CATRF: Codec-Adaptive TriPlane Radiance Fields for Volumetric Content Delivery

Supplementary Material

A. Caching for SCL Training

If we naively invoke the codec round trip $\mathcal{C}_q$ on every mini-batch, the encoding cost quickly dominates training time, especially for dynamic scenes where a video codec must operate on a sequence of frames (see Tab. 4). To make our standard-codec-in-the-loop (SCL) training practical, we maintain a per-frame cache of decoded feature planes and density volumes $(\widehat{\mathcal{P}}, \widehat{D})$, and only refresh this cache when necessary. For each frame t and each axis $ax \in \{xy, xz, yz\}$ we cache the decoded TriPlane $\widehat{P}_t^{ax}$ and a snapshot of the raw plane P_t^{ax} at the time of refresh. We do the same for the density grid D_t. Our refresh policy consists of two cases:

Step-based refresh. We keep track of the last refresh step g_{cache} and the current global step g. Every M training steps, i.e., when $g - g_{\text{cache}} \geq M$, we re-run the codec over the entire video segment to update the cache.

Change-based refresh. Step-based refresh alone may lag behind rapid parameter changes early in training. We therefore also monitor how much the current TriPlane has drifted from the last cached snapshot. If the relative change between the current planes and their cached snapshots exceeds a threshold ϵ (measured by a normalized ℓ_2 difference), we trigger an early refresh even if $g - g_{\text{cache}} < M$. In practice, this makes the cache more responsive at the beginning of training while keeping codec round trips infrequent once the model stabilizes.

Putting these pieces together, the SCL training loop with caching can be summarized as:

1. Sample a frame index t, rays $\{(\mathbf{o}, \mathbf{d})\}$, and camera pose π.
2. If the cache is empty or $g - g_{\text{cache}} \geq M$, mark *refresh* as true.
3. If the relative change between the current (P_t^{ax}, D_t) and their cached snapshots exceeds a threshold ϵ, also mark *refresh* as true.
4. If *refresh* is true, then run the encode–decode codec round trip (as illustrated in Fig. 2) for the video segment.
5. Update $g_{\text{cache}} \leftarrow g$.
6. STE substitution using cached reconstructions:
   (a) For each axis $ax \in \{xy, xz, yz\}$:
   $$\widehat{P}_t^{ax} \leftarrow \text{cached decoded plane},$$
   $$\widetilde{P}_t^{ax} \leftarrow \widehat{P}_t^{ax} + \big(P_t^{ax} - \texttt{detach}(P_t^{ax})\big).$$
   (b) For density:
   $$\widehat{D}_t \leftarrow \text{cached decoded density},$$
   $$\widetilde{D}_t \leftarrow \widehat{D}_t + \big(D_t - \texttt{detach}(D_t)\big).$$
7. Render and compute losses:
   $$I \leftarrow \mathcal{R}(\widetilde{\mathcal{P}}_t, \widetilde{D}_t, \pi; \phi),$$

Tab. 4 shows that increasing the refresh interval M yields a favorable trade-off between accuracy and training efficiency. It suggests that relatively infrequent cache updates (e.g., $M = 128$) already capture most of the benefit of SCL training, while keeping the overhead of expensive codec round trips manageable.

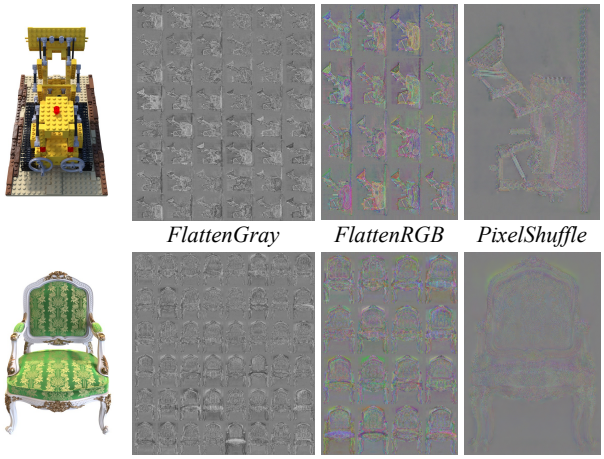

Figure 7. Visualization of appearance-plane canvases under different packing strategies. All three layouts represent the same TriPlane features, but they induce different spatial statistics and texture patterns, which affect how well standard codecs exploit redundancy and preserve feature semantics after compression.

