CAMERA POSITIONAL EMBEDDING FOR TEST-TIME TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

To understand the world in 3D from many images, a model must take into account the cameras that captured them. Camera embeddings for attention do so by relating two tokens only through the relative geometry of their cameras, but they also inherit the quadratic cost of attention, which limits the number of input views. Test-time training (TTT), which stores past tokens in a small network trained at test time, offers an efficient alternative with linear cost. However, how to embed cameras into TTT has not been studied, and camera embeddings designed for attention do not transfer directly. We present CaPET (Camera Positional Embedding for TTT), a camera embedding that acts inside the TTT layer. We build it by asking where a camera embedding should be placed in a TTT layer and what form it should take. For the placement, we show that a TTT layer holds implicit scores inside its fast weights, which let a camera embedding relate two views through their relative camera. For the form, we find that a TTT layer prefers orthogonal embeddings, since they preserve the norms of its keys and queries and so keep training at test time stable. On novel view synthesis across diverse datasets, CaPET improves over the TTT model without camera embedding by up to 1.80 dB PSNR and outperforms camera embeddings for attention, at the linear cost of TTT.

1 INTRODUCTION

The world is three-dimensional, yet every image shows it from a single viewpoint. The camera that captured an image, described by its intrinsic and extrinsic parameters, determines where in 3D space each pixel looks. To understand the world in 3D from several images, a model must therefore take their cameras into account, since only the cameras reveal which pixels in different views show the same point in space. Camera embedding, the way a model encodes these cameras, has become a central design choice for multi-view models, especially as 3D vision (Wang et al., 2025a) and robotics (Jiang et al., 2026) come to rely on many views of each scene.

For transformers, recent camera embeddings follow the success of relative positional encoding in language models (Su et al., 2024) and encode the cameras inside attention (Miyato et al., 2024; Li et al., 2025; Wu et al., 2026). Rather than tagging each token with its absolute camera, they make the attention between every pair of tokens depend on the relative geometry of their cameras, so that the model does not depend on how the world frame is chosen. These embeddings, however, also inherit the cost of attention. Attention compares every pair of tokens, so its cost grows quadratically with the number of tokens and limits how many views a model can take as input.

Test-time training (TTT) has recently emerged as an efficient alternative whose cost grows only linearly with the number of tokens (Sun et al., 2025; Zhang et al., 2026b). A TTT layer stores past tokens in a small network, called the fast weights, which it trains at test time. However, how to embed cameras into TTT has not been investigated. Existing TTT models inject the cameras only at the input, as extra features of each pixel (Zhang et al., 2026b). Camera embeddings designed for attention do not transfer directly, because a TTT layer relates tokens through its fast weights rather than through explicit attention scores. Applied as they are, they can even degrade performance.

In this work, we develop **CaPET** (Camera Positional Embedding for TTT), a camera embedding that acts inside the TTT layer. We build it by asking where a camera embedding should be placed in a TTT layer and what form it should take. For the placement, we show that a TTT layer, although it computes no attention scores explicitly, still compares queries with keys through implicit scores

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.

inside its fast weights. CaPET embeds the camera at each of these scores, and also at the values and outputs, as camera embeddings for attention do. For the form, we find that a TTT layer prefers an orthogonal embedding. It has no softmax to normalize its scores, so an embedding that changes norms would let the camera, rather than the content, decide how strongly each view is stored. CaPET therefore uses rotary embeddings built from the ray and the estimated 3D point of each image patch, which keep two views of one surface close even when their cameras are far apart.

We evaluate CaPET on novel view synthesis, a task that requires relating many posed views of a scene, using RealEstate10K (Zhou et al., 2018), DL3DV (Ling et al., 2024), and Objaverse (Deitke et al., 2023). CaPET improves over the TTT model without camera embedding on all three datasets, by up to 1.80 dB PSNR, and outperforms camera embeddings designed for attention when they are applied to the same model. The gains of the individual embeddings add up, and they hold across diverse fast-weight networks, from the SwiGLU layer (Shazeer, 2020) to MLPs of depth four. Throughout, CaPET adds only 0.13% FLOPs, and the TTT layer keeps its linear cost.

The contributions of this paper are as follows:

- To our knowledge, CaPET is the first camera embedding designed for TTT.
- We analyze where a camera embedding should be placed in a TTT layer and what form it should take, and the analysis extends to a wide range of fast-weight networks.
- We show on three novel view synthesis benchmarks that CaPET outperforms both the TTT baseline and camera embeddings designed for attention, at the linear cost of TTT.

2 Related Work

Camera Positional Embedding. Multi-view models encode the cameras either at the input or inside attention. At the input, most models attach a per-pixel ray map to the image, often in Plücker coordinates (Sitzmann et al., 2021; Sajjadi et al., 2022; Zhang et al., 2024; Gao et al., 2024; He et al., 2025; Jin et al., 2025), or embed the 3D points along each ray (Liu et al., 2022). Such inputs depend on the choice of the world frame, so later work moves the cameras inside attention, either by restricting attention to epipolar lines (He et al., 2020; Charatan et al., 2024) or by making the score of two tokens depend on the relative geometry of their cameras (Kong et al., 2024). GTA transforms the queries, keys, values, and outputs by the rigid pose of each camera (Miyato et al., 2024), PRoPE extends the pose to the full projection matrix (Li et al., 2025), and RayRoPE encodes each patch by its ray and a predicted point on that ray (Wu et al., 2026). Related encodings rotate features by the viewing ray (Xiang et al., 2026) or the 3D point (Wu et al., 2025) of each patch, project key patches into the query view (Xie et al., 2026), or separate the rotation and the translation of the camera (Kenney & Suzuki, 2026). For camera-controlled video generation, other encodings give each ray its own frame (Zhang et al., 2026a) or repurpose the rotary embedding of a pretrained model (Li et al., 2026a). All of these relative encodings act inside softmax attention, whose cost grows quadratically with the number of tokens. CaPET instead brings camera embeddings into the fast weights of a TTT layer, which keeps a linear cost, and builds them from orthogonal rotations of the ray and the estimated 3D point of each image patch, as Section 4.3 describes.

Test-Time Training. TTT layers store the context in fast weights, a small network trained by gradient descent at test time, and their cost grows linearly with the number of tokens (Sun et al., 2025). They build on fast weights that change during inference (Schmidhuber, 1992; Ba et al., 2016) and on test-time training, which adapts a model to each test input with a self-supervised loss (Sun et al., 2020). Titans and test-time regression describe the update of fast weights as online regression on key-value pairs (Behrouz et al., 2025; Wang et al., 2025b), and a TTT layer that binds keys to values acts as a learned linear attention (Liu et al., 2026b). LaCT updates the fast weights once per large chunk of tokens, which uses modern hardware efficiently and scales TTT to novel view synthesis (Zhang et al., 2026b). TTT layers now serve video generation (Dalal et al., 2025), vision backbones (Han et al., 2026; Li et al., 2026b), and 3D reconstruction (Wang et al., 2026; Jin et al., 2026; Chen et al., 2026; Ma et al., 2026; Liu et al., 2026a). In these 3D models, the camera either enters at the input as a ray map (Zhang et al., 2026b; Wang et al., 2026; Ma et al., 2026) or is predicted from the images (Jin et al., 2026; Chen et al., 2026; Liu et al., 2026a). CaPET instead embeds cameras inside the TTT layer, at the implicit scores that the update and the apply form in the fast weights.

