17

Towards Open-World Spatial AI

Liam Paull, Sacha Morin, Dominic Maggio, Martin Büchner, Cesar Cadena, Abhinav Valada, and Luca Carlone

Previously we have seen many different types of potential map representations, including *sparse* representations that tend to represent the world as graphical structures (Chapter 1), *dense* representations that store the detailed geometric information about the 3D space (Chapter 5), as well as *hierarchical* representations, such as 3D scene graphs, that attempt to flexibly balance these two extremes (Chapter 16). Furthermore, we have seen how these representations are created in a way that is dependent on the input sensor modality. Examples include vision (Chapter 7) LiDAR (Chapter 8), and Radar (Chapter 9), among others. However, the types of information contained within those representations have been largely restricted to geometrical (or possibly photometric, e.g., Chapter 14) properties of the environment.

In the previous chapter (Chapter 16), we have seen that modern SLAM systems can move beyond mere geometry to encode semantics and thus enable some form of *Spatial AI*, or reasoning in a more abstract and high-level way. This has the potential to enable robots to perform more complex tasks that can be specified at a more intuitive level. For example, goals can be specified in high-level natural language as opposed to low-level geometric primitives. This is enabled by building representations that contain more semantically meaningful information (such as objects). These representations tend to be built by querying perceptual models that are trained with deep learning. However, in Chapter 16, the models used to build these representations were typically trained in a *closed-set* manner, meaning that the dictionary of potential classes, object types, or concepts was pre-determined at the time the perceptual model (e.g., object detector) was trained and is therefore fixed and finite.

In recent years, we have seen the rise of "foundation models" that are trained on massive datasets, and have shown the ability to learn "open-world" representations, meaning that they potentially generalize to any class, object type, or concept that one might encounter.

These open-world representations can be useful for incorporating semantic information inside of the SLAM map, as shown in Figure 17.1. In this chapter, we

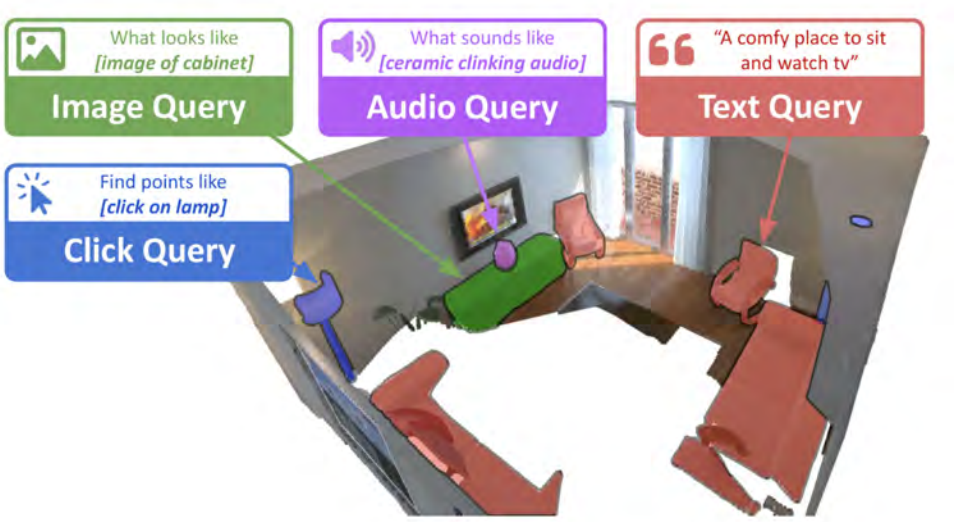

Figure 17.1 By embedding representations from multi-modal foundation models like CLIP [906] into SLAM maps, we can interact and query them in new ways. In this example, adapted from ConceptFusion[520], we have a 3D point cloud where points also store embeddings from other types of encoders. For example, when we query the map with a text query "A comfy place to sit and watch tv", this text is passed through a text encoder and then we can look for all of the points in the map whose representation is sufficiently close (in terms of cosine distance) to the embedded query. The result is that we find all of the chairs and couches. The same methodology can be applied to any modality for which we have a multi-modal foundation model with aligned embedding spaces between the modalities. More details in Section 17.3.

.

will explore in detail how foundation models can be leveraged within the context of SLAM.

17.1 Introduction

17.1.1 A Brief History of Foundation Models

A foundation model is a machine learning model trained on a sufficiently vast dataset that we may reasonably expect that it should *generalize* to a wide variety of operating conditions. Training such models requires two key components: 1) deep learning architectures that are able to efficiently update a very large number of parameters, and 2) a very large (e.g., internet-scale) dataset to train the model. The first model that may apply to this definition in computer vision (although not typically referred to as a foundation model) is AlexNet [616], which was a CNN model trained in a *supervised* fashion (meaning that the dataset contained

labels that were provided by human annotators) on the ImageNet database [268], which comprises 3.2 million labeled images. One issue with training such large models using traditional CNN architectures is that they suffer from the problem of *vanishing gradients*, where gradients that have to pass through many layers become very small and therefore training the parameters based on these gradients becomes very inefficient.

This was initially addressed through the introduction of residual networks [450], or ResNets, which introduce a direct pathway from one layer to the next, making training more stable and efficient. Vision encoders based on ResNets, which were pre-trained on ImageNet, formed the basis for the development of deep-learning-based *closed-vocabulary* visual perception systems, such as object detection models including YOLO [6], RCNN [392], and many others.

However, constructing a large manually labeled dataset such as ImageNet is an arduous task. In the field of natural language processing (NLP), it was possible to automate this process. This was demonstrated by Bidirectional encoder representations from transformers (BERT) [276], which uses a corpus of data to generate a task with associated labels in an automated way as a method for learning meaningful representations. In the case of text, this is done by simply removing a word from a sentence and then trying to predict the missing word. The set of all sentences in the English Wikipedia (∼2.5 billion words) and BookCorpus(∼800 million words) were two of the first datasets used in this manner. This process is referred to as *unsupervised learning* [1] since no human labels are needed.

BERT also showcased the benefits of a new type of model architecture called the *Transformer* [1127], which was based on the concept of *attention* [50]. Attention enabled the capturing of longer-term dependencies in input data. Concurrently with BERT, the first Generative pre-trained transformer (GPT) model was released, which was also based on the transformer and trained on the BookCorpus dataset and showed similar capabilities.

While it was relatively intuitive to perform this type of unsupervised training with language, it was not immediately clear how to extend this concept to higher-dimensional inputs such as images. Words contain specific semantic meanings in a way that random patches of an image may not. This problem began to be unlocked through the development of *contrastive* learning-based approaches combined with vision transformers [288].

In constrastive learning approaches, such as SimCLR[191], representations are learned by augmenting the input image in some way (such as removing patches, flipping it etc.) and then creating a loss function that pushes the original and augmented image closer in representation space (positives), while pushing all other

[1] It is fairly common in the literature to distinguish between *self-supervised* learning, where a label is manually generated, and *unsupervised* learning, where there is no explicit label, but we make no such distinction here and refer to all methods that do not require human annotations as *unsupervised*.

inputs or modified inputs away (negatives). Through this process, the hope is that the representations that are learned are invariant to the data augmentation and are therefore robust for downstream visual perception tasks (see Figure 17.2).

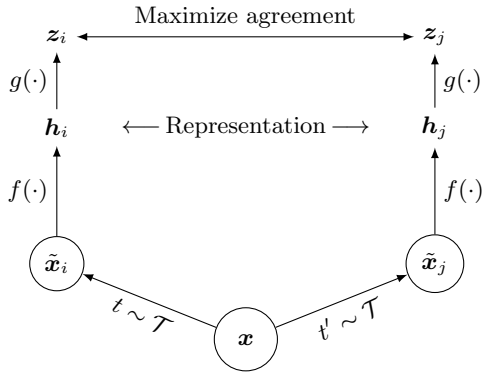

Figure 17.2 Reproduced from [191]. An input $\boldsymbol{x}$ is red through two different data augmentation operators sampled ($t \sim T$ and $t' \sim T$) and applied to each data example to obtain two correlated views. These augmented inputs are then fed through an encoder network $f(\cdot)$ to obtain a representation, and a projection head $g(\cdot)$, which is used for training through a contrastive loss.

Concurrent with the development of contrastive learning was the porting of the transformer to the visual domain [288], shown in Figure 17.3. By leveraging a similar idea of "tokenization" of an image and multi-headed self-attention, the vision transformer was applied to classification and shown to be competitive with state-of-the-art supervised ResNet models on classification tasks.

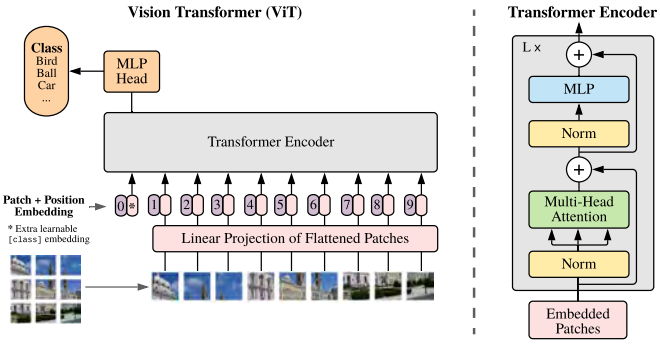

Figure 17.3 Reproduced from [288]. The vision transformer splits an image is split an image into fixed-size patches and linearly embeds each of them together with position embeddings. The resulting sequence of vectors is fed as input to a standard Transformer encoder. To perform classification, an extra learnable "classification token" is added to the to the sequence.

