Zeqi Xiao, Yushi Lan, Yifan Zhou, Wenqi Ouyang, Shuai Yang, Yanhong Zeng, and Xingang Pan. WorldMem: Long-term consistent world simulation with memory. In *NeurIPS*, 2025.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Frédo Durand, and William T. Freeman. Improved distribution matching distillation for fast image synthesis. In *NeurIPS*, 2024a.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Frédo Durand, William T. Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *CVPR*, 2024b.

Tianwei Yin, Qiang Zhang, Richard Zhang, William T. Freeman, Frédo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *CVPR*, 2025.

Jiwen Yu, Jianhong Bai, Yiran Qin, Quande Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Xihui Liu. Context as Memory: Scene-consistent interactive long video generation with memory retrieval. In *SIGGRAPH Asia*, 2025.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *ICCV*, 2023.

Sixiao Zheng, Minghao Yin, Wenbo Hu, Xiaoyu Li, Ying Shan, and Yanwei Fu. VerseCrafter: Dynamic realistic video world model with 4D geometric control. In *CVPR*, 2026.

Jiayi Zhu, Jianing Zhang, Yiying Yang, Wei Cheng, and Xiaoyun Yuan. ShareVerse: Multi-agent consistent video generation for shared world modeling. arXiv preprint arXiv:2603.02697, 2026.

APPENDIX CONTENTS

A State model in detail 13
B Distributed deployment in detail 15
C Training stages and scaling 16
D Evaluation protocols and long horizons in detail 19
E Scene state in detail 22
F The player state field in detail 24
G Limitations and scope 26
H Additional qualitative comparisons 27

Throughout the appendix, *GT states* are the recorded player states, whose positions are the GT player positions of Sec. 4, with visibility from the map geometry; *predicted states* replace these positions and the visibility with those predicted by the state model.

A STATE MODEL IN DETAIL

This appendix details the state model of Sec. 3.4. Its *motion head* predicts the player's displacement between consecutive latent frames, its *place head* recognizes where on the map a frame was taken, and the complementary filter of Eq. (4) advances one estimate by the former and corrects it with the latter.

Components. The state model reads windows of 41 noise-free latent frames ($48 \times 24 \times 42$) of the client's own video; one latent frame spans four video frames at 16 fps, or sixteen engine ticks. Five convolutional blocks (three strided) map each latent frame from 48 to 512 channels, and a linear projection yields a token of width 1280. A causal transformer trunk (16 layers, 16 heads, pre-norm, $4\times$ feed-forward) processes these tokens, and a linear motion head on its outputs predicts the three-dimensional displacement. The sixteen control values per tick (thirteen buttons, pitch and yaw deltas, a validity flag) are projected to 256 dimensions, encoded by the action encoder (a four-layer causal transformer) and added to the trunk tokens. The place head takes detached trunk tokens, a pooled early feature map, and the unpooled 12×21 grid, concatenates a map embedding, and outputs an address (below). The trunk and action encoder have 338M parameters and the motion and place heads 11M.

Address. Positions are in the game engine's unit of length, u (not the token index u of Appendix F); a player's body is 32 u (0.61 m) wide. To predict position as a class, each map is tiled into cells of $256 \times 256 \times 64$ u, each divided $4 \times 4 \times 1$ into sub-cells; a cell, a sub-cell, and an offset of at most ± 32 u from the sub-cell center represent a position exactly. The place head predicts the cell, then the sub-cell conditioned on the cell's 64-dimensional embedding, then the offset through a tanh. A map has at most 8,768 sub-cells in its occupied cells.

Complementary filter. Eq. (4), used for all results with predicted states, weights the integrated position and the place head's estimate equally at every latent frame and has no learned parameters. The estimate is initialized at the player's GT position at the start of the round (or of the window, for ten-second windows). The place head's estimates enter as displacements from its first estimate, so its constant bias cancels.

Within a block. As in networked games, each client extrapolates the motion of all players locally and is authoritative for its own player. The cameras of a block's four latent frames and the other players' positions over it are extrapolated from the controls (Bernier, 2001), since the block has no frames yet. Clients exchange player state once per block (Sec. 3.1); with predicted states, block n starts from the positions published when it begins, each player's estimate at the last latent frame of block $n-1$. Each player's movement keys are integrated at 64 Hz with per-key walking speeds, without collisions, and the facing direction accumulates the view controls from the last known facing direction. After the block, each client corrects only its own player's position, with the state model

and Eq. (4) on its new frames, and publishes it; every client then extrapolates over the next block from these corrected positions. With GT states, the GT cameras and positions are used.

Depth head. The depth head, a separate 44M-parameter residual network with ten blocks at width 384 and a five-block half-resolution branch, predicts the depth-video latent of the client's own view from three latent frames of context (generated at inference), supervised offline by depth rendered from the public 3D map assets. A 0.21M-parameter read-out maps this latent, without VAE decoding, to one depth value per 16×16 pixel patch, on which visibility (Sec. 3.4) is computed; Fig. 5 shows the prediction decoded at full resolution on a recorded frame. The map assets contain no players, so the depth excludes occlusion between players. On held-out recorded frames of the clients of Table 19, the read-out depth differs from the rendered map depth by a median factor of 1.07, with 85% of patches within a factor of 1.25. On generated frames, its effect is measured through retrieval (Table 18).

Training. All components are trained on rounds disjoint from both evaluation sets, with AdamW, a one-cycle schedule, and weight decay 0.01. The trunk and action encoder are trained with an L_1 displacement loss for 80,000 steps at batch 16 (about four hours of training) on 150k windows from 8,317 rounds; the place head is then trained on the same windows, with the trunk frozen, for 60,000 steps at batch 16, both at a learning rate of 3×10^{-4}. The depth head is trained for 6,000 steps at a global batch of 192 on 32 GPUs, on 283k windows from 10,469 rounds, with a learning rate of 4×10^{-4} and an L_1 loss in latent space against the rendered map depth video, encoded by the same VAE. Per one-second block, the trunk with its heads takes 17 ms and the depth head with its read-out about 12 ms.

Contribution of each component. Table 6 compares the state model with extrapolation from the controls, which networked games use between server updates (Bernier, 2001), and isolates its motion and place heads. The place head anchors the global position, height included. On recorded video the state model keeps the median error near 0.5 body widths over the minute, with 87% of clients within two body widths at 60 s, whereas extrapolation drifts to 7.6 and leaves 2%; it does not know the height of the ground either, erring in height by a median of 3.4 body widths over the minute against 0.06 for the state model. On video generated with GT states, the state model's error, which includes each rollout's departure from the GT trajectory, grows to 2.6 at 50–60 s, a third of that of extrapolation. The motion head reads local motion from the frames, such as a stop against a wall: in the Blocked windows it misplaces the one-second displacement by 0.05 body widths on recorded and 0.38 on generated video, against 2.38 and 2.42 for extrapolation, although alone it drifts, to 2.0 and 22.7 at 50–60 s. Eq. (4) combines the two. Over the minute it is on par with the place head alone, which is slightly ahead on generated video (2.51 against 2.64 body widths at 50–60 s); per latent frame, the step at which the complementary filter and the player state field are updated, its displacement error is about half that of the place head alone on recorded video, 0.21 against 0.45 body widths, and a third lower on generated video, 0.45 against 0.69.

Table 6: Position error in body widths (32 u, 0.61 m) of the clients of the whole rounds of Table 3, all alive at 60 s, on recorded video and on video generated by WorldCast with GT states: per-client medians over each ten-second interval, then medians over clients, the share of clients within two body widths at 60 s, and, as Blocked, the median error of the one-second displacement over the one-second windows in which the controls imply more than one body width of motion and the GT position moves less than a third of it. Bold marks the better of the state model and extrapolation; the last two rows of each part isolate the motion and place heads of the state model. Extrapolation does not read the video; its two rows differ only in the instants at which the error is evaluated.

