Supplementary Material

Ray-LLM: Suppressing Runge's Phenomenon in Data-Sparse Fisheye Calibration via Neuro-Symbolic Synthetic Anchoring

January 20, 2026

Note. This supplementary document provides additional details and visualizations that are omitted from the main manuscript due to space constraints. The contents focus on zone-based diagnostics, geometric candidate metrics, and active pool update behaviors (ADD/SWAP), including memory module terminology and experimental validation on on-site data.

S1. 9-Zone Partition and Zone-wise Error Heatmaps

This section provides implementation details and qualitative visualizations of the 9-zone diagnostic used in Ray-LLM.

S1.1 Zone Partition

Let an image have width W and height H. For each frame n, we compute the checkerboard corner centroid $\mathbf{c}_n = (c_{n,x}, c_{n,y})$ and assign a zone index $z_n \in \{0, \dots, 8\}$ by:

$$i_n = \min\left(2, \max\left(0, \left\lfloor \frac{3c_{n,x}}{W} \right\rfloor\right)\right), \quad j_n = \min\left(2, \max\left(0, \left\lfloor \frac{3c_{n,y}}{H} \right\rfloor\right)\right), \quad z_n = 3j_n + i_n. \tag{S1.1}$$

S1.2 Zone-wise RMS Heatmap

Given corner correspondences $\{(\mathbf{X}_{n,k}, \mathbf{u}_{n,k})\}$, the fisheye projection model $\pi(\cdot; \Theta)$ produces predicted pixels $\hat{\mathbf{u}}_{n,k} = \pi(\mathbf{X}_{n,k}; \Theta)$. The reprojection residual is:

$$\mathbf{e}_{n,k} = \mathbf{u}_{n,k} - \hat{\mathbf{u}}_{n,k}. \tag{S1.2}$$

For each zone z, we collect corner indices:

$$\mathcal{I}_z = \{(n,k) \mid z_n = z\}. \tag{S1.3}$$

The zone-wise RMS is computed as:

$$\epsilon_z = \sqrt{\frac{1}{|\mathcal{I}_z|} \sum_{(n,k)\in\mathcal{I}_z} \|\mathbf{e}_{n,k}\|_2^2}, \quad z \in \{0, \dots, 8\}. \tag{S1.4}$$

The heatmap vector is defined as:

$$\mathbf{H} = [\epsilon_0, \dots, \epsilon_8]^\top \in \mathbb{R}^9. \tag{S1.5}$$

S1.3 Visualization: Early vs. Converged Heatmaps

Fig. 1 compares zone-wise RMS heatmaps at the beginning of optimization and after convergence, highlighting the reduction of peripheral errors.

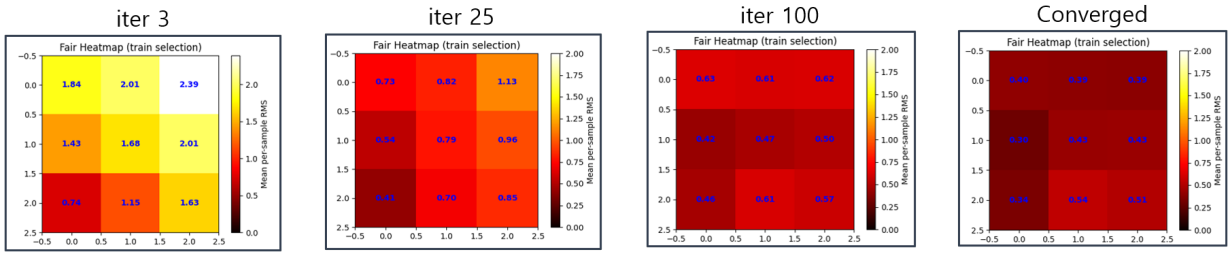

Figure 1: Comparison of zone-wise RMS heatmaps: (left) at the beginning of optimization to (right) after convergence.

S2. Geometric Candidate Metrics (dist / tilt)

This section provides implementation details and visual examples of the candidate metrics used in Ray-LLM.

S2.0 Candidate Construction

For each frame, we detect checkerboard corners using the same detector as the main calibration pipeline. A frame is considered a valid candidate if it satisfies a minimum corner count requirement ($N \geq N_{\min}$). Let $\{\mathbf{u}_k\}_{k=1}^{N}$ be the detected 2D corners in image coordinates, where $\mathbf{u}_k = (x_k, y_k) \in \mathbb{R}^2$.