The combination of these two approaches led to breakthroughs in unsupervised visual representation learning. One canonical example was Deeper into neural networks (DINO) [160], which used a teacher-student distillation setup (instead of contrastive learning) for unsupervised training of a vision transformer encoder, and was remarkably impressive at learning visual representations.

Contrastive learning also enabled the alignment of representation spaces *across modalities*. For example, contrastive language-image pre-training (CLIP) leverages image-text pairs collected from the internet and uses a contrastive loss to push an image and its associated text caption closer together in feature space, while pushing all other examples away [906].

At this point, the majority of the theoretical tooling was present to enable feature-based and generative foundation models (see Section 17.2 for further details about this distinction). Further improvements to models, datasets, and workflows, combined with ever-increasing computational capabilities, continue to improve the representational capabilities of these models, showing impressive ability to generalize to unseen tasks and settings. However, it is somewhat unclear whether this progress will continue at the same rate in the future.

As we turn our focus to the ways in which these models can be useful in the context of SLAM, it is important to keep in mind that the existence of truly massive, internet-scale datasets has been one of the key ingredients in the incredible representational power of these models. As a result, several data modalities that are commonly used in SLAM systems, and have been discussed in previous chapters, such as LiDAR or Radar, among others, lack readily available data at this scale.

17.1.2 Nomenclature and Scope

We seek to make precise exactly what types of methods we consider to be in scope for the remainder of this chapter. A first concept to make precise is exactly what we mean by "open-world", as this represents the key distinction between this chapter and Chapter 16.

Open-World Queries. There are (at least) two ways that we could envisage that this open-world concept could be applied in the context of SLAM. In the first paradigm, we could consider using open-world variants of front-end perception systems. Examples of such object detection systems include YOLO-World [202], which is an open-vocabulary variant of YOLO [6], and OV-DETR [1259], which is an open-vocabulary variant of the "Detection Transformer" (DETR) [152]. However, it is important to emphasize that while these models do admit open-vocabulary text-conditioned queries, this conditioning must happen *at the time that the perception model is queried*, which in the case of SLAM is at the time of map construction. Consequently, the resulting map should be considered to be *closed-world* since the

decision about which concepts should be represented is enforced at the time of map creation.

A second paradigm typically embeds open-world representations into the SLAM map in a way that enables *open-world queries* when the map is actually being used. We will focus primarily on this second paradigm in Section 17.3.

Localization vs. Mapping vs. SLAM. Foundation model representations have been used for visual place recognition (VPR) and localization/loop closure detection. For example, AnyLoc [555] studies which layers and features from DINO [160] can be aggregated to derive image descriptors for VPR under spatial, temporal, and viewpoint variation. While this is certainly an important line of work related to SLAM, we will focus more directly in this chapter on the data structures and algorithms that enable open-world querying and task specification in SLAM maps. The vast majority of the methods that we will discuss in the remainder of the chapter are actually pure *mapping* algorithms since they assume the availability of *posed* camera frames (for example from some upstream SLAM system such as one of the ones previously discussed). As such, in most cases, the problem reduces to the question of *how to embed foundation model representations into SLAM maps?*

Chapter Outline. The remainder of the chapter is structured as follows: in Section 17.2, we continue with a more detailed technical discussion of some of the most influential and useful foundation models for open-world mapping; in Section 17.3 we present several examples of how these open-world foundation model representations are being incorporated and embedded into SLAM maps to enable joint geometric/semantic reasoning; and, finally, in Section 17.4, we look forward to what might be next in this rapidly evolving field.

17.2 Foundation Models for Spatial AI

In this section, we will introduce three classes of foundation models helpul for building open-world map representations. We first distinguish between **feature-based** and **generative** foundation models. Feature-based foundation models such as CLIP [906] extract real-valued vectors from text or images (i.e., features) and rely on feature space computations to achieve tasks. In contrast, generative foundation models can generate novel data given text and/or image prompts. The distinction is not always clear-cut: some of the feature-based models we will introduce are actually used as subcomponents in generative models, and conversely, features can technically be extracted from generative models. However, differences in outputs and typical use cases in the mapping context support the distinction. Generally speaking, we also note that inference with generative models, while potentially more powerful, tends to be more computationally expensive than extracting features. We conclude this section with a brief overview of useful computer vision models for

class-agnostic image segmentation since these models are very widely used in the various approaches that we will cover in Section 17.3.

17.2.1 Feature-Based Foundation Models

Features, also known as embeddings, representations, or codes, are real-valued vectors resulting from the encoding of an input data point by a deep neural network. We will only briefly mention network architectures and training objectives and simply denote different feature extractors as functions, leaving the reader to consult works such as [400] for a proper overview of deep learning. For the purposes of this chapter, we will consider that we have an encoder f_{x} (typically a deep neural network) that can be applied to an input x of modality x resulting in d-dimensional features $\boldsymbol{f}_x$ according to $\boldsymbol{f}_x = f_{\mathrm{x}}(x) \in \mathbb{R}^d$.

The field of deep learning aims to learn such encoding functions with useful properties. A key property for this chapter will be the ability to easily measure meaningful **similarities** between data points. Distances in feature space will help us answer queries such as *"How similar are these two text excerpts?"* or *"How similar is this text caption to this image?"*

Research in self-supervised learning objectives and deep architectures has led to the emergence of feature extractors that can act as a foundation for a wide range of applications. Modern feature extractors are pre-trained on large internet-scale NLP and computer vision datasets. While it is possible to fine-tune these models for specific tasks or even use their features as inputs to downstream classifiers or regressors, the mapping methods we consider in this chapter mostly use the features "as is". Here, we present a brief overview of some available text, vision, and multimodal feature extractors.

Natural Language. BERT [276] transformed the NLP landscape and introduced bi-directional transformers pre-trained on a masked language modeling objective. The resulting model achieves strong performance on a number of downstream NLP tasks, with and without fine-tuning. While BERT provides features at the level of tokens (i.e., substrings of the input), SentenceBERT [930] adds a pooling operation and fine-tunes BERT to output sentence-level features that can efficiently be compared with cosine similarities (the cosine of the angle between the two embeddings treated as vectors originating from the origin). SentenceBERT model checkpoints are available in a number of languages and have been fine-tuned for a variety of NLP tasks.

Vision. DINO [160] trains vision transformers [288] with a self-supervised learning scheme involving a student network matching the predictions of a teacher network. A key training ingredient is multi-crop augmentation: the teacher is exposed

to global views ($> 50\%$ of the image) while the student has to process both local views ($< 50\%$ of the image) and global views of the same image, encouraging "local-to-global" learning. DINOv2 [837] proposes various improvements to the training procedure and introduces a curated dataset of 142M images. The result is a powerful visual feature extractor with a strong holistic understanding of images. DINO outputs **image-level** visual features that generalize zero-shot to downstream tasks such as image classification or image retrieval. For example, distances in the DINO feature space can serve as a foundation to build k-nearest neighbors classifiers for specific supervised tasks. Even simple models such as logistic regression can achieve strong performance in the image domain when trained on DINO features. The transformer architecture underpinning DINO also allows for the prediction of **patch-level features** (e.g., a feature vector for every 8x8 patch in the image). Training additional models on these dense features can yield strong performance on pixel-resolution downstream tasks such as semantic segmentation, depth estimation, and surface normal estimation [837, 52]. Directly using distances in feature space has also been explored for multi-view correspondence estimation [52].

Multimodal. A learned feature space can be a powerful tool to reason about similarities between data points from the same modality. Different modality-specific encoders can also map to a common **aligned feature space** to enable multimodal capabilities. CLIP [906] is a notable example of this approach in the vision-language setting, training an image encoder f_{image} and a text encoder f_{text} (Fig. 17.4). Given an RGB image $\boldsymbol{I}$ and a text query T, CLIP extracts unit-norm features $\boldsymbol{f}_I = f_{\text{image}}(\boldsymbol{I})$ and $\boldsymbol{f}_T = f_{\text{text}}(T)$. This enables image-text comparisons using the cosine similarity $\langle \boldsymbol{f}_I, \boldsymbol{f}_T \rangle$.

CLIP encoders are trained using a contrastive learning scheme on 400M image-caption pairs collected from the internet, yielding features encoding a broad array of visual and textual concepts out-of-the-box. CLIP features can be repurposed for a variety of downstream tasks without any fine-tuning or retraining. For example, we can construct a **zero-shot image classifier** by defining a set C of class labels in text form, possibly using a template sentence (e.g., {`a picture of a dog`, `a picture of a cat` }), and use CLIP similarities to predict a class for an image $\boldsymbol{I}$ with

$$\arg\max_{c \in \mathsf{C}} \langle \boldsymbol{f}_I, \boldsymbol{f}_c \rangle. \tag{17.1}$$

Reformulating a classifier for a new problem is simply a matter of specifying a new class set through the definition of the text prompts, allowing transfer to a wide array of tasks. The classifier in (17.1) can be extended to other modalities such as sound [428], given an appropriate encoder.

Extracting dense patch-level image features from CLIP vision encoders is also possible. However, one should not expect these features to be language-aligned,

and some studies have found them to be more limited for vision and 3D-related tasks in comparison to models such as DINO [837, 52].