B. Decoder-Side Payload of Baseline Methods

In a streaming scenario, NeRF compression methods that train a *scene-specific* codec (e.g., NeRFCodec [32], Rate-aware NeRF [59], and CNC [10]) must transmit not only the encoded radiance-field representation but also the decoder-side codec parameters required to reconstruct NeRFs at the client. While prior work typically reports only the feature bitstream or backbone model size, in our evaluation these decoder parameters are part of the content payload and therefore must be included in the bitrate budget. In contrast, CATRF is encoded and decoded using standard codecs that are ubiquitous on commodity devices, avoiding this additional overhead (see Tab. 2 in the main paper).

Observed payload for NeRFCodec. For NeRFCodec [32], we adopt their official implementation, which uses `Cheng2020Anchor` neural image codec from CompressAI. Empirically, we observed that the following decoder-side modules are fine-tuned together with the radiance-field backbone on a per-scene basis, and thus must be transmitted alongside the encoded NeRFs for decoding:

- `decoder_adaptor`
- `context_prediction`
- `entropy_parameters`
- `entropy_bottleneck.quantiles`

Note that NeRFCodec employs two separate neural image codecs for the appearance and density streams, respectively.

Encoder-side modules, although also fine-tuned, are excluded from the bitrate budget since they are not required at the client for decoding. The authors of NeRFCodec do not release their bitrate budget estimation code. Rather than reporting the raw parameter size, we apply per-tensor quantization and entropy coding to estimate a more realistic compressed size:

1. **Per-tensor quantization.** For each tensor, we collect its values $x \in \mathbb{R}$ and compute

$$x_{\min} = \min(x), \quad x_{\max} = \max(x).$$

We quantize to q_{bits} bits (we use $q_{\text{bits}} = 8$ in our experiments). Let $L = 2^{q_{\text{bits}}}$ and

$$\text{scale} = \frac{x_{\max} - x_{\min}}{L - 1}.$$

Each element is mapped to an integer level

$$q = \text{round}\left(\frac{x - x_{\min}}{\text{scale}}\right),$$

and clamped to $[0, L-1]$.

2. **Entropy estimate.** We compute a histogram over the quantized levels and derive the empirical PMF p_i over $\{0, \ldots, L-1\}$. The Shannon bound gives an idealized bit cost

$$H_{\text{bits}} = N \sum_i -p_i \log_2 p_i,$$

where N is the number of elements and the sum is taken over non-zero probabilities.

3. **Per-tensor header.** To reconstruct a tensor from its quantized form, the decoder also needs lightweight side information: $(x_{\min}, x_{\max})$, q_{bits}, and the tensor shape. We therefore add a small header

$$\text{header\_bits} = 2 \cdot 32 + 8 + 32 \cdot d,$$

where d is the number of dimensions.

The total bit cost for a tensor is therefore

$$\text{bits} = H_{\text{bits}} + \text{header\_bits},$$

which is how we estimate the bitrate budget required for NeRFCodec's decoder-side parameters.

The compression procedure described above yields a raw decoder payload of roughly 24 MB in total (about 11.9 MB for the density codec and 12.8 MB for the appearance codec). After compression, this reduces to about 1.6 MB. These quantities are what we report as NeRFCodec's memory footprint, which represents the actual payload in a streaming application if we use NeRFCodec as the 3D representation.

Observed payload for CNC. For CNC [10], we directly reuse the bitrate estimation utilities provided in their official implementation, which have already included quantization and entropy coding. The difference between the sizes reported in our evaluation and those in the original paper stems from whether one includes the compressed MLP sizes and entropy model parameters in the total bitrate budget.

