Language embedded 3D Gaussians with vector database accelerated queries

Laszlo Szilagyi
laszlosz@stanford.edu

Abstract

This project's goal is to implement semantic environment mapping with open vocabulary natural language queries. Beyond visualizing results the queries should return coordinates of object locations, should be fast and should be easy to integrate with downstream tasks such as robotics and augmented reality. The implementation should use a scene representation that has the potential to handle dynamic scenes and can enable queries to return parts of the scene as 3D assets. Related work leverages 2D image language embedding models in combination with 3D novel view synthesis techniques to achieve this goal. A prime example is LERF [3] which combines neural radiance fields [5] with CLIP [7] language embeddings. As demonstrated by LangSplat [6] and LEGaussians [8] using Gaussian splatting as a scene representation in tandem with CLIP embeddings can greatly improve performance.

This work leverages a Gaussian splatting-CLIP stack, however, instead of adding the language embeddings directly to the scene model it employs a vector database to separate the concerns of scene representation and the processing of embeddings to improve scalability. Another difference is using weighted averaging for determining the final per-Gaussian embeddings instead of optimization, which can simplify implementation. The initial results are promising, a follow-up work could further analyze embedding processing alternatives, refine masking and verify vector database scalability in large scenes.

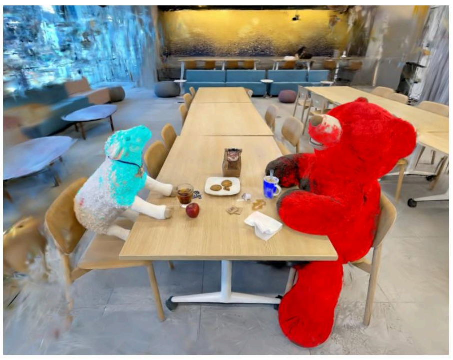
Figure 1. Queries in the "teatime" scene.

1. Introduction

The primary goal is to have the ability to ask for the precise coordinates of objects in a mapped environment using natural language queries. Beside the visualization the goal is to get the locations of query matching objects, even beyond line of sight. The representation should also have the future capability to returns subsets of the scene, e.g. the model of the matching object for further processing (physical dimension for robotics, graphical overlay for AR). Dynamic scene handling is also a potential future preference.

The combination of 3D novel view synthesis with 2D language embeddings is a rapidly developing field. LERF demonstrated how neural radiance field can be extended with CLIP image embeddings, LangSplat and LEGaussians achieved significant performance with the adoption of Gaussiaqn splatting [2] for the same purpose. Beside rendering performance Gaussian splatting also has the favorable property of being an exdplicit scene model which is directly editable too (vs. NeRF). Therefore for this work Gaussian splatting has been selected as the 3D scene representation model. As the CLIP image embeddings are representing the image as a whole additional processing is needed to achieve object specificity. Previous works leveraged cropped images withe CLIP, DINO [1], (hierarchical) SAM [4] to achieve dense (per-pixel) language embedding. This work leverages a stack wimilar to LangSplat, which combines SAM with CLIP for this purpose.

To implement a performant and scalable system this work evaluates the use of a vector database to store and process the language embeddings of the Gaussians. To identify the embeddings to be loaded into the vector databes as keys for each Gaussians in the scene a reverse rendering - segment anything model (SAM) masking - CLIP embedding - per-Gaussian weighted averaging pipeline needs to be implemented and assessed.

2. Related work

This work has been inspired by related work attempting to combine 2D image embedding models with 3D novel view synthesis techniques to achieve dense 3D language embedded scene representations. These include LERF, LangSplat and LEGasussians.