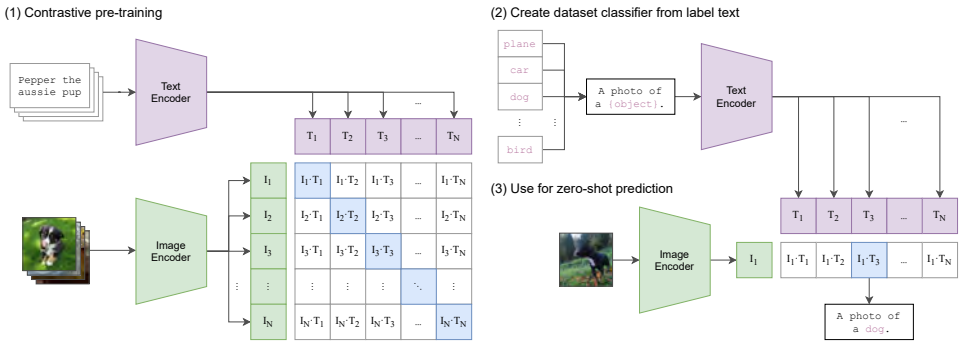

Figure 17.4 Figure reproduced from [906]. (1) CLIP learns image and text features with contrastive learning. Image and caption features from the same image-caption pair are incentivized to be close in feature space (positives, in blue) while image features and features from other randomly sampled captions are pushed away (negatives, in white). We note that the pre-training of CLIP does not rely on the definition of any particular class set. (2) After training, class text labels (possibly in a template) can be encoded to define a classifier. (3) The feature vector extracted from an image can be compared with class features for zero-shot prediction.

17.2.2 Generative Foundation Models

Large language models (LLMs) are transformers trained for text generation, generally using self-supervised objectives. The models are typically large (i.e., many learnable parameters) and trained on a massive text corpus, resulting in impressive generalization. Here we outline key aspects of LLM training and inference and refer the reader to [1127, 1117, 1108] for an in-depth treatment of transformers in the NLP context.

The first step in language modeling is tokenization. Tokenization is the process of segmenting the input text into tokens (e.g., words, subwords, characters, or bytes) defined in a vocabulary. Each token is assigned to a unique integer value and has a corresponding learnable embedding. LLMs will process text as a sequence of token embeddings.

The specifics of LLM training are beyond the scope of this chapter, but the core learning objective is **next-token prediction**. Given a context of $T-1$ tokens $w_{1:T-1}$, LLMs learn to predict the probabilities $p(w_T|w_{1:T-1})$ of the next token. At inference time, LLMs will accept an initial context of text instructions (a prompt) and autoregressively generate text by predicting the next token, adding it to the context, and repeating the process until some stopping condition is met.

LLMs can, in principle, be used for any task where instructions, inputs, and outputs can be specified in text form. Examples include interactive chatbots, code generation, text summarization, or symbolic planning. Prompt design will significantly impact task performance and has driven research on prompt engineering techniques such as chain-of-thought (prompting the model to think step-by-step) and in-context learning (providing input-output examples in the prompt). Some widely used LLMs include the LLaMa family of models [1108] and the GPT models [12]. Many others are available with newer, more powerful models being released frequently.

There is great interest in extending the general abilities of LLMs to other modalities, such as vision. Vision language models (VLMs) share most of the components of LLMs in addition to accepting image tokens as part of their input. LLaVA [655] achieves multimodal capabilities by connecting the CLIP vision encoder to an LLM and training on vision and language data. Recent GPT models also support vision inputs and outputs. Given an adequate prompt, VLMs can perform tasks such as image classification, image captioning, and more general visual question answering.

17.2.3 Class-Agnostic Image Segmentation

Image segmentation is a key component in many mapping methods and will be essential for reasoning about the semantics of specific image regions. We will be interested in detecting a set of R segmentation masks $\{\boldsymbol{Z}_i \in \{0,1\}^{H \times W}\}_{i=0}^{R}$ representing the likely objects in an RGB image I of width W and height H. Critically, we need the mask proposals to be class-agnostic to properly capture the breadth of objects typically found in robot environments (and to exploit the open-vocabulary nature of the foundation models that we just introduced). Throughout this chapter, we will denote this function as $\texttt{Seg}(\cdot)$.

The main tool to implement this function is SAM [589, 918]. SAM is based on large-scale supervised learning for class-agnostic image segmentation. The result is a promptable segmentation model that takes as input pixel coordinates, a bounding box, or even another segmentation mask and predicts a segmentation mask. SAM can also automatically process a full image to generate a set of masks describing different instances in the image. While SAM can be applied directly as a standalone segmentation model, it can also process the bounding box detections from an open-vocabulary object detector [688] to achieve better object definition and reduced inference time.

17.3 Open-World Mapping

In Chapter 5, we saw how geometric quantities such as occupancy or distance fields could be stored in maps, and Chapter 16 extended this to semantics, primarily through the storage of additional class information. While certainly offering a

reasonable discrete approximation of scene semantics, the class framework faces a number of limitations in practice. Chief among them is the **closed-set** assumption: classical semantic maps require *a priori* knowledge of the **vocabulary** of classes we wish the map to support. For example, any mapping system relying on image classifiers or object detectors inherently assumes that the world can be fully described by the vocabulary of their perception front-end.

In practice, predefined object classes only offer a very coarse approximation of reality. Consider the typical objects found in a toolbox. Most classifiers will lack the vocabulary to fully describe the usual objects in our box. While "tool" might be an acceptable general description for each object, more specific labels such as "screwdriver" or "hammer" are certainly more desirable. Even then, we might be interested in additional object properties such as the color of specific tool instances ("red hammer", "blue screwdriver"), their material ("metal", "rubber") and potentially other physical properties ("hard", "soft", "hot", "cold") that are typically lost if we assign a single class to a map element. Of particular interest to robots, we might want to encode additional information on the typical uses and affordances of a given object ("cutting", "measuring"). In the case of objects with depictions ("a painting of Ronaldo"), an ideal map would encode both the medium ("painting") and the content, even in the case of a specific cultural reference ("Ronaldo"). Meeting all these objectives with a predefined set of classes is an arduous task. A vocabulary containing hundreds of classes may still fall short at some level, even on common objects, and the issue is magnified for rare "long-tail" objects that are unlikely to be included in the vocabulary in the first place. Other continuous properties, such as the typical sound an object can make, are inherently hard to represent with a discrete class label.

To remove the closed-set assumption and enable support for a broader range of concepts, we can replace class labels in the map with some high-dimensional continuous features extracted by the foundation models we introduced in Section 17.2. The large datasets used to train these feature extractors ensure that the features can represent a vast, **open** set of objects, along with their associated properties. While grounded features themselves are not directly usable, their multimodal alignment allows for comparison with queries expressed in different modalities, primarily through natural language, and enable the ability to perform a number of tasks in a zero-shot manner.

Throughout this section, the primary sensor of interest will be RGB-D, which supports both RGB feature extraction using image-based foundation models and pixel-level conversion to 3D representations like point clouds. We generally assume a sequence of **posed** RGB-D frames in a static scene, therefore focusing on the mapping problem as opposed to the full SLAM problem. We process each frame sequentially to incrementally build our maps.

17.3.1 Dense Representations

In this section, we study the three main components involved in building a dense open-vocabulary map: 1) the choice of 3D representation, 2) the extraction of open-vocabulary features from sensor data using foundation models, and finally 3) the aggregation across frames and its spatial grounding of these features. Finally, we will discuss the process of performing inference or querying the map.

Map Representation. Open-vocabulary mapping embeds semantic features from foundation models into one of the dense spatial representations introduced in Chapter 5. Without loss of generality, we will assume that a map representation is comprised of a set of map elements $\mathsf{M} \triangleq \{m_k\}$ where each map element $m_k \triangleq \{\boldsymbol{p}_k, \boldsymbol{f}_k\}$ is composed of a geometric component $\boldsymbol{p}_k$ (voxel, point, etc.), and a semantic component or feature vector $\boldsymbol{f}_k$.

The choice of representation is mainly driven by the downstream application: navigation methods will favor cell-level feature aggregation in a 2D grid [492] while a mobile manipulation pipeline may opt for voxel-grids [685]. ConceptFusion [520] and OpenScenes [870] model space and semantics in a map with an unordered set of 3D points. Open-Fusion utilizes a volumetric representation with a TSDF [1209]. A hybrid approach taken by VLMaps is to accumulate the information in a 3D point cloud but then project it into a 2D grid map [492].

It should be emphasized that the geometric component of the map elements, $\boldsymbol{p}_k$, is assumed to be provided by an upstream SLAM system, such as one of the ones that we have seen in previous chapters.

Feature Extraction. We turn our attention now to the choice of which features to embed in the 3D representation and how to obtain them. A common approach is to extract intermediate semantic features from the RGB data of a given frame $\boldsymbol{I}$ using a foundation model f, such as one of the ones described in Section 17.2.1. One issue is that foundation models typically yield image-level features encoding information on the whole collection of objects in the camera frustrum. While this larger context is valuable, we are also interested in more fine-grained, local semantics to properly describe the different parts of the image, potentially at a pixel-level resolution. But there is a question of how to obtain these more fine-grained features.

One approach is to use a foundation model that directly outputs pixel-level [654] or region-level [387] features. These models are trained by fine-tuning a foundation model like CLIP on a segmentation dataset. OpenScenes and VLMaps directly extract CLIP features from the RGB images through LSeg [654]. Open-Fusion [1209] adopts a different strategy by employing SEEM [1307] to directly extract region-based features, aiming for better computational efficiency.

However, recent evidence suggests that these dense features do not capture as many long-tail concepts as the original unaltered CLIP features [520]. An alterna-

tive approach taken in ConceptFusion aims to preserve the original CLIP features and relies on feature arithmetic to construct pixel-level feature vectors. However, this requires some heuristics to balance the need for local and global semantics information.