	0–10 s ↓	50–60 s ↓	Within two body widths at 60 s ↑	Blocked ↓
Recorded video				
State model, Eq. (4)	**0.46**	**0.49**	**0.87**	**0.19**
Extrapolation from the controls	1.86	7.63	0.02	2.38
Place head only	0.56	0.56	0.87	0.24
Motion head only	0.39	2.00	0.53	0.05
Generated video, WorldCast				
State model, Eq. (4)	**1.49**	**2.64**	**0.39**	**1.91**
Extrapolation from the controls	1.86	7.60	0.03	2.42
Place head only	1.57	2.51	0.42	1.80
Motion head only	1.20	22.74	0.00	0.38

B DISTRIBUTED DEPLOYMENT IN DETAIL

This appendix specifies the client protocol, the measurements behind Fig. 6 (Table 7), the attention patterns of the joint-generation methods, and the cost of a client over a long stream and of training a joint-generation method.

Deployment protocol. Every client, once per block, publishes one message and reads the latest message of every other client, without a central process that merges them. For block n, client j (i) publishes its player state $\mathbf{s}_n^j$, that is, the positions that Eq. (4) estimates at the four latent frames of block $n-1$, the facing direction, and the controls and weapon for block n; with scene state, also block $n-1$ as a memory entry (four cameras and depths) and the memory entries it withdraws; and (ii) generates block n: it extrapolates the other players from their latest player states over the block (Appendix A), applies the withdrawals and retrieves at most one memory entry completed before block n, whose latents it fetches from the client that generated it. A message takes 3.7 kB for the player state and 8.3 kB for a memory entry (cameras in single precision, a $4 \times 24 \times 42$ depth in half precision); fetching the latents of a memory entry takes 387 kB. With GT states, player states come from the recording, and messages carry only the scene state. In our evaluations, each client runs on its own GPU, messages and latents are exchanged through a shared file system (for the timings of Table 7 with $P \leq 8$, through shared memory on one machine), and the clients advance in lockstep, so that every client reads the messages of the same block.

Joint-generation methods. In *Solaris*, each player attends to the windows of all players in one joint attention; *frame concatenation* attends over the same tokens with the players' frames interleaved in time; Gamma-World's *sparse hub* adds eight hub tokens per frame that attend to every player, while each player attends to its own window and the hubs. As released, Solaris and Gamma-World generate every player's view on one GPU, and we time them unchanged, all players on one GPU; frame concatenation is retrained only with our backbone (Appendix C) and is not timed. Joint attention reads the keys and values of all players at every layer, so, placed one player per GPU, they would exchange keys and values at every layer of every denoising step (Table 7). WorldCast places each player on its own GPU.

Long streams. Because the attention window is bounded, a client's time per block and GPU memory footprint do not keep growing with the length of a stream. Over a three-minute stream of a single client with scene state and CUDA graphs, a block whose window holds memory frames takes 633 ms in the first minute and 654 and 650 ms in the second and third, the increase being mostly the writes to the memory bank once the pool of the client's own memory entries is full; the footprint grows

Table 7: Serving P players on H200 GPUs. WorldCast runs one client per player, each on its own GPU, with scene state, as deployed with CUDA graphs, which reproduce its output bit for bit. All P clients run concurrently and exchange messages and memory entries, through shared memory on one machine for $P \leq 8$ and through a shared file system across two machines for $P = 10$ and 16. A client's frame rate counts its own time per block and excludes the time it waits for the other clients at the lockstep of our evaluations, since, without lockstep, a client reads the latest messages (Appendix B); the table reports the median over clients and blocks. Gamma-World (2.2B parameters, 320×480; its distilled four-step causal student with a KV cache, the fastest configuration it releases) and Solaris (1.8B, 352×640) run as released, with random weights, all players on one GPU, which is their design. FPS per player includes decoding; memory per GPU is the peak of each framework's allocator, for WorldCast the largest over clients. At $P = 16$, a WorldCast client receives 3.5 Mb/s on average: per block, a 12.0 kB message from each other client and, when it retrieves another client's memory entry, that entry's 387 kB of latents. Traffic is received per GPU; for Gamma-World and Solaris it is that of splitting the players one per GPU. Attention compute is the attention FLOPs needed to generate one second of video at 16 fps for all players on the GPU, computed analytically from the query–key products, $2d$ operations per pair, d being the model width; *the configuration runs out of memory. OOM: the configuration exceeds the memory of one H200.

Players P	2	4	8	10	16
FPS per player, including decoding ↑					
WorldCast, one GPU per player	23.4	23.3	22.5	21.7	18.7
Gamma-World, one GPU	13.0	7.7	4.1	3.3	2.2
Solaris, one GPU	14.2	3.4	1.1	0.7	OOM
Memory per GPU, GiB					
WorldCast, one GPU per player	27.1	27.0	27.0	27.1	27.1
Gamma-World, one GPU	13.4	21.9	38.6	47.0	72.2
Solaris, one GPU	14.6	22.4	46.2	64.2	OOM
Traffic received per GPU ↓					
WorldCast, one GPU per player	2.2 Mb/s	2.3 Mb/s	2.7 Mb/s	2.9 Mb/s	3.5 Mb/s
Gamma-World, one GPU per player	149 Mb/s	227 Mb/s	273 Mb/s	285 Mb/s	310 Mb/s
Solaris, one GPU per player	26 Gb/s	78 Gb/s	182 Gb/s	234 Gb/s	389 Gb/s
Attention compute per GPU, TFLOP/s ↓					
WorldCast, one GPU per player	5.9	5.9	5.9	5.9	5.9
Gamma-World, one GPU	40.7	81.4	163	203	326
Solaris, one GPU	34	137	548	856	2,192*

once, by 0.9 GiB, when a CUDA graph is captured for the first window without memory frames, and not afterwards.

Training a joint-generation method. The training cost of WorldCast does not depend on P, since a sample is always the window of one player: under teacher forcing, 41 context and 41 noisy latent frames, or, when the sample holds memory frames, the shorter window of Appendix E. A joint-generation method must place all players in one sample, $82P$ frames for the same model. On one H200, with frozen parameters and gradient checkpointing, a forward and backward pass over one sample takes 11.3 GiB beyond the 10.8 GiB of weights for 82 frames, 22.5 GiB for 164, 64.7 GiB for 328 and 100.9 GiB for 410, and runs out of memory for 656. WorldCast stays at the first value for any number of players, whereas Solaris and frame concatenation need 22.5 GiB at two players and 100.9 GiB at five and do not fit in the memory of one GPU at eight.

C Training stages and scaling

This appendix details the training stages and runs (Fig. 8, Table 8), the foreground weight, distillation, the compute and baselines of Table 1, and the scaling of the player state field with data (Table 9).

Foreground weight. From stage 2 on, the weight in Eq. (5) is $\alpha_k = 1 + (\lambda - 1)\max_p g_p(k)\,\beta_p$ with $\lambda = 3$, where g_p is a unit-peak Gaussian centered on the visible other player p on the 24×42 latent grid, and $\beta_p = \min\big(3, \max(1,\ 1.5/r_p)\big)$ grows as the player's projected radius r_p on that grid

Table 8: Training runs of the reported models (Batch: global batch in windows); stages 3 and 4 are run once from each stage-2 model, with and without scene state. The ablations of Table 5 retrain the stage-2 model without scene state from stage 1 for 6,000 steps at a batch of 64 (384k windows). WorldCast denotes the stage-2 model without scene state at 25,000 steps in Table 1, its 6,000-step checkpoint in Table 5 (the 384k column of Table 9a), and the four-step model with scene state in Tables 3, 4, 6, 7, 15, 16 and 17–19.

