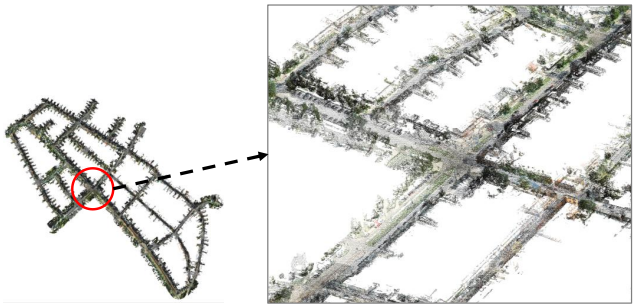
VLM-Loc: Localization in Point Cloud Maps via Vision-Language Models

Supplementary Material



(a)

(b)

Figure 6. Example point clouds from the *CityLoc* benchmark. (a) A roadside LiDAR scene from KITTI-360 [32]. (b) A photogrammetric urban block from CityRefer [42].

A. Overview

In the supplementary material, we provide additional benchmark construction and statistics (Sec. B), implementation details of textual query and system prompt (Sec. C), and further visualization results (Sec. E).

B. The *CityLoc* Benchmark Details

Our *CityLoc* benchmark consists of two subsets, including *CityLoc-K* for training, validation and testing, *CityLoc-C* for cross-domain testing, as shown in Fig. 6. The two splits differ significantly in semantic composition, sensing modality, point cloud characteristics, and geographic region, offering a diverse and challenging setting for evaluating the generalization ability of T2P localization models.

In this section, we provide additional details that complement the description in Sec. 3. We first describe the construction procedures of *CityLoc-K* and *CityLoc-C* in Sec. B.1 and Sec. B.2, respectively. In Sec. B.3 we then present the statistics of the proposed *CityLoc* benchmark.

B.1. CityLoc-K Construction

Data source. *CityLoc-K* is constructed based on the KITTI-360 dataset[1], which contains large-scale LiDAR point clouds collected by vehicle-mounted sensors along urban roads in Karlsruhe. We use 5 training sequences (*00, 02, 04, 06, 07*), 1 validation sequence (*10*), and 3 testing sequences (*03, 05, 09*).

Map center sampling. When constructing the maps, we sample map centers from the vehicle trajectory. Specifically, we perform distance-based subsampling along the trajectory to ensure that any two selected centers are at least 10 m apart. This ensures that sampling remains evenly distributed across the entire trajectory, preventing points from clustering in specific regions and avoiding biased localization outcomes.

Query location sampling. For each sampled map center, we further generate four query positions by randomly perturbing its horizontal coordinates within ± 15 m along both the East and North directions, thereby expanding the number of query samples and positional diversity.

B.2. CityLoc-C Construction

Data source. *CityLoc-C* is derived from SensatUrban [20] and CityRefer [42][2]. For convenience, we denote blocks in Birmingham and Cambridge as *B#* and *C#*, respectively. The dataset contains high-resolution photogrammetric point clouds with nearly three billion points collected across three UK cities, covering an area of 7.6 km^2. We use the validation and test splits from Birmingham and Cambridge, selecting 4 blocks in Birmingham (*B0, B5, B6, B12*) and 7 blocks in Cambridge (*C2, C3, C8, C10, C14, C21, C26*) for cross-city generalization analysis.

Map center sampling. CityRefer does not provide the sensor trajectory of the data acquisition process. Therefore, for map center sampling, we perform grid sampling on each point cloud block and retain maps that contain more than six objects. For query location sampling, we follow the same strategy used for *CityLoc-K*, as described in Sec. B.1.

B.3. Benchmark Statistics

Dataset-Level Statistics. *CityLoc-K* contains 2,767, 300, and 1,027 maps, together with 16,113, 1,772, and 6,109 queries for training, validation, and testing, respectively. These maps span areas of 1.66, 0.19, and 0.60 km^2.

[1] Available under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 license.

[2] Both datasets are released under the MIT License.