The method balances the need for local and global semantics by computing:

- **Global features** $\boldsymbol{f}^G$ by encoding the entire image with $\boldsymbol{f}^G = f(\boldsymbol{I})$;
- **Local features** $\boldsymbol{f}_i^L$ by processing all the masks $\mathsf{R} = \texttt{Seg}(\boldsymbol{I})$ inferred by a class-agnostic segmentation method (such as SAM [589] see Section 17.2). Each mask is converted to a bounding box to create tight image crop $\boldsymbol{I}_i$. Applying f results in mask-level features $\boldsymbol{f}_i^L = f(\boldsymbol{I}_i)$;
- **Pixel-level mask features** $\boldsymbol{f}_i^P = w_i \boldsymbol{f}^G + (1 - w_i)\boldsymbol{f}_i^L$. The weights $w_i \in [0, 1]$ for the linear combination are computed to emphasize local features that are i) distinct from the global features and ii) unique when compared to the other local features; and
- **Pixel-level features** $\boldsymbol{f}_{u,v}^P$ by summing all the $\boldsymbol{f}_i^L$ such that $\boldsymbol{Z}_i[u, v] = 1$ (i.e. the pixel-level features of the masks covering the pixel) and unit-normalizing the result.

Certainly, other methods of local and global feature merging could be possible, and performing this merge in a more principled manner could be an interesting avenue for future work.

In summary, assuming a pinhole camera model and pixel-feature correspondences, we can follow the procedure introduced in Section 5.1 to generate an open-vocabulary semantically-enhanced organized point-cloud observation. Each point has a corresponding semantic feature vector calculated using one of the methods above (i.e., either querying a pixel-resolution model or performing global/local feature merging). We refer to these as *range-feature* observations, analogous to the *range-category* observations introduced in Chapter 16.

Feature Aggregation and Grounding. Given a choice of spatial representation and a method for generating *range-feature* observations, the next component deals with how the features from different RGB-D frames that correspond to the same 3D location get combined into one that *annotates* the corresponding map element.

A straightforward approach is adopted by VLmaps where the *range-feature* observations in 3D from all frames are averaged if they correspond to the same 2D cell projection [492].

ConceptFusion [520] converts the *range-feature* observation to a global coordinate frame, then, for every element m_k in the map, all pixel-level features corresponding to the point $\boldsymbol{p}_k$ are weighted and averaged by the feature confidences. The features,

$\boldsymbol{f}_k$ and confidences, c_k are updated using an averaging scheme:

$$\boldsymbol{f}_k \leftarrow \frac{c_k \boldsymbol{f}_k + \alpha \boldsymbol{f}_{u,v}^P}{c_k + \alpha} \tag{17.2}$$

$$c_k \leftarrow c_k + \alpha \tag{17.3}$$

where $\alpha = e^{-\gamma^2/2\sigma^2}$ is the confidence assigned to each pixel-feature associated to the point being aggregated, γ is the radial distance to the camera center, and σ is a scaling term. Intuitively, α assigns higher confidence to features closer to the camera center, similar to previous 3D reconstruction work [557].

Similarly, OpenScene [870] averages all the per-frame 2D features. However it also distills them into 3D features. The fusion of averaged 2D features and the distilled 3D ones is carried out through an ensemble mechanism. Open-Fusion [1209] takes a different approach and maintains in each occupied voxel only a key to a dictionary of features. The decision on which key to assign to the voxel results from a temporal feature matching.

Other works have investigated alternative feature aggregation or ensembling strategies to i) avoid averaging high-dimensional feature vectors, and ii) increase robustness to outlier feature vectors. For instance, accumulating a feature set $\mathsf{F}_k = \{\boldsymbol{f}_{k,1}, ..., \boldsymbol{f}_{k,t}\}$ over time for all map elements and retaining the feature vector with the lowest Euclidean distance to the other features in the buffer (the medoid vector) [870]. Clustering methods can reduce F_k to a few representative cluster centroids [712] or make feature selection more robust by selecting the feature vector in F_k that is closest to the centroid of a majority cluster [1179].

Inference. Once the features are grounded and aggregated in the map, they can be used to perform various tasks by comparing them to other features extracted from different modalities. For example, in the case of vision and language feature-alignment, as is the case with CLIP, **zero-shot semantic segmentation** can be achieved by encoding the textual class labels in a prompt list C. If the map contains map elements m_k with corresponding features $\boldsymbol{f}_k$, then point-level labels y_k can be predicted with

$$y_k = \arg\max_{C \in \mathsf{C}} \langle \boldsymbol{f}_k, \boldsymbol{f}_C \rangle \tag{17.4}$$

where $\boldsymbol{f}_C$ is the feature representation obtained by encoding the prompt C with the text encoder: $\boldsymbol{f}_C = f_{\text{text}}(C)$. Note that this mirrors exactly the image classifier discussed in Section 17.2. Critically, C does not need to be known when building the map and can be redefined at will to predict different labels. The segmentation process for other map representations, such as grid maps and voxel maps, is analogous.

However, the main use case of open-vocabulary maps is arguably **querying**.

Open-ended querying is achieved by encoding a query and comparing query features with all map features. In the CLIP context, one can encode a free-form text query T to extract the corresponding feature vector $\boldsymbol{f}_T = f_{text}(T)$ and score the points in M based on the query similarities $\langle \boldsymbol{f}_k, \boldsymbol{f}_T \rangle$.

The resulting similarities can be thresholded to return a set of relevant map elements. The spatial coordinates of the returned elements can also be clustered to identify different instances. Again, no prior knowledge of the queries is required at map building time and the generality of the multi-model foundation model will ensure that even the most specific queries can be reasonably answered.

While language is a natural way of specifying queries, any modality with an aligned encoder can be used to extract query features. Query features can also be retrieved through user interaction. As an example, in addition to text, ConceptFusion explores a range of **multimodal queries** (see Figure 17.1):

- **Click query** is taken as the feature vector $\boldsymbol{f}_k$ of a clicked point in a visualization of M;
- **Image query** $\boldsymbol{f_I} = f_{\text{image}}(\boldsymbol{I}_{\text{query}})$ using the image encoder f_{image} on a query image $\boldsymbol{I}_{\text{query}}$;
- **Audio query** $\boldsymbol{f}_s = f_{\text{audio}}(s)$ using the AudioCLIP [428] sound encoder f_{audio} on a sound clip s.

Zero-shot inference is more limited when using vision-only features (e.g., DINO [160]). Such maps will still support image queries, local image queries (e.g., with image clicks), and map click queries. Users can click on a few objects in some reference images to achieve accurate segmentation of objects and parts in the map [752].

17.3.2 Object Maps and 3D Scene Graphs

While dense open-vocabulary maps can model semantics at a very granular level, they scale poorly with the size of the environment due to the high dimensionality of commonly used feature vectors. In our ConceptFusion example, we saw the value of considering masks in images to extract local features. This strategy leverages spatial correlations: nearby pixels in an image tend to belong to the same objects and share the same semantics. This observation also applies to maps in general. We expect nearby map elements (points, voxels) to generally share similar semantics, which suggests that semantic features could be stored at the level of objects without incurring a large loss of information.

In this section, we study the construction of open-vocabulary object maps and 3D scene graphs. Building on the maps introduced in Section 5.1, we model every object in the scene with a geometric representation and only store one feature vector per object, drastically reducing the memory footprint of the map. Object-centric representations are also particularly well-suited for embodied AI applications since they lend themselves more naturally to object retrieval queries and object-centric

task planning. We first describe how to build an open-vocabulary object-centric map using steps from OVIR-3D [712] and ConceptGraphs [415] as examples. We then discuss different ways to model relationships between objects (object scene graphs) and how to integrate objects in spatial hierarchies (hierarchical scene graphs) and highlight HOV-SG [1179] as an example of such an approach.

Map Representation. We represent the map as a set of J objects $\mathsf{V}_O = \{o_j\}_{j=1}^{J}$. Each object o_j is characterized by a geometric representation and a feature vector. The geometric representation can range from sparse abstractions such as object centers [184] and 3D bounding boxes to dense representations like 3D point clouds [415, 1179, 1071]. For the examples in this section, we will describe each object with a 3D point cloud P_{o_j}, a feature vector $\boldsymbol{f}_{o_j}$, and a detection count n_{o_j}. We incrementally build the map by processing incoming posed RGB-D frames, adding or initializing objects as needed.

Feature Extraction. Object-centric maps typically rely on a class-agnostic segmentation method (such as SAM [589]) to extract a set of masks $\mathsf{R} = \texttt{Seg}(\boldsymbol{I})$ from the current image. We convert each mask to a bounding box, create the corresponding image crop $\boldsymbol{I}_i$ and extract some mask-level features $\boldsymbol{f}_i^M = f(\boldsymbol{I}_i)$. This procedure is identical to the extraction of local features in ConceptFusion.

Using the depth data, camera pose and camera intrinsics, we backproject the pixels of every mask to 3D resulting in mask point clouds P_i^M which are converted to the map frame. Mask point clouds can also be denoised at this stage (e.g., with DBSCAN [319]). In tandem with the mask features, we obtain a *mask-feature* observation $\{(\mathsf{P}_i^M, \boldsymbol{f}_i^M)\}_{i=0}^{R}$.

Object Association. We now need to measure the similarity of our masks with existing objects in V_O. Methods will usually rely on some notion of geometric and semantic similarities. Geometric similarity can be measured as the nearest neighbor ratio $\phi_{geo}(\mathsf{P}_i^M, \mathsf{P}_{o_j}) \in [0, 1]$ defined as the proportion of points in point cloud P_i^M that have nearest neighbors in point cloud P_{o_j}, within a distance threshold of δ_{nn} [415, 1179]. Semantic similarity can be measured using a function of feature similarity such as $\phi_{sim}(\boldsymbol{f}_i, \boldsymbol{f}_{o_j}) = \langle \boldsymbol{f}_i, \boldsymbol{f}_{o_j} \rangle / 2 + 1 \in [0, 1]$.