3 PRELIMINARIES

Softmax Attention. Attention lets each token gather information from other tokens. Each token carries a query, a key, and a value (q_i, k_i, v_i), which describe what it looks for, what it contains, and what it passes on. The output for token j averages the values, each weighted by how well its key matches the query q_j, as measured by their inner product,

$$o_j^{\text{Attn}} = \sum_i \text{softmax}_i(\underbrace{\langle q_j, k_i \rangle}_{\text{attention score}}) v_i. \tag{1}$$

Camera Embedding for Attention. For posed images, camera embeddings make attention depend on the cameras of the tokens. We write the camera of token i as an extrinsic matrix $T_i \in SE(3)$ and an intrinsic matrix K_i, which together form the projection matrix $P_i = \left[\begin{smallmatrix} K_i & 0 \\ 0 & 1 \end{smallmatrix}\right] T_i$. Each token receives an embedding matrix E_i, built from T_i in GTA (Miyato et al., 2024) and from P_i in PRoPE (Li et al., 2025). The query q_j is multiplied by $E_j^\top$, the key k_i and the value v_i by E_i^{-1}, and the output is mapped back by E_j,

$$\begin{aligned} o_j^{\text{cam}} &= E_j \sum_i \text{softmax}_i(\langle E_j^\top q_j, E_i^{-1} k_i \rangle) E_i^{-1} v_i \\ &= \sum_i \text{softmax}_i(\langle q_j, \underbrace{E_j E_i^{-1}}_{\text{relative camera}} k_i \rangle) \underbrace{E_j E_i^{-1}}_{\text{relative camera}} v_i. \end{aligned} \tag{2}$$

Both the score and the value thus see the two cameras only through the relative camera $E_j E_i^{-1}$, which does not change when the world frame moves. Since neither camera matrix is orthogonal, the two sides of the score use the transpose $E^\top$ and the inverse E^{-1}.

Test-Time Training (TTT). TTT is a sequence layer that stores the context in the weights W of a small network f_W, called the *fast weights* (Sun et al., 2025; Schmidhuber, 1992). Past tokens are compressed into W and never revisited, so the cost of the layer grows linearly with sequence length. A TTT layer takes the queries, keys, and values as attention does, but processes them in two operations. The update operation trains the network on the key-value pairs, each of which defines a self-supervised task, returning v_i when given k_i,

$$\textbf{Update:} \qquad W' = W + \Delta W, \qquad \Delta W = -\eta \sum_i \frac{\partial L(f_W(k_i), v_i)}{\partial W}, \tag{3}$$

where η is the learning rate and the loss L is typically a mean squared error or a negative inner product. The apply operation then runs the trained network on the query of a later token j,

$$\textbf{Apply:} \qquad o_j = f_{W'}(q_j). \tag{4}$$

Note that these TTT equations show no counterpart of Eq. 2. In attention, each query interacts with each key and value explicitly, one token pair at a time, and a camera embedding acts on this pairwise interaction, both on its score and on the value it carries. In a TTT layer, the update compresses all keys and values into the fast weights, and the apply lets each query interact only with the trained network, not with any other token directly. A TTT layer thus has no explicit pairwise interaction for a camera embedding to act on, and current TTT models feed the camera only at the model input, as per-pixel Plücker rays concatenated to the image (Jin et al., 2025; Zhang et al., 2026b).

4 METHOD

4.1 IMPLICIT SCORES IN TTT

A TTT layer has no explicit pairwise interaction, but we show that it has an implicit one. Reinterpreting the update and the apply operation, we find that a query still meets each key in an inner product of the same form as the attention score, and receives the stored values weighted by these implicit scores. In the rest of this subsection, we locate the scores and the values inside the fast weights.

For simplicity, we take the fast weights to be a two-layer MLP whose hidden size equals its input size,

$$f_W(x) = W_2\, h(x), \qquad h(x) = \sigma(W_1 x), \tag{5}$$

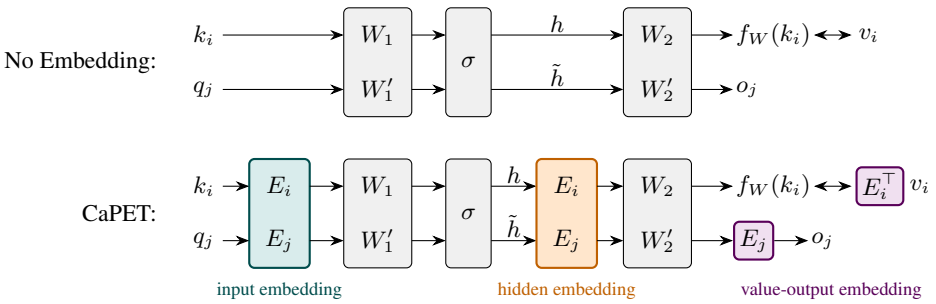


Figure 1: **No Embedding vs CaPET.** The key k_i (top) updates the fast weights from W to W', and the query q_j (bottom) applies W'. CaPET multiplies both by orthogonal matrices built from their own cameras, E_i and E_j, before W_1 and W_2 and on the value and the output. Inside each score of Eqs. 8–9, the two matrices then meet only as a relative camera.

where σ is an activation function, and we assume that the loss of token i is the negative inner product $L_i = -\langle f_W(k_i),\, v_i \rangle$. With this loss, the update of Eq. 3 adds to W_2 one matrix for each token, the outer product of its value v_i and the hidden vector $h(k_i)$ of its key,

$$\Delta W_2 \;=\; -\eta \sum_i \frac{\partial L_i}{\partial W_2} \;=\; \eta \sum_i \frac{\partial \langle W_2\, h(k_i),\, v_i \rangle}{\partial W_2} \;=\; \eta \sum_i v_i\, h(k_i)^\top. \tag{6}$$

For W_1, the chain rule through the pre-activation $z_i := W_1 k_i$ gives terms of the same kind,

$$\Delta W_1 \;=\; -\eta \sum_i \frac{\partial L_i}{\partial W_1} \;=\; -\eta \sum_i \frac{\partial L_i}{\partial z_i}\, \frac{\partial z_i}{\partial W_1} \;=\; \eta \sum_i g_i\, k_i^\top, \tag{7}$$

with $g_i := -\partial L_i / \partial z_i$. Appendix A derives both equations step by step. In both, each term is an outer product with the key on its right side, $h(k_i)$ for W_2 and k_i for W_1. The update thus stores every key in the fast weights, once per weight matrix.

The apply of Eq. 4 runs the updated network $W' = W + \Delta W$ on the query q_j of a later token. An outer product turns a vector into an inner product times a fixed vector, $(a\, b^\top)\, c = \langle b,\, c \rangle\; a = \langle c,\, b \rangle\; a$, so each stored term returns its content scaled by an inner product between the hidden vectors $\tilde{h}(q_j)$ and $h(k_i)$ of the query and the key,

$$o_j \;=\; (W_2 + \Delta W_2)\, \tilde{h}(q_j) \;=\; W_2\, \tilde{h}(q_j) \;+\; \eta \sum_i \underbrace{\Big\langle \tilde{h}(q_j),\, h(k_i) \Big\rangle}_{\text{hidden score}}\; v_i, \tag{8}$$

where $\tilde{h}(q_j) := \sigma\big((W_1 + \Delta W_1)\, q_j\big)$ is the hidden vector of the query. The first term expands in the same way at the first layer,

$$W_2\, \tilde{h}(q_j) \;=\; W_2\, \sigma\big((W_1 + \Delta W_1)\, q_j\big) \;=\; W_2\, \sigma\Big(W_1 q_j \;+\; \eta \sum_i \underbrace{\langle q_j,\, k_i \rangle}_{\text{input score}}\; g_i\Big). \tag{9}$$

The query meets each stored key only in the two marked inner products, one at the input of each weight matrix. They act as attention scores, although no softmax normalizes them and the input score weights the term g_i that the update stores rather than a value. Each of them receives its own camera embedding in Sec. 4.2, as the attention score does in Eq. 2.

4.2 Placement of Camera Embedding

CaPET embeds the camera at the three locations that Fig. 1 shows: the hidden vectors before W_2, the keys and queries before W_1, and the values and outputs. Each embedding is an orthogonal matrix E_i built from the camera of token i, as Sec. 4.3 explains, and it acts in the update and the apply alike. We derive the three embeddings one at a time, each on the plain layer, and show that each makes its score or its value see the two cameras only through their relative camera, as in Eq. 2. Since the

three act at different places in the layer, CaPET uses them together, and their gains add up. We start with the hidden score, where CaPET differs from camera embeddings for attention, and then treat the input score and the values and outputs.