Stage	Initialization	Steps	Batch	Windows
1: bidirectional tuning, five-second windows	Wan2.2-TI2V-5B	34,000	256	8.70M
then ten-second windows	five-second model	6,000	256	1.54M
2: bidirectional training, player state field	stage 1	25,000	64	1.6M
then scene state	stage 2 at 20,000 steps	5,000	64	320k
3: autoregressive training	stage 2	5,000	64	320k
4: four-step distillation	stage 3	600	64	38.4k

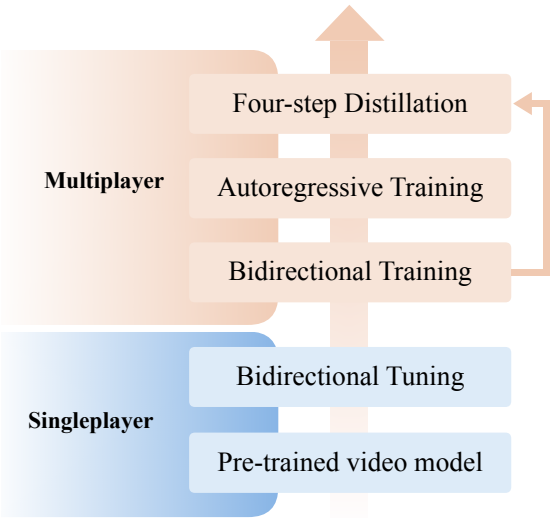

Figure 8: Training stages, from a pretrained video model to four-step distillation.

(at least half a cell) shrinks. The weight at a player's center is thus 3 for a near player and at most 7 for a distant one; the weight map is normalized by its batch mean.

Distillation. Stage 4 runs 600 steps at a global batch of 64 ten-second windows, updating the generator every fifth step and the critic otherwise. The generator is initialized from the stage-3 model, and the teacher and critic from WorldCast, the stage-2 model at 25,000 steps (Table 1), which has no scene state. From the recorded first frame, the generator produces the other 40 latent frames of each window as ten blocks of four, each conditioned on its own earlier outputs; the teacher and critic score the whole window bidirectionally. Distillation windows contain no memory frames, yet the four-step model retains the use of memory frames learned in stages 2 and 3 (Appendix E): after 30 s, WorldCast exceeds the same weights with scene state off at inference by 1.23 dB PSNR $[0.91, 1.55]$ (Table 18).

Compute. The three models of Table 1 used 2,637 GPU-hours for WorldCast (25,000 steps, global batch of 64 windows, 64 GPUs); 3,146 for frame concatenation, which continues a 5,000-step run without player states (a further 640k windows and 1,224 GPU-hours); and 3,110 for Solaris (12,500 steps in total, first on 16 and then on 32 GPUs, at a global batch of 64 view pairs, i.e. 128 windows). Each model sees 1.6M windows with player states.

Baselines. The two baselines keep jointly generated views consistent through attention between them; they receive the player states as tokens, in training and at inference, so that they know where the other players are, as the field does; the comparison is between joint generation and the player state field, not between ways of feeding states. Each player's state enters as one token per latent frame, with the token contents of the model trained without camera alignment (Appendix F) and

the GT visibility label, folded per latent frame as for the field. The tokens join the cross-attention context, their mean over players is added to the frame's action modulation, and training uses the uniform loss.

Every model that receives the states as tokens, coupled or not, renders below 0.1, against 0.763 for WorldCast at the same 384k windows: the model trained without camera alignment (Table 5), a single view given the states as tokens, renders 0.093, against 0.074 for frame concatenation and 0.056 for Solaris at 1.6M windows (Table 1). We attribute the gap to where the state enters. The state injector writes each player's state explicitly, aligned with the camera, at the tokens where the player appears, so that the pretrained DiT can draw the player there; attention between views must learn this placement implicitly and, at these budgets, does not. The model trained without camera alignment differs from the baselines in three further ways: it has the foreground weight, lacks the visibility label and sees 384k windows. Nor does the budget explain the gap: the baselines stay below 0.09 at every budget, including 256k and 512k on either side of 384k, whereas WorldCast renders 0.62 of the other players at 64k windows and improves with data (Table 9).

Scored only on the other player of the pair that the joint-generation methods generate together, WorldCast renders 0.84, against 0.03 and 0.02 for frame concatenation and Solaris, over the views with at least eight scored pairs. Although that player's own view seldom shows its body, the joint-generation methods generate that view and receive its state token, so they have at least as much information about its position as WorldCast.

Table 9: Scaling with training data on the ten-second windows of Table 1, with the same entity criterion. **(a)** The player state field. **(b)** The render rate of WorldCast, frame concatenation and Solaris at the budgets at which all three were trained and scored, each at its own checkpoint. Frame concatenation starts from a 5,000-step run (640k windows) without player states; its budgets count only windows with states.

(a) Training windows	64k	128k	256k	384k	512k	768k	1.0M	1.6M
Rendered ↑	0.617	0.667	0.738	0.763	0.759	0.774	0.784	**0.817**
Unmatched ↓	0.154	0.102	0.075	0.076	0.060	0.047	0.044	**0.039**
PSNR ↑	15.31	15.51	15.70	15.54	15.66	15.86	15.98	**16.17**
LPIPS ↓	0.469	0.456	0.448	0.455	0.444	0.436	0.428	**0.418**
FVD ↓	53.0	44.3	43.3	45.7	42.1	43.3	42.9	**39.7**
(b) Training windows		128k	256k		512k	768k		1.6M
WorldCast		**0.667**	**0.738**		**0.759**	**0.774**		**0.817**
Frame concatenation		0.065	0.065		0.081	0.067		0.074
Solaris		0.045	0.060		0.041	0.036		0.056

Consistency between views. Joint generation couples the views of a window to keep them consistent. Table 10 compares the three models of Table 1 on this property with the measure of Table 19, applied to the two views of each ten-second window. Over the last five seconds, the depths of WorldCast's two views agree on 0.207 of pixels, against 0.170 for frame concatenation and 0.179 for Solaris; the recorded video reaches 0.309.

Table 10: Agreement between the depths of the two views of a ten-second window at the same instant, for the three models of Table 1 with GT states: share of pixels whose reprojected depths agree within the larger of 24 u and 5%, averaged over the windows; chance places a view's depth from the most distant instant of its window at its current camera. Differences are paired over windows; their 95% bootstrap intervals over windows (10,000 draws) all exclude zero.

	First 5 s	Last 5 s
WorldCast	0.257	0.207
Frame concatenation	0.232	0.170
Solaris	0.233	0.179
Recorded video	0.317	0.309
Chance, depth from the most distant instant	0.117	0.116
WorldCast minus		
Frame concatenation	$+0.025$	$+0.037$
Solaris	$+0.023$	$+0.028$

D EVALUATION PROTOCOLS AND LONG HORIZONS IN DETAIL

Data. All video comes from OpenCS2 (Blanchon, 2026) (CC-BY-4.0), which replays Counter-Strike 2 demos published on HLTV and renders the ten players' first-person views at 1280×720 and 32 fps, with every player's controls and state at each 64 Hz engine tick. We use four maps (Dust2, Mirage, Nuke, Ancient): 569 match-maps from 55 events with 11,853 rounds, split by match into 303 training matches (10,480 rounds) and 40 held-out matches (1,373 rounds). Models read the video at 384×672 and 16 fps. The detector, YOLO11x, is fine-tuned on Dust2 frames labeled automatically with the pretrained detector's person boxes that match boxes projected from the GT positions. On Dust2 frames from matches unused in fine-tuning, labeled the same way, it reaches 0.87 precision and 0.90 recall at its F1-optimal threshold; evaluation uses a confidence of 0.05.

Evaluation sets. Both sets are drawn from the 1,373 rounds of the 40 held-out matches by rules fixed before any reported result and based only on the recordings and the map assets: the visibility labels computed from them, which players' first-person views the dataset holds, and the length of each recording. The set of ten-second windows takes 64 rounds per map, 256 in total, in a seeded random order and, in each, the ten-second window, among windows starting every 2.5 s, and the pair of players with a first-person view of it that maximize the smaller of the two shares of frames in which one player has the other in view. These two players give the two generated views, the pair that the joint-generation methods generate together; this smaller share ranges from 16% to 100% (median 48%), and windows start a median of 23 s into the round. The whole-round set scores each group of three players whose recordings all last at least 60 s by the share of seconds in which one player has another in view in at least half of the frames, averaged over its two pairs with the highest shares and multiplied by the rollout length capped at 120 s; it keeps the eight rounds per map with the best-scoring groups. Although the rule does not require one team, in all 32 rounds the three selected players are teammates who remain alive throughout their rollouts, which run from the round start to the end of the shortest recording (67 to 120 s). All methods and ablations use the same windows and rounds.