Geometric and semantic similarities can be combined in an overall score or thresholded separately to identify potential mask-object matches. A greedy strategy [415] can associate incoming masks to a single object in V_O. Other more sophisticated schemes allow one mask to be associated with multiple objects [712, 1179]. If a mask has no match, its geometry and features can be used to initialize a new object in V_O.

Object Fusion and Feature Aggregation. In the event of a match between o_j and the i^{th} mask, object o_j needs to be updated. One approach is to combine point

clouds and average features:

$$\boldsymbol{f}_{o_j} \leftarrow \frac{n_{o_j}\boldsymbol{f}_{o_j} + \boldsymbol{f}_i}{n_{o_j}+1}$$
$$\mathsf{P}_{o_j} \leftarrow \mathsf{P}_{o_j} \cup \mathsf{P}_i^M$$
$$n_{o_j} \leftarrow n_{o_j} + 1$$

with P_{o_j} being subsequently downsampled to remove redundant points. The alternative feature aggregation strategies discussed for dense maps also apply here.

Open-vocabulary methods will also rely on additional processing steps to refine object estimates. Methods may periodically (i.e., every few frames) run association and fusion algorithms between the objects in V_O to minimize the number of redundant objects [415, 712]. It is also possible to remove objects with a low detection count n_{o_j} either periodically during mapping or after processing all frames.

3D Segmentation. An alternative to 2D image segmentation and mask association would be to accumulate all frames in a global scene point cloud and then directly perform object segmentation in 3D, using either a learned model [1071] or geometric techniques such as region growing [547]. By reprojecting 3D object masks, methods can identify representative images for each object and extract features accordingly. 3D segmentation approaches have the potential to be faster since they avoid segmenting individual RGB frames and associating their segments. However, they do not lend themselves naturally to incremental mapping since the full scene point cloud is required.

Inference. The zero-shot segmentation and querying mechanics are very similar to the dense open-vocabulary mapping scenario. For zero-shot segmentation, 3D points inherit the predicted class label of the object to which they belong. Assuming a point $\boldsymbol{p}_k \in \mathsf{P}_{o_j}$ belonging to an object with some CLIP-aligned object features $\boldsymbol{f}_{o_j}$, the predicted point-level class label is defined as

$$y_k = \arg\max_{C \in \mathsf{C}} \langle \boldsymbol{f}_{o_j}, \boldsymbol{f}_C \rangle$$

where C is a set of classes in text form and $\boldsymbol{f}_C$ is the CLIP text feature vector of a given class prompt. Objects and their similarities can also be adapted for instance segmentation [1071].

Queries become significantly more useful as they can now be used for object retrieval. Given some text query, T, its features $\boldsymbol{f}_T$, we can retrieve the most relevant object in the object set V_O:

$$o^* = \arg\max_{o \in \mathsf{V}_O} \langle \boldsymbol{f}_o, \boldsymbol{f}_T \rangle$$

Object retrieval capabilities are extremely useful for language-driven goal specification in the context of navigation and manipulation. For instance, a query such as "`Something to drink`" will correlate strongly with the object features of a soda can. The corresponding point cloud can be forwarded to a motion planning pipeline to navigate to, inspect and potentially manipulate the soda can.

Language Descriptions. Language-aligned object features work best when compared to the language features of short descriptive queries. They typically fail to capture the nuances of more complex queries involving negation or affordance information. "`Something to drink other than a soda can`" will likely correlate with a soda can features regardless of the query intent. Moreover, features are not *explicit* and cannot be directly understood by human users or LLMs.

An alternative to object features is to directly use free-form language to describe objects in the map. Language as an alternative representation of semantics is made possible thanks to recent progress in developing VLMs that can reasonably describe most objects. A simple way to extract object descriptions is to track the best image crops of each object (e.g., by tracking the masks that contributed the most points to an object) and pass them to a VLM along with a prompt such as "`describe the central object in the image`". Individual object views can be captioned before obtaining on final summary per object using an LLM. Some VLMs also directly accept multiple images as input. We note that some methods use both features and language descriptions [415, 170].

Object captions are much more expressive than closed-set class labels and benefit from the general object understanding of VLMs. For querying, an LLM can be instructed to "search" for the most relevant object in the map given the list of object captions and a user query. This LLM-based object retrieval strategy has been found to address some of the limitations of feature-based object retrieval with more complex object queries [415]. We note, however, that LLM captioning and inference is typically much slower than feature-based alternatives. Alternative text-based representations such as language tags have also been leveraged effectively [1270].

Object Scene Graphs. We can consider a more general map representation $\mathsf{G} = (\mathsf{V}_O, \mathsf{E}_O)$ where the edge set E_O explicitely models the pairwise relationship between the different objects in V_O, forming a 3D scene graph. Much like object semantics, we aim to infer open-vocabulary edge descriptions either using language or features. ConceptGraphs [415] relies on object captions and positions to infer spatial relationships such as "`a on b`" or "`b in a`" with an LLM and discusses the extension of spatial edge types to other relationships a language model can interpret, such as "`a backpack` *may be stored in* `a closet`" and "`sheets of paper` *may be recycled in* `a trash can`". Edge features can be derived by encoding images where two objects are co-visible with a foundation model [596] and there is a growing interest in learning relationships using graph neural networks [595, 597]. Querying

in the context of object scene graphs requires additional thought: OVSG [170] maps contextual language queries to graph structures using LLMs and proposes a graph matching approach to compare them with the scene graph [170].

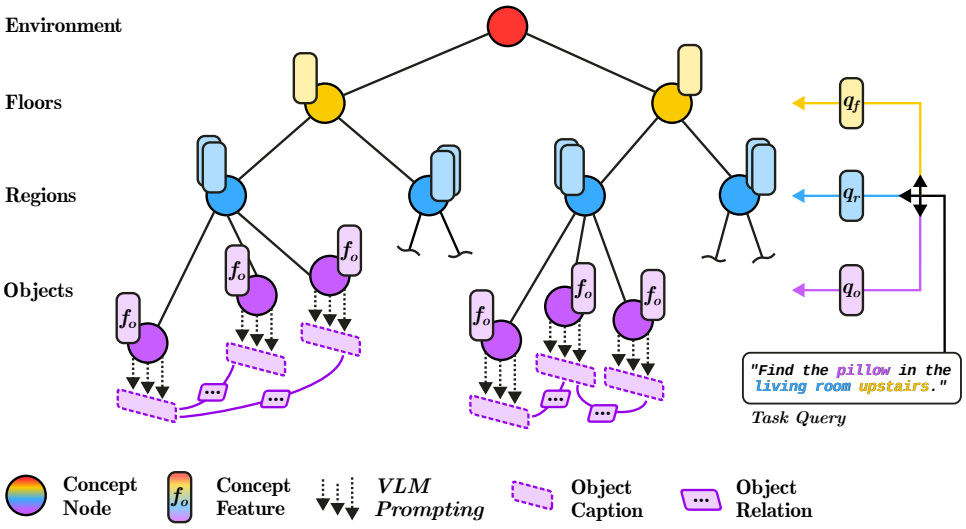

Figure 17.5 Open-vocabulary 3D scene graphs including hierarchical environment partitioning [1179] (HOV-SG) and language-level object-object relations [415] (ConceptGraphs).

Hierarchical Scene Graphs. Orthogonal to the object-object relations discussed before, multiple works have investigated the partitioning of hierarchical scenes while relying on open-vocabulary semantics [1179, 476, 1038]. Following the definition of hierarchical graphs introduced in Section 16.4, the observed environment is sparsely modeled as a graph $\mathcal{G} = (\mathsf{V}, \mathsf{E})$ that is hierarchically partitioned into l ordered concept layers. This yields a set of nodes $\mathsf{V} = \cup_{i \in 1:l} \mathsf{V}_i$, which can come in various forms depending on the target environment. While HOV-SG [1179] separates indoor environments into floors, regions, objects, and Voronoi nodes covering navigable free space, Clio [731] models regions, places, and objects. Other works adapt to typical urban outdoor environments that are decomposed into, *e.g.,* roads, lanes, and static or dynamic objects [1038, 271].

In the following, we discuss typical indoor scenes and follow the approach introduced by HOV-SG. We assume the following map factorization employing pair-wise disjoint sets of nodes and edges, respectively:

$$\mathcal{V} = \mathsf{V}_E \cup \mathsf{V}_F \cup \mathsf{V}_R \cup \mathsf{V}_O \qquad \mathsf{E} = \mathsf{E}_{EF} \cup \mathsf{E}_{FR} \cup \mathsf{E}_{RO}, \tag{17.5}$$

where V_E represents a single root node and V_F, V_R, V_O denote the floor, region, and

object nodes. Thus, we extend the prior object factorization of ConceptGraphs [415] to multiple conceptual layers. Accordingly, the set of edges E_{EF} connects floors with the root node, E_{FR} floors with regions, and E_{RO} regions with objects, respectively. This layering enables hierarchical relation identification, smaller memory footprints, and fast graph search. HOV-SG supports language grounding of all concepts, ranging from floors to objects, by equipping each conceptual node with at least one open-vocabulary concept feature that is attained using CLIP. This is visualized in Fig. 17.5 and detailed in the following.

Similar to the setting of ConceptGraphs, HOV-SG performs incremental RGB-D mapping and class-agnostic instance prediction while assuming access to accurate camera poses. This produces a point cloud that is separated into multiple floors based on binning observed points along the gravitationally-aligned height axis of the scene and subsequent peak identification. In order to refer to the obtained floors via language, each floor gets assigned a templated CLIP text feature of the form "*floor* {*X*}" or "*upstairs*".