2.1. LERF

LERF [3] combines neural radiance fields (NeRF) with CLIP embeddings. It creates a language embedding field through associating CLIP embeddings with 3D volumes in the scene with multiple scales to capture objects with varying sizes. The volume embeddings are learned from calibrated input images with CLIP embedded. CLIP embeddings are note dense, to better localize the embeddings CLIP is employed on image crops with different scales. To improve embedding quality a DINO feature field is also added for regularization leveraging its object decomposition properties. In rendering time rendering the RGB and depth channels is unchanged from NeRF. The language embeddings of each pixel on the 2D image however are determined by integrating the language embeddings of the 3D volumes along the sampling rays. LERF is a widely referenced work in the domain, however the compute requirements of NeRF introduces a performance bottleneck.

2.2. LangSplat

LangSplat [6] addresses the inherent performance limitations of LERF caused by NeRF with using Gussian splatting as the 3D scene representation technique. It also uses CLIP, however, instead of image cropping LangSplat employs object masking with the segment anything model (SAM) to localize the language embeddings. To capture objects (and their parts) with varying sizes 3 different semantic levels of masks are generated (whole, subpart, part). LangSplat augments each Gaussians with language embeddings and uses the object masked language embedded images to learn the Gaussian embeddings in an object specific manner. Adding the 512-dimension CLIP embeddings for every Gaussians can be very expensive on memory, to address this LangSplat trains a scene-specific autoencoder to reduce the size of the embeddings.

2.3. LEGaussians

LEGaussians [8] is another recent work leveraging the combination of Gaussian Splatting and CLIP embeddings. It concatenates the dense CLIP and DINO features extracted from multi-view images to form hybrid language feature maps. This work also concludes that associating these concatenated features as-is to each Gaussians is too expensive considering the memory footprint and proposes a quantization scheme to reduce the size of features. The authors developed a discrete language feature space and use integer index to retrieve the nearest language feature. The authors then expand the channels of the Gaussians to add the index. To address semantic ambiguity arising from visual disparities across various viewpoints, a mechanism is introduced to reduce the spatial frequency of language embeddings through an adaptive learning loss based on the learned uncertainty values on each point.

2.4. Differences to related work

The proposed approach in this work is closest to the LangSplat stack, applying a combination of Gaussian splatting and SAM masked CLIP embeddings, with the following differences:

- Instead of adding the language embeddings directly to the scene model it employs a vector database to separate the concerns of scene representation and the storage and query of embeddings. Vector databases are optimized for storing and processing embedding vectors and can greatly improve the scalability of the system. As the embeddings are no more directly tied to the scene representation the number and size of the embeddings are no more limited by the GPU memory. This can enable much larger scenes to be embedded without the compression (and quality degradation) of the CLIP embeddings.
- The language query results are the list of matching Gaussians across the whole scene vs. on the current image plane. This can be useful for downstream tasks needing to query location of objects beyond line of sight (with clustering the results). Providing the 3D representation of matching objects to clients is also possible via sharing the subset of Gaussians indexed by the query results. This can be useful for AR clients creating overlays over real world objects or robots needing more details of specific object's dimensions without storing the model of the whole scene on the device.
- The implementation uses weighted averaging for determining the per-Gaussian embeddings instead of sophisticated optimization. As shown later the results are promising with a greatly simplified implementation, with the potential of quicker processing times. However, the performance of this approach needs further analysis.
- Masking is simplified with no hierarchy. Results suggests that this harms performance, more sophisticated masking can be implemented in a follow-up work.

3. Technical Approach

As described on Figure 4 the semantic mapping system consists of the following building blocks: 1. 3D reconstruction, 2. Reverse rendering, 3. Gaussian embedding, 4. Database population, 5. Query processing and 6. Visualization. Step 1-4 corresponds to the scene mapping and embedding stage which is more processing heavy and less time critical, while step 4-5 is about retrieving semantic information from the mapped scene. This part should to be real-time.