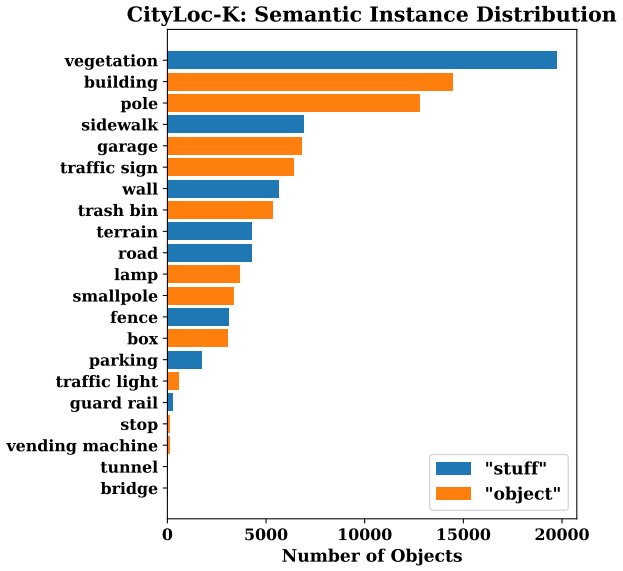


Figure 7. Semantic instance distribution in *CityLoc-K*.

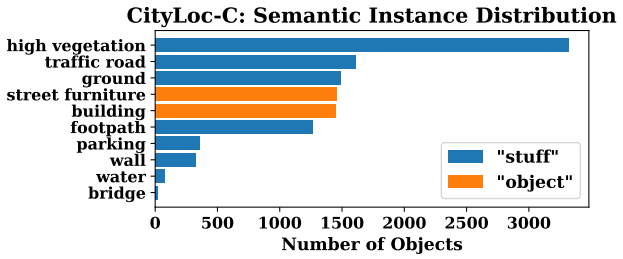


Figure 8. Semantic instance distribution in *CityLoc-C*.

CityLoc-C includes 875 maps and 4,487 queries, covering 0.90 km^2 for cross-domain testing. The reported map area is computed as the union of the minimum bounding boxes of all maps.

Distribution of semantic instances. For all semantic instances contained in the point cloud maps, we compute their distribution, as shown in Fig. 7 and Fig. 8. Here, "object" and "stuff" follow the definitions in KITTI-360 [32], referring to countable and uncountable instances, respectively, and are rendered in orange and blue.

C. Implementation Details

C.1. Textual Query Generation

Components. Each textual query $\mathcal{T}$ contains N_t hints. Each hint h describes an object by specifying its color, semantic category, and its directional relation with respect to the query location, following the template format: `"The pose is <direction> of <color> <semantic>."`

Direction computation. For each object o referenced in the textual description, we first project all its 3D points onto the horizontal plane and identify the closest point $\mathbf{p}_{\text{close}} \in \mathbb{R}^2$ to the query position ξ. If the distance between $\mathbf{p}_{\text{close}}$ and ξ is below $\delta = 2.5$ m, the direction is assigned as "on-top". Otherwise, we determine the relative orientation by comparing the horizontal offsets dx and dy between ξ and the centroid of o, as summarized in Eq. (3).

$$\text{Direction}(\xi, o) = \begin{cases} \text{“on-top”}, & \|(\xi - \mathbf{p}_{\text{close}})\|_2 < \delta, \\ \text{“east”}, & |dx| \geq |dy| \text{ and } dx \geq 0, \\ \text{“west”}, & |dx| \geq |dy| \text{ and } dx < 0, \\ \text{“north”}, & |dx| < |dy| \text{ and } dy \geq 0, \\ \text{“south”}, & |dx| < |dy| \text{ and } dy < 0, \end{cases} \tag{3}$$

Color mapping. Each object in the map is already associated with an RGB color $\bar{\mathbf{c}}$ (as described in Eq. (1) in the main paper). To obtain its textual color attribute, we map the object's RGB value to the nearest entry in a predefined discrete color palette $\texttt{COLOR\_NAMES} = \{\text{dark-green, gray, gray-green, bright-gray, black, green, beige}\}$ as in KITTI360Pose [25], using the corresponding template RGB centers $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$ for nearest-neighbor assignment. The assigned textual label is obtained by

$$\text{color}(o) = \texttt{COLOR\_NAMES}\left[\arg\min_k \|\bar{\mathbf{c}} - \mathbf{c}_k\|\right]. \tag{4}$$

where $k \in \{1, \ldots, K\}$.

Constructing large-scale, free-form textual descriptions from point cloud data is highly challenging and costly. Due to the complexity of real-world point cloud scenes, manually extracting scene information to compose detailed descriptions is prohibitively expensive. Although some recent methods [13, 61] attempt to generate text by first extracting keywords and then prompting a large language model (LLM) (*e.g.*, GPT-4 [1]). This process is generally not free and still requires manual verification, making it unsuitable for scalable dataset construction.