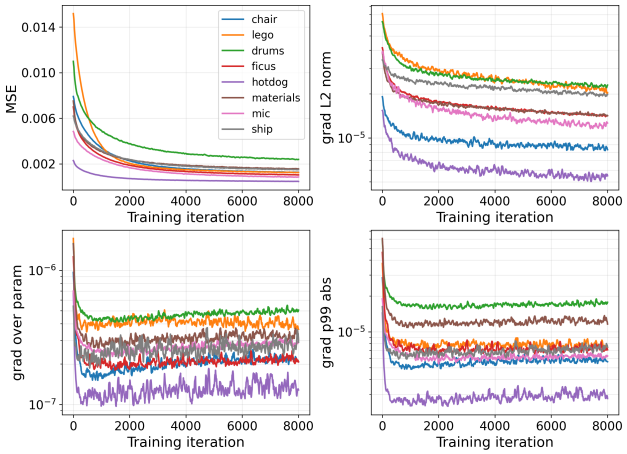

Figure 8. Training diagnostics of CATRF with STE as the gradient surrogate through the codec round trip. We plot the reconstruction MSE, gradient ℓ_2 norm of the feature plane parameters, the scale-normalized gradient magnitude, and the 99th-percentile absolute gradient magnitude (computed from 200k sampled parameters). All metrics remain bounded while the loss decreases, indicating stable optimization without exploding or vanishing gradients.

C. More Implementation Details

For the radiance-field backbone, we train for 30k iterations and adopt progressive scaling widely used in the literature. The plane resolutions (and line vectors) are bilinearly upsampled at steps [2000, 3000, 4000, 5500, 7000], and the alpha-mask is recomputed on a schedule [2000, 4000, 6000, 11000, 16000, 21000, 26000]. The MLP render has two hidden layers (128 units, ReLU) and a 3-channel head with sigmoid. Optimization uses Adam with learning rates $lr = 0.02$ for the radiance-field model and $lr = 0.001$ for the renderer MLP. When employing AbsMax (absolute max) quantization, we employ hard-coded upper and lower bounds. For the static branch (TensoRF backbone), density and appearance planes are globally quantized within preset ranges, $[-25, 25]$ and $[-5, 5]$, respectively. For the dynamic branch (DVGO backbone), density and appearance planes are both globally quantized within the preset range, $[-20, 20]$. For our STE-based caching, the change-based refresh uses a threshold $\epsilon = 0.05$ (5% normalized ℓ_2 drift) to trigger early codec refreshes.

D. Gradient Diagnostics

The core of CATRF is treating the entire codec encoding-decoding round trip as a black box and using a Straight-Through Estimator (STE) as its gradient surrogate. While this enables backpropagation, the resulting gradients are biased: standard codecs introduce non-linear distortions that an identity STE does not explicitly model, potentially causing gradient mismatch and training instability. Given the complexity of standard codecs (e.g., quantization, block transforms, entropy coding, and in-loop filtering), providing a formal convergence proof is intractable. Instead, we provide empirical evidence that STE yields stable and useful surrogate gradients in our training setup. Specifically, we