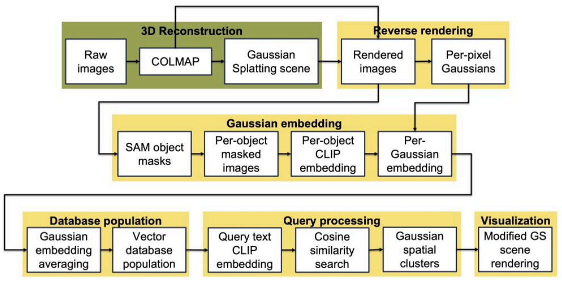

Figure 2. The system architecture

In the following paragraphs, detailed descriptions will be given of the implementation details of each of them.

3.1. 3D reconstruction

For the 3D reconstruction, the Nerfstudio project [9] is being leveraged, which has its own Gaussian Splatting implementation, the gsplat library [10]. To reconstruct a scene, we typically need couple of hundreds of well-selected (overlapping) images of the area. For comparability with other projects the LERF ”teatime” dataset was used, and also added the LERF ”kitchen” dataset to asses performance across different scenes. The raw images need to be pre-processed with the COLMAP algorithm to estimate their positions before being fed into the Gaussian Splatting algorithm. All this functionality is part of the Nerfstudio splatfacto Gaussian splatting pipeline, so no significant work was necessary to build the Gaussian splatting scenes.

3.2. Reverse rendering and Gaussian embedding

Ultimately, we want to associate language embeddings to the 3D Gaussians so we can perform search for relevant Gaussians with cosine similarity using CLIP encoded text queries. Recent literature is leveraging OpenAI’s CLIP model to generate language embeddings, which includes both an image and a text encoder producing 512 dimension embedding vectors. However, the image embedding is sparse, works only for whole images, not for specific objects on the image. To break down an image into embeddings for every object, Meta AI’s Segment Anything Model (SAM) was used, then CLIP embeddings has been generated for each resulting segments masked with a white background as seen on Figure 8.

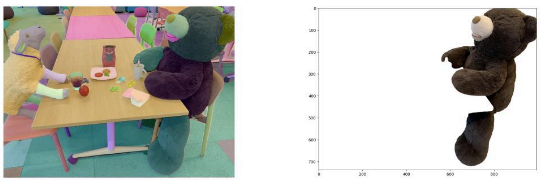

Figure 3. SAM segmentation in the ”teatime” scene (left) and a masked object (right).

As mentioned, hierarchical masking with SAM was not implemented in this work due to time limitations. A more sophisticated masking implementation is due in a follow-up work, potentially considering Grounded SAM and DINO too.

With such a segmented embedding we are still in the 2D image space, we need to bring embeddings into the 3D scene represented by the Gaussians. For this with a ”reverse rendering” method we need to identify which Gaussians contribute with what weights of each pixel on a rendered image in the scene then associate the previously described segmented embeddings of the rendered image with those Gaussians. To implement the reverse rendering step the Nerfstudio gsplat projection and rasterizer pipeline code had to be modified channelling out the per-pixel Gaussian indexes and weights of the rendered images. Then the Gaussian indexes of the non-zero pixels of the specific rendered image could be associated with the masked embedding with the corresponding Gaussian weights.

A representative number of rendered images with proper view points needs to be processed this way. To achieve this I’ve reused the COLMAP positions generated in the scene 3D reconstruction step, rendered the images from the same positions tapping out the per-pixel Gaussian weights information. I’ve used a PyTorch implementation of the rendering pipeline originally used by the authors of the Nerfstudio gsplat library for debugging the CUDA implementation, which reduced speed (still sub-second per frame), however this stage is not time critical and limited to several hundreds of image renders per scene.

3.3. Database population: Gaussian embedding averaging

The per-Gaussian embeddings associations in the previous step associates multiple (typically hundreds) of different embeddings for each Gaussians and cosidering a non-hierarchical embedding a single embedding has to be calculated for each Gaussians. As discussed, in related works optimization is used to minimize the embedding loss across multiple viewing angles’ ground truth embedding of the masked images. In this implementation the final per-Gaussian embeddings are determined by weighted averaging. As described in the previous step along the per-pixel

Gaussian indexes their contribution weights are also determined. These weights are used when accumulating the embeddings associated with a specific Gaussian, mulplying the embeddings with the corresponding Gaussian weight before addition. Once the per-Gaussian weighted embedding accumulation is done the per-Gaussian sum embedding is divided by the per-Gaussian sum of the weights. While cosine similarity is not affected by the norm of the vector, this keeps the embeddings in the same magnitude.