For these reasons, we follow prior work [25] and adopt a rule-based template to encode the essential object-level cues for localization, including color, semantic category, and relative direction. This approach allows us to construct informative and scalable textual descriptions at minimal cost. It is fully free to generate, requires no LLMs, and avoids any manual post-verification, while still preserving the key information necessary for fine-grained localization.

C.2. System Prompt

Our system prompt consists of six components designed to guide the VLM in performing the T2P localization task, as detailed in Tab. 7. The prompt could be formulated as :

Table 7. System prompt for VLM-Loc.

You are a helpful assistant for spatial reasoning and grounding in BEV (bird's-eye-view) images. You are also given a scene graph describing the environment as: {node_id, label, pixel_center}, where: - label = semantic label (e.g., "road", "vegetation", "building", "pole", etc.) - pixel_center = pixel coordinates [x, y] (origin at top-left, x→right, y→down).

COORDINATE & DIRECTION RULES: In BEV pixel coordinates: Up = North = y decreases; Down = South = y increases; Left = West = x decreases; Right = East = x increases.

GOAL: (1) Parse the user's natural-language description of a target location. (2) Extract all mentioned object phrases (e.g., "terrain", "parking") in order of appearance. (3) For each object phrase: Find the matching node in the given scene graph, determine whether grounding is successful (boolean), and record the matched node's ID if grounded; otherwise set it to None. (4) Finally, infer the 2D pixel coordinate of the **target location** based on the described spatial relations (north/south/east/west/on-top).

RULES: All reasoning and distance computation MUST be in pixel coordinates. Length of assignments MUST equal the number of mentioned objects in the user's text. Each matched node_id must exist in the scene graph; if not found or out of context, set "grounded": false and "matched_node": None. Output exactly ONE JSON object in the schema below **— no explanations, no extra text**.

OUTPUT FORMAT (STRICT): { "assignments": ["object_label": <string>, "grounded": <bool>, "matched_node": <int|None>, ...], "point_2d": [<int>, <int>] }

EXAMPLE (ILLUSTRATION ONLY): OUTPUT: { "assignments": ["object_label": "parking", "grounded": true, "matched_node": 0, "object_label": "terrain", "grounded": true, "matched_node": 8, "object_label": "road", "grounded": true, "matched_node": 4, "object_label": "vegetation", "grounded": true, "matched_node": 11], "point_2d": [45, 135] }

- **Role:** specifies the localization task for the VLM and clearly defines the expected input structure, including the types of information provided, their format, and the order in which they are supplied.
- **Coordinate system**: provides the correspondence between the pixel coordinate system and the geospatial coordinate system, which is essential for enabling the model to interpret directional cues in the text and map them to the correct orientation on the map.
- **Goal:** guides the model in estimating the current pose step-by-step. The procedure is: 1) comprehend the input text description; 2) extract the textual cues by sequentially processing each mentioned object; 3) assess the visibility of described elements and form text-node matching pairs for valid entries; 4) estimate the target location based on the aggregated geographic information.
- **Rules:** specifies the constraints for the model's output, *e.g.*, that distances should be measured in pixel coordinates. These rules ensure that the pose estimation adheres to a parsable format and avoids invalid results.
- **Output format:** defines the required output structure, mandating that the model presents its results for node assignment and pose estimation in a predefined format.
- **Example:** provides a concrete example that illustrates the predefined output format for the model to follow. Specifically, the output should first present the visibility and assignment status for each textual hint, followed by the estimated 2D pixel coordinates.

D. Additional Experiments

D.1. Results on KITTI360Pose

To enable a fair comparison on KITTI360Pose [25], we follow its retrieval-then-localization protocol and adopt the same Top-1 retrieval results from CMMLoc [58] for localization. We retrain VLM-Loc for localization on KITTI360Pose and evaluate it on the test set. Other methods perform localization using their pretrained models (Text2Pos [25] is not available). Results are reported in Tab. 8. Under same retrieval protocol, VLM-Loc achieves competitive localization performance compared to CMMLoc.

D.2. Inference Analysis

Inference analysis is conducted on two RTX 4090 GPUs with the batch size of 1, and results are reported in Tab. 9.

Method	R@5	R@10	R@15	Method	R@5	R@10	R@15
Text2Pos	/	/	/	MNCL	33.71	50.33	54.69
Text2Loc	37.27	51.32	54.79	CMMLoc	37.81	**51.84**	**55.02**
VLM-Loc	**40.36**	51.69	54.74				

Table 8. Localization on KITTI360Pose test set (11404 samples).

Backbone	FPS	Peak Mem. (GB)	Params (B)
Qwen-VL-2B-Instruct	0.36	8.50	2.14
Qwen-VL-8B-Instruct	0.23	33.65	8.79

Table 9. Inference analysis of VLM-Loc on the *CityLoc-K* val set.