Corner centroid.

$$\mathbf{c} = \frac{1}{N} \sum_{k=1}^{N} \mathbf{u}_k. \tag{S2.1}$$

Minimum-area rectangle (minAreaRect): We compute the minimum-area bounding rectangle enclosing all detected corners and denote its side lengths by $(w_{\min}, h_{\min})$. We use the aspect ratio as a stable proxy for viewpoint obliqueness.

Candidate pool: All valid frames are stored in a candidate pool $\mathcal{P}$, each associated with $\mathbf{c}$, $(w_{\min}, h_{\min})$, and the derived metrics below.

S2.1 dist: Normalized Peripheral Coverage

We define dist as the normalized distance between the centroid $\mathbf{c} = (c_x, c_y)$ and the image center $\mathbf{o} = (W/2, H/2)$:

$$\mathrm{dist}(\mathbf{c}) = \frac{\|\mathbf{c} - \mathbf{o}\|_2}{\sqrt{(W/2)^2 + (H/2)^2}}. \tag{S2.2}$$

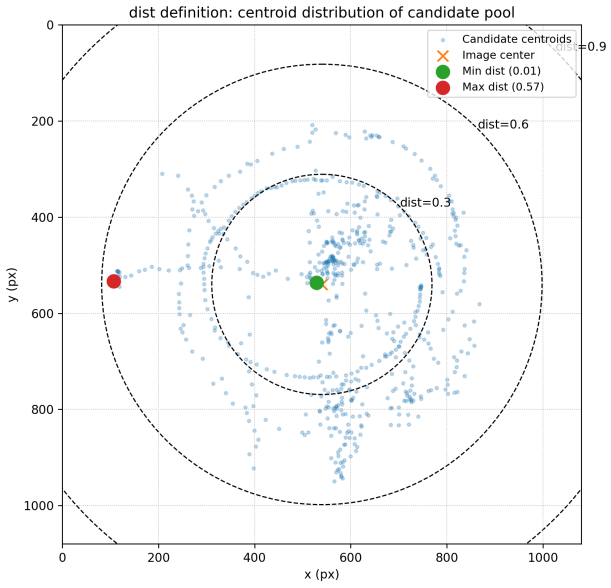

Figure 2: Definition of dist and example distribution of candidates from center to periphery.

S2.2 tilt: Viewpoint Obliqueness Proxy

We define tilt from the aspect ratio of the minimum-area rectangle:

$$\mathrm{tilt} = \frac{\max(w_{\min}, h_{\min})}{\min(w_{\min}, h_{\min}) + \epsilon}, \tag{S2.3}$$

where ϵ is a small constant (e.g., 10^{-6}) to avoid division by zero.

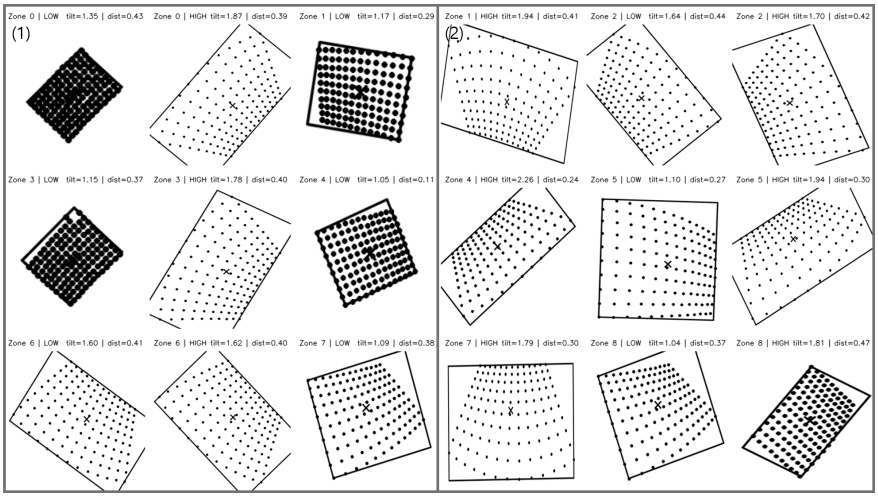

Figure 3: Definition of tilt with qualitative examples of low-tilt (1) vs. high-tilt (2) frames.

S2.3 Candidate Ranking in Implementation

In implementation, candidates can be prioritized by a weighted score:

$$s = \alpha \cdot \mathrm{dist} + (1 - \alpha) \cdot \mathrm{tilt}, \tag{S2.4}$$

where $\alpha \in [0, 1]$ is selected depending on the objective.