While this approach is somewhat naive, it shows promising results and worth further investigation in a follow-up work whether the simplicity (and potential speedup) worth considerations compared to more sophisticated optimization methods.

3.4. Database population: Vector database population

As described earlier the proposed approach in this project is to store the final per-Gaussian embeddings in a dedicated vector database with keys being the vector embeddings and values the Gaussian IDs, coordinates and color values. This on the one hand can improve scalability as the rather large footprint embeddings are not bound by the GPU memory, on the other hand can speed up vector similarity search in query time.

The challenge with vector search arises when we need to find similar embedding vectors in a big set of objects. If we want to find the closest examples, the naive approach would require calculating the distance to every vectors. That might work with dozens or even hundreds of examples but may become a bottleneck if we have more than that. When we work with relational data, we set up database indexes to speed things up and avoid full table scans. And the same is true for vector search. Vector databases, like the database of choice in this project, Qdrant, speed up the search process by using a graph-like structure to find the closest objects in sublinear time. So calculating the distance to every object from the database is not necessary, only for some candidates. Vector databases are optimized for storing and querying high-dimensional (embedding) vectors efficiently, using specialized data structures and indexing techniques such as Hierarchical Navigable Small World (HNSW, which is used to implement Approximate Nearest Neighbors) and Product Quantization, among others. These databases enable fast similarity and semantic search while allowing users to find vectors that are the closest to a given query vector based on some distance metric. The most commonly used distance metrics are Euclidean distance, cosine similarity, and dot product. CLIP relies on cosine similarity, so this implementation uses that one.

In this project CLIP embeddings are loaded and stored in the vector database database as-is. This implementation is less memory bound than the prior art, no compression is was

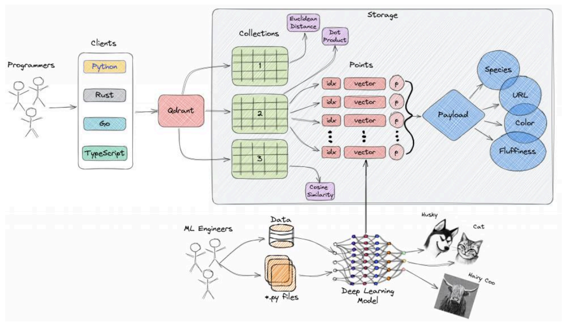

Figure 4. High-level Qdrant architecture

performed beforehand with encoding or quantization. However, Qdrant supports various quantization schemes (scalar, product, binary) to further tune storage footprint and search performance. The advantage of the database's native quantization is that no scene-specific prepossessing is necessary. The performance of such configurations and further optimizations with memory and storage configurations can be assessed in a follow-up work.

3.5. Query processing

Once the vector database is populated with the embedding vector-Gaussian pairs we can query it with natural language. The natural language needs to be embedded using CLIP's text encoder. The embedded text can be used to query the vector database which will return the IDs of the Gaussians with the closest cosine similarity embeddings. A cosine similarity threshold can be configured, to define what is considered to be a query match. In the Experiment/Analysis section the results with a cosine similarity threshold of 0.275 and 0.26 presented. These threshold values provided meaningful results across multiple scenes. The query execution time depends on the size of the scene and the number of returned Gaussians ranging from sub-10 milliseconds to 100s of milliseconds and beyond. Note, however, that the list of Gaussians could belong to multiple objects physically far apart on the scene. We need to perform spatial clustering to determine the number and location of separate objects. Once clustering is performed, we can return the list of centroids of the objects as their (relative) coordinates in the 3D space and the Gaussians belonging to each object. Apps could also get the 3D representations of the query matching objects via returning the subset of Gaussians matching the query to be used for AR overlaying or for spatial information (e.g. grabbing in robotics). Due to the limited time I've only performed basic averaging to identify the centroids of the objects, in a follow-up work scikit-learn's kmeans implementation could be leveraged in in combination with different methods to identify the right number of clusters, including the elbow method, silhouette score, Davies-Bouldin index and gap statistic.