The latency is acceptable for the T2P localization setting. Moreover, the inference cost can be significantly reduced via quantization, smaller backbones, and optimized deployment frameworks, making VLM-Loc more practical for real-world systems.

E. Qualitative Results

Additional qualitative results of our approach and baselines on the *CityLoc-K* and *CityLoc-C* splits are shown in Fig. 9 and Fig. 10. The results demonstrate the superior performance of the proposed method over baselines across diverse scenes, validating its robustness and accuracy.

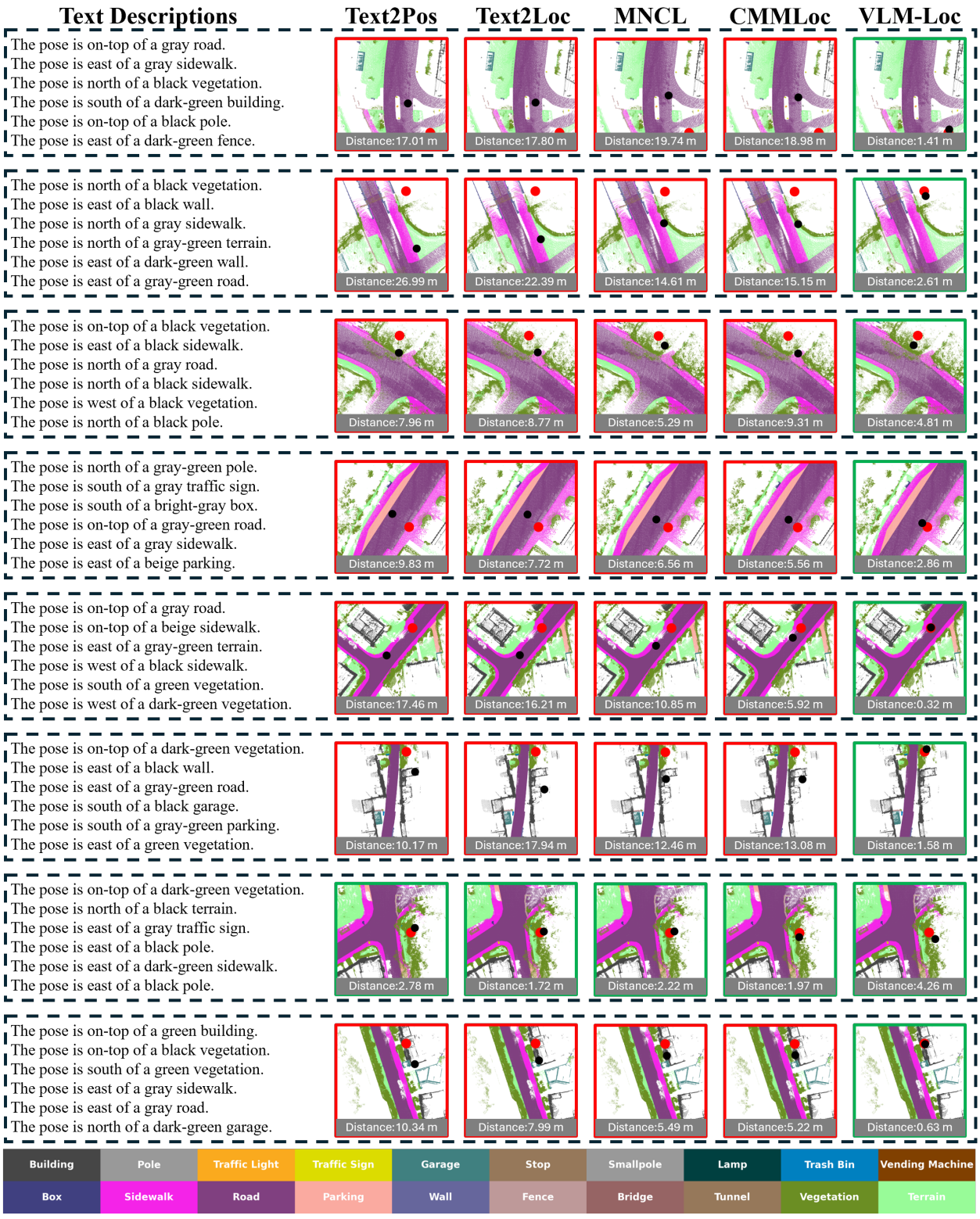


Figure 9. Qualitative results of VLM-Loc and baseline methods on the *CityLoc-K*. Each example visualizes the predicted and GT positions on colorized BEV maps rendered with semantic labels. The red circles ● and black circles ● denote the GT and predicted positions, respectively. The localization error is shown below each image, and green/red borders indicate localization error below/above 5 m.

