Table 4: **Comparison of ObjectNav methods.** Open-world methods are not limited to a closed set of object categories. Zero-shot methods do not use `ObjectNav` annotations for training.

Method	Open-World	Zero-Shot
Fully-Supervised Methods [10–13, 16]	✗	✗
EmbCLIP [25]	✓	✗
ZER [18]	✗	✓
CoW [21]	✓	✓
ZSON (ours)	✓	✓

A Extended Discussion of Related Work

In Table 4, we compare object-goal navigation (`ObjectNav`) methods along two dimensions: *open-world* and *zero-shot*. Open-world methods are not restricted to object categories from a closed predetermined vocabulary (e.g., *"chair"*, *"bed"*, *"sofa"*, etc.). Zero-shot methods do not use `ObjectNav` annotations (e.g., labeled environments or large-scale human demonstrations [10]) for training.

Traditional, fully-supervised `ObjectNav` methods [10–13, 16] rely on a closed-world assumption and task-specific training data – i.e., they are neither open-world nor zero-shot. EmbCLIP [25] can be used for open-world `ObjectNav` because it pre-processes object-goals (e.g., *"chair"*) with a CLIP [19] text encoder CLIP_t. Thus, the EmbCLIP interface allows describing objects using the open-vocabulary supported by CLIP_t. However, EmbCLIP is trained directly with `ObjectNav` annotations based on labeled 3D environments. Consequently, EmbCLIP is not a zero-shot method.

The method proposed in [18] (ZER) is zero-shot because it uses the image-goal navigation (`ImageNav`) task for training. However, ZER cannot perform open-world `ObjectNav` because it relies on a mapping from a closed-set of object categories into the image-goal embedding space for transfer to object-goal navigation. CoW [21] is a zero-shot method that does not require training a navigation policy. Instead, CoW uses a heuristically defined policy that has no ability to learn about indoor layouts of home environments (e.g., the fact that *"stoves"* are found in *"kitchens"* as illustrated in Fig. 3). However, CoW is able to perform open-world `ObjectNav` through the use of CLIP visual and text encoders.

By contrast, our approach (ZSON) uses CLIP to project image-goals and object-goals into a common semantic-goal embedding space, which converts `ImageNav` and `ObjectNav` into semantic-goal navigation (`SemanticNav`). This enables training with semantic-goals derived from images, followed by *zero-shot* transfer to *open-world* `ObjectNav`.

An interesting direction for future work might be to train `SemanticNav` agents with multi-task training using semantic-goals derived from both image- and object-goals. Such a solution would not be zero-shot. However, it might combined the advantages of large-scale training with image-goals (as used in our approach) with the advantages of smaller-scale task-specific training with object-goals (as used in EmbCLIP). We present initial results in this direction in Appendix B.

Table 5: **Results of finetuning** with ObjectNav annotations. **indicates reproduced results*

#	Method	Dataset	SPL	SR
1	OVRL [16]	MP3D	7.0%	**25.3%**
2	ZSON (ours)	MP3D	4.8%	15.3%
3	ZSON finetuned 25M steps (ours)	MP3D	**9.2%**	22.9%
4	OVRL [16]	HM3D	12.3%*	32.8%*
5	ZSON (ours)	HM3D	12.6%	25.5%
6	ZSON finetuned 100M steps (ours)	HM3D	**27.0%**	**49.6%**

B Results of ObjectNav Finetuning

In this section, we study the benefits of additional task-specific training by finetuning ZSON agents using `ObjectNav` annotations – i.e., manually labeled training environments. Specifically, we initialize with a `SemanticNav` agent trained for 500M steps of experience in HM3D environments using image-goals. Then, we finetune for ObjectNav in either MP3D [8] or HM3D [20] annotated environments using reinforcement learning (RL) with the finetuning approach from [37].

In Table 5, we find that finetuning ZSON agents results in 7.6% - 24.1% absolute improvements in `ObjectNav` success rates (SR). Specifically, in row 3 we finetune for 25M steps in MP3D environments. This leads to a 7.6% absolute improvement in SR (15.3% $\rightarrow$ 22.9%) and a 4.4% absolute improvement in SPL (4.8% $\rightarrow$ 9.2%). This 9.2% SPL surpasses the state-of-the-art in the RGB-only setting of 7.0% SPL, which was set by OVRL [16] (row 1) using direct supervision from 40k human demonstrations in MP3D environments. Similarly, in row 6, we finetune for 100M steps in HM3D environments. This results in a 24.1% absolute improvement in SR (25.5% $\rightarrow$ 49.6%) and a 14.4% absolute improvement in SPL (12.6% $\rightarrow$ 27.0%). These results exceed the OVRL [16] baseline presented in row 4 (which was trained in MP3D environments and is identical to the agent in row 1) by 16.8% absolute in SR (32.8% vs. 49.6%) and 14.7% absolute in SPL (12.3% vs. 27.0%).