3.6. Visualization

To visualize query results a copy of the Nerfstudio Gaussian scene model has been directly edited, the DC colors of the matching Gaussians has been changed. Loading this modified model into the viewer enabled assessing the results with the Nerfstudio Viewer interface. The implementation was kept simple enabling a single query visualization only, however the Viewer framework has the ability to handle real-time scene manipulation and queries and a follow-up work can leverage that to not only handle subsequent queries in one session, but also visualize vector database population and updates as well in real-time.

4. Experiments/Analysis

For the experiments the LERF ”teatime” and ”kitchen” datasets have been used. The former contains 180 images and the Nerfstudio Splatfacto method builds a Gaussian scene with 530k Gaussians in 15 minutes on an Nvidia RTX 3090 GPU. The kitchen dataset has 617 images and turns into a 900k Gaussian scene in 45 minues. Performing the reverse rendering and per-Gaussian embedding steps takes 20 minutes and 70 minutes for the teatime and kitchen scenes respectively, populating the database takes additional 10 and 20 minutes (due to database indexing).

The query processing time depends on the size of the scene (the size of the corresponding embedding collection in the database). Depending on the limit of matching Gaussian embeddings the query time can vary between 10s and 100s of milliseconds in the 1000s and 10000s limit range in case of a database with 1M embeddings. Further analysis could benchmark how performance changes as we grow the scene (Qdrant claims sub-linear scaling) and how the database performance be further tuned with quantization and memory settings. The CLIP text embedding typically took 10 ms. An analysis can also consider sparser returns (limited number of matches), as it can speed up queries. Tuning quantization and memory also has potentials for speeding up the queries. Such tuning could be useful with large scenes to keep the responses close to real-time.

Figure 5 compares the query results of “bear”, “sheep” and “tea” queries in the ”teatime” scene with a cosine similarity threshold of 0.275 and 0.260. Figure 6 depicts results in the ”kitchen” scene with the same cosine similarity thresholds for the “guitar”, “piano” and “hat” queries. These results are promising, even somewhat consistent across scenes, however there are definitely objects which has more ambiguities (e.g. wooden furniture parts can be hard to tell apart from each other or from the floor by the embedding model), and there are clearly visible masking artifacts (e.g. the legs of the sheep).

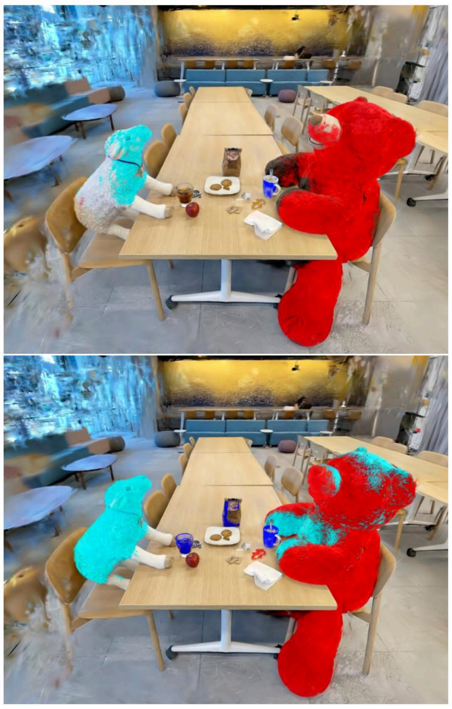
Figure 5. The “teatime” scene with “bear”, “sheep” and “tea” queries with a cosine similarity threshold of 0.275 (top) and 0.26 (bottom).