S3. ADD vs. SWAP Examples (Before → After)

Purpose. Ray-LLM updates the calibration pool at each iteration using two discrete actions:

- **ADD:** injects physically anchored synthetic observations into under-covered zones.
- **SWAP:** replaces high-residual samples with geometrically superior candidates to stabilize unstable zones.

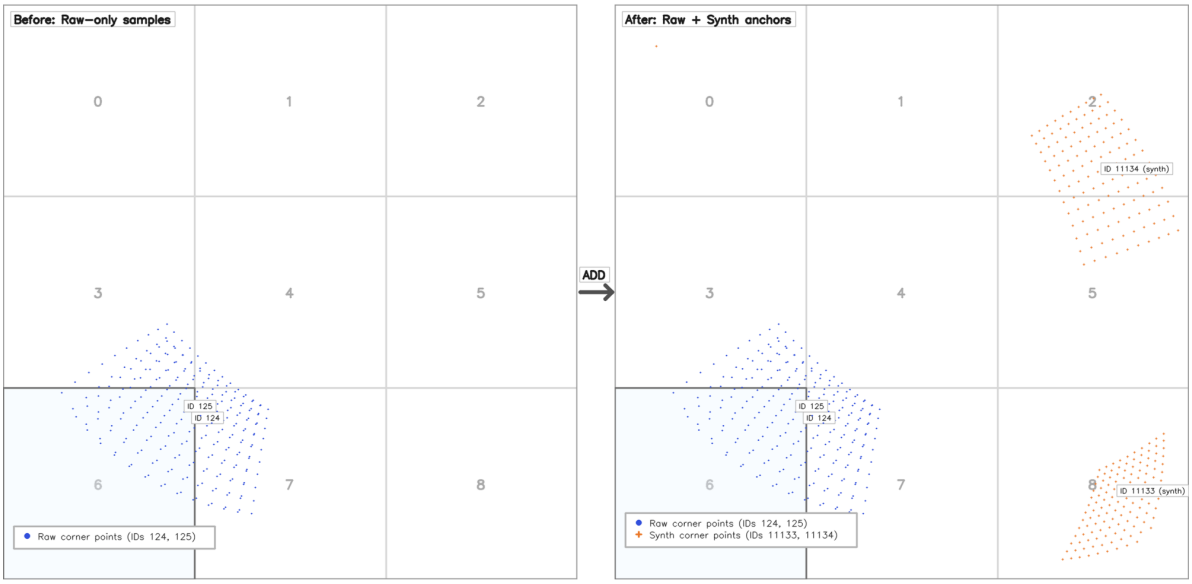

Figure 4: **ADD** example (Before → After): synthetic anchors injected into a sparse peripheral zone.