Hidden Rotary Embedding. The hidden score of Eq. 8 compares the hidden vectors of the query and the key at the input of W_2. We multiply the hidden vector of the key by E_i and that of the query by E_j, as the orange box in Fig. 1 shows. The update then stores $E_i h(k_i)$ in place of $h(k_i)$ in Eq. 6, and the apply feeds $E_j \tilde{h}(q_j)$ in place of $\tilde{h}(q_j)$, so Eq. 8 becomes

$$\begin{aligned}(W_2 + \Delta W_2)\, E_j \tilde{h}(q_j) \;&=\; W_2 E_j \tilde{h}(q_j) \;+\; \eta \sum_i \langle E_j \tilde{h}(q_j),\, E_i h(k_i) \rangle\, v_i \\ &=\; W_2 E_j \tilde{h}(q_j) \;+\; \eta \sum_i \langle \tilde{h}(q_j),\, \underbrace{E_j^\top E_i}_{\text{relative camera}}\, h(k_i) \rangle\, v_i.\end{aligned} \tag{10}$$

The hidden score thus sees the two cameras only through their relative camera $E_j^\top E_i = E_j^{-1} E_i$, as in Eq. 2, although the TTT layer never forms an attention score. In other words, the hidden vector of the key is moved from its own camera to that of the query before the two are compared.

Input Rotary Embedding. The input score of Eq. 9 receives the same treatment at the input of W_1, with $E_i k_i$ in the update and $E_j q_j$ in the apply. Leaving out $W_2\, \sigma(\cdot)$ of Eq. 9, its argument becomes

$$(W_1 + \Delta W_1)\, E_j q_j \;=\; W_1 E_j q_j \;+\; \eta \sum_i \langle q_j,\, \underbrace{E_j^\top E_i}_{\text{relative camera}}\, k_i \rangle\, g_i, \tag{11}$$

and its two matrices again meet only as the relative camera. This is the step that camera embeddings for attention take on the query and the key, applied here to the keys that the update stores.

Value and Output Rotary Embedding. Camera embeddings for attention also move each value to the frame of the query (Eq. 2), and we take the same step. We store each value as $E_i^\top v_i$ and map the layer output back by E_j, so Eq. 8 becomes

$$E_j\, (W_2 + \Delta W_2)\, \tilde{h}(q_j) \;=\; E_j W_2 \tilde{h}(q_j) \;+\; \eta \sum_i \langle \tilde{h}(q_j),\, h(k_i) \rangle\, \underbrace{E_j E_i^\top}_{\text{relative camera}}\, v_i, \tag{12}$$

and each value reaches the query through the same relative camera $E_j E_i^{-1}$. In other words, each stored value is moved from the camera of its key to that of the query before it enters the output.

What Stays Absolute. Unlike attention in Eq. 2, the TTT layer as a whole does not see the two cameras only through their relative camera. In each of Eqs. 10–12, the first term involves the query alone and keeps its absolute camera. Section 4.3 returns to this term when it chooses the form of E_i.

Combining the Three Embeddings. We derive the three embeddings independently, but Table 1 shows empirically that they also work together. The table adds them to the baseline one at a time, and each addition improves the PSNR averaged over three datasets. The three together give the best result, so their gains add up, on each dataset as well (Table 4).

Table 1: Evaluation across the different embeddings. Detailed results in Table 4.

hidden	input	value-output	PSNR ↑
			20.08
✓			20.79
✓	✓		20.87
✓	✓	✓	**21.03**

Extension to Other Fast Weights. CaPET is not limited to the two-layer MLP. In any fast-weight network, each weight matrix compares the query with the stored keys at its input, so CaPET places one embedding at each such input. The number of embeddings thus follows the network, two for SwiGLU, as W_1 and W_3 act on the same input, and n for an n-layer MLP (Fig. 2). Appendix B derives the implicit scores of both networks step by step. In Sec. 5.2, CaPET improves MLPs of depth two to four as well.

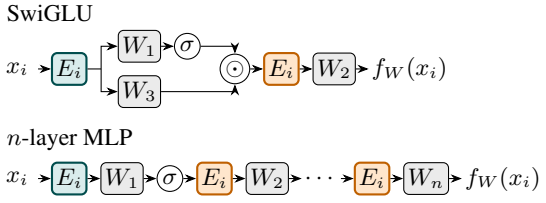


Figure 2: CaPET on other fast weights.

4.3 Form of Camera Embedding

TTT prefers orthogonal embedding. An orthogonal matrix keeps the norm of every vector it multiplies, as Fig. 3 shows, since its transpose is its inverse, $E_i^\top = E_i^{-1}$. We build the embedding from such matrices, because norms matter in a TTT layer. First, the fast weights are a small MLP trained by gradient descent, and such training is stable when the inputs are normalized (LeCun et al., 1998), so TTT layers normalize their keys and queries (Zhang et al., 2026b; Li et al., 2026b). Second, a TTT layer sums its stored terms with no softmax to renormalize them, so the norm of an embedded key sets how strongly its token is stored, and that of an embedded query sets how large its output is.

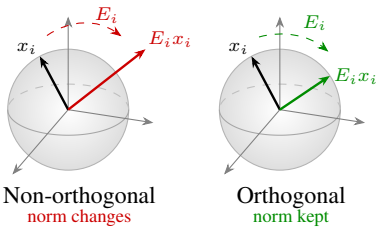


Figure 3: A non-orthogonal E_i moves the normalized vector x_i off the unit sphere, while an orthogonal E_i keeps it on the sphere.

The camera embeddings for attention in Sec. 3 are not orthogonal, so they change these norms by an amount that depends on the absolute camera, such as its distance from the arbitrary world origin. In Eq. 2, this dependence cancels, since the matrices of two tokens always meet as $E_j E_i^{-1}$. A TTT layer also uses each camera matrix alone, in the term that the update stores for each key and in the first terms of Eqs. 10–12. A matrix that changes norms would thus let the absolute camera, rather than the content, decide how strongly each view is stored. An orthogonal matrix avoids this, and one matrix then embeds both the key and the query.

Embedding with Ray and 3D Point. We build the orthogonal matrices with rotary position embedding (RoPE) (Su et al., 2024), which turns the coordinates of a token into rotations. What remains is to choose the coordinates. Each token covers an image patch, and the ray through the center of the patch is a natural first choice, since its direction $r_i \in \mathbb{R}^3$ in world coordinates tells where the token looks. A direction alone, however, cannot tell whether two tokens see the same surface, since two views of one surface point look along different directions, as Fig. 4 shows. What they share is the 3D point, and finding it requires the depth of the surface along each ray. Following RayRoPE (Wu et al., 2026), we add one linear layer that predicts the log depth of token i from the input x_i of the TTT layer, and we train it with the model without depth supervision. The token is then lifted to its 3D point,

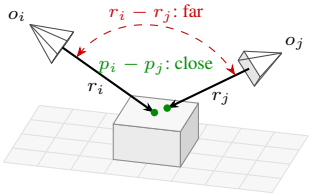


Figure 4: Two views of the same surface point have far-apart rays but nearly the same 3D point.

$$p_i = o_i + d_i\, r_i, \qquad \log d_i = w^\top x_i + b, \qquad \pi_i = (r_i,\, p_i), \tag{13}$$

where o_i is the camera origin, the translation of T_i^{-1}, and w and b are the parameters of the linear layer. Two views of the same surface point now share nearly the same p_i, while r_i keeps the part that depends on the viewpoint. The embedding is built from 2×2 rotations $R(\theta)$, one for each coordinate of π_i and each of the log-spaced frequencies $\omega_1, \ldots, \omega_F$,

$$E_i = \bigoplus_{c=1}^{6} \bigoplus_{f=1}^{F} R(\omega_f\, \pi_{i,c}). \tag{14}$$

where $\oplus$ concatenates matrices along the diagonal into a block-diagonal matrix. The relative camera $E_j^\top E_i$ then rotates each pair by $\omega_f\,(\pi_{i,c} - \pi_{j,c})$, the difference of the coordinates of the two tokens. At the value and output, E_i uses the three coordinates of p_i alone, so that two views of the same surface point exchange their values almost unrotated.

Checking the Two Choices. Table 2 supports both choices of this subsection. Placed at the same locations, the non-orthogonal projection matrix falls below the baseline, while the orthogonal ray embedding improves on it. Adding the 3D point improves further on the ray and gives the best PSNR averaged over three datasets.

Table 2: Evaluation across geometries. Detailed results in Table 5.

geometry	orth.	PSNR ↑
No Embedding	–	20.08
Projection	✗	19.93
Ray	✓	20.47
Ray + 3D Point	✓	**21.03**

Table 3: **Novel view synthesis.** No Embedding is LaCT (Zhang et al., 2026b), which receives the cameras only as Plücker rays at the input. All models are trained and tested with 8 input views.