Tables 13, 6 and 15 report the first 60 s, which every client of these rounds reaches. Tables 3, 16 and 17 include every window to the end of each round.

Whole rounds (Tables 3 and 4a). The whole-round results of Tables 15 and 16 are obtained with WorldCast, the four-step model with scene state, and the whole-round rows of Table 13 and the clips of Table 14 with the four-step model trained without scene state, all on the 32 rounds of Table 3. In every round, the three selected teammates are clients, sharing one memory bank when the model has scene state, and the other seven players follow their GT states; every client is rolled out from the round start, given the GT states of all players. In Table 4a, WorldCast runs on the same rounds in closed-loop deployment, with all five players of the team as clients until their recordings end; in closed-loop deployment, every client runs the whole deployed loop with predicted states and takes nothing from the recording beyond its first frames and position: after every block, each client estimates its own position from its new frames with the state model and Eq. (4) and publishes it, the other clients render that client's player at this position in their next block, and all visibility is predicted as in Sec. 3.4.

Ten-second windows (Tables 1 and 2). Table 2 evaluates the state model: its two rows use the same generator and differ only in whether the states are GT or predicted. Its generated views (one per client) differ from those of Table 1 (two per window), so rates are compared only within a table. In both rows of Table 2, every player whose first-person view of the window is in the dataset runs a client; the others, nearly all dead before or during the window, keep their GT states. For the *Predicted positions* row of Table 2, every client runs in closed loop: after every block, it estimates its own position from its new frames with the state model and Eq. (4), initialized at the GT position, and publishes it; the other clients of the window render that client's player there in their next block, advancing it over the block by extrapolation from its controls (Appendix A), and all visibility is predicted as in Sec. 3.4 (Figs. 16 and 17). Table 11 reports the corresponding paired differences with 95% bootstrap intervals, all of which exclude zero, and Table 12 repeats them under other settings of the entity criterion.

One-hour rollouts. Each of the 16 one-hour rollouts (four per map, for different players) runs one WorldCast client with scene state, which estimates its own position from its generated video, on one player's recorded controls, with all other players following their GT states. It starts from the recorded first frame of the first round and continues over 66 to 102 consecutive rounds from four to six matches held out from training; each round lasts as long as the player's recording of it and continues from the last generated frame of the previous one. At the start of every round, the player respawns: the scene state is cleared and the position estimate restarts from the recorded spawn position. The rollout is cut at one hour.

Table 11: Paired differences on the ten-second windows of Tables 1 and 2, whose 95% intervals from 10,000 bootstrap resamples of the windows, with the views of a window resampled together, all exclude zero. PSNR, SSIM and LPIPS report mean per-view differences, while Rendered and Unmatched report median per-view differences, which need not equal the difference between the medians of these tables. FVD compares two distributions of clips and therefore has no paired difference.

	ΔPSNR	ΔSSIM	ΔLPIPS	ΔRendered	ΔUnmatched
WorldCast minus frame concatenation	$+0.80$	$+0.015$	-0.042	$+0.631$	-0.571
WorldCast minus Solaris	$+0.51$	$+0.006$	-0.032	$+0.636$	-0.592
Predicted minus GT positions	-0.38	-0.007	$+0.021$	-0.058	$+0.032$

Table 12: Table 11 under other settings of the entity criterion (Appendix D), each changed in the generated and the recorded video alike: WorldCast minus frame concatenation (Concat.) and minus Solaris, and predicted minus GT positions (Pred.). Every difference keeps its sign, and its 95% bootstrap interval, computed as in Table 11, excludes zero.

	ΔRendered			ΔUnmatched		
Setting of the criterion	Concat.	Solaris	Pred.	Concat.	Solaris	Pred.
Match within half a body width	$+0.648$	$+0.656$	-0.095	-0.692	-0.709	$+0.101$
Match within two body widths	$+0.553$	$+0.547$	-0.038	-0.364	-0.409	$+0.012$
Detector confidence 0.25	$+0.585$	$+0.584$	-0.064	-0.603	-0.636	$+0.029$
Detector confidence 0.5	$+0.556$	$+0.552$	-0.090	-0.616	-0.624	$+0.025$
Boxes of at least 48^2 pixels	$+0.638$	$+0.632$	-0.053	-0.515	-0.521	$+0.062$
Mean instead of median	$+0.596$	$+0.593$	-0.070	-0.546	-0.568	$+0.075$

VBench. Table 13 reports every VBench dimension computed from video alone; Table 14 measures, on 24 random ten-second windows of the whole rounds, which of them, except temporal flickering, respond to image damage. Under seven corruptions, imaging and aesthetic quality fall by up to 48% and 20%, whereas subject consistency, background consistency and motion smoothness change by at most 3.5% and rise when frames are held, and dynamic degree responds only to held frames.

Table 13: Reference-free VBench scores of the ten-second windows of the whole rounds (GT states, four-step model trained without scene state) and of the one-hour rollouts of WorldCast with scene state, per twenty-minute interval. Dynamic degree is the fraction of clips classified as moving.

	Subject	Background	Motion	Flicker	Dynamic	Imaging	Aesthetic
Whole rounds							
0–10 s	0.732	0.888	0.953	0.926	1.000	70.8	0.480
10–20 s	0.792	0.917	0.965	0.947	1.000	71.1	0.491
20–30 s	0.822	0.929	0.966	0.948	0.969	71.5	0.499
30–40 s	0.824	0.926	0.966	0.948	0.948	71.8	0.497
40–50 s	0.827	0.929	0.964	0.946	0.958	72.1	0.497
50–60 s	0.829	0.926	0.966	0.948	0.979	72.0	0.495
One-hour rollouts, recorded controls							
0–20 min	0.803	0.910	0.965	0.947	0.950	70.3	0.486
20–40 min	0.797	0.912	0.965	0.946	0.975	70.3	0.489
40–60 min	0.797	0.910	0.965	0.947	0.966	69.2	0.486

Table 14: VBench sensitivity: percent change under paired corruptions of random ten-second windows of the whole rounds, generated by the four-step model trained without scene state (Table 13).

Damage	Subject	Background	Motion	Dynamic	Aesthetic	Imaging
Blur, $\sigma = 0.5$ px	-0.1	$+0.6$	$+0.3$	0.0	-0.1	-5.7
Blur, $\sigma = 2$ px	-1.1	$+1.3$	$+1.1$	0.0	-15.7	-48.1
JPEG, quality 30	-0.5	-2.1	-0.2	0.0	-7.7	-6.3
JPEG, quality 12	-1.6	-0.7	-0.3	0.0	-20.4	-19.6
Noise, $\sigma = 10$	-0.8	-1.4	-0.8	0.0	-8.6	-14.9
Gamma 1.5	-0.1	-0.1	$+0.1$	0.0	-1.3	-7.4
Every 8th frame held	$+3.5$	$+2.3$	$+2.0$	-37.5	$+0.8$	$+0.8$

The entity criterion. Rendered and Unmatched are computed in all tables with the rule of Sec. 4, against the state given to the generator (with GT states, the recorded position and the visibility computed from the map assets). A pair of another player and a frame is scored if the player is marked visible in the state given to the generator, the player's box clipped to the 384×672 frame covers at least 24^2 pixels, and the detector finds the player in the recorded video; the player is rendered if a person is detected within one body width, the width of the player's projected box, of its position in the given player state, each detected person matching at most one player. A detected person is unmatched if neither a player marked visible in the given state, with a projected box at least 24 pixels on its short side, nor a person detected in the recorded video lies within one body width of it; the unmatched rate is the unmatched share of persons detected in the generated frames. Both rates are computed for each view, one player's view of a ten-second window, with a whole round split into consecutive ten-second windows, and are summarized by their median over the views with at least eight pairs or eight detected persons, respectively; the views in the unmatched median thus depend on how many persons a generator draws and can differ between rows. Table 16 instead pools the scored pairs of each level of crowding.