Next, we derive region estimates that satisfy geometric constraints and exhibit semantic coherence. Downstream tasks involving navigation require region primitives that adhere to spatial continuity and structural boundaries represented by, *e.g.,* walls or doors [476]. In contrast, semantic higher-level reasoning policies favor semantically consistent grounding of regions. A typical corner case of this is a combined kitchen and living room area covering large extents but sharing a significant number of semantically-distinct sub-regions. While HOV-SG employs a pure geometric approach at region segmentation by thresholding Euclidean distance fields per floor, others cluster navigational nodes obtained through a Generalized Voronoi Diagram (GVD) given a task instruction [731], or harness doors as logical indicators of region boundaries [476].

To enrich region concept nodes with open-vocabulary features, we obtain the camera views of all camera poses falling into the interior set of a region segment, respectively. For each assigned camera view, we obtain the corresponding image-level CLIP embedding. In order to represent a distinct region, we distill the set of camera views into K representatives $\{\boldsymbol{f}_{r,k}\}_{k=1}^{K}$ using k-means clustering. We identify region categories by scoring those representatives against a putative set of candidate region categories C_R that are encoded via CLIP, yielding $\{\boldsymbol{f}_{R,c}\}_{c=1}^{|\mathsf{C}_R|}$. A typical set could include the following regions categories: {"*kitchen*", "*bathroom*", "*living room*", "*corridor*"}. We obtain category votes for each representative $\boldsymbol{f}_{r_k}$ through dot-product scoring:

$$c_{r_k}^{*} = \arg\max_{c \in \mathsf{C}_R} \langle \boldsymbol{f}_{r_k}, \boldsymbol{f}_{R,c} \rangle, \tag{17.6}$$

where $c_{r_k}^{*}$ represents the highest-scoring region category per region representative $\boldsymbol{f}_{r_k}$. By computing the majority vote among all representatives, a single region category is identified.

Furthermore, HOV-SG demonstrates natural language-based navigation within large-scale environments by hierarchically scoring against concept layers. Given a complex text query such as "*go to the plant in the living room on the first floor*", a generative foundation model (GPT) decomposes the long query into multiple distinct captions based on the concepts imposed during the mapping stage. In turn, those concepts ("*plant*", "*living room*", "*first floor*") are encoded to match the CLIP feature space of the constructed scene graph. As depicted in Fig. 17.5, the concepts are progressively scored against all hierarchical layers, going from *higher* to *lower* concepts, *e.g.,* floors to regions to objects. In the case of regions, this procedure allows queries against a set of region representatives $\{\boldsymbol{f}_{r,k}\}_{k=1}^{K}$ instead of queries against a single text embedding denoting the region category, which ultimately fosters robust region retrieval.

Pose Maps. While we adopted an object-centric perspective in this section, we note that a number of related methods instead store open-vocabulary semantic information at the level of camera poses or viewpoints [548, 1090, 1270]. This results in sparse spatial representations that can be leveraged for room segmentation and loop closure detections [548] and relocalization [1090]. Querying the map features returns the most relevant viewpoints and can be used to specify navigation goals. The semantic and spatial overlaps between different viewpoints can localize specific object instances [1270], enabling object querying capabilities comparable to those of explicit object maps.

17.3.3 Implicit Functions

In Chapter 14 we saw the use of radiance fields for creating photorealistic and geometric map representations using Neural Radiance Field (NeRF) [770] and with 3D Gaussian Splatting (3DGS) [560]. Numerous works have extended these to include open-set semantics. In general, most of the NeRF-based methods are trained to add language features to NeRF's implicit radiance field map and methods using Gaussian Splatting assign embeddings to the Gaussians which can then be rasterized to the image plane. Since storing unique language embeddings to each Gaussian can lead to high memory usage (CLIP's ViT-L/14 model for example uses language embeddings of size 768), it is common practice to either learn a compressed CLIP embedding or group Gaussians and assign a single CLIP vector to the group. We will look at LERF [563] and LangSplat [902] as examples of NeRF and 3DGS based approaches, respectively.

LERF. LERF adds an output to NeRF for a semantic embedding given a point $\mathbf{x}$ and a scale s, which is the width of a cube centered at $\mathbf{x}$. View-dependency is omitted to enforce consistent output at different viewing directions. The embedding is rendered similarly to color in NeRF as covered in Chapter 14, with a frustum along

each ray increasing proportionally from the initial scale s_{img} in the image to define the volume for rendering the semantic embedding, as shown in Fig. 17.6. During training, to get CLIP embeddings for training supervision at multiple resolutions per image, LERF precomputes CLIP embeddings at multiple randomly sampled scales of image crops (i.e., multiple values for s_{img}) for a given training image, resulting in an image pyramid of semantic embeddings. During training, rays are assigned ground truth embeddings at randomly selected scales in the pyramid for supervision. To produce crisper semantics along boundaries of potential objects, LERF uses supervision from DINO [160] embeddings. We will look at LERF [563] and LangSplat [902] as examples of NeRF and 3DGS based approaches respectively.

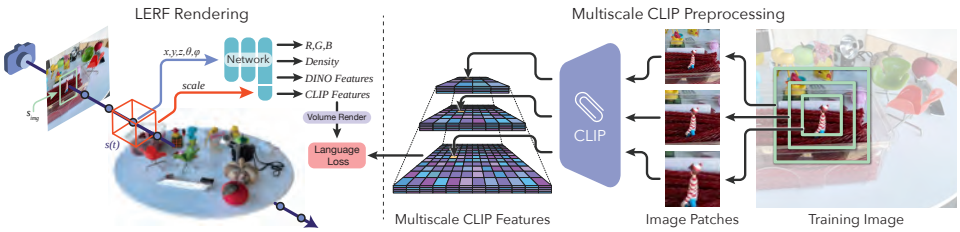

Figure 17.6 Reproduced from [563]. LERF [563] trains a neural radiance field to render CLIP features. To get supervised semantic embeddings at multiple resolutions to be used during training, LERF precomputes a pyramid of embeddings by passing image crops at multiple scales to CLIP.

When querying a rendered image for similarity to a text embedding f_{text}, LERF computes a relevancy score to each language embedding f_{render} in the image as

$$\min_i \frac{\exp(\langle f_{\text{render}}, f_{\text{text}} \rangle)}{\exp(\langle f_{\text{render}}, f^i_{\text{canonical}} \rangle) + \exp(\langle f_{\text{render}}, f_{\text{text}} \rangle)}, \tag{17.7}$$

where $f^i_{\text{canonical}}$ are embeddings from a set of canonical phrases such as "object", "things", "stuff", and "texture." Using the canonical phrases is intended to help denoise the relevancy score.

LangSplat. While LeRF computes CLIP vectors for multiple-sized image crops across an image pyramid to get semantic embeddings at multiple resolutions, LangSplat [902] computes CLIP vectors of class-agnostic image segments obtained from SAM at three levels of granularity. Not only does this help capture local and global context, but it also induces crispness into the map of language embeddings, which makes it unnecessary to additionally use DINO supervision as in LERF.

Each 3D Gaussian is assigned an added attribute for each of the three levels of language embeddings. As assigning CLIP vectors to each Gaussian greatly increases memory requirements (LangSplat uses a CLIP model which has embeddings of size

512), LangSplat trains a scene-specific network that compresses CLIP vectors for a particular scene from their original size of 512 to size 3, and trains a decoder that recovers the original embedding. The intuition is that the semantic variation of a typical scene is much smaller than that of CLIP's training corpus, and thus a smaller size embedding vector is sufficient. The 3D Gaussians then use compressed embedding vectors at the three levels of granularity as training supervision. Semantic rendering is performed as in the RGB case to splat the compressed embeddings at the three granularity levels to the image plane, and is then upscaled to the full-size embeddings. Querying is done using the same relevancy score as LERF, which in the case of LangSplat returns three scores (one for each granularity level) and the level with the highest score is selected when locating an object of interest.

17.3.4 Task-Driven Applications of Foundational Models

While open-set mapping can enable a broader range of concepts with richer descriptions than closed-set mapping, it brings up an interesting question of how to define objects. In the closed-set scenario, objects are defined by a list of labels such as "tool" and "cup". In the open-set scenario, there is ambiguity of **what is the correct granularity to define objects.** For example, consider making an open-set semantic map of a scene consisting of a piano in an otherwise empty room. One possible map could have the piano as a single object. If the task of the agent using the map is to move the piano, this would be an appropriate granularity. However, if the agent's task is to play the piano, the scene must be mapped as over 90 objects — considering each key and pedal as a separate object. An agent tasked with tuning the piano would need to consider the scene as hundreds of objects — considering the strings, tuning pins, and so forth. Likewise, if a forest should be represented as a single area of landscape or as branches, leaves, trunks, etc., remains ill-posed until we specify the tasks that the representation has to support. Most practitioners would agree that the correct scene representation depends on the tasks an agent must complete [739, 1025], but until the recent availability of vision-language foundation models, it has been unclear how to incorporate knowledge of tasks in practice.

We have previously looked at examples of selecting objects for an object-set map which typically assume objects can be defined by clustering regions based on semantic or geometric similarity (see for example, ConceptGraphs in Sec. 17.3.2). While this can approximate the correct object granularity for some scenes and applications, to more generally ensure the correct open-set map representation is generated, Clio [731] formally introduces the problem statement of **task-driven open-set mapping** as follows: given a set of tasks in natural language such as "clean backpacks" and "get condiment packets", construct a map with objects at the correct granularity to support the tasks.