	RealEstate10K			DL3DV			Objaverse			Average		
method	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
No Embedding	21.61	0.623	0.295	16.42	0.340	0.481	22.21	0.836	0.145	20.08	0.600	0.307
GTA	21.55	0.620	0.299	16.55	0.342	0.474	22.12	0.835	0.145	20.07	0.599	0.306
PRoPE	22.05	0.637	0.288	16.38	0.336	0.489	22.25	0.838	0.140	20.23	0.604	0.306
RayRoPE	22.43	0.661	0.271	16.79	0.349	0.473	22.26	0.838	0.144	20.49	0.616	0.296
CaPET (ours)	**23.41**	**0.702**	**0.259**	**17.00**	**0.355**	**0.469**	**22.67**	**0.842**	**0.139**	**21.03**	**0.633**	**0.289**

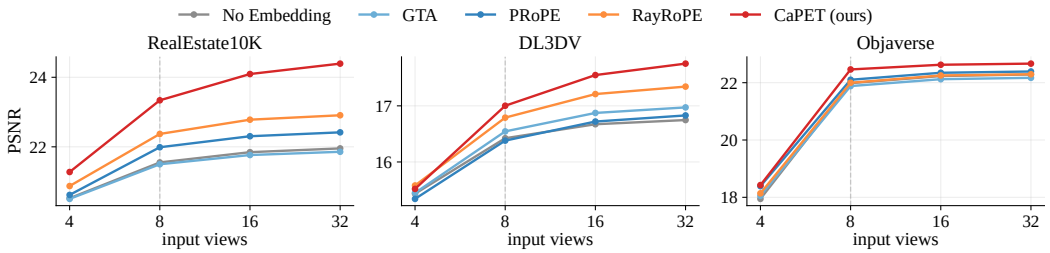


Figure 5: **Number of input views.** Models are trained on 8 input views and tested on 4 to 32.

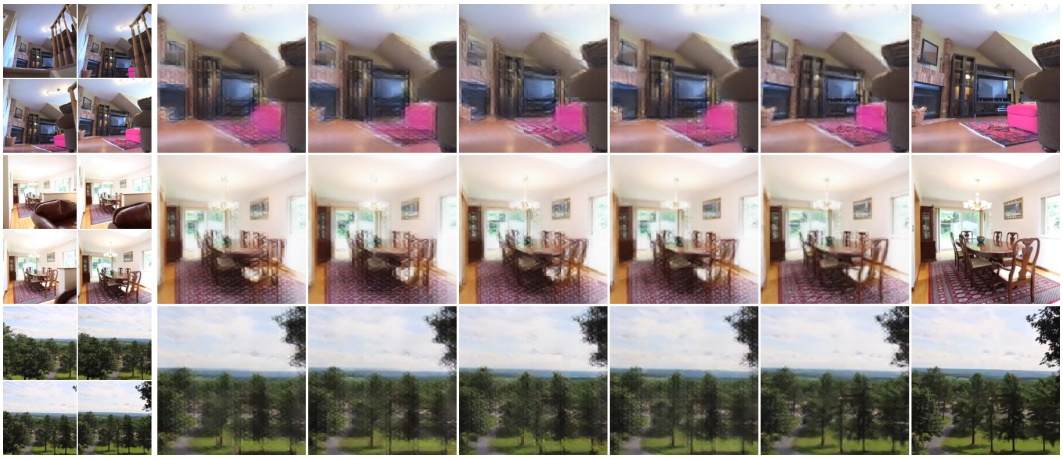


Figure 6: **Qualitative comparison on RealEstate10K.** Each row is one test scene, and the Input column shows four of its eight input views.

5 Experiments

We evaluate CaPET mainly on novel view synthesis (Sec. 5.1), analyze its placement, its form, and other fast weights (Sec. 5.2), and apply it to two other camera-conditional tasks (Sec. 5.3).

5.1 Novel View Synthesis

Model and Training. We build on the LaCT model for novel view synthesis (Zhang et al., 2026b), whose TTT layer uses SwiGLU fast weights (Shazeer, 2020), as many recent TTT models do (Wang et al., 2026; Ma et al., 2026; Liu et al., 2026a). These fast weights are updated with Muon (Jordan et al., 2024). As in LaCT, every model receives the camera of each pixel as a Plücker ray, and No Embedding adds no other camera embedding. Given our limited compute, we use a small configuration with 6 blocks, width 256, and patch size 16 (9.6M parameters). CaPET adds 0.04% parameters and 0.13% FLOPs (Appendix E). Every model is trained from scratch with 8 input and 8 target views for 30K iterations with batch size 16, using mean squared error and LPIPS losses. All models share this recipe and the random seed, and differ only in the camera embedding inside the TTT layer.

Table 4: **Ablation on embedding placement.** Check marks show which of the hidden, input, and value-output (v/o) embeddings of Sec. 4.2 are used. The first row is No Embedding.

embeddings			RealEstate10K			DL3DV			Objaverse			Average		
hidden	input	v/o	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
			21.61	0.623	0.295	16.42	0.340	0.481	22.21	0.836	0.145	20.08	0.600	0.307
✓			22.94	0.681	0.267	16.88	0.350	0.471	22.55	0.841	0.140	20.79	0.624	0.293
	✓		22.54	0.665	0.274	16.76	0.345	0.471	22.50	0.838	0.140	20.60	0.616	0.295
✓	✓		22.99	0.681	0.267	16.98	0.353	**0.468**	22.64	**0.842**	**0.139**	20.87	0.625	0.291
✓	✓	✓	**23.41**	**0.702**	**0.259**	**17.00**	**0.355**	0.469	**22.67**	**0.842**	**0.139**	**21.03**	**0.633**	**0.289**

Table 5: **Ablation on embedding form.** Each row builds the embedding from a different geometry, placed at the same three locations. The orth. column marks whether the embedding is orthogonal.

		RealEstate10K			DL3DV			Objaverse			Average		
geometry	orth.	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
No Embedding	–	21.61	0.623	0.295	16.42	0.340	0.481	22.21	0.836	0.145	20.08	0.600	0.307
Extrinsic	✗	21.36	0.610	0.308	16.32	0.337	0.492	22.18	0.838	0.142	19.95	0.595	0.314
Projection	✗	21.37	0.610	0.311	16.20	0.331	0.505	22.23	0.838	0.143	19.93	0.593	0.320
Ray	✓	23.05	0.689	0.260	16.69	0.349	0.472	21.67	0.832	0.152	20.47	0.623	0.295
Ray + Camera origin	✓	23.00	0.683	0.263	16.60	0.345	0.474	20.73	0.821	0.164	20.11	0.616	0.300
Ray + 3D Point	✓	**23.41**	**0.702**	**0.259**	**17.00**	**0.355**	**0.469**	**22.67**	**0.842**	**0.139**	**21.03**	**0.633**	**0.289**

Datasets and Evaluation. We use three datasets that range from small to large camera motion between views, RealEstate10K (Zhou et al., 2018), DL3DV (Ling et al., 2024), and the G-Objaverse renderings (Qiu et al., 2024) of Objaverse (Deitke et al., 2023), in which 40 cameras surround each object. For each test scene, we choose 8 input views at uniform spacing and 4 target views between them, and report PSNR, SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018) on the target views of 256 RealEstate10K scenes, the 140 DL3DV benchmark scenes, and 499 Objaverse objects.

Baselines. We compare with the camera embeddings designed for attention, GTA (Miyato et al., 2024), PRoPE (Li et al., 2025), and RayRoPE (Wu et al., 2026). The self-attention of this model acts within one image, where a relative camera embedding reduces to the identity, so we apply each in the TTT layer as in its original design. Appendix C gives the remaining details of the setup.

Results. Table 3 compares CaPET with the baselines. CaPET is the best on every dataset and metric. It improves over No Embedding by 1.80 dB PSNR on RealEstate10K, 0.58 dB on DL3DV, and 0.46 dB on Objaverse, and over the strongest baseline, RayRoPE, by 0.54 dB on average. The embeddings designed for attention gain much less in a TTT layer. GTA does not improve over No Embedding on average, and PRoPE improves on RealEstate10K but falls below No Embedding on DL3DV. Both use non-orthogonal camera matrices (Sec. 4.3).