Agreement on player positions. To evaluate whether independently generated views place the same player at the same location, Table 15 measures, for two clients of a round, whether both render the player of the third client and where. The evaluation includes the frames in which that player is marked visible for both clients, its clipped box covers at least 24^2 pixels in both views, and the detector finds it in both recordings. A client is considered to render that player if a person is detected within one body width of the player's GT position.

Table 15: Agreement of two clients on the position of the player of the third client of their round, for WorldCast on the whole rounds with GT states. Frames denotes the number of frames in which that player is scored in both views, and Rendered by both the fraction of these frames in which both clients render it. Disagreement reports, over the frames rendered by both, the median closest distance between the two clients' viewing rays to the persons matched to that player (one body width is 32 u).

	Frames	Rendered by both	Disagreement (u) ↓
0–10 s	955	0.687	27.4
10–20 s	492	0.626	18.7
20–30 s	652	0.644	19.3
30–40 s	831	0.712	17.0
40–50 s	1,033	0.804	19.2
50–60 s	642	0.715	16.7

Crowded frames. To examine whether rendering degrades in crowded views, Table 16 groups the whole-round render rate of WorldCast by the number of other players in view, counting the players marked visible whose clipped box covers at least 24^2 pixels. The render rate remains between 0.79 and 0.82 for one to four other players, and no frame of the 32 rounds contains more.

Table 16: WorldCast on the whole rounds by the number of other players in view. Rendered is pooled over the scored pairs of each level of crowding, since one window mixes levels of crowding.

Other players in view	1	2	3	4
Frames	44,395	15,274	3,498	631
Scored pairs	31,525	21,980	7,540	1,941
Rendered ↑	0.809	0.797	0.795	0.821

E SCENE STATE IN DETAIL

The 32 rounds of Table 3 come from held-out matches on the training maps, as the place head requires, but outside the training data of the state model and of every generator stage. All rows of Table 3 use the same ten-second windows of the whole rounds.

Each ablation changes one aspect of WorldCast on the same rounds. *Scene state off at inference* keeps WorldCast's weights and removes only the memory frames, isolating the scene state; the *model trained without scene state* also has clean context, as in its training (*Training* below). After 30 s, WorldCast exceeds both on PSNR, SSIM and LPIPS (Table 18; examples in Figs. 14 and 15); in the first 30 s, the model trained without scene state is slightly ahead, with intervals that include zero. The other ablations change what is retrieved: *Only its own earlier blocks* restricts retrieval to the client's own memory entries, *Least-covering memory* selects the memory entry covering the fewest missing pixels, and *GT frames in the memory bank* replaces the other clients' memory entries with their GT frames, which also supply the depth that locates them, keeping which memory entries exist and when.

Table 17: Table 3 over whole rounds, against the recorded video. PSNR, SSIM and LPIPS: means over the ten-second windows; FVD: over their clips; Rendered and Unmatched: medians over views (Appendix D).

	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Rendered ↑	Unmatched ↓
Trained without scene state	78.4	13.60	0.403	0.572	0.815	0.053
Scene state off at inference	90.5	13.40	0.402	0.593	0.820	0.049
Only its own earlier blocks	91.1	13.87	0.407	0.574	0.825	0.042
Least-covering memory	98.5	13.21	0.393	0.602	0.804	0.054
WorldCast	89.3	14.40	0.416	0.545	0.841	0.037
GT frames in the memory bank	75.0	14.99	0.433	0.513	0.853	0.030

Table 18: WorldCast minus each other row of Table 3, paired over the same ten-second windows: mean differences; † marks a 95% cluster-bootstrap interval over rounds (10,000 draws) that includes zero. Positive ΔPSNR and ΔSSIM and negative ΔLPIPS favor WorldCast. The last row, *GT frames in the memory bank*, whose memory entries hold GT frames that clients cannot have, is ahead (negative ΔPSNR and ΔSSIM, positive ΔLPIPS).

WorldCast minus	ΔPSNR	ΔSSIM	ΔLPIPS
First 30 s			
Trained without scene state	$-0.05^{\dagger}$	$-0.0078^{\dagger}$	$+0.0138^{\dagger}$
Scene state off at inference	$+0.51$	$+0.0072^{\dagger}$	-0.0248
Only its own earlier blocks	$+0.32$	$+0.0059^{\dagger}$	-0.0189
Least-covering memory	$+0.61$	$+0.0097$	-0.0293
GT frames in the memory bank	-0.29	-0.0113	$+0.0163$
After 30 s			
Trained without scene state	$+1.20$	$+0.0227$	-0.0466
Scene state off at inference	$+1.23$	$+0.0178$	-0.0588
Only its own earlier blocks	$+0.63$	$+0.0102$	-0.0347
Least-covering memory	$+1.46$	$+0.0294$	-0.0712
GT frames in the memory bank	-0.75	-0.0188	$+0.0397$

Agreement on the scene. Since each rollout departs from the recording, the scene state is also evaluated by the consistency among clients within the world they generate. Mat. Pix. counts the pixels that GIM (Shen et al., 2024) matches between two clients' frames at the same instant with confi-

dence above 0.05, as in SynCamMaster (Bai et al., 2025). Inliers counts the LoFTR (Sun et al., 2021) matches between the two frames that fit one essential matrix, estimated with MAGSAC++ (Barath et al., 2020), and RotErr, as in CameraCtrl (He et al., 2025a) and SynCamMaster, is the angle between the rotation recovered from that matrix and the relative rotation of the two cameras. All three are computed over the pairs of clients whose views are co-visible in the recording. On the recorded video of the pairs scored in Table 4a, same-instant pairs score far better than pairs offset by 10 s (Mat. Pix. 163.1K against 52.3K, Inliers 128.3 against 36.4, RotErr 16.7° against 64.9°; better in all but two rounds for Mat. Pix. and in every round for the others), so the measures track agreement between views. To test whether the clients of a round generate the same scene, Table 19 compares the depths that two clients predict for their generated frames at the same instant. The depth that the depth head predicts for one client's latent frame is back-projected with that client's camera and reprojected into the other client's camera; a pixel agrees if the reprojected depth and the other client's own depth there lie within the larger of 24 u and 5%. The share of agreeing pixels is averaged over the same pairs and over the rounds. The recorded video gives the upper bound, and a client's depth from 10 s later, placed at its current camera, the chance level. The scene state raises this agreement, particularly after 30 s, and every ablation falls below WorldCast. It leaves each client's agreement with its own next latent frame unchanged, so the gain does not come from a steadier depth estimate. In the closed-loop deployment of Table 4a, where the clients use predicted states and each client's depth is reprojected with the cameras it publishes, the scene state likewise raises the agreement after 30 s, from 0.092 to 0.120, and leaves it unchanged in the first 30 s. On the same pairs, relative to scene state off, WorldCast changes Mat. Pix. by +20.1K [5.1, 35.7] and +13.2K [4.5, 22.5], Inliers by +5.2 [1.1, 9.3] and +9.0 [5.1, 13.1], and RotErr by −2.8° [−4.8, −0.9] and −3.4° [−6.7, −0.3] in the first 30 s and after 30 s, respectively, with 95% round-cluster bootstrap intervals. Every view shows the weapon and hands at the same screen position; with the lower right of each frame and its lower-left corner masked, the differences after 30 s remain +8.0K [2.2, 14.1] in Mat. Pix., +5.2 [2.7, 7.9] in Inliers and −12.9° [−22.1, −4.8] in RotErr, and those in the first 30 s also exclude zero.