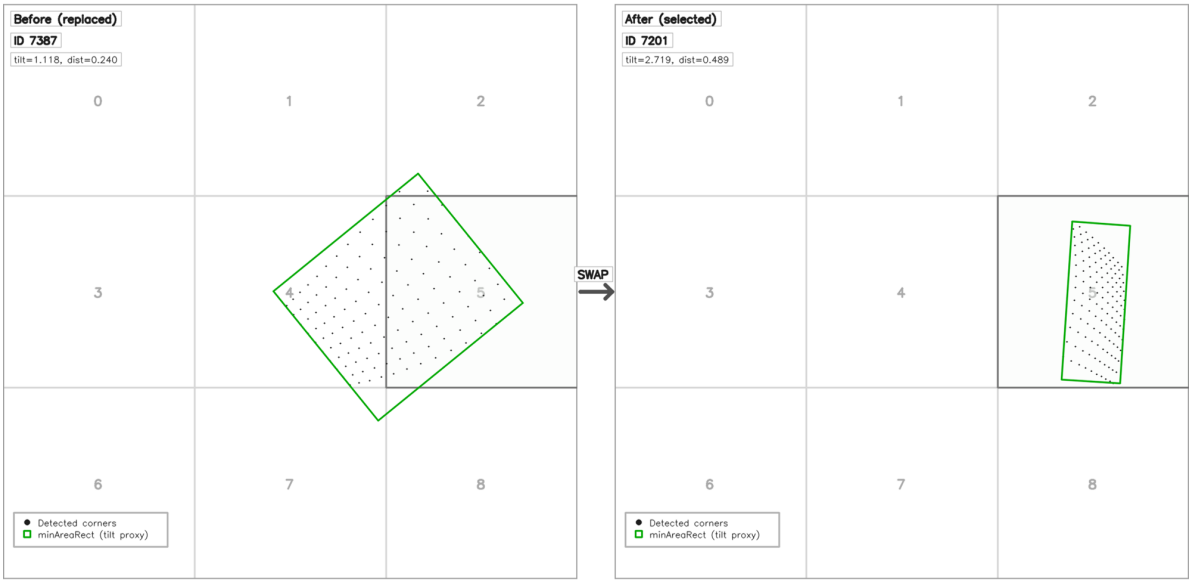

Figure 5: **SWAP** example (Before → After): high-residual samples replaced to stabilize a problematic zone.

S4. Terminology Definitions: Calibration Brain / Experience Buffer

Purpose. To reduce regression and repetitive ineffective actions during long iterative optimization, Ray-LLM maintains two lightweight memory modules: **Calibration Brain** and **Experience Buffer**. This section defines each module and provides real examples from our implementation logs.

S5.1 Calibration Brain (JSON)

Definition. The **Calibration Brain** is a persistent JSON memory that accumulates **zone-wise strategy statistics** (e.g., effectiveness of ADD/SWAP under different criteria). It is indexed by `zone` ($0 \sim 8$) and `criteria` (e.g., `worst_err`, `moreedge`).

Field	Type	Meaning
`zone_scores[z][criteria]`	dict	Per-zone strategy tracker for a selection criterion (one entry per zone in this run).
`score`	float	Accumulated effectiveness score (implementation-defined; may be negative if improvements were not sustained).
`wins` / `tries`	int / int	Number of successful updates vs. total attempts recorded for this zone and criterion.
`context_score`	dict[str $\rightarrow$ float]	Context-dependent effectiveness (keyed by tags such as `OVERFIT|SWAP`, `WEAK_DIST|ADD`).
`history`	list[dict]	Compact history of past trials. Each record contains: `n` (trial index), `dt` (test RMS change), `dz` (zone RMS change), `mode` (ADD/SWAP), `tags`, `imp` (improvement score).

Table 1: Calibration Brain schema (JSON).

Real JSON snippet (from `calibration_brain.json`).


```
"zone_scores": {
      "5": {
          "brain_best": {
              "score": 15.688160987177,
              "wins": 136,
              "tries": 406,
              "context_score": {
                  "MODE_SWAP": 39.6923530945177
              },
              "history": [
                  {
                      "n": 3,"dt": -0.00249,
                      "dz": -0.00347,
                      "mode": "SWAP",
                      "tags": [
                          "MODE_SWAP"
                      ],
                      "imp": -0.17
                  },
                  {
                      "n": 3,
                      "dt": 0.00306,
                      "dz": 0.00788,
                      "mode": "ADD",
                      "tags": [
```

```
            "MODE_ADD"
        ],
        "imp": 0.32
    },
    {
        "n": 5,
        "dt": -0.01105,
        "dz": -0.02264,
        "mode": "SWAP",
        "tags": [
            "MODE_SWAP"
        ],
        "imp": 0.98
    }
```


This JSON structure allows Ray-LLM to reuse zone-specific action priors (e.g., which zones tend to benefit from ADD vs. SWAP under certain instability contexts).

S5.2 Experience Buffer (JSONL)

Definition. The **Experience Buffer** is a JSONL log file where each line stores one step of a **state–action–outcome** trajectory. It is used to retrieve previously effective decisions under similar geometric conditions.

Table 2: Schema definition of `experience_buffer.jsonl` (one JSON record per iteration).

Field	Type	Description
`ts`	float	Unix timestamp of the record.
`round`, `iter`	int	Round and iteration index.
`state`	dict	Telemetry snapshot (RMS errors, intrinsics, distortion coefficients, worst zones).
`action`	dict	Selected action: target zone, mode (ADD/SWAP), and criteria.
`accepted`	bool	Whether the action was applied after masking/validation.
`outcome`	dict	Performance deltas after the action (e.g., Δtest RMS, Δedge RMS).
`rollback_reason`	str/null	Rejection reason if `accepted=false`.