Number of Input Views. Figure 5 tests the same checkpoints with 4 to 32 input views. The lead of CaPET over No Embedding grows with the number of views, from 1.8 dB at 8 views to 2.4 dB at 32 views on RealEstate10K, and from 0.6 dB to 1.0 dB on DL3DV. On Objaverse, every method gains little beyond 8 views, and CaPET stays ahead of all of them at every view count.

Qualitative Results. Figure 6 shows three test images of RealEstate10K, on which CaPET leads RayRoPE by 2.3 to 2.5 dB PSNR. Appendix F shows ten more images of RealEstate10K.

5.2 Analysis

Placement. The ablation in Table 4 switches on the three embeddings one at a time. Each embedding alone improves over No Embedding on every dataset, and the hidden embedding gains more than the input one on every dataset. Adding the input embedding to the hidden one improves every dataset, and adding the value-output embedding gives the best PSNR on every dataset.

Geometry. The ablation in Table 5 keeps the three locations and changes the coordinates that the embedding codes. The extrinsic and projection matrices, which are not orthogonal, fall below No Embedding on RealEstate10K and DL3DV and on average. The ray improves RealEstate10K, where

nearby cameras look in similar directions, but falls below No Embedding on Objaverse, where two views of the same point look along very different directions. Adding the camera origin to the ray does not help. The 3D point gives the best result on all three datasets.

Fast-Weight Networks. Table 6 replaces the SwiGLU fast weights with MLPs of depth two to four. CaPET places one embedding at the input of each weight matrix (Fig. 2), together with the value-output embedding, and improves every network on RealEstate10K, by 1.80 to 3.23 dB.

Table 6: **CaPET on other fast weights.** PSNR on Re10K for diverse fast-weight networks of Fig. 2.

fast weights	No Embedding	CaPET (ours)	Δ
SwiGLU	21.61	23.41	+1.80
2-layer MLP	20.51	**23.74**	**+3.23**
3-layer MLP	21.70	23.59	+1.89
4-layer MLP	21.67	23.55	+1.88

5.3 Other Camera-Conditional Tasks

3D Reconstruction. We apply CaPET to tttLRM (Wang et al., 2026), which predicts 3D Gaussians (Kerbl et al., 2023) from posed images with LaCT layers, and train its full-size model from scratch on DL3DV (Appendix D). CaPET improves PSNR by 0.81 dB over No Embedding and by 0.53 dB over PRoPE, the strongest baseline, and is the best in all three metrics (Table 7). Its PSNR is higher than that of PRoPE in 137 of the 140 test scenes.

Table 7: **3D reconstruction.** tttLRM trained and tested on DL3DV.

method	PSNR ↑	SSIM ↑	LPIPS ↓
No Embedding	15.28	0.367	0.640
PRoPE	15.56	0.375	0.615
RayRoPE	15.42	0.378	0.626
CaPET (ours)	**16.09**	**0.396**	**0.598**

Camera-Controlled Video Generation. We follow the task of ReCamMaster (Bai et al., 2025): given a source video and a new camera trajectory, the model generates the same scene as seen along the new trajectory. We use the autoregressive video model of LaCT (Zhang et al., 2026b), initialized from Wan2.1-T2V-1.3B (Wan Team, 2025), which combines sliding-window attention with a TTT layer. No Embedding keeps only the camera encoder of ReCamMaster, and CaPET here uses an earlier variant of its depth (Appendix D). We measure how well each video follows the target trajectory with RotErr and CamMC (He et al., 2025; Bai et al., 2025) on COLMAP poses (Schönberger & Frahm, 2016), and its quality with FVD (Unterthiner et al., 2018). CaPET lowers FVD from 557.7 to 354.4, the rotation error from 19.4 to 17.1 degrees (lower in 29 of the 43 pairs), and CamMC from 0.589 to 0.544 (Table 8).

Table 8: **Camera-controlled video generation.** RotErr (degrees) and CamMC over the 43 pairs that COLMAP registers for both methods, FVD over all 64.

method	RotErr ↓	CamMC ↓	FVD ↓
No Embedding	19.44	0.589	557.7
CaPET (ours)	**17.10**	**0.544**	**354.4**

6 Conclusion

We presented CaPET, a camera embedding that acts inside the TTT layer. Although a TTT layer computes no attention scores, its update and apply compare queries with keys through implicit scores, one at the input of each weight matrix. CaPET embeds the camera at these scores and at the values and outputs, where two views meet only through their relative camera, and builds each embedding as an orthogonal rotation of the ray and the 3D point of an image patch, since the fast weights have no softmax. It outperforms the TTT baseline and camera embeddings designed for attention on three novel view synthesis datasets, gains on every fast-weight network we test, and improves 3D reconstruction and camera-controlled video generation, at the linear cost of TTT.

Limitations and Future Work. The TTT layer as a whole is not fully relative, since the first term of each apply keeps the absolute camera of the query, and making this term relative is left for future work. Because of our limited compute, most of our experiments use a small model and one random seed, so larger TTT models and longer view sequences, which the linear cost of TTT makes affordable, remain to be tested. Finally, the implicit scores do not depend on cameras, so the same placement can carry other positional embeddings in TTT layers, such as time in video.

AI USE STATEMENT

In this work, we used generative AI tools to assist in writing proofs, to propose or refine hypotheses, to implement methods, to assist with translation, and to interpret results. We have not used generative AI tools to formulate mathematical claims, to provide critical ingredients for proving them, to help develop theoretical models or conceptual frameworks, to design or give feedback on the research methodology or experiments, or to support qualitative and thematic data analysis. Generating synthetic data sets and cleaning or reformatting datasets are not applicable to this work.

Additionally, we used generative AI tools to brainstorm, to search for information, to identify and summarize relevant literature, to create or edit software code, to create figures from the authors' hand-drawn sketches, to edit the paper for readability, and to format references. We have reviewed all AI-assisted work, and the authors verified the code and checked the experimental results produced with generative AI. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Jimmy Ba, Geoffrey E. Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.

Jianhong Bai, Menghan Xia, Xiao Fu, Xintao Wang, Lianrui Mu, Jinwen Cao, Zuozhu Liu, Haoji Hu, Xiang Bai, Pengfei Wan, and Di Zhang. ReCamMaster: Camera-controlled generative rendering from a single video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 113506–113543, 2025.

David Charatan, Sizhe Lester Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelSplat: 3D gaussian splats from image pairs for scalable generalizable 3D reconstruction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Xingyu Chen, Yue Chen, Yuliang Xiu, Andreas Geiger, and Anpei Chen. TTT3R: 3D reconstruction as test-time training. In *International Conference on Learning Representations (ICLR)*, 2026.

Karan Dalal, Daniel Koceja, Jiarui Xu, Yue Zhao, Shihao Han, Ka Chun Cheung, Jan Kautz, Yejin Choi, Yu Sun, and Xiaolong Wang. One-minute video generation with test-time training. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17702–17711. IEEE, 2025.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3D objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Ruiqi Gao, Aleksander Hołyński, Philipp Henzler, Arthur Brussee, Ricardo Martin-Brualla, Pratul Srinivasan, Jonathan T. Barron, and Ben Poole. CAT3D: Create anything in 3D with multi-view diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Dongchen Han, Yining Li, Tianyu Li, Zixuan Cao, Ziming Wang, Jun Song, Yu Cheng, Bo Zheng, and Gao Huang. ViT^3: Unlocking test-time training in vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 51–61, 2026.

Hao He, Yinghao Xu, Yuwei Guo, Gordon Wetzstein, Bo Dai, Hongsheng Li, and Ceyuan Yang. CameraCtrl: Enabling camera control for video diffusion models. In *International Conference on Learning Representations (ICLR)*, 2025.

Yihui He, Rui Yan, Katerina Fragkiadaki, and Shoou-I Yu. Epipolar transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Tianchong Jiang, Jingtian Ji, Xiangshan Tan, Jiading Fang, Anand Bhattad, Vitor Guizilini, and Matthew R. Walter. Do you know where your camera is? View-invariant policy learning with camera conditioning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

Haian Jin, Hanwen Jiang, Hao Tan, Kai Zhang, Sai Bi, Tianyuan Zhang, Fujun Luan, Noah Snavely, and Zexiang Xu. LVSM: A large view synthesis model with minimal 3D inductive bias. In *International Conference on Learning Representations (ICLR)*, 2025.