If we plot the sorted cosine similarity scores along with the number of embeddings that have been averaged for the final per-Gaussian embeddings on Figure 7 we see that that Gaussians with more averaging (the ones contributing to more pixels) tend to be more ambigous. They are neither clearly a match or not a match. A potential future work is to plot the same figure using optimizers to determine the per-Gaussian embeddings and see if they perform better, showing more robust differentiation between matching and non matching Gaussians. Another aspect of a future analysis is how more sophisticated masking schemes with improved cross-frame consistency would impact the differentiation. Better masking could eliminate ambiguities generated by the noise of inconsistent embeddings across viewpoints.

As a quick experiment a simple object localizer has been also implemented which simply calculates the centroids of the query matching Gaussian means. As mentioned earlier a potential future work is to employ a more sophisticated clustering algorithm which has the ability to distinguish multiple object instances.

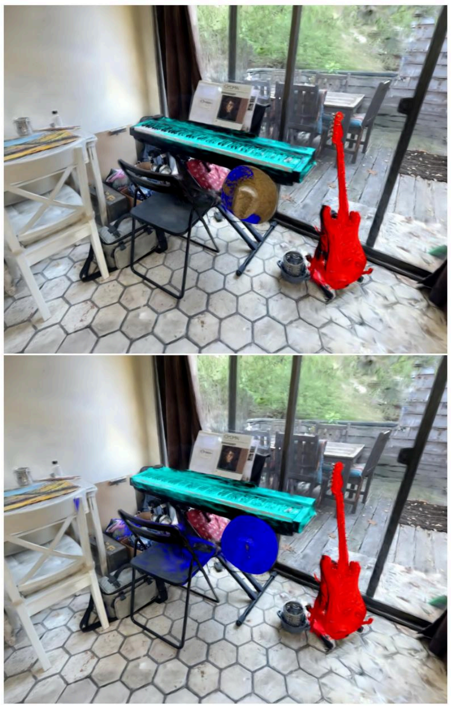

Figure 6. The “kitchen” scene with “guitar”, “piano” and “hat” queries with a cosine similarity threshold of 0.275 (top) and 0.26 (bottom).

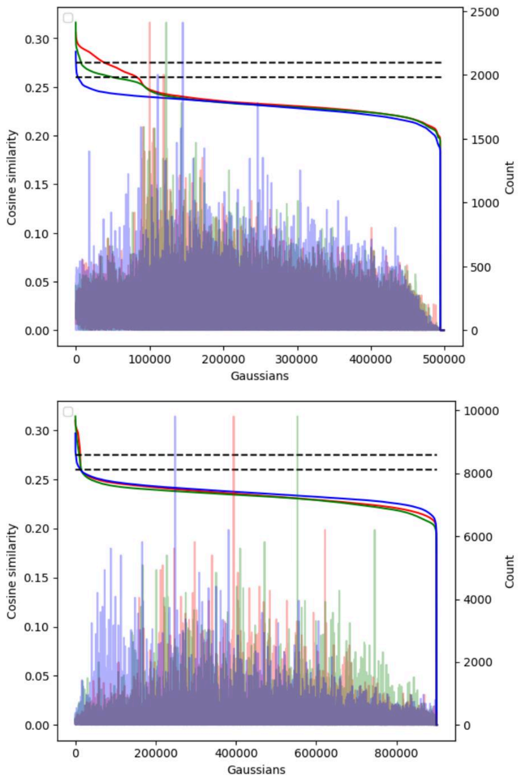

Figure 7. The “teatime” scene with “bear”, “sheep” and “tea” query result Gaussians sorted by cosine similarity scores (top) and the “kitchen” scene with “guitar”, “piano” and “hat” query results. The dashed lines mark the 0.275 and 0.26 cosine similarity thresholds. With lighter colors the number of averaged embeddings per Gaussians are visualized.