Table 6: **Additional ablations** of the visual encoders used for training our `SemanticNav` agents.

			`ImageNav` (Gibson)		`ObjectNav` (Gibson)	
#	Encoder	Dataset	SPL	SR	SPL	SR
1	ResNet-9 (scratch)	HM3D	23.4%	31.1%	9.5%	20.9%
2	ResNet-50 (scratch)	HM3D	22.4%	28.2%	6.9%	16.6%
3	ResNet-50 (OVRL)	HM3D	**28.0%**	**36.9%**	**12.0%**	**31.3%**

C Additional Ablation Experiments

Table 3 contains ablations of the visual encoder used to process RGB observations within our agent. In Table 6, we provide additional ablations that compare the ResNet-9 and ResNet-50 architectures with and without pretraining – i.e., from scratch vs. using OVRL [16]. We find that without pretraining, switching to the larger ResNet-50 encoder leads to a 2.9% drop in `ImageNav SR` and a 4.3% drop in `ObjectNav SR` (rows 1 vs. 2). By contrast, using OVRL pretraining, performance improves on all tasks across all metrics. For instance, `ImageNav SR` improves by 5.8% and `ObjectNav SR` improves by 10.4% (rows 1 vs. 3).

D Additional Training Details

Table 7 details the default hyperparameters used in all of our training runs. For ablation experiments in Gibson [4] environments we reduce the "number of environments per GPU" to 28 for ResNet-9 experiments and 20 for ResNet-50 due to GPU memory constraints. We use the ResNet-9 implementation from [18] and ResNet-50 implementation from [2].

E Additional Qualitative Results

In Figs. 4 to 11, we provide additional qualitative results. Figs. 4 to 6 show successful navigation to *"stairs"*, *"table"*, and *"television"* (respectively), highlighting the versatility of our ZSON agent.

Figs. 7 to 11 illustrate various failure modes of the learned policy. In Fig. 7, the agent successfully finds a *"refrigerator"* in 4 out of 5 trials. However, in one case the agent stops before entering the kitchen, despite having a view of the *"refrigerator"*. In this case, the failure mode is stopping short.

In Fig. 8, the agent successfully finds a *"bathtub"* when it enters the bathroom nearest to the starting position. However, in 2 of 5 trials it fails to find that bathroom, thus does not find a *"bathtub."* In this case, the failures are due to poor exploration. Similarly, in Fig. 9 the agent does find a seating area

Table 7: **Hyperparameters** used to train `SemanticNav` agents.

Parameter	Value
Number of GPUs	8
Number of environments per GPU	32
Rollout length	64
PPO epochs	2
Number of mini-batches per epoch	2
Optimizer	Adam
Learning rate	2.25×10^{-4}
Weight decay	1.0×10^{-6}
Epsilon	1.0×10^{-5}
PPO clip	0.2
Generalized advantage estimation	True
γ	0.99
τ	0.95
Value loss coefficient	0.5
Max gradient norm	0.2
DDPPO sync fraction	0.6

that resembles a *"sofa"* in 3 of 5 trials. However, it stops in the dining area in 2 trials. Furthermore, it never enters the room to the right, which contains two sofas – again, demonstrating poor exploration.

In Fig. 10 the agent never enters the room to the bottom right, which contains a *"desk,"* thus failing in all five runs. In 3 of 5 trials it stops near a table that does not appear to be functioning as a desk (e.g., there is no chair nearby). In these examples, the agent might be confusing objects that can have a similar appearance. Finally, in Fig. 11, we start the agent from two different positions (A and B) and provide the instruction *"Find a dresser."* When the agent is initialized near the bedroom (A) it is able to find the dresser in 4 out of 5 trials. However, from position B the agent navigates into the kitchen and stops near the cabinets. Again, these failures may be due to similarities in the appearance of dressers and cabinets.

Figure 4: Qualitative example of successful navigation to the *"stairs."* The number of steps taken by our agent over five trials ranges from 80 to 102.

Figure 5: Qualitative example of successful navigation to a *"table."* The number of steps taken by our agent over five trials ranges from 60 to 98.