Haian Jin, Rundi Wu, Tianyuan Zhang, Ruiqi Gao, Jonathan T. Barron, Noah Snavely, and Aleksander Hołyński. ZipMap: Linear-time stateful 3D reconstruction via test-time training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21748–21759, 2026.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. https://kellerjordan.github.io/posts/muon/.

Shun Kenney and Teppei Suzuki. DPPE: Rethinking camera-based positional encoding for scaling multi-view transformers. *arXiv preprint arXiv:2606.31585*, 2026.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023.

Xin Kong, Shikun Liu, Xiaoyang Lyu, Marwan Taher, Xiaojuan Qi, and Andrew J. Davison. EscherNet: A generative model for scalable view synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pp. 9–50. Springer, 1998.

Chunyang Li, Yuanbo Yang, Jiahao Shao, Hongyu Zhou, Katja Schwarz, and Yiyi Liao. ReRoPE: Repurposing RoPE for relative camera control. In *ACM SIGGRAPH Conference Papers*, 2026a.

Ruilong Li, Brent Yi, Junchen Liu, Hang Gao, Yi Ma, and Angjoo Kanazawa. Cameras as relative positional encoding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yining Li, Dongchen Han, Zeyu Liu, Hanyi Wang, Yulin Wang, and Gao Huang. Linearizing vision transformer with test-time training. In *International Conference on Machine Learning (ICML)*, 2026b.

Lu Ling, Yichen Sheng, Zhi Tu, Wentian Zhao, Cheng Xin, Kun Wan, Lantao Yu, Qianyu Guo, Zixun Yu, Yawen Lu, et al. DL3DV-10K: A large-scale scene dataset for deep learning-based 3D vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Changkun Liu, Jiezhi Yang, Zeman Li, Yuan Deng, Jiancong Guo, and Luca Ballan. Mem3R: Streaming 3D reconstruction with hybrid memory via test-time training. *arXiv preprint arXiv:2604.07279*, 2026a.

Junchen Liu, Sven Elflein, Or Litany, Zan Gojcic, and Ruilong Li. Test-time training with KV binding is secretly linear attention. In *International Conference on Machine Learning (ICML)*, 2026b.

Yingfei Liu, Tiancai Wang, Xiangyu Zhang, and Jian Sun. PETR: Position embedding transformation for multi-view 3D object detection. In *European Conference on Computer Vision (ECCV)*, 2022.

Ziqiao Ma, Xueyang Yu, Haoyu Zhen, Yuncong Yang, Joyce Chai, and Chuang Gan. Fast spatial memory with scalable elastic test-time training. In *European Conference on Computer Vision (ECCV)*, 2026.

Takeru Miyato, Bernhard Jaeger, Max Welling, and Andreas Geiger. GTA: A geometry-aware attention mechanism for multi-view transformers. In *International Conference on Learning Representations (ICLR)*, 2024.

Lingteng Qiu, Guanying Chen, Xiaodong Gu, Qi Zuo, Mutian Xu, Yushuang Wu, Weihao Yuan, Zilong Dong, Liefeng Bo, and Xiaoguang Han. RichDreamer: A generalizable normal-depth diffusion model for detail richness in text-to-3D. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Mehdi S. M. Sajjadi, Henning Meyer, Etienne Pot, Urs Bergmann, Klaus Greff, Noha Radwan, Suhani Vora, Mario Lučić, Daniel Duckworth, Alexey Dosovitskiy, Jakob Uszkoreit, Thomas Funkhouser, and Andrea Tagliasacchi. Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Vincent Sitzmann, Semon Rezchikov, William T. Freeman, Joshua B. Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *PMLR*, pp. 9229–9248, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (Learn at test time): RNNs with expressive hidden states. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, pp. 57503–57522, 2025.

Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphaël Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

Wan Team. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Chen Wang, Hao Tan, Wang Yifan, Zhiqin Chen, Yuheng Liu, Kalyan Sunkavalli, Sai Bi, Lingjie Liu, and Yiwei Hu. tttLRM: Test-time training for long context and autoregressive 3D reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual geometry grounded transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025a.

Ke Alexander Wang, Jiaxin Shi, and Emily B. Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *arXiv preprint arXiv:2501.12352*, 2025b.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004.

Yu Wu, Minsik Jeon, Jen-Hao Rick Chang, Oncel Tuzel, and Shubham Tulsiani. RayRoPE: Projective ray positional encoding for multi-view attention. In *European Conference on Computer Vision (ECCV)*, 2026.

Yuqi Wu, Wenzhao Zheng, Jie Zhou, and Jiwen Lu. Point3R: Streaming 3D reconstruction with explicit spatial pointer memory. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Chendong Xiang, Jiajun Liu, Jintao Zhang, Xiao Yang, Zhengwei Fang, Shizun Wang, Zijun Wang, Yingtian Zou, Hang Su, and Jun Zhu. Geometry-aware rotary position embedding for consistent video world model. *arXiv preprint arXiv:2602.07854*, 2026.

Yichen Xie, Depu Meng, Chensheng Peng, Yihan Hu, Quentin Herau, Masayoshi Tomizuka, and Wei Zhan. URoPE: Universal relative position embedding across geometric spaces. In *European Conference on Computer Vision (ECCV)*, 2026.

Cheng Zhang, Boying Li, Meng Wei, Yan-Pei Cao, Camilo Gambardella, Dinh Phung, and Jianfei Cai. Unified camera positional encoding for controlled video generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026a.

Kai Zhang, Sai Bi, Hao Tan, Yuanbo Xiangli, Nanxuan Zhao, Kalyan Sunkavalli, and Zexiang Xu. GS-LRM: Large reconstruction model for 3D gaussian splatting. In *European Conference on Computer Vision (ECCV)*, 2024.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T. Freeman, and Hao Tan. Test-time training done right. In *International Conference on Learning Representations (ICLR)*, 2026b.

Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *ACM Transactions on Graphics*, 37(4), 2018.

A Step-by-Step Derivation of the Update Equations

This appendix derives Eqs. 6 and 7 from scratch. It needs only the chain rule, with sums written out entrywise.

Setup. The update is defined by Eq. 3: one gradient descent step on the per-token losses L_i. Each L_i is the negative of a scalar s_i, so descent on L_i is ascent on

$$s_i \;:=\; \langle v_i,\, f_W(k_i)\rangle \;=\; v_i^\top W_2\, \sigma(W_1 k_i), \qquad \Delta W_m = \sum_i \mathrm{lr}_i\, \frac{\partial s_i}{\partial W_m}. \tag{15}$$

It suffices to differentiate the scalar s_i of a single token with respect to W_2 and W_1, then sum over the tokens. The result will be exactly Eqs. 6 and 7 of the main text, so this appendix is the proof of both.

Tool: the gradient of $a^\top W b$ is the outer product $ab^\top$. Write the scalar entrywise:

$$a^\top W b \;=\; \sum_r \sum_c a_r\, W_{rc}\, b_c. \tag{16}$$

Each entry W_{rc} appears exactly once, multiplied by $a_r b_c$, so

$$\frac{\partial\,(a^\top W b)}{\partial W_{rc}} \;=\; a_r\, b_c, \qquad \text{that is,} \qquad \frac{\partial\,(a^\top W b)}{\partial W} \;=\; a\, b^\top. \tag{17}$$

In words, differentiating a scalar of the form (left vector) W (right vector) with respect to W leaves exactly the outer product of the two vectors. Both updates below are this one fact applied at the right place.

Gradient with respect to W_2. Only the factor W_2 in 15 depends on W_2; abbreviate the hidden vector $h_i := \sigma(W_1 k_i)$, so $s_i = v_i^\top W_2\, h_i$. Apply 17 with $a = v_i$, $b = h_i$:

$$\frac{\partial s_i}{\partial W_2} \;=\; v_i\, h_i^\top \qquad\Longrightarrow\qquad \Delta W_2 = \sum_i \mathrm{lr}_i\, v_i\, h(k_i)^\top. \tag{18}$$

This is Eq. 6, with the value v_i on the left and the hidden vector $h(k_i)$ of the key on the right.