Real JSONL examples (from `experience_buffer.jsonl`).


```
{"ts": 1767.6877432, "round": 1, "iter": 471, "state":
{"ret": 0.7613044020340065, "test_rms": 0.7940791394157624, "edge_rms": 1.1389232834463858,
"fx": 257.24377069797214, "fy": 255.95847005674273,
"k_vals": [0.055724680228209734, 0.13008083344438282, -0.08673881703572073,
    0.01686081964376545], "k_tag": "NOMINAL", "bad_zones": [2, 6, 8], "fov_diag":
    164.68442008901894},
"action": {"zone": 2, "mode": "SWAP", "criteria": "moreedge", "synth_params":
{"source_model": "AUTO",
 "force_corner": true, "tilt_range": [15, 45], "noise_sigma": 0.15, "recipe_tag":
    "ZONE_DEFAULT", "zone": 2, "criteria": "moreedge"}}, "accepted": true, "outcome":
    {"delta_test_loss": 0.0, "delta_test_rms": 0.0, "delta_edge_rms": 0.0}, "rollback_reason":
    null
}
```


By keeping a trajectory-level record, Ray-LLM can avoid repeatedly trying ineffective actions, and instead prioritize previously successful interventions under matching states.

S5. Experimental Validation on High-Resolution On-site Data

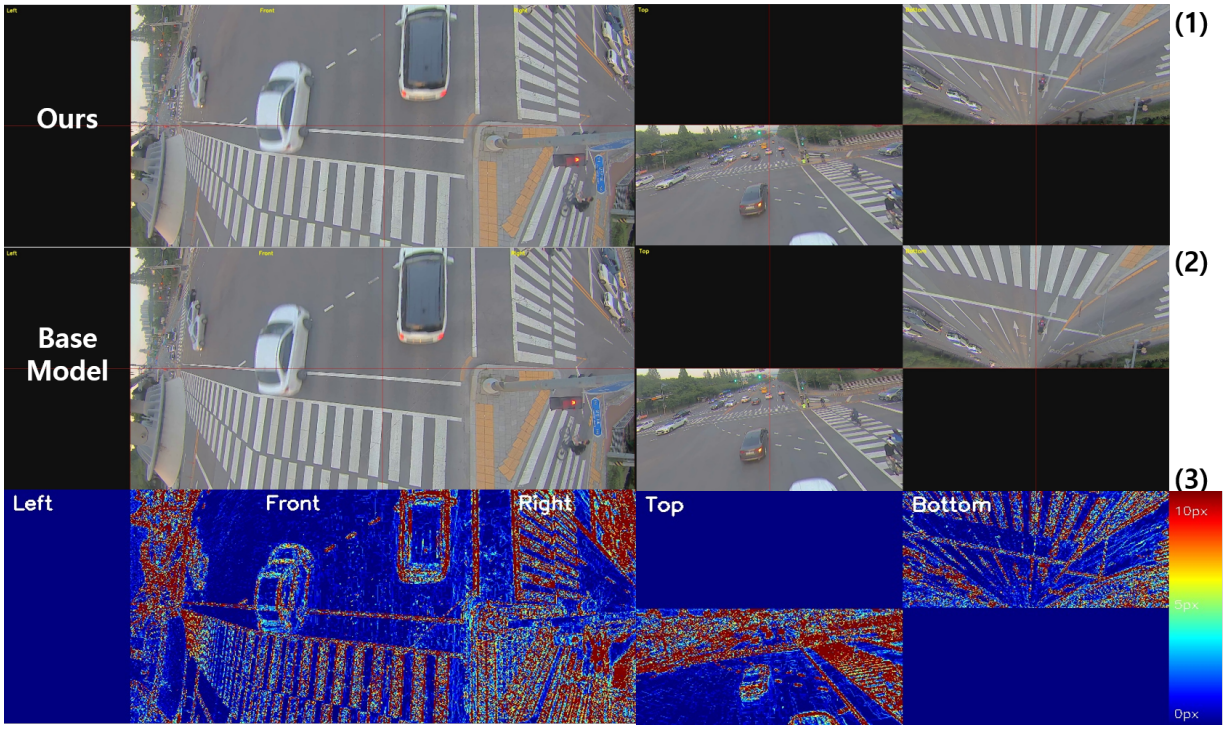

Figure 6: Comparative analysis of on-site extrinsic calibration. (1) Ours: Implementation of LLM-synthesized peripheral data to mitigate sparsity. (2) Base Model: OpenCV-based calibration exhibiting significant overfitting to central data clusters. (3) Error Heatmap: Spatial distribution of pixel-wise registration errors (0–10 px). Notably, while the model was trained at 1080p resolution, this on-site application demonstrates performance at 3008p resolution.