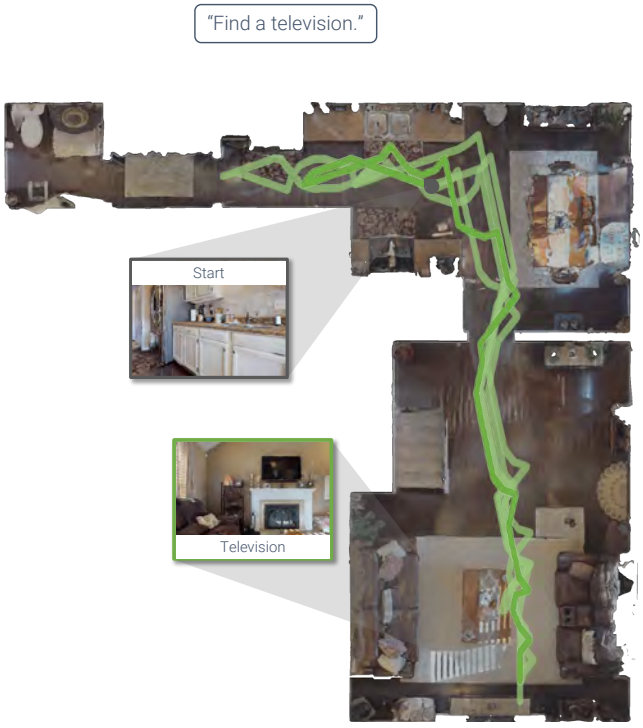

Figure 6: Qualitative example of successful navigation to a *"television."* The number of steps taken by our agent over five trials ranges from 78 to 148.

Figure 7: Qualitative example of *"Find a refrigerator."* The agent succeeds in 4 of 5 trials (green) from the same starting position. In the failure (red) the agent stop short of fridge. The number of steps ranges from 83 to 99.

Figure 8: Qualitative example of *"Find a bathtub."* The agent succeeds in 3 of 5 trials (green) from the same starting position. In the two failures (red) the agent never enters the bathroom. The number of steps ranges from 58 to 114.

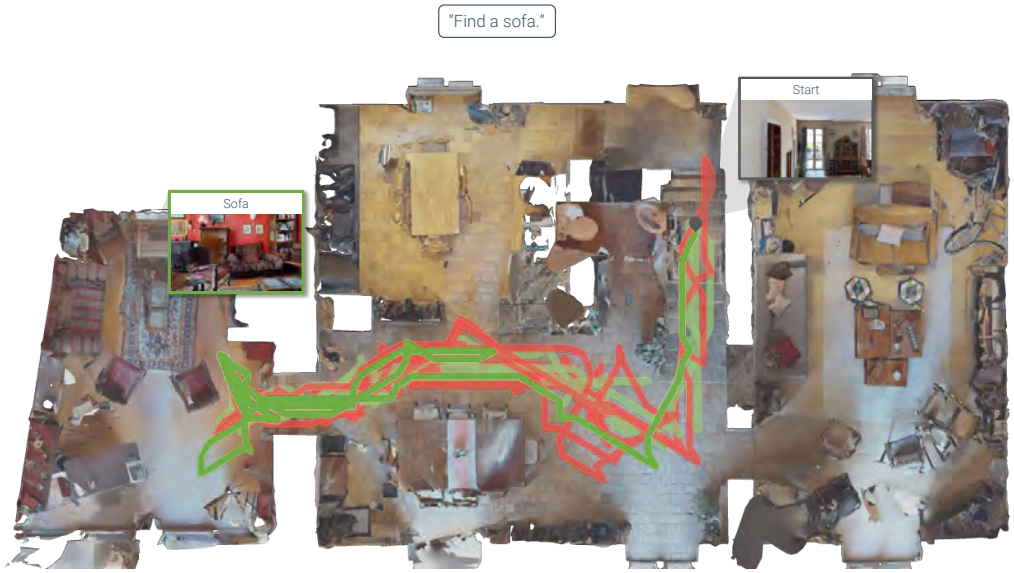

Figure 9: Qualitative example of *"Find a sofa."* The agent succeeds in 3 of 5 trials (green) from the same starting position. In the two failures (red) the agent stops in the dining area (center). The agent never enters the room to the right with two *"sofas"*. The number of steps ranges from 174 to 501.