Gradient with respect to W_1. Move W_2 onto the value and name the two vectors that matter:

$$w := W_2^\top v_i \in \mathbb{R}^{d_h}, \qquad a := W_1 k_i \in \mathbb{R}^{d_h}, \qquad s_i \;=\; w^\top \sigma(a) \;=\; \sum_{r=1}^{d_h} w_r\, \sigma(a_r). \tag{19}$$

In words, s_i is a sum over hidden coordinates, and coordinate r involves only the numbers w_r and a_r, so we can differentiate coordinate by coordinate:

$$\frac{\partial s_i}{\partial a_r} \;=\; w_r\, \sigma'(a_r) \qquad\Longrightarrow\qquad \frac{\partial s_i}{\partial a} \;=\; \sigma'(a) \odot w \;=\; g_i, \tag{20}$$

which is exactly the g_i of Eq. 7 (there $g_i := -\partial L_i/\partial(W_1 k_i)$, and $L_i = -s_i$). Since $a = W_1 k_i$ is linear in W_1 (row r: $a_r = \sum_c (W_1)_{rc} (k_i)_c$), the chain rule and 17 give

$$\frac{\partial s_i}{\partial W_1} \;=\; \frac{\partial s_i}{\partial a}\, k_i^\top \;=\; g_i\, k_i^\top \qquad\Longrightarrow\qquad \Delta W_1 = \sum_i \mathrm{lr}_i\, g_i\, k_i^\top. \tag{21}$$

In words, g_i is the value pulled back through $W_2^\top$ and scaled by the slope of the nonlinearity, and the key k_i sits on the right side of the outer product.

Why the derivation is exact. No approximation was made: s_i is linear in W_2, and its dependence on W_1 passes through one linear map followed by a coordinatewise function. The gradients above are exact, and since the update is defined as one gradient step, Eqs. 6–7 are identities, not expansions. The term g_i is not defined by hand, since the chain rule produces it at the first layer.

The update in our experiments. Our TTT layers follow LaCT (Zhang et al., 2026b), whose update differs from the plain gradient step of Eq. 3 in three ways. First, each token has its own learning rate, which only rescales its outer product in Eqs. 6–7. Second, for each weight matrix, the sum G of these outer products passes through the Newton–Schulz iteration of Muon (Jordan et al., 2024) before it is added, and the iteration returns $\phi(GG^\top)\, G$ for a matrix function ϕ. Applied to an embedded query, this expression expands into products of the stored terms in which every camera matrix meets another one, so the cameras still enter only as relative cameras. Third, each row of an updated weight matrix is rescaled to its initial norm. The scale factors depend on the current weights, so, like the first terms of Eqs. 10–12, they see the absolute camera of each token.

B SwiGLU and n-Layer MLP Fast Weights

The fast weights of LaCT, used in all our experiments, replace the hidden map of Eq. 5 by the gated SwiGLU

$$f_W(x) \;=\; W_2\, h(x), \qquad h(x) \;=\; \sigma(W_1 x) \odot (W_3 x), \qquad W_1, W_3 \in \mathbb{R}^{d_h \times d}, \tag{22}$$

with a gate branch $\sigma(W_1 x)$ and a linear branch $W_3 x$. Everything in the main text holds unchanged, and the only new element is a second term stored at the first layer.

Updates. Repeating the derivation of Appendix A with $s_i = \sum_r w_r\, \sigma(a_r)\, b_r$, where $w = W_2^\top v_i$, $a = W_1 k_i$, $b = W_3 k_i$:

$$\Delta W_2 = \sum_i \mathrm{lr}_i\, v_i\, h(k_i)^\top, \qquad \Delta W_1 = \sum_i \mathrm{lr}_i\, g_i\, k_i^\top, \qquad \Delta W_3 = \sum_i \mathrm{lr}_i\, u_i\, k_i^\top, \tag{23}$$

$$g_i = \sigma'(W_1 k_i) \odot (W_3 k_i) \odot (W_2^\top v_i), \qquad u_i = \sigma(W_1 k_i) \odot (W_2^\top v_i). \tag{24}$$

In words, the gate branch receives the pulled-back value weighted by the gate's slope and the linear branch's activation (g_i), and the linear branch receives it weighted by the gate itself (u_i). Both follow the coordinatewise computation of Appendix A with one extra factor.

Same two implicit scores. W_1 and W_3 share the same input k_i, so both first-layer updates carry the *same* key on their right side, and the expanded apply of Section 4.1, Eqs. 8–9, gains one line with the same input score,

$$(W_1{+}\Delta W_1)\, q_j = W_1 q_j + \sum_i \mathrm{lr}_i \,\langle q_j,\, k_i\rangle \; g_i, \qquad (W_3{+}\Delta W_3)\, q_j = W_3 q_j + \sum_i \mathrm{lr}_i \,\langle q_j,\, k_i\rangle \; u_i, \tag{25}$$

while the W_2 line is Eq. 8 with the SwiGLU hidden vector $\tilde{h}$. The three weight matrices thus hold only two implicit scores, the input score $\langle q_j,\, k_i\rangle$, which now weights both g_i and u_i, and the hidden score $\left\langle \tilde{h}(q_j),\, h(k_i)\right\rangle$. CaPET therefore places its embeddings at the same two scores as for the two-layer MLP, one input embedding shared by W_1 and W_3 and one hidden embedding before W_2 (Fig. 2). With the hidden embedding, $W_2^\top v_i$ in g_i and u_i becomes $E_i^\top W_2^\top v_i$.

n-layer MLP. A deeper network adds implicit scores instead. Write an n-layer MLP through its activations,

$$a^{(0)}(x) = x, \qquad a^{(m)}(x) = \sigma\big(W_m\, a^{(m-1)}(x)\big) \;\; (m < n), \qquad f_W(x) = W_n\, a^{(n-1)}(x). \tag{26}$$

The derivation of Appendix A applies at every layer. The chain rule splits the gradient of s_i at the pre-activation $z_i^{(m)} := W_m\, a^{(m-1)}(k_i)$, and with $g_i^{(m)} := \partial s_i / \partial z_i^{(m)}$, which is v_i for the last layer, the update of each weight matrix is

$$\Delta W_m \;=\; \sum_i \mathrm{lr}_i\, g_i^{(m)}\, a^{(m-1)}(k_i)^\top, \qquad m = 1, \ldots, n. \tag{27}$$

Each term is again an outer product with the key, now its activation at the input of W_m, on its right side. In the apply, the query reaches W_m as $\tilde{a}^{(m-1)}(q_j)$, the same activation under the updated earlier layers, and

$$(W_m + \Delta W_m)\, \tilde{a}^{(m-1)}(q_j) \;=\; W_m\, \tilde{a}^{(m-1)}(q_j) \;+\; \sum_i \mathrm{lr}_i \left\langle \tilde{a}^{(m-1)}(q_j),\, a^{(m-1)}(k_i)\right\rangle g_i^{(m)}. \tag{28}$$

Every weight matrix thus holds one implicit score at its input, and CaPET places one embedding there, n in total (Fig. 2). For $n = 2$, with $a^{(1)} = h$, $g_i^{(2)} = v_i$, and $g_i^{(1)} = g_i$, Eq. 28 gives back Eqs. 8 and 9. The number of embeddings therefore follows the network: one for a linear fast weight, two for the two-layer MLP and SwiGLU, and n for an n-layer MLP.

C Implementation Details

Model. The LaCT model for novel view synthesis (Zhang et al., 2026b) is a variant of LVSM (Jin et al., 2025) whose blocks each contain a self-attention layer within each image, a TTT layer over the tokens of all views, and an MLP. The TTT layer is thus the only layer in which tokens of different views interact. Its SwiGLU fast weights are updated with Muon (Jordan et al., 2024), which orthogonalizes each update, and Appendix A describes how this update differs from the plain gradient step of Eq. 3.

CaPET. CaPET rotates the query and key with $F = 21$ frequencies per coordinate and the hidden vectors with $F = 42$, so its rotations cover 252 of the 256 query and key channels and 504 of the 512 hidden channels, and the remaining channels stay unrotated. Each frequency is scaled by a learnable gain initialized at one. The depth layer of Eq. 13 is initialized at zero, so every depth starts at one in normalized scene units, and its log depth is soft-clamped to $[-2.5, 2.5]$. The rotary pairs are split evenly between the ray and the 3D point on RealEstate10K and DL3DV, and one to three on Objaverse, whose views are far apart.