Clio (Fig. 17.7) shows that the problem statement can be formalized mathemat-

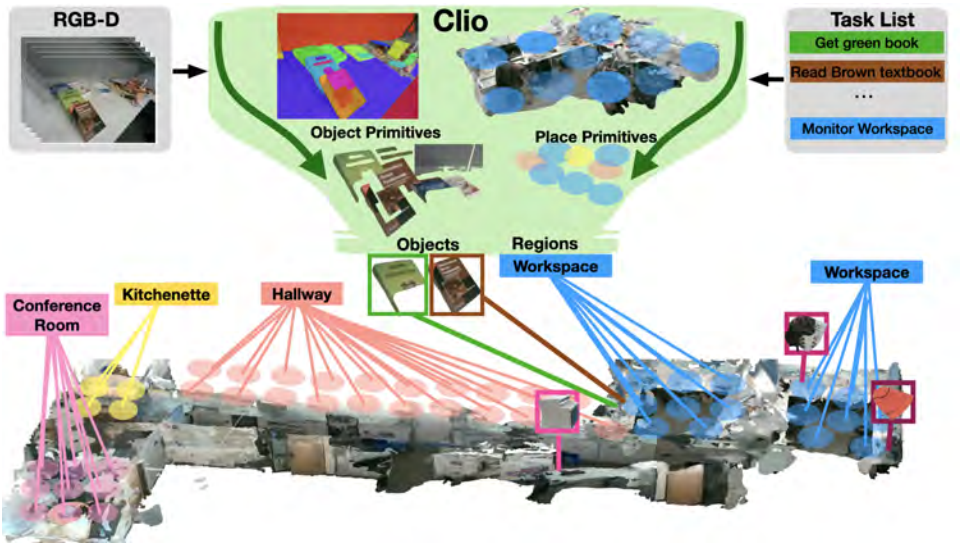

Figure 17.7 Reproduced from [731] (©2024 IEEE). Clio generates an open-set task-driven 3D scene graph given RGB-D images and a list of tasks in natural language. Clio clusters object primitives (derived from associating image segments in 3D) into task-relevant objects at the right granularity to support the task list. Likewise, the idea extends up the scene hierarchy by clustering place primitives (representing regions of free space) into task-relevant regions such as rooms.

ically using the classical Information Bottleneck theory [1102] where given a set of overly-fine primitive observations X (for example, in an extremely fine sense this could be the set all pixels observed from every visual observation), the objective is to form a compressed scene representation $\tilde{X}$ (i.e., the task-relevant objects) that preserves the necessary information needed to complete desired tasks Y.

Writing soft assignments in terms of probabilities $p(\tilde{x}|x)$ (*i.e.,* the probability that a primitive $x \in X$ is clustered into an object $\tilde{x} \in \tilde{X}$), the task-driven clustering problem can be written using the Information Bottleneck as

$$\min_{p(\tilde{x}|x)} I(X;\tilde{X}) - \beta I(\tilde{X};Y), \tag{17.8}$$

where β is a parameter that controls the trade-off between compression and preserving information about the tasks and I is the mutual information.

The Information Bottleneck requires defining $p(y|x)$, which in the case of task-driven mapping, describes the relevance between a primitive x and one of the given tasks y. To make the problem more tractable, Clio defines X as being a set of over-segmented 3D object primitives which are created by projecting class agnostic image segments, $\texttt{Seg}(I)$ to 3D meshes and tracking them throughout frames. The primitives are assigned semantic embedding vectors by aggregating (through

averaging) CLIP embedding vectors computed for each segment associated to the primitive. Clio then uses the cosine score between image and text embeddings to inform an approximation of $p(y|x)$.

The optimization problem (17.8) can be solved incrementally using the Agglomerative Information Bottleneck algorithm [1023], which defines a graph where nodes correspond to primitives, and edges connect nearby primitives, and then gradually clusters primitives together. This produces a hard clustering assignment $p(\tilde{x}|x)$. More in detail, each edge (i, j) is assigned a weight, d_{ij} which measures the amount of distortion in information caused by merging primitive clusters $\tilde{x}_i$ and $\tilde{x}_j$ as:

$$d_{ij} = (p(\tilde{x}_i) + p(\tilde{x}_j)) \cdot D_{\text{JS}}[p(y|\tilde{x}_i), p(y|\tilde{x}_j)], \tag{17.9}$$

where $p(\tilde{x}_i)$ and $p(\tilde{x}_j)$ are computed by the algorithm (and intuitively store the number of primitives merged in clusters i and j), and D_{JS} is the Jensen-Shannon divergence. At each iteration, edges are merged greedily based on their weight.

Clio extends the idea of task-driven mapping up the scene hierarchy by also clustering 3D regions of free space (referred to as places primitives) in the scene into task-relevant regions such as rooms, creating a hierarchical 3D scene graph with layers of object primitives, objects, place primitives, and regions.

While the mathematical representation in (17.8) is a general way to frame the task-driven mapping objective, in practice, solving it by approximating the information between image and text (i.e., estimating $p(y|x)$) and determining how to aggregate semantic knowledge across views are potential sources of error that can lead to incorrect representations. These two issues, which are currently active areas of research, are discussed and provided with a more probabilistically driven solution in Bayesian Fields [730]. Bayesian Fields also uses a similar Agglomerative Information Bottleneck method as Clio, but instead of coarse meshes, uses a Gaussian Splatting map, where the task relevant objects, X, are groupings of 3D Gaussians which form high-resolution, photorealistic objects.

The current examples of task-driven mapping we have discussed require that the tasks be supplied in specific language such as "clean backpacks", but looking ahead, task-driven mapping can be more broadly thought of as robots on-the-fly determining how to represent relevant information of a scene that will be required to complete their broader mission objective such as keeping a home clean or monitoring a factory.

17.4 Future Trends

In this section, we will try to project into the future, a difficult task given how rapidly things are evolving in this space.

17.4.1 Grounding Foundation Models with Maps

We saw in Section 17.2 how LLMs can be prompted to achieve a wide array of tasks conditioned on any textual information included in their prompts. Their rich prior knowledge of the world and reasoning abilities hold great promise for spatial AI, and there is significant interest in grounding LLMs in the physical world, potentially to deploy them as agents. This gives rise to a natural question that may guide research in this space in the coming years: *how can we fully integrate LLMs and map representations?* Here we highlight two other ways in which foundation models and map representations may become more integrated.

Map Prompting. Directly providing the map in text form to an LLM is sometimes possible. We previously saw an example of this approach with 3D scene graphs. When objects and edges are described with natural language by a VLM, a scene graph can be naturally represented as a structured text file (e.g., JSON or YAML) and added to the prompt [415, 733]. Some sparse geometric descriptions of objects (e.g., centroid position, size, bounding box coordinates) can, in principle, be included but how well LLMs can effectively leverage this information is unclear [733]. Such a modular system, where a VLM-based map is passed in text form to an LLM, can be interpreted as a Socratic model [8], a modular framework in which pretrained models can be composed.

Map API. Textual map prompting fails to convey detailed geometric information about the world and is restricted to maps with a sparse, text-compatible structure. An alternative is to treat the map as a completely separate module and expose some functionalities through a symbolic map API. Examples of such functions could include queries such as `exists(object)`, `where_is(object)` or `distance(object_A, object_B)` where inputs are described in natural language and the actual function implementations rely on the query mechanisms we discussed in Section 17.3. Map function calls can be interleaved with LLM calls and robot API calls to build complex and capable systems. As an example, NLMap [184] proposes a planning framework where an LLM breaks down user queries into different proposed objects and verifies object existence and location by querying an open-vocabulary map. The map API can also be leveraged for "tool calling", a framework in which an LLM is free to call external functions and use their outputs during answer generation. LLMs can query open-vocabulary maps to generate robot plans [1270] or answer diverse scene queries expressed in natural language [520, 480].

17.4.2 Revisiting the Question of the Need for Maps

With such impressive visual perceptual representations learned from internet-scale datasets, it is reasonable to revisit one of the first questions from the Prelude to this

SLAM Handbook - the question of the need for SLAM maps, or at least explicit map representations. In this section, we will detail some ways in which explicit maps could, in some cases, be replaced directly by foundation models.

Long-context VLMs/ Models such as Gemini and GPT4+ [1082, 12] can directly process sequences of RGB images that we would otherwise use to build a map, although to what extent long-context VLMs have a coherent understanding of the spatial structure underlying the images is unclear. OpenEQA [733] benchmarks various integrations of VLMs and scene graphs for embodied question answering tasks. They find that directly prompting GPT-4V with 50 frames outperforms using an explicit scene graph prompt on their benchmark. Mobility VLA [209] explores long-context VLMs in the context of navigation: it prompts a VLM with a full office tour (948 frames) and asks it to identify a high-level goal image given a user query. The goal image is then forwarded to a map-based navigation pipeline. The authors experimented with a map-free variant of their system, where the VLM was tasked to output navigation actions directly given the current observation and the office tour, and found this to be ineffective. Based on these results and others, we argue that, at present, *the need for an explicit map representation in the context of LLM or VLM-based physical agents appears to largely depend on the spatial and temporal horizons of the considered tasks* and remains an active area of research.

Physics or geometry-aware VLMs. Some other approaches have attempted to enhance foundation models with an awareness of geometric and physical properties of the environment, something that has been demonstrated to be lacking out-of-the-box for most VLM models [52, 1166]. For example, [183] takes this perspective that the sensor is not *embodied* in the sense that it can be actuated or controlled and is only considering a single view. Nevertheless, these enhanced models have shown an impressive ability to reason about 3D space from a single camera image. However, while the performance on benchmarks such as OpenEQA is impressive, at present there is still an important gap between this and the level of embodied intelligence that we have seen demonstrated with the inclusion of SLAM in the pipeline. It is possible that this gap could be filled through other learning-based paradigms, such as imitation or reinforcement learning, a subject we will cover more thoroughly in the next section.