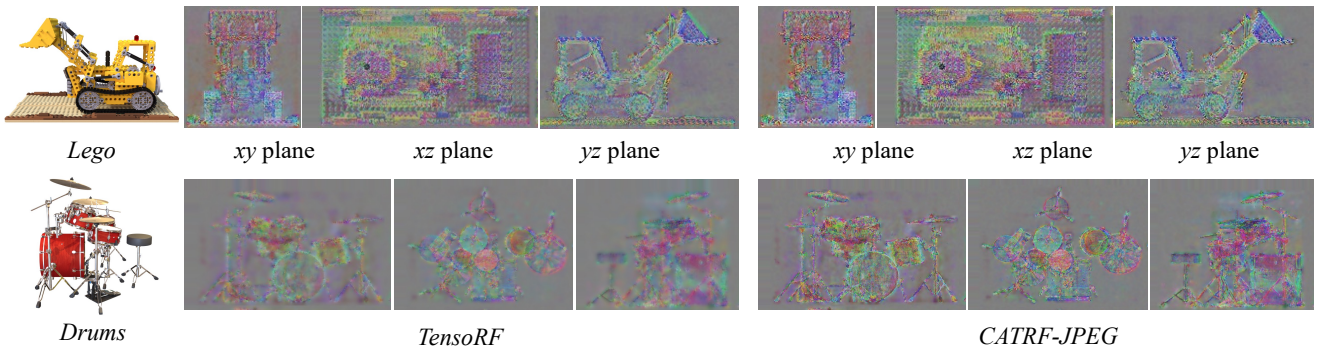

Figure 9. Comparison of visualized appearance planes packed with *FlattenRGB*. For a given scene, we show the xy, xz, and yz planes of vanilla TensoRF (left) and our CATRF-JPEG model (right). The CATRF canvases exhibit richer high-frequency structure than the vanilla TensoRF planes, reflecting that our SCL training explicitly adapts feature planes to the codec round trip. Because the codec's transform, quantization, and in-loop filtering tend to smooth the canvases, CATRF learns to encode more detail pre-encoding. Consequently, after decoding, the decoded patterns still preserve the underlying feature semantics needed for high-quality rendering.

log four statistics throughout training: (1) *MSE*, the reconstruction loss between rendered RGB and ground-truth images; (2) *grad L2 norm*, the ℓ_2 norm of gradients on the feature-plane parameters; (3) *grad over param*, a scale-normalized gradient magnitude $\|\nabla_\theta \mathcal{L}\|_2/(\|\theta\|_2 + \epsilon)$; and (4) *grad p99 abs*, the 99th percentile of $|\nabla_\theta \mathcal{L}|$ computed from a random sample of 200,000 gradient entries. As shown in Fig. 8, these metrics remain bounded (with no NaN/Inf entries) while the MSE decreases smoothly, indicating that optimization proceeds normally under STE despite the biased gradients. This diagnostic supports our design choice that an STE-through-codec surrogate is sufficient for stable SCL training.

E. Gradient-Surrogate Ablation

Surrogate-gradient design for non-differentiable operators is a long-standing problem in quantized and discrete optimization. Beyond the vanilla identity Straight-Through Estimator (STE), many alternatives have been proposed to reduce gradient bias or better approximate the local effect of quantization. Since CATRF treats the full codec round trip as a black box, it is natural to ask whether a more sophisticated surrogate than vanilla STE would further improve optimization. We therefore compare three strategies: vanilla STE [3], modified STE (mSTE) [36], and a hybrid STE+SPSA estimator inspired by black-box gradient approximation [41, 48]. Tab. 6 reports rate–distortion performance (PSNR/SSIM/LPIPS), compressed size, and training time.

Vanilla STE uses the identity surrogate in the backward pass, i.e., the gradient is copied directly to the pre-codec feature plane. Although this surrogate is biased, recent work has noted that STE can nevertheless provide high-quality biased gradients in practice [56]. Our results are consistent with this observation. Vanilla STE provides the most robust optimization overall, particularly in the low-bitrate regime where codec-induced distortions are strongest. We therefore use vanilla STE as the default gradient surrogate in all main experiments.

The modified STE (mSTE) rescales the detached codec error using the standard deviation of the quantization error. Let $\epsilon = \hat{\mathbf{P}} - \mathbf{P}$ denote the codec-induced perturbation between the decoded feature plane $\hat{\mathbf{P}}$ and the pre-codec feature plane $\mathbf{P}$. mSTE takes the form

$$\widetilde{\mathbf{P}} = \mathbf{P} + \mathrm{sg}(\epsilon) \cdot \frac{\sigma(\epsilon)}{\mathrm{sg}(\sigma(\epsilon))},$$

where $\mathrm{sg}(\cdot)$ is the stop-gradient operator and $\sigma(\epsilon)$ is the standard deviation of the perturbation. Intuitively, this preserves the STE forward pass while modulating the backward signal using the observed perturbation scale. In our experiments, mSTE performs competitively at high bitrate, but degrades noticeably at low bitrate. A plausible reason is that, under stronger codec distortion, the perturbation becomes less like a small local quantization residual and more strongly reflects the non-linear artifacts from the black-box codec. Thus, a standard-deviation reweighting can become a poor proxy for the true local sensitivity instead, making optimization less stable and less effective than plain identity STE.