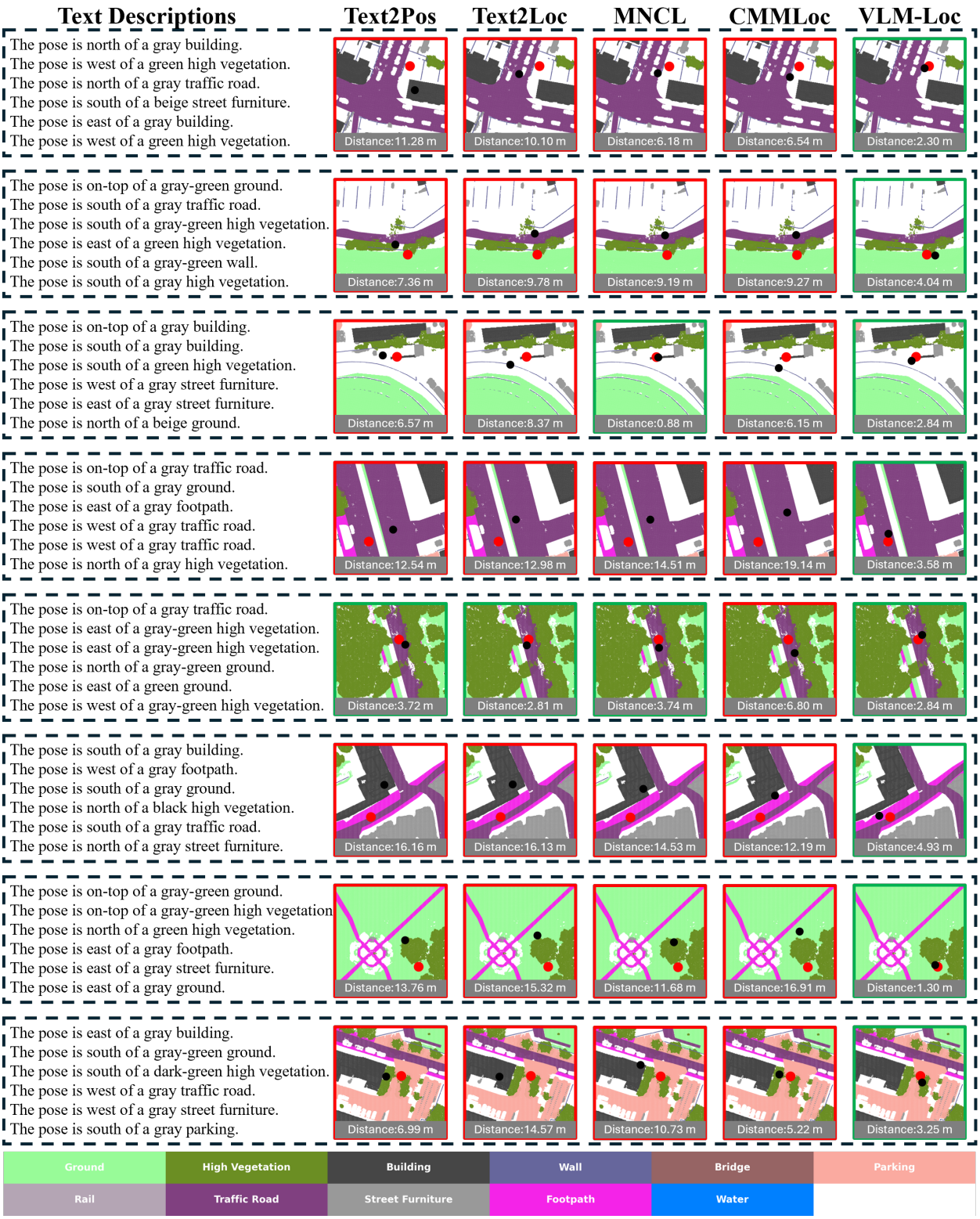


Figure 10. Qualitative results of VLM-Loc and baseline methods on the *CityLoc-C*. Each example visualizes the predicted and GT positions on colorized BEV maps rendered with semantic labels. The red circles ● and black circles ● denote the GT and predicted positions, respectively. The localization error is shown below each image, and green/red borders indicate localization error below/above 5 m.