

Peer Review

Review of: "LiLMaps: Learnable Implicit Language Maps"

Evgenii Arsentev¹

1. Independent researcher

1. What the work claims

The paper proposes a way to build a 3D language map while a robot is still exploring, inside a learned implicit map, so that language models can query it later (Abstract, Section 1). The map is a sparse octree with low-dimensional learnable features at voxel corners on three levels, plus one high-dimensional vector F per voxel of the coarse level (Section 3.1, Figure 2). A decoder of three fully connected layers turns these into a CLIP-space language feature, and training uses a cosine-similarity loss, Eq. (1).

Two contributions sit on top of this. The first is Adaptive Language Decoder Optimization (Section 3.2, Algorithm 1). When a frame contains language features not seen before, found by cosine similarity with a threshold τ , only the decoder and the new parameters are optimized, and stored known features are replayed so the old ones are not forgotten. The second is the Measurement Update Strategy (Section 3.3, Eq. (4)-(5)). The training target is an exponential smoothing of the new observation and the feature already stored in the map, meant to reduce the view-to-view inconsistency of vision-language features. For evaluation (Section 4), the authors use five Matterport3D sequences in the Habitat simulator, with inputs from ground truth labels (GT), simulator semantic images (SEM), or the LSeg encoder. They compare with OpenScene 3D and with VLMaps numbers (Table 2), and with pre-trained OpenScene heads as decoders (Table 3). The conclusion (Section 5) states that LiLMaps outperforms similar works in language mapping quality and is currently the only approach capable of large-scale incremental implicit language mapping.

2. Whether the conclusion holds on the data

I found it valuable that the ablation with and without the measurement update is reported for every input type and every sequence (Table 2, rows LiLMaps and LiLMaps*), and that Figure 3 shows the effect

visually. With simulator semantic input (SEM), the update helps on all five sequences in Table 2. For example, mIoU on 5LpN3gDmAk7_1 rises from 57 (LiLMaps* SEM) to 66 (LiLMaps SEM), and on JmbYfDe2QKZ_2 from 63 to 74. Forgetting is also tested directly in Figure 5 and Table 3: the decoder grown online (first LiLMaps row of Table 3) has 20 classes with F1 above 90%, the same count as the decoder pre-trained on all scene labels (second LiLMaps row, also 20). I appreciate that the authors are open that GT and SEM inputs are an upper bound with close-to-ideal data (Section 4.1, text under Table 2).

My first remark concerns the conclusion "we outperform similar works in terms of language mapping quality" (Section 5). It is clear for GT and SEM inputs, but those inputs come from labels, not from a vision-language encoder. With a real encoder (LSeg rows of Table 2, LiLMaps with the update against OpenScene), accuracy is higher on one sequence (70 vs 61 on gTV8FGcVJC9_1), equal on two (68 vs 68 on 5LpN3gDmAk7_1, 56 vs 56 on JmbYfDe2QKZ_2), and lower on two (57 vs 63 on YmJkqBEsHnH_1, 73 vs 77 on jh4fc5c5qoQ_1). Mean recall is lower on three of five sequences (37 vs 45, 39 vs 52, 42 vs 51). So with real encoder input, the result is a tie with OpenScene, not a win, and I would suggest that the conclusion says so.

One more thing about Table 2 is the measurement update with GT input. Without the update (LiLMaps* GT), mIoU is slightly higher than with it (LiLMaps GT) on three sequences: 93 vs 91 on 5LpN3gDmAk7_1, 94 vs 92 on gTV8FGcVJC9_1, and 89 vs 87 on JmbYfDe2QKZ_2. The authors say the update "does not change the performance significantly" for precise input (Section 4.1). A sentence on why it costs 2 points on clean data would help the reader.

The VLMaps row of Table 2 repeats the same numbers (accuracy 28, mIoU 19) in all five sequences. As the last paragraph of Section 4.1 explains, these are mean values reported in the VLMaps paper for 2D maps, so they are not measured on the same sequences or in 3D. I would mark this row as not directly comparable, or rerun VLMaps on the five sequences.

All results are reported for five simulated Matterport3D sequences, without repeated runs or spread (Section 4). It would help to add repeated runs with the spread and, if possible, some real-robot data.

Finally, I would soften the claim in Section 5 that "currently, LiLMaps is the only approach capable of large-scale incremental implicit language mapping." The evaluation covers five indoor sequences, and Section 2 describes ConceptFusion as fusing language features into 3D maps with SLAM. A more measured wording would match the evidence better.

3. Reproducibility

Much of what is needed to repeat the experiments is given. Table 1 lists the main hyperparameters (similarity threshold τ 0.02, octree levels 8, 9, 10, fine resolution 0.05 m, learnable feature size 16, F vector size 512, 100 iterations per decoder optimization, 100 iterations per mapping loss). The data are depth, semantic, and RGB images provided by the VLMaps authors through Habitat, Matterport3D (Section 4), and the five sequence IDs are in the Table 2 header.

I searched the text for "github," "code," "GPU," and "seed" and found no statement on code release, hardware, or random seeds. The learning rate and optimizer are also missing, as is the time per frame of the mapping itself. The only timing is 4 frames per second for decoder optimization (end of Section 4.2). I would add these details.

The paper reports "only slight overhead" and "a small overhead" (Sections 1 and 5), but no table compares time or memory with and without LiLMaps. A short table of time per frame and memory would support this. It would also help to say whether τ 0.02 (Table 1) is applied to cosine similarity or to 1 minus cosine similarity, since with cosine similarity it would treat almost any two features as similar.

4. What specifically to fix

1. Eq. (5) and the sentence before it disagree. The text says α is set to higher values when the new measurements are more different from the previously optimized map features. But Eq. (5), $\alpha = \text{CosineSimilarity} / (0.5 + \text{CosineSimilarity})$, grows when the features are more similar: cosine 1 gives $1/1.5$, about 0.67, while cosine 0.2 gives $0.2/0.7$, about 0.29. Since α weights the stored map feature in Eq. (4), I would fix either the sentence or the formula.
2. Eq. (5) also has no guard for negative cosine similarity: at -0.5 the denominator is zero. And with the stored feature initialized to 0 in Eq. (4), cosine similarity with a zero vector is undefined on the first observation. A short note on how the code handles both cases would help.
3. In Table 3, the row labels lost their subscripts on Qeios, so the two LiLMaps rows and the three OpenScene rows cannot be told apart. The text names four OpenScene variants (OpenSceneMIT, OpenSceneSC, OpenSceneMS, OpenSceneHE, Section 4.2), while the table has three OpenScene rows. It would help to label the rows with full names and to give the training dataset of each head.
4. Section 4.2 says "we do not update pre-trained models (LiLMapssimple, OpenSceneMIT, OpenSceneSC, OpenSceneMS)." LiLMapssimple is the adaptively trained decoder, so I think

LiLMapspretrained was meant here.

5. Section 5 refers to "3D language-based object detection (7)." I would write Figure 7.

6. Reference 1 (CLIP) gives the venue as "ICLM." It should be ICML.

5. Recommendation

Minor revisions. The method is clear, the ablation of the measurement update and the forgetting test (Table 3, Figure 5) are useful, and the work fits current research on language maps for robots. Before acceptance, I would align the conclusion with the LSeg rows of Table 2, which show a tie with OpenScene on real encoder input. It would help to fix the text or the formula of Eq. (5), label the rows of Table 3 fully, and add a code statement and the timing of mapping. A test on real robot data would make the paper much stronger, but I don't make it a condition.

Declarations

Potential competing interests: No potential competing interests to declare.