We also evaluate SPSA, a gradient estimator based on symmetric random perturbations [48]. For an objective $C_q(.)$, SPSA estimates its gradient by running two function evaluations:

$$\hat{g}_i(\mathbf{P}) = \frac{\mathcal{C}_q(\mathbf{P} + \epsilon\Delta) - \mathcal{C}_q(\mathbf{P} - \epsilon\Delta)}{2\epsilon\Delta_i},$$

where Δ is a random perturbation vector sampled from a symmetric Bernoulli distribution, i.e., $\Delta_i = \pm 1$ with probability of 0.5; ϵ is a small perturbation scale. In our implementation, we use a hybrid STE+SPSA scheme, where STE is used on cached steps, while SPSA is applied only when the codec cache is refreshed. This design keeps the method computationally feasible, since applying SPSA at every step would require multiple additional codec round trips per iteration and would be prohibitively expensive in our setting. Empirically, STE+SPSA yields the worst reconstruction quality and the highest training time. We attribute this to the high variance of zeroth-order estimates and the poor signal-to-noise ratio introduced by severe codec non-linearity. Although SPSA is less biased in principle, its gradient estimates can be substantially noisier for a complex, black-box codec roundtrip, making them less aligned with the descent direction needed for radiance-field optimization.

	PSNR	SSIM	LPIPS	Size	Training
STE (high-rate) [3]	33.56	0.969	0.182	0.27 MB	1h 14m
STE (low-rate) [3]	31.91	0.957	0.267	0.13 MB	1h 9m
mSTE (high-rate) [36]	33.62	0.968	0.195	0.32 MB	1h 25m
mSTE (low-rate) [36]	30.96	0.948	0.368	0.17 MB	1h 28m
STE+SPSA (high-rate) [48]	29.63	0.930	0.459	0.14 MB	3h 18m
STE+SPSA (low-rate) [48]	27.94	0.909	0.650	0.07 MB	3h 18m

Table 6. Ablation of gradient-surrogate methods for SCL training. STE demonstrates superior robustness at low rates, whereas mSTE degrades under heavy quantization. STE+SPSA yields suboptimal R-D performance and incurs substantial training latency due to repeated codec round-trips.

Overall, this ablation shows that reducing bias in the surrogate gradient does not necessarily improve optimization for SCL Tri-Plane training. Instead, vanilla STE offers the best trade-off between stability, RD performance, and efficiency.

F. More Quantitative and Qualitative Results

In this section, we present detailed quantitative and qualitative results that complement the main paper. First, we report per-scene quantitative results for the NeRF Synthetic and Tanks and Temples benchmarks, allowing a more fine-grained comparison across individual scenes (see Tab. 7 and Tab. 8). We then provide extra qualitative visualizations for both static and dynamic benchmarks to further illustrate how CATRF behaves across a wide range of operating points and scene types (see Fig. 10 and Fig. 11).