17.4.3 A Foundation Model for Robotics?

Language and 2D vision foundation models have proved comparatively straightforward to build due to the availability of internet-scale data that can be used for training. Given the incredible advances that have ensued as a result of the representational power of these models, it is natural to consider if we can find a way to generate enough data to build foundation models for other domains. One great

example is Dust3R [1162] discussed in Chapter 13, which has already had a significant impact on the field of 3D vision. But the question remains as to whether this is directly possible for robotics, and, if so, does this cause us to revisit the question of the need for explicit maps. Therefore, to close this chapter, we consider a recent effort in the robotics community to explicitly build a foundation model for robots.

The objective of such a robotics foundation model is to serve as a *base generalist policy* that can perform a diversity of skills across a variety of robot embodiments. The policy would have to learn how to somehow *implicitly* represent the world and then leverage this representation to execute tasks. The data therefore should be of the form of input-output pairs (sensor data and associated actions). As noted in Section 17.1, creating foundation models requires a massive amount of data, and this is preclusively difficult on real robot hardware. Nevertheless, there have been some efforts to collect this scale of data on both real hardware and using simulation data.

In the next sections, we will briefly overview some of the recent efforts at collecting such large-scale robotics datasets, and then we will subsequently overview some of the models that have been built to consume these datasets with the objective of producing generally capable robot agents that do not require explicit world representations.

Datasets and Benchmarks. RT1: Robotics Transformer[124] was an early dataset effort comprising 130k robot trajectories of a fleet of 13 robots performing 700 different tasks over a period of 17 months. The "BridgeData V2" [1146] further enhanced the diversity of tasks compared to RT1, including over 50k demonstrations of 13 skills over 24 environments. Although the hardware is fixed, there is an effort to randomize the external camera locations to provide robustness. The DROID dataset [567] was a multi-institutional effort to create a diverse dataset of robot interaction data that could be used primarily to train manipulation policies. All of these datasets and several others were integrated to create the Open-X-Embodiment dataset [234], which comprises over 1M episodes of 22 different robot embodiments performing 527 different "skills" in diverse environments (34 different university labs).

An important factor in scaling up the collection of data has been to build efficient and intuitive APIs for humans to provide demonstrations, often together with associated high-fidelity simulations. For example, DROID [567] leveraged a VR headset for teleoperation. The ALOHA [1081] setup is relatively low-cost and comprises two sets of synchronized manipulators so that humans can provide demonstrations that will be mimicked directly on the robot hardware. The "Universal Manipulation Interface" [208] is a low-cost handheld gripper setup that exactly matches a robot gripper and eye-in-hand sensor configuration and therefore can be easily used to collect robot demonstrations in a flexible and low-cost manner.

For the most part, The data in these aforementioned datasets takes the form

of trajectories of sensor action pairs $\mathcal{D} = \{z_t, u_t\}_{t=1}^{T}$ where T is the length of the trajectory. As such, they are really primarily targeting the learning of robot skills, such as grasping and manipulation, across a diversity of operating conditions and robot configurations. However, the objective is that complex tasks can be achieved either through increasingly complex demonstrations (combined with language conditioning), or by chaining together shorter demonstrations of sub-tasks.

Model Categories. There are several modeling paradigms that have been developed to *consume* the datasets presented in the previous section with the objective of producing *generalist* agents. These agents should be able to perform a wide array of abstractly-specified tasks in a variety of environments and on a diversity of robot configurations/morphologies.

A simple and intuitive approach to learn a behavior model from data is simply to perform behaviour cloning (BC): train a model in a supervised fashion to learn the mapping from observations to actions [888]. However, this approach is known to be brittle and prone to divergence since little sub-optimal data is contained in the demonstrations and small deviations in action from the demonstration lead to the model becoming out of distribution [950]. The "behaviour transformer" model has shown initial promise in being able to overcome some of these challenges [998], even when trained and evaluated exclusively in simulation.

As a result, the majority of the models trained on these large-scale datasets, which form the "robotics foundation model" candidates, follow a similar structure as the foundation models introduced in Sec. 17.2, except now the output tokens of the transformers are interpreted *directly as control actions*, referred to generally as "Vision Language Action" (VLA) models, as shown in Figure 17.8.

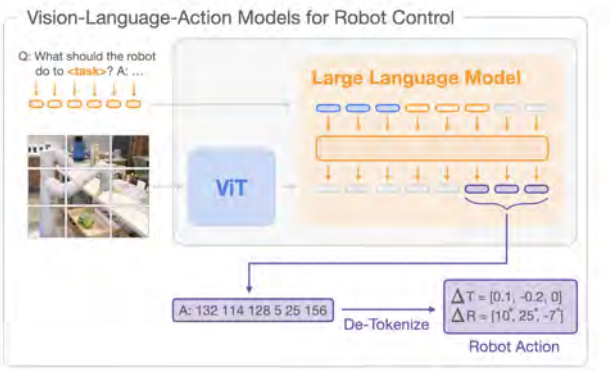

Figure 17.8 Figure reproduced from [125]. In a "Vision Language Action" model, a transformer is modified to directly output robot actions.

Perhaps the first such example was GATO [927], in which they showed that one

generalist agent can potentially perform a huge diversity of tasks, including robot control, with a single model by tokenizing the input and output. Subsequently, PaLM-E [289], built an "embodied multimodal language model," which was trained on vision-language data as well as robot manipulation data, to perform a diversity of language, motion planning, and manipulation tasks directly. RT1 [124] uses a FiLM image encoder rather than a ViT as was the case with PaLM-E, but otherwise the structure of the models is similar. RT2 [125] takes the data from PaLM-E and RT1 and combines them. RT1-X and RT2-X are variants of RT1 and RT2 trained on the Open-X-Embodiment dataset [234].

A variant of the standard VLA model is the "Action Chunking Transformer" (ACT) [1081]. Specifically designed to handle fine manipulation challenges with the ALOHA low-cost data collection platform, the ACT model outputs a sequence of actions rather than just an individual action (as was done in RT1), and has subsequently been shown to be an important modification to the original VLA transformer achitecture to obtain better generalization performance [91].

Perhaps unsurprisingly, gains are potentially achievable by further integrating the representational power of existing VLMs and LLMs that we introduced in Section 17.2 into the language and vision encoders of these generalist policies. For example, The OpenVLA model is built on entirely open-source models, including VLM (DINOV2) and LLM (Llama 2) components, and, after finetuning for (15 days on a 64 A100 GPU cluster) produces a very performant model for the task of robot manipulation as compared to models trained on the Open-X-Embodiment dataset [581] alone. A similar approach was taken in the training of the π_0 model [91].

A somewhat different paradigm for obtaining generalist robot policies is through the use of diffusion models [471]. A diffusion model is a type of generative model that learns to create data by reversing a gradual noising process. During training, it takes clean data and adds noise to it incrementally until it becomes pure noise. The model then learns how to reverse this process, denoising the noisy input in stages to recover the original data. When generating new samples, it starts from random noise and iteratively refines it to generate a sample from the training dataset. This process has been successfully applied to generate robot actions through a "diffusion policy" by representing a visuomotor policy as a conditional denoising process [207], and is currently a very active area of research.

It should also be noted that these two paradigms are not necessarily mutually exclusive. Octo, for example, leverages a large transformer for producing outputs which are fed to an "action head" that runs a forward diffusion process [824].

Reinforcement learning (RL) is also an increasingly popular paradigm for training robot policies. However, since the large datasets mentioned above do not contain an explicit notion of *rewards*, the reward would need to be inferred or learned as in an inverse RL setup. RL methods are particularly amenable to training in simulation where reward functions can be easily defined and evaluated, and where rollouts are relatively inexpensive to perform compared to real hardware. RL also has the

potential to act as a fine-tuning mechanism that acts on the policies generated by the previously mentioned VLA or diffusion methods.

This section is by no means meant to be an exhaustive list of the state of the art of training generalist policies. This field is rapidly evolving with new, more capable models being released very frequently. As the capacity of these models grows, their ability to implicitly encode some representation of their environment also seems to improve, and may modify in what contexts we still need an explicit map representation.

17.4.4 Concluding Remarks

It is undeniable that foundation models of all kinds have transformed robotics in a similar manner to many other fields. It remains to be seen, however, to what degree this progress continues. On the one hand, it seems conceivable that simply further scaling the robotics datasets and dedicated foundation models combined with internet-scale VLMs and LLMs could produce increasingly impressive capabilities that may render explicit map representations obsolescent. However, it has been demonstrated that transformers in particular struggle to perform compositional or chain-of-thought reasoning [305]. For this reason, we are seeing the re-emergence of neural memory mechanisms, such as "structured state space models" [414], which are specifically designed to have a notion of statefulness, in much the same way as recurrent neural networks. But the limits of these neural memory architectures and how they become integrated with generalist policies are still unclear.

Equally as likely (or perhaps more so) at this time is that true generalization and scalability to compositional tasks in large and complex environments could be achieved through some form of explicit structure that is learned through a process such as SLAM. One view could be that, in fact, these two paradigms (explicit world modeling through SLAM and planning vs. generalist robotics policies) are entirely complementary. Perceiving, representing, and understanding the world is a prerequisite for taking action (at least in the types of complex problems in which we are interested), and this type of explicit world modeling should only help our ability to perform complex actions robustly.