5. Conclusion, future work

The results suggests that the application of vector databases in semantic environment mapping has potentials. Separating the concerns of scene representation and storing and processing language embeddings provides more scalability and control to tune system performance. Low-latency query results across the whole scene can be possible with flexible further processing options depending on the use case. It addresses potential scalability challenges appearing with large scenes and also makes integration to downstream tasks easier.

The work also demonstrated that a relatively simple weighted averaging to determine the per-Gaussian embeddings could provide meaningful results, however this aspect definitely needs further analysis in a follow-up work with detailed comparisons to alternative optimization methods. The results also suggest that the embedding quality is largely sensitive to the segmentation mask quality. More sophisticated masking techniques used in related works could be assessed in a follow-up.

There are two horizons of potential future work. As mentioned in this report the current implementation has multiple open questions and improvement areas, including the following research and engineering areas:

- The scalability of the of the vector database in (very) large scenes (reserach/engineering question).
- Detailed assessment of the weighted averaging per-Gaussian embedding against optimization alternatives (research question).
- Assessment of more sophisticated masking techniques including hierarchical SAM masking and Grounded SAM (research question).
- Assesment of clustering algorithms to determine the number of distinct 3D objects including k-means with the elbow method, silhouette score, Davies-Bouldin index and gap statistic (research question).
- If worthwhile, the optimization of the weighted averaging and reverse rendering implementation with

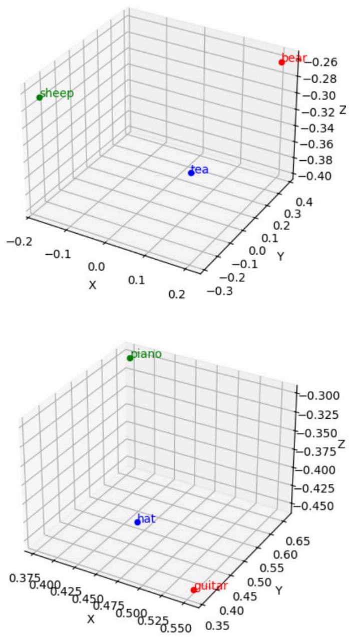

Figure 8. Centroids of the “bear”, “sheep” and “tea” query results in the ”teatime” scene (top) and the “guitar”, “piano” and “hat” query results in the ”kitchen” scene (bottom) with a cosine similarity threshold of 0.275.

CUDA (engineering question).

- Assessment of vector database optimization techniques: quantization, memory and storage tuning, optimized batch upload and queries (engineering question).

- Assessment of integration best practices with downstream tasks: location and 3D asset sharing with robotics, augmented reality applications (engineering question).

There are also opportunities to expand the scope of the work:

- Introduction of a Gaussian splatting-based SLAM into the stack.

- Collaborative environment mapping.

- Continuous mapping and embedding pipeline.

- Handling of dynamic scenes with multiple fixed and mobile cameras.

References

[1] Caron et. Al. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021. 1

[2] Kerbl et. al. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 2023. 1

[3] Kerr et. al. Lerf: Language embedded radiance fields. In *ICCV*, 2023. 1, 2

[4] Kirillov et. al. Segment anything. *arXiv:2304.02643*, 2023. 1

[5] Mildenhall et. al. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1

[6] Qin et. al. Langsplat: 3d language gaussian splatting. *arXiv preprint arXiv:2312.16084*, 2023. 1, 2

[7] Radford et. al. Learning transferable visual models from natural language supervision, 2021. 1

[8] Shi et. al. Language embedded 3d gaussians for open-vocabulary scene understanding. *arXiv preprint arXiv:2311.18482*, 2023. 1, 2

[9] Tancik et. al. Nerfstudio: A modular framework for neural radiance field development. SIGGRAPH ’23, 2023. 3

[10] Ye et. al. Mathematical supplement for the `gsplat` library, 2023. 3