Table 19: Agreement between the depths predicted for two clients' generated frames at the same instant, on the whole rounds of Table 3 with GT states and, in the lower part, in the closed-loop deployment of Table 4a with the cameras each client publishes: share of pixels whose reprojected depths agree within the larger of 24 u and 5%; chance places a client's depth from 10 s later at its current camera. Differences are paired over rounds; † marks a 95% cluster-bootstrap interval over rounds (10,000 draws) that includes zero. Own next latent frame applies the same measure to each client and its own next latent frame; scene state does not change it, so its gain between clients is not a steadier depth estimate.

	First 30 s	After 30 s
GT states		
Trained without scene state	0.259	0.135
Scene state off at inference	0.229	0.133
Only its own earlier blocks	0.231	0.147
Least-covering memory	0.225	0.119
WorldCast	0.264	0.206
Recorded video	0.411	0.365
Chance, depth from 10 s later	0.113	0.101
WorldCast minus scene state off		
Between clients	+0.035	+0.073
Own next latent frame	+0.004†	+0.000†
Closed-loop deployment, predicted states		
WorldCast	0.247	0.120
Scene state off at inference	0.243	0.092
WorldCast minus scene state off		
Between clients	+0.004†	+0.028
Own next latent frame	−0.001†	−0.002†

FVD. FVD compares the generated and recorded ten-second windows of the whole rounds as unpaired distributions. It is similar for WorldCast and *Scene state off at inference* (89.3 and 90.5), which share weights; the lower FVD of the model trained without scene state (78.4) thus stems from its weights or its clean context.

Ray embedding. The MLP ϕ of Sec. 3.3 is Linear–SiLU–Linear with a zero-initialized last layer, so the ray embedding starts at zero; its 9.5M parameters are trained with the full backbone. The length scale is $\ell = 420$ u, and each viewing direction uses its frame's field of view.

Managing the memory entries. All clients share one scene state per round. A memory entry is one block (four latent frames, one second of video), stored with each latent frame's camera and the depth predicted by the depth head from the block's own latents, one value per 16×16-pixel patch (Sec. 3.4). Each latent frame spans four video frames and takes the camera and depth of the last, when its content is complete and the state model reads its position. A set of memory entries covers a pixel of a frame if the nearest of their reprojected points matches the frame's depth there within the larger of 24 u and 5%. Every generated block is published as a memory entry; each client keeps at most $B = 64$ of its own, withdrawing beyond this bound the one best covered by its others (the older on a tie). With reprojections between a client's memory entries cached, publishing adds at most $2B$ reprojections and withdrawal requires none. The scene state thus holds at most B memory entries per client for any rollout length, and BP for P clients. Retrieval projects every memory entry completed before a block, so its per-block cost grows with the number of memory entries, at most linearly in P, unlike the generator's; the frame rates of Table 7 include it.

Retrieval. Retrieval runs before the first denoising step, when the block has no frames, and so uses only its four cameras and its missing pixels, which no 3D point of the client's twelve most recent latent frames reaches. A memory entry covers a missing pixel if its point lies, within the tolerance, on the visible surface given by the nearest point of any memory entry. Of all memory entries completed before the block, the client's own included, the one covering the most missing pixels is retrieved (the more recent on a tie), and none if no memory entry covers any. Retrieving at most one memory entry bounds the generator's input at twenty latent frames (four memory frames, twelve recent context frames, four target frames) for any number of clients. In the rollouts of Table 3, a memory entry is retrieved for 95% of the blocks that read the scene state, 0.65 $[0.61, 0.68]$ of them from the other two clients; in the closed-loop deployment of Table 4a, for 96%, 0.47 $[0.44, 0.50]$ of them from the other clients. Replayed on the generated blocks of the whole rounds whose view, by the map geometry, shows surface unseen from the recent context, with coverage measured on the map geometry rather than on the predicted depth that retrieval uses, the retrieved memory entry attains 0.81 $[0.78, 0.83]$ of the coverage of the best earlier memory entry of the other clients, against 0.70 $[0.67, 0.72]$ for the memory entry outside the recent context whose camera is nearest the block's; the paired difference, $+0.11$ $[+0.09, +0.13]$, grows with head rotation. In every evaluation the scene state is read from the 25th latent frame of a rollout onward, so the first six seconds use the recent context alone.

Training. With probability 0.8, a training sample includes GT frames as memory frames: the four latent frames of a same-round teammate's view that end no later than the last target frame and show most of what the recent context lacks, as counted by m_k; otherwise it has none, so that the generator also learns to generate without scene state. In stage 3, under block-causal attention, the memory frames precede the twelve most recent and four target latent frames; in stage 2, they precede 32 recent latent frames, noised with the target, and four target latent frames, all under bidirectional attention. The loss at position k of target frame f is multiplied by $\alpha_k c_k$ in Eq. (5), with $c_k = \frac{1}{2}\big(1 + m_k/\bar{m}_f\big)$. The mask $m_k \in \{0, 1\}$ marks the positions whose surface the memory frames show and the recent context does not, computed during training from the map geometry and the recorded cameras, and $\bar{m}_f$ is the share of marked positions in frame f; in a frame without marked positions, $c_k = 1$. Otherwise, marked positions receive the weight $\frac{1}{2}(1 + 1/\bar{m}_f)$ and the other positions $\frac{1}{2}$, so that c_k averages to one over each frame and the total weight of the frame is unchanged. The recent context frames of the model with scene state are noised to a timestep drawn between 16 and 32 of 1,000 in stage 3, and to 16 in distillation and at inference.

F THE PLAYER STATE FIELD IN DETAIL

This appendix specifies the player state field of Sec. 3.2.

Channels. The field has twenty-three channels on the model's token grid, 12×21 per latent frame (the 24×42 latent grid of a 384×672 frame, patchified by two): coverage; log-compressed depth;

the sine and cosine of the yaw relative to the observer; team; headcount; nine control channels, each the fraction of the latent frame during which a key is held (forward, back, strafe left, strafe right, jump, crouch, walk, attack, reload); four weapon-embedding channels; dying and corpse channels, written where the player fell; and two identity channels.

Splatting. Each player to be written is splatted onto the token grid with the isotropic Gaussian kernel of Wang et al. (2025), centered on the body at the projected position raised by the projected radius (the engine's origin is the ground between the feet). The weight of player p at token u is $w_p(u) = e_p \exp\big(-\|\mathbf{x}_u - \boldsymbol{\mu}_p\|^2/2\sigma^2\big)$, with σ half a token for all players; e_p is zero for a player not written and otherwise a confidence between 0.3 and 1 that smooths visibility over the latent frames of a block. Player size enters through the depth channel, not the kernel. The spread of the weights over neighboring tokens encodes the sub-token position of the center: with the coarse field of Table 5, which resolves position only to 2×2 tokens, the render rate is 0.708, against 0.763 for WorldCast.

Composition. At each token the field keeps the two players with the largest depth-discounted weight $w_p(u)\, e^{-\tilde{z}_p}$, with $\tilde{z}_p$ the log-compressed depth, and combines them as

$$c(u) = \min\Big(1, \sum_{p \in S(u)} w_p(u)\Big), \qquad \mathbf{F}_a(u) = \sum_{p \in S(u)} w_p(u)\, a(s_p), \tag{6}$$

where $S(u)$ is the selected pair, c the coverage channel and $a(s)$ the value of a state in channel a. The headcount channel sums each player's peak weight in the frame, capped at four. A frame without players to write is exactly zero in all channels.

Visibility. Another player is written into the field only if it lies in the view frustum and is visible (Sec. 3.4). Visibility is tested at the feet raised by 40 u and at eight points on a horizontal circle of radius 40 u around that point: the player is visible if, at any test point, the 3D points of the recent context frames reach its projection and their median depth over the 2×2 depth-map cells there (each 16×16 pixels) lies at most 24 u in front of it. Where no 3D point reaches, the player is not written at the first denoising step; after it, visibility is re-tested once on the depth of the block's clean estimate $\hat{\mathbf{x}}_0$ and held for the remaining steps. In evaluations using GT states, the GT visibility labels, computed from the recorded positions and the public 3D map assets, are used instead.