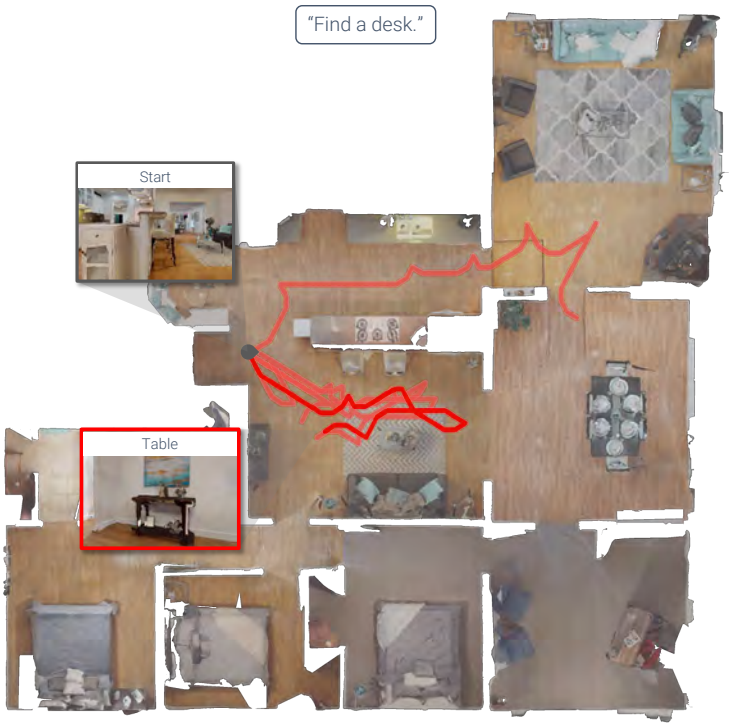

Figure 10: Qualitative example of *"Find a desk."* The agent fails in all five trials (red), stopping at a table in 3 of 5 runs. The agent never enters the room in the bottom right that contains a *"desk"*. The number of steps ranges from 49 to 114.

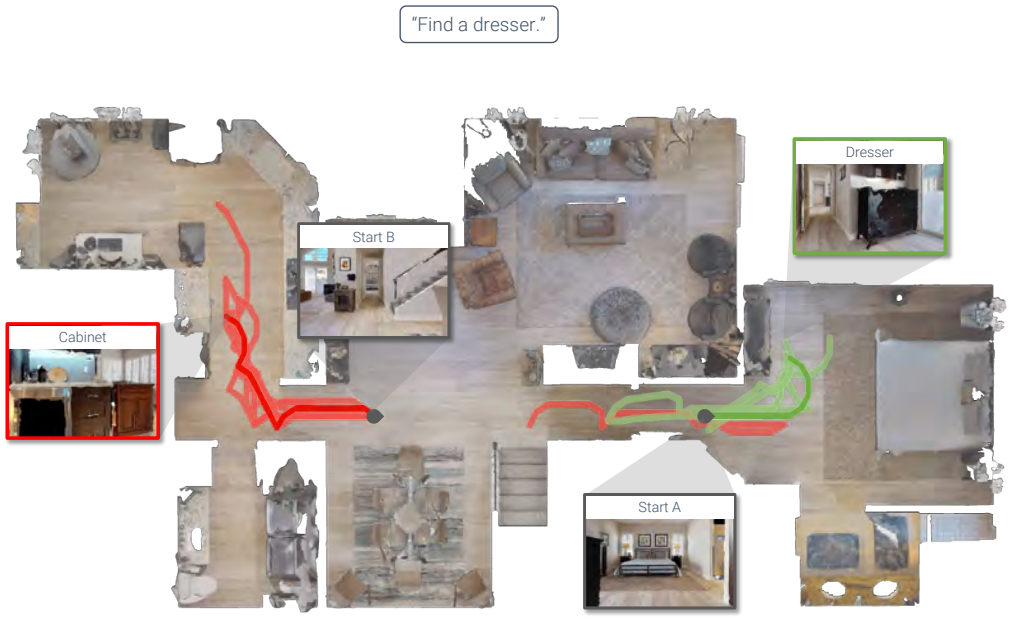

Figure 11: Qualitative example of *"Find a dresser."* We run five trails from two starting positions (A and B). From Start A the agent is able to find a dresser in 4 of 5 trials (green). However, when the starting location is shifted further from the bedroom (Start B) the agent enters the kitchen and fails in all five runs (red), stopping near the kitchen cabinets. These failures might be due to the similarity in appearance between cabinets and dressers. The number of steps ranges from 29 to 109.