Method	chair	drums	ficus	hotdog	lego	materials	mic	ship	Avg.
PSNR									
TensoRF [9]	35.89	26.39	34.34	36.76	34.98	30.96	34.68	30.43	33.05
NeRFCodec [32]	34.28	25.77	33.25	35.07	35.28	29.08	34.16	30.63	32.19
CNC [10] ($F = 8$, $\lambda = 4e-3$)	32.99	25.33	32.31	35.66	32.96	29.95	34.78	29.61	31.70
CNC [10] ($F = 8$, $\lambda = 2e-3$)	34.45	25.43	33.27	35.92	34.50	30.06	36.62	29.65	32.49
CNC [10] ($F = 8$, $\lambda = 7e-4$)	35.56	26.18	32.88	37.39	34.64	30.27	36.59	31.83	33.16
HAC++ [12] ($\lambda = 3e-3$)	34.42	26.28	34.46	36.76	34.36	29.99	34.90	30.49	32.71
HAC++ [12] ($\lambda = 2e-3$)	34.89	26.34	34.55	37.00	34.71	30.21	35.54	30.70	32.99
HAC++ [12] ($\lambda = 1e-3$)	35.55	26.49	34.68	37.52	35.10	30.33	35.80	30.70	33.27
CATRF-JPEG (QP=20)	34.74	26.03	33.75	36.15	33.08	30.52	34.55	30.02	32.17
CATRF-JPEG (QP=35)	35.06	26.10	34.18	36.38	33.61	30.78	34.77	30.77	32.49
CATRF-JPEG (QP=65)	35.79	26.24	34.43	36.71	34.59	31.02	35.11	31.22	33.13
SSIM									
TensoRF [9]	0.983	0.931	0.983	0.981	0.978	0.957	0.988	0.888	0.961
NeRFCodec [32]	0.976	0.923	0.978	0.968	0.974	0.950	0.988	0.877	0.954
CNC [10] ($F = 8$, $\lambda = 4e-3$)	0.980	0.941	0.983	0.978	0.978	0.958	0.991	0.901	0.964
CNC [10] ($F = 8$, $\lambda = 2e-3$)	0.982	0.942	0.984	0.980	0.980	0.959	0.992	0.909	0.966
CNC [10] ($F = 8$, $\lambda = 7e-4$)	0.984	0.942	0.984	0.982	0.982	0.960	0.993	0.915	0.968
HAC++ [12] ($\lambda = 3e-3$)	0.980	0.951	0.985	0.981	0.977	0.961	0.989	0.902	0.966
HAC++ [12] ($\lambda = 2e-3$)	0.982	0.951	0.986	0.982	0.978	0.963	0.991	0.903	0.967
HAC++ [12] ($\lambda = 1e-3$)	0.985	0.952	0.986	0.983	0.980	0.964	0.991	0.903	0.968
CATRF-JPEG (QP=20)	0.974	0.925	0.977	0.973	0.958	0.954	0.983	0.874	0.952
CATRF-JPEG (QP=35)	0.978	0.928	0.979	0.975	0.963	0.957	0.985	0.881	0.956
CATRF-JPEG (QP=65)	0.981	0.930	0.982	0.978	0.970	0.970	0.987	0.886	0.961
Size (MB)									
TensoRF [9]	69.19	69.26	72.02	85.24	17.99	86.14	67.96	72.24	67.51
NeRFCodec† [32]	0.525	0.602	0.563	0.521	0.555	0.556	0.553	0.593	0.559
NeRFCodec [32]	1.819	1.896	1.863	1.721	1.849	1.850	1.847	1.887	1.842
CNC [10] ($F = 8$, $\lambda = 4e-3$)†	0.406	0.488	0.365	0.332	0.377	0.485	0.332	0.560	0.418
CNC [10] ($F = 8$, $\lambda = 2e-3$)†	0.511	0.649	0.444	0.367	0.454	0.610	0.366	0.717	0.515
CNC [10] ($F = 8$, $\lambda = 7e-4$)†	0.689	1.003	0.588	0.470	0.602	0.851	0.471	1.106	0.722
CNC [10] ($F = 8$, $\lambda = 4e-3$)	0.652	0.641	0.595	0.518	0.545	0.647	0.524	0.780	0.613
CNC [10] ($F = 8$, $\lambda = 2e-3$)	0.668	0.802	0.622	0.553	0.622	0.772	0.744	0.938	0.715
CNC [10] ($F = 8$, $\lambda = 7e-4$)	0.847	1.130	0.703	0.656	0.770	1.013	1.486	1.327	0.991
HAC++ [12] ($\lambda = 3e-3$)	0.707	1.220	0.766	0.620	0.892	1.027	0.574	1.256	0.883
HAC++ [12] ($\lambda = 2e-3$)	0.830	1.546	0.917	0.721	1.223	1.166	0.654	1.541	1.075
HAC++ [12] ($\lambda = 1e-3$)	1.098	1.787	1.219	0.802	1.434	1.498	0.868	2.150	1.357
CATRF-JPEG (QP=20)	0.332	0.364	0.368	0.362	0.138	0.473	0.335	0.481	0.357
CATRF-JPEG (QP=35)	0.453	0.509	0.501	0.477	0.186	0.636	0.408	0.664	0.479
CATRF-JPEG (QP=65)	0.731	0.840	0.797	0.767	0.370	1.049	0.559	1.001	0.764

Table 7. Per-scene quantitative results on the NeRF Synthetic dataset [37]. When computing the model size, we include all scene-specific codec and entropy-model parameters required to decode the radiance fields at the client. We also report the size of the encoded radiance fields alone, indicated by †. For per-scene results of other baselines, please refer to ECRF [26].