Injection. The field enters after the second of the 30 DiT blocks through one zero-initialized stem of thirty-two channels, so stage 2 starts from the stage-1 generator; the stem's learning rate is 1.4×10^{-3}, the backbone's 2.8×10^{-5}. The remaining 28 DiT blocks process the field; injected after the 23rd DiT block, it passes through only seven, and the render rate falls from 0.763 to 0.549 while the unmatched rate rises from 0.076 to 0.203 (Table 5). The observer's controls (eleven buttons, i.e. the thirteen read by the state model except the scoreboard and use keys, plus the pitch and yaw deltas) and held weapon (a 52-way embedding) enter every DiT block through adaptive layer normalization; the other players' positions and facing directions enter only through the field.

Field ablations. Each retrained row of Table 5 changes the field or its loss. Without camera alignment, per-player state tokens replace the field, one per latent frame and player (world position and position relative to the observer; position, bearing and log-distance in the observer's camera frame; facing direction; team; alive; presence and observer flags), appended to the cross-attention context and pooled into the latent frame's action modulation. Lacking an image position, these tokens have no visibility gate and, unlike the baseline tokens of Table 1, no visibility label; the loss keeps its foreground weight, computed outside the generator by projecting the players' positions through the observer's camera. Without visibility, every living player in the view frustum is written, occluded or not, with no visibility label. Without foreground weight, $\lambda = 1$, so $\alpha_k = 1$. Late injection writes the field after the 23rd DiT block; the coarse field is max-pooled by two and upsampled back. The two inference-time rows keep the WorldCast weights and change one input: *Field off at inference* discards the stem's entire output, biases included; *Visibility off at inference* marks every player visible, so every living player other than the observer in the view frustum is written at full confidence, as by the model trained without visibility.

Table 20 reports each row of Table 5 minus WorldCast, paired on the same views, with 95% bootstrap intervals over the ten-second windows. Every retrained row has a lower render rate, by 0.03 to 0.55

in the median per-view difference, with all intervals below zero. Visibility off at inference raises the unmatched rate by 0.14 ($[0.11, 0.17]$), with a render-rate difference in $[-0.017, 0]$. Training without camera alignment and both inference-time changes lower PSNR and SSIM and raise LPIPS; training without visibility, without foreground weight or with a coarse field raises PSNR by 0.13 to 0.17 dB; for late injection the PSNR and LPIPS intervals include zero.

Table 20: Each row of Table 5 minus WorldCast, paired over the views both rows score (PSNR, SSIM, LPIPS: mean difference; Rendered, Unmatched: median per-view difference, which need not equal the difference between the medians of Table 5); † marks a 95% interval from 10,000 bootstrap resamples of the windows, the two views of a window drawn together, that includes zero. Negative ΔPSNR, ΔSSIM and ΔRendered and positive ΔLPIPS and ΔUnmatched mean that the row is behind WorldCast. FVD compares two distributions of clips and has no paired difference.

Row minus WorldCast	ΔPSNR	ΔSSIM	ΔLPIPS	ΔRendered	ΔUnmatched
Trained without camera alignment	-0.37	-0.005	$+0.024$	-0.545	$+0.560$
Trained without visibility	$+0.13$	$+0.003$	-0.006	-0.047	$0.000^\dagger$
Trained without foreground weight	$+0.17$	$+0.006$	-0.009	-0.043	$0.000^\dagger$
Trained with late injection	$-0.02^\dagger$	$+0.004$	$+0.001^\dagger$	-0.148	$+0.051$
Trained with a coarse field	$+0.15$	$+0.003$	-0.010	-0.034	$0.000^\dagger$
Field off at inference	-0.20	-0.003	$+0.012$	-0.690	$+0.192$
Visibility off at inference	-0.38	-0.009	$+0.020$	$0.000^\dagger$	$+0.143$

G LIMITATIONS AND SCOPE

This appendix states the measurement conditions of Sec. 4 and what they leave untested.

Player states. The whole-round results of Tables 3, 15, 16, 17 and 18 and of the upper part of Table 19 give every client the GT states of all players, to measure the generator and the scene state separately from the predicted states (Appendix D). Table 4a and the lower part of Table 19 use closed-loop deployment, with predicted states. With predicted states (Tables 2 and 4a), the entity criterion scores each generated frame against the positions and visibility given to the generator, that is, whether each player is drawn where that player's own client has placed it. Every rollout departs from the recording as its errors accumulate, so the criterion asks whether the clients agree on one world, kept shared by the predicted states and, in Table 4a, the scene state, rather than whether that world follows the recording. On video generated with GT states, the state model's median position error against the GT trajectory grows to 2.64 body widths at 50–60 s, and 39% of clients are within two body widths at 60 s (Table 6).

The entity criterion. Rendered counts a player as drawn when the fine-tuned detector finds a person within one body width of its position in the given player state; Unmatched counts detections with no such player nearby and no person detected nearby in the recorded video. Neither checks the team, identity, facing direction, weapon or controls that the field also carries; both inherit the detector's errors, and only players that the detector finds in the recorded video are scored for Rendered, so those it misses there are not evaluated (Sec. 4).

Scene state. Relative to the model trained without scene state, the gain in PSNR, SSIM and LPIPS holds after 30 s but not in the first 30 s (Appendix E). After 30 s, the depths predicted for two clients' generated frames at the same instant, reprojected into one camera, agree within the larger of 24 u and 5% on 0.206 of pixels for WorldCast, against 0.365 when the same measure is applied to the recorded video (Table 19).

Number of players. The generator's computation is independent of P, but the scene state holds up to B memory entries per client, so the memory entries projected at retrieval grow with P (Appendix E); the frame rates of Table 7 include this cost. Where players overlap in the image, each token of the field describes at most two players, and the headcount channel saturates at four (Appendix F); since no frame of the 32 rounds has more than four other players in view with a box of at least 24^2 pixels (Table 16), denser crowds are not evaluated.

Clients and the network. In our evaluations, clients advance in lockstep, one block at a time (Sec. 3.1); the evaluations and timing measurements (Appendix B) vary neither network delay, jitter nor loss, and do not study clients that fall behind. Within a block, a client extrapolates the other players' latest published positions from their controls (Appendix A); the effect of staleness on rendering is not measured. Player states and memory entries from other clients are used as received, without cross-checking, so a faulty client alters what the others render.

What comes from the recording. Every client starts from recorded first frames and its GT position (Appendices A and D). In the whole rounds, three players of one team run clients, and all five in Table 4a, while the other players follow their GT states; in Table 2, players without a recorded first-person view of the window keep their recorded states; each one-hour rollout follows one player's recorded controls and restarts its position estimate at that player's recorded spawn position in every round (Appendix D). The state model estimates only position and depth; the other attributes of a player state come from the player's own inputs (Sec. 3.2). No component of WorldCast determines hits, damage or deaths.

Maps and domain. The place head classifies positions into cells of the training maps and reads a map embedding (Appendix A), so a new map requires recorded positions for training; the depth head is supervised with depth rendered from the public 3D map assets. All results are on one game and its training maps.

Baselines and runs. Solaris and frame concatenation in Table 1 are those designs' attention patterns in our stage-2 model, retrained with our data and budget; they keep consistency implicit in cross-view attention and receive the player states as tokens so that they know where the other players are, as the field does (Appendix C). The released Solaris and Gamma-World are timed with random weights (Appendix B) but not scored. Each model of Table 8 and each retrained baseline is trained once, and no table reports variation across training runs; the bootstrap intervals resample windows or rounds.

H Additional qualitative comparisons

Figs. 9–13 show five further held-out rounds, each from a different match and from the start of the round, in the layout of Fig. 4: the recording with its controls overlaid, WorldCast, Solaris and frame concatenation at the same six frames. In the same six-column layout, Figs. 14 and 15 show whole rounds with and without scene state (Table 3), and Figs. 16 and 17 ten-second windows with GT and predicted states (Table 2), and Figs. 18 and 19 closed-loop deployment with scene state on and off (Table 4a); all rows start from the same recorded first frame.