Training. Each model is trained from scratch on one GPU for 30K iterations with batch size 16 and AdamW, with learning rate 10^{-4} after 1.5K warm-up iterations. Each training example takes 8 input and 8 target views from a random window of at most 192 frames. The loss is the mean squared error, and an LPIPS loss is added after 5K iterations.

Datasets. RealEstate10K (Zhou et al., 2018) contains videos of houses with smooth camera paths, and we use 66K clips for training and 7.3K for testing. DL3DV (Ling et al., 2024) contains videos

in which the camera walks around a real scene, and we train on 10K scenes and test on the 140 scenes of the DL3DV benchmark. For Objaverse (Deitke et al., 2023), we use the G-Objaverse renderings (Qiu et al., 2024), hold out 500 objects for testing, and train on the other 19.5K. Images are 256×256, or 256×448 on DL3DV, which keeps nearly the full frame.

Evaluation. For each test sequence, we take its central 128 frames, or all frames if it is shorter, before choosing the 8 input views and the 4 target views, which lie midway between input views. On Objaverse, we evaluate the 499 held-out objects that have all 40 renderings.

Baselines. For GTA, we apply its rigid camera matrix to half of the channels of the query and key. For PRoPE, we apply the projective matrix P_i to half of the channels and RoPE of the 2D image coordinates to the other half, on the query, key, value, and output as in the original. For RayRoPE, we use its coordinates, built from the camera origin and the rays of each patch at a depth and a depth uncertainty predicted by a linear layer, on the query, key, and value, with the inverse on the output. RayRoPE expresses its coordinates in the frame of each query camera, which would need separate fast weights for every target view in a TTT layer, so we express them in the world frame.

D Other Camera-Conditional Tasks

3D Reconstruction. We train the full-size tttLRM model (Wang et al., 2026), with 24 blocks and width 768, from scratch on DL3DV at 272×480 for 15K iterations, with 8 input and 8 target views, batch size 8, and the losses of tttLRM. We render the target views of the 140 DL3DV benchmark scenes, with the input views chosen by the k-means view selection of tttLRM, and report PSNR, SSIM, and LPIPS. PRoPE, RayRoPE, and CaPET act in the TTT layers as in novel view synthesis.

Camera-Controlled Video Generation. The autoregressive video model of LaCT (Zhang et al., 2026b) generates three latent frames at a time after the clean source video. The camera encoder of ReCamMaster (Bai et al., 2025), which every model keeps, adds the target camera to the features of each block. In this model, CaPET uses an earlier variant of the depth in Eq. 13, in which the log depth comes from one channel of the value and is measured relative to the depth of the point closest to all camera axes. We train on the MultiCamVideo dataset of ReCamMaster, with 81 frames at 480×832, for 14K iterations, and evaluate 64 held-out pairs of source videos and target trajectories with 40 sampling steps. COLMAP (Schönberger & Frahm, 2016) estimates the camera poses of each generated video, and RotErr, TransErr, and CamMC compare them with the target trajectory after aligning the scale. Over the 22 of the 43 pairs with camera translation, the translation error is 0.468 for No Embedding and 0.481 for CaPET. The same pipeline applied to the real target videos gives a rotation error of 3.1 degrees, which bounds how low the metric can go.

E Computational Cost

Table 9 lists the parameters and FLOPs of each embedding in the evaluation setting of Section 5.1. CaPET adds 0.04% parameters and 0.13% FLOPs, a cost similar to that of RayRoPE, and keeps the linear cost of the TTT layer.

Table 9: Computational cost of each camera embedding (8 input + 4 target views, 256^2).

method	# Params (M) ↓	FLOPs (G) ↓
No Embedding	9.646	93.01
PRoPE	9.646	93.12
RayRoPE	9.649	93.06
CaPET (ours)	9.650	93.13

F Additional Qualitative Results

Figure 7 shows the ten test images of RealEstate10K, at most one per scene, on which CaPET leads RayRoPE in PSNR by the most, leaving out the three images of Fig. 6. On these images, CaPET leads RayRoPE by 2.3 to 2.7 dB. The input column shows four of the eight input views.

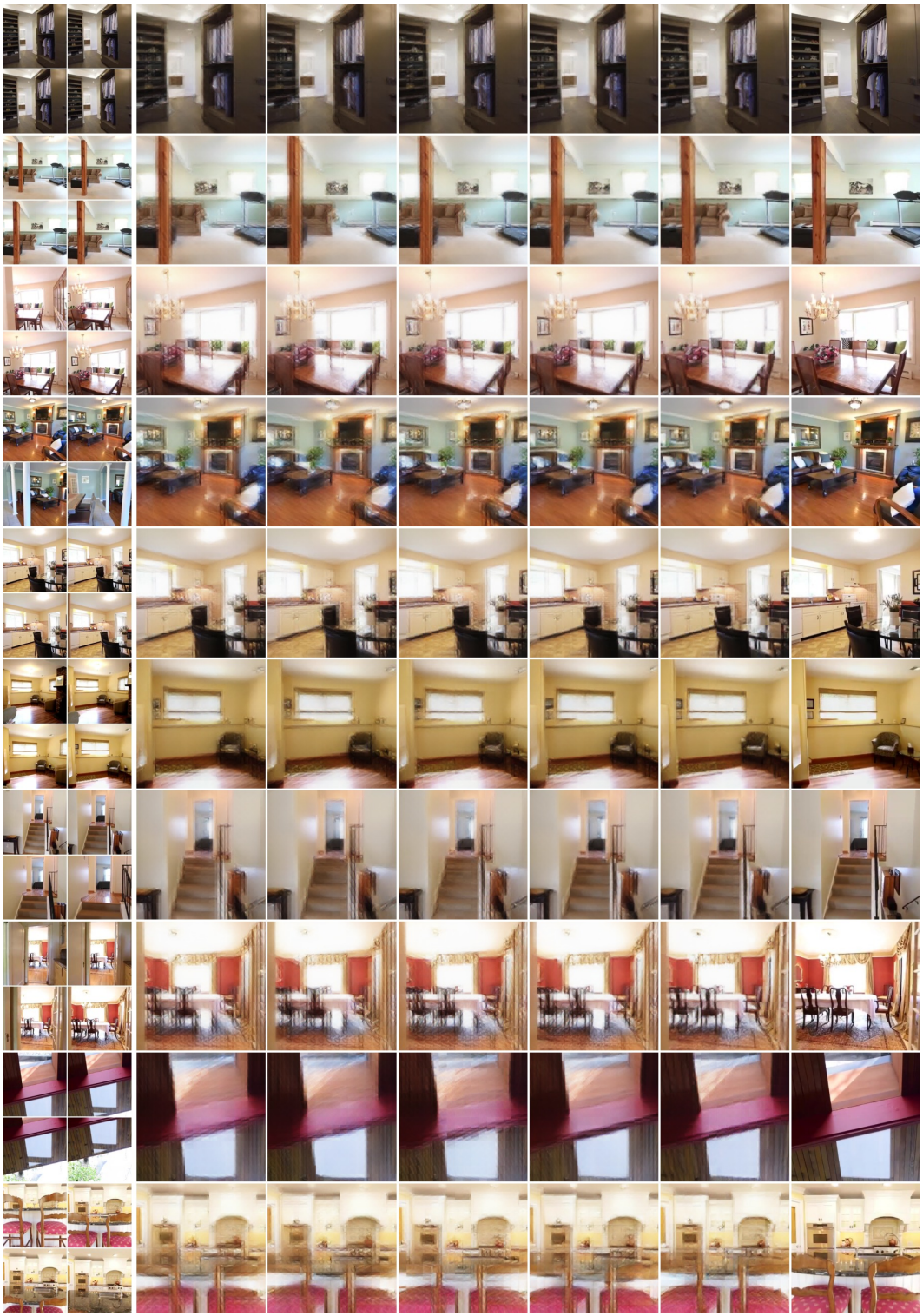


Figure 7: Qualitative comparison on RealEstate10K (Zhou et al., 2018).