Method	Barn	Caterpillar	Family	Ignatius	Truck	Avg.
PSNR						
TensoRF [9]	29.48	26.84	33.69	28.64	26.88	29.11
NeRFCodec [32]	28.35	25.12	33.39	27.26	25.45	27.91
CNC ($F = 8$, $\lambda = 8e-3$) [10]	28.15	25.85	32.48	27.52	26.38	28.08
CNC ($F = 8$, $\lambda = 4e-3$) [10]	28.56	25.70	32.68	27.20	26.42	28.11
CNC ($F = 8$, $\lambda = 2e-3$) [10]	28.75	26.22	32.72	27.53	26.23	28.29
CNC ($F = 8$, $\lambda = 7e-4$) [10]	28.82	26.44	32.86	28.02	27.12	28.65
CATRF-JPEG (QP=20)	28.69	25.94	31.86	28.37	26.10	28.19
CATRF-JPEG (QP=35)	29.02	26.30	32.21	28.45	26.44	28.48
CATRF-JPEG (QP=50)	29.45	26.86	33.42	28.52	26.60	28.97
CATRF-JPEG (QP=65)	29.67	26.88	33.72	28.48	26.76	29.10
SSIM						
TensoRF [9]	0.901	0.913	0.965	0.949	0.903	0.926
NeRFCodec [32]	0.849	0.891	0.957	0.940	0.863	0.901
CNC ($F = 8$, $\lambda = 8e-3$) [10]	0.866	0.911	0.955	0.941	0.910	0.917
CNC ($F = 8$, $\lambda = 4e-3$) [10]	0.872	0.914	0.959	0.944	0.914	0.921
CNC ($F = 8$, $\lambda = 2e-3$) [10]	0.879	0.917	0.961	0.946	0.917	0.924
CNC ($F = 8$, $\lambda = 7e-4$) [10]	0.884	0.920	0.965	0.947	0.921	0.927
CATRF-JPEG (QP=20)	0.893	0.895	0.968	0.946	0.892	0.919
CATRF-JPEG (QP=35)	0.904	0.905	0.971	0.949	0.900	0.926
CATRF-JPEG (QP=50)	0.909	0.908	0.973	0.951	0.904	0.929
CATRF-JPEG (QP=65)	0.923	0.918	0.976	0.956	0.916	0.938
Size (MB)						
TensoRF [9]	82.06	73.22	68.42	68.18	78.52	74.08
NeRFCodec† [32]	0.574	0.625	0.561	0.552	0.613	0.584
NeRFCodec [32]	1.867	1.925	1.861	1.852	1.913	1.884
CNC ($F = 8$, $\lambda = 8e-3$)† [10]	0.546	0.579	0.384	0.432	0.511	0.490
CNC ($F = 8$, $\lambda = 4e-3$)† [10]	0.726	0.824	0.455	0.559	0.708	0.654
CNC ($F = 8$, $\lambda = 2e-3$)† [10]	0.976	1.067	0.543	0.721	0.992	0.860
CNC ($F = 8$, $\lambda = 7e-4$)† [10]	1.465	1.652	0.710	1.146	1.539	1.302
CNC ($F = 8$, $\lambda = 8e-3$) [10]	0.807	0.799	0.598	0.671	0.726	0.720
CNC ($F = 8$, $\lambda = 4e-3$) [10]	0.987	1.044	0.669	0.798	0.923	0.884
CNC ($F = 8$, $\lambda = 2e-3$) [10]	1.237	1.287	0.757	0.960	1.207	1.090
CNC ($F = 8$, $\lambda = 7e-4$) [10]	1.726	1.872	0.924	1.385	1.754	1.532
CATRF-JPEG (QP=20)	0.368	0.356	0.501	0.296	0.365	0.377
CATRF-JPEG (QP=35)	0.504	0.483	0.671	0.393	0.486	0.507
CATRF-JPEG (QP=50)	0.675	0.626	0.851	0.501	0.634	0.657
CATRF-JPEG (QP=65)	0.816	0.746	1.008	0.599	0.755	0.785

Table 8. Per-scene results on the Tanks and Temples dataset [35]. When computing the model size, we include all scene-specific codec and entropy-model parameters required to decode the radiance fields at the client. We also report the size of the encoded radiance fields alone, indicated by †. For per-scene results of other baselines, please refer to ECRF [26]

Figure 10. More qualitative comparisons of NeRF Synthetic and Tanks and Temples benchmarks.