Figure 9: Nuke round 13, player 2; rows and overlays as in Fig. 4.

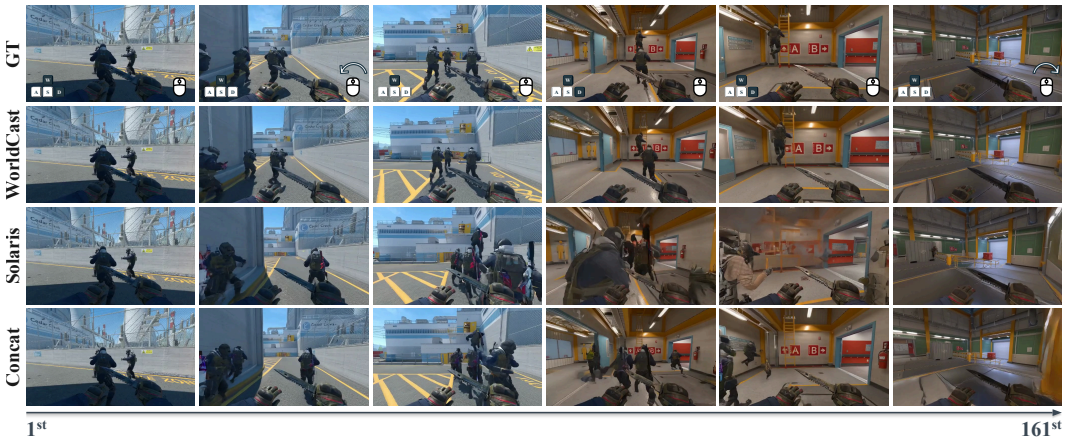

Figure 10: Nuke round 13 of another match, player 1; rows and overlays as in Fig. 4.



Figure 11: Nuke round 2, player 7; rows and overlays as in Fig. 4.

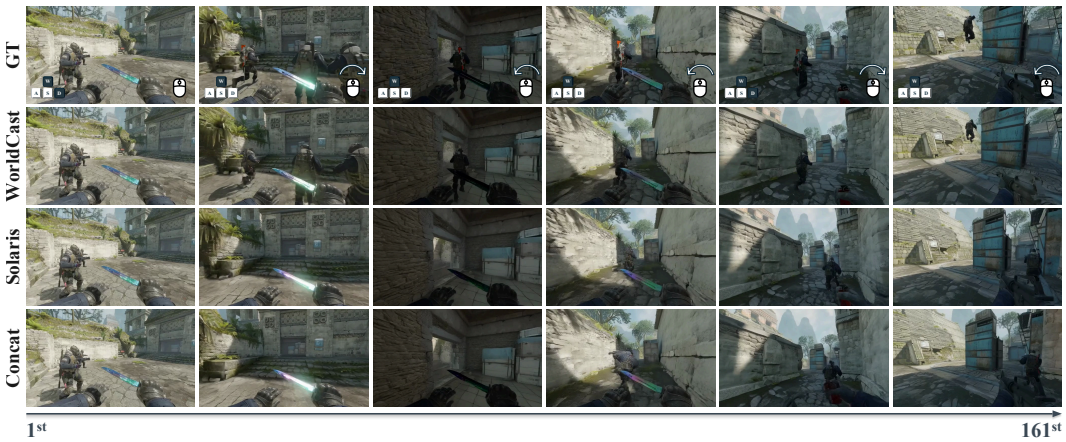

Figure 12: Ancient round 22, player 5; rows and overlays as in Fig. 4.

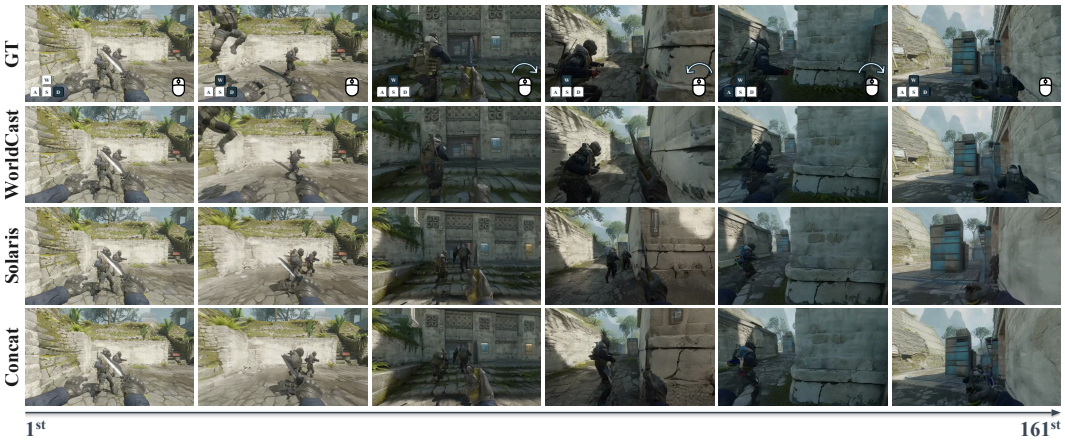

Figure 13: Ancient round 18, player 1; rows and overlays as in Fig. 4.



Figure 14: Mirage round 16, one client, whole round with GT states. GT: recording; SceneOff: scene state off at inference; NoScene: trained without scene state. After 30 s, WorldCast keeps the scene of the recording and in the third column returns to the wall of its first frame; SceneOff and NoScene generate other parts of the map.

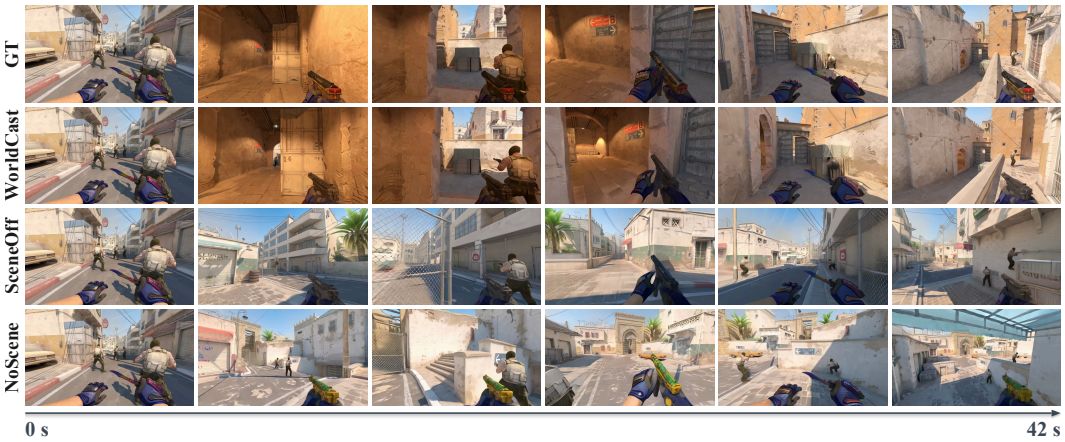

Figure 15: Dust2 round 9, one client; rows as in Fig. 14.



Figure 16: Ancient round 11, one player, ten-second window of Table 2. GT: recording; Oracle: GT positions; Predicted: positions predicted in closed loop, every client estimating its own position from its generated video, with visibility from the depth head.



Figure 17: Dust2 round 18, one player; rows as in Fig. 16.



Figure 18: Mirage round 11, closed-loop deployment of Table 4a: all five players of one team run clients with predicted states; three are shown. Off/On: scene state off/on; a player's Off and On rows differ only in scene state. With scene state, the three views stay around the A-site ramp and agree on its landmarks, and Players 2 and 3 draw the teammates ahead of them (supplementary videos).



Figure 19: Dust2 round 22; rows as in Fig. 18. With scene state, the three players push from mid to A site and see their teammates ahead; without it, Players 1 and 3 remain in a corner by a wooden door and Player 2 faces a wall.