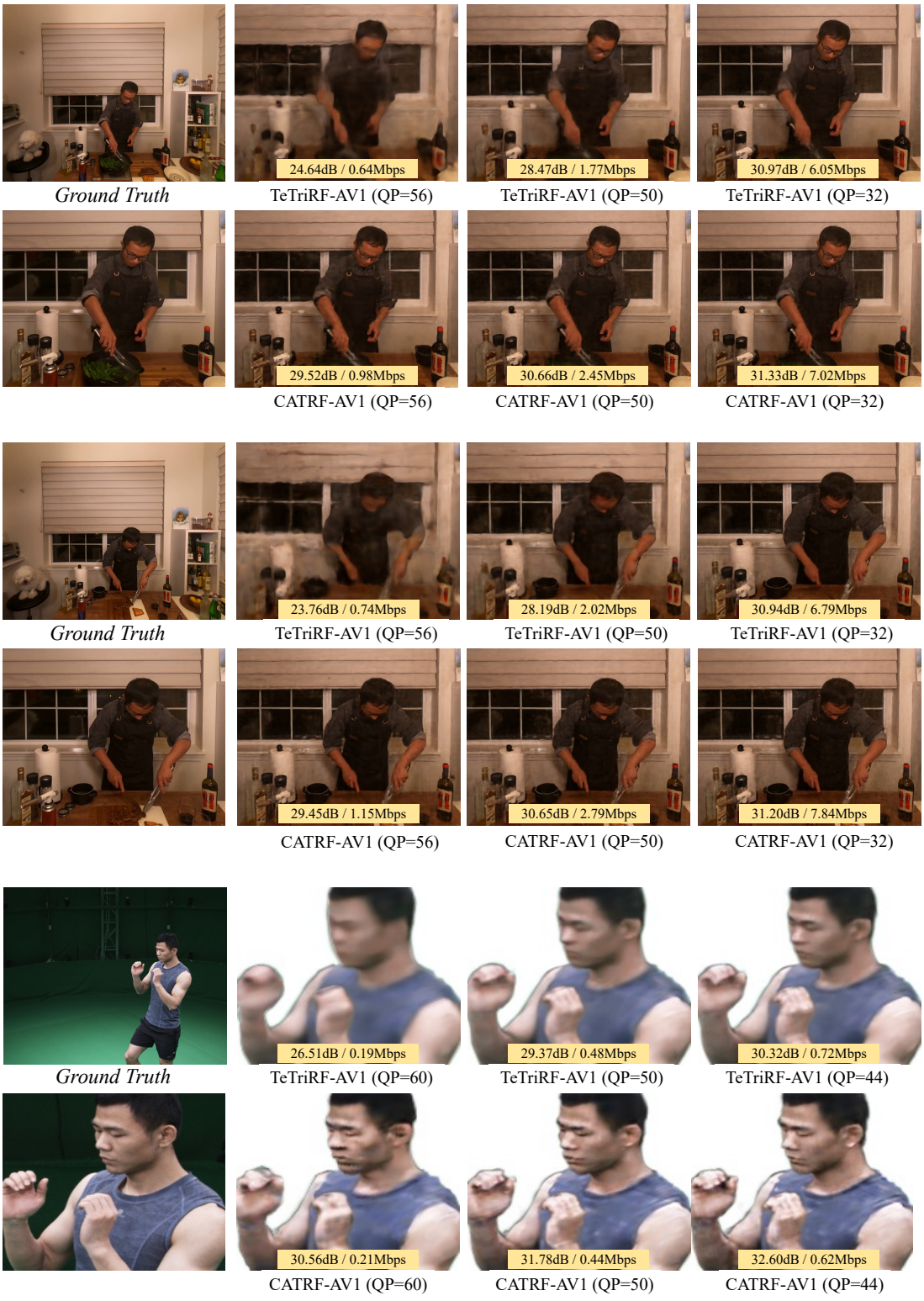

Figure 11. More qualitative comparisons of Neural 3D Video and NHR benchmarks.