A LAMPP FOR SEMANTIC SEGMENTATION

A.1 METHODS

We derive the following decision rule from the model in Fig. 2:

$$p(y_i \mid \underline{x}) \precsim p(y_i \mid d_i = d_i^*)p(d_i = d_i^* \mid x_i)\left(\sum_r p(r)p(y_i \mid r)\prod_{j=1\cdots n}\left(\sum_{y_j}\frac{p(r \mid y_j)p(d_j = y_j \mid x_j)}{p(r)}\right)\right) \tag{5}$$

We obtain this decision rule as described below. Here we denote rooms r, true object labels y, noisy object labels d, and observations x. (Underlines denote sets of variables, so e.g., $\underline{x} = \{x_1, \ldots, x_n\}$.) Finally, we write d_i^* to denote the base model's prediction for each image segment ($d_i^* = \arg\max p_{\text{seg}}(d_i \mid x_i)$). To see this:

$$\begin{aligned}
p(y_i \mid \underline{x}) &\propto \sum_r \sum_{\underline{y}\setminus\{y_i\}} \sum_{\underline{d}} p(\underline{x}, \underline{y}, \underline{d}, r) \\
&= \sum_r \sum_{\underline{y}\setminus\{y_i\}} \sum_{\underline{d}} p(r)\Big(p(y_i \mid r)p(d_i \mid y_i)p(x_i \mid d_i)\Big)\prod_j p(y_j \mid r)p(d_j \mid y_j)p(x_j \mid d_j) \\
&= \sum_r p(r)\Big(p(y_i \mid r)\sum_{d_i} p(d_i \mid y_i)p(x_i \mid d_i)\Big)\Big(\prod_j \sum_{y_j} p(y_j \mid r)\sum_{d_j} p(d_j \mid y_j)p(x_j \mid d_j)\Big)
\end{aligned}$$

Rather than marginalizing over all choices of d, we restrict each sum to a single term. For d_i, we choose the most likely detector output $d_i = d_i^*$. For d_j, we choose the corresponding y_j in the outer sum. Together, these simplifications reduce the total number of unnecessary LM queries about unlikely object confusions, and give a lower bound:

$$\geq p(d_i = d_i^* \mid y_i)p(x \mid d_i = d_i^*)\sum_r p(r)(p(y_i \mid r)\Big(\prod_j \sum_{y_j} p(y_j \mid r)p(d_j = y_j \mid y_j)p(x_j \mid y_j)\Big)$$

Applying Bayes' rule locally:

$$\begin{aligned}
&= \frac{p(y_i \mid d_i = d_i^*)p(d_i = d_i^*)}{p(y_i)}\frac{p(d_i = d_i^* \mid x_i)p(x_i)}{p(d_i = d_i^*)} \\
&\quad \sum_r p(r)p(y_i \mid r)\Big(\prod_i \sum_{y_j} \frac{p(r \mid y_j)p(y_j)}{p(r)}p(d_j = y_j \mid y_j)\frac{p(d_j = y_j \mid x_j)p(x_j)}{p(d_j = y_j)}\Big)
\end{aligned}$$

Finally, we make two modeling assumptions. First, we assume that of the form $p(y)$ and $p(d)$—the marginal distributions of true and noisy object labels—are uniform. This allows us to use LMs as a source of information about object co-occurrence probabilities without relying on their assumptions about base class frequency. Second, for non-target detections x_j, we assume the probability that noisy labels match the true labels is constant over object categories. Then, dropping constant terms gives:

$$\propto p(y_i \mid d_i = d_i^*)p(d_i = d_i^* \mid x_i)\sum_r p(r)p(y_i \mid r)\Big(\prod_j \sum_{y_j} \frac{p(r \mid y_j)}{p(r)}p(d_j = y_j \mid x_j)\Big)$$

A.2 SOCRATIC MODEL BASELINE

We attempt to make the Socratic models inference procedure for this task as analogous to the LAMPP approach as possible: the LM must account for both room-object co-occurrence likelihoods and object-object resemblance likelihoods when predicting true labels. However, here, the LM must

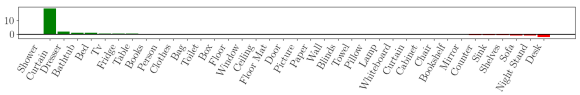
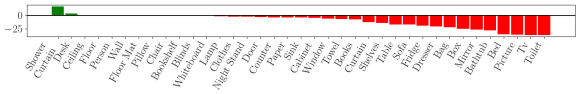
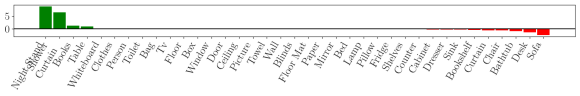
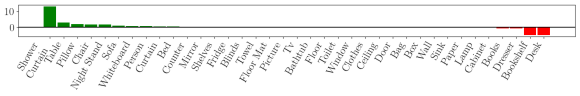
Base Model	mIoU	ΔIoU by Object Category
RedNet (ID)	47.8	
+ LAMPP	48.3	
+ SM	37.5	
RedNet (OOD)	33.8	
+ LAMPP	34.0	
CLIPSeg (ZS)	27.7	
+ LAMPP	28.2	

Table 4: Image Segmentation results. We report mIoU, or intersection-over-union averaged over each object category. We also report the improvements to IoU for each object category when LAMPP or Socratic Models is applied to RedNet trained on in-domain or out-of-domain data, and CLIPSeg applied zero shot.

implicitly incorporate these likelihoods into its text-scoring, rather than integrating them into a structured probabilistic framework.

Specifically, the Socratic model baseline is given model predictions $\widehat{d}_i$ for *each segment* x_i and re-labels each segment by querying GPT-3 with:

> *You can see:* $[\underline{\widehat{d}}]$
>
> *You are in the* $[r]$
> *The thing that looks like* $[\widehat{d}_i]$ *is actually* $[y_i]$.

The LM is given the non-highlighted portions and asked to generate the portions highlighted in yellow. $\underline{\widehat{d}}$ is the set of all unique objects detected by the base model, written out as a comma-separated list. r is a room type generated by the LM based on these objects (inferred by normalizing over possible room types), and y_i is the actual identity of the object corresponding to this segment. We replace all pixels formerly predicted as $\widehat{d}_i$ with y_i.

A.3 FULL RESULTS

See Table 4 for the full results over object categories. In the ID case, note that shower curtain improved dramatically with the incorporation of LAMPP. This is because the base model almost always classifies shower curtains as window curtains, while explicitly incorporating room priors from language models fixes this major class of error. Moreover, in the OOD setting, the base image model sees far fewer examples of nightstands and consequently never predicts nightstands on the test data. (Nightstands are frequently predicted to be tables and cabinets instead). This is likewise rectified with LAMPP: background knowledge from language reduces model sensitivity to a systematic bias in dataset construction

he goal objects that improved the most with the incorporation of LAMPP (shower curtain, nightstand in the OOD case) are all ones that strongly co-occur with specific rooms, indicating the usefulness of incorporating room-object co-occurrence priors.

B LAMPP FOR NAVIGATION

B.1 SOCRATIC MODEL BASELINE

As in the image segmentation case, we have a Socratic model baseline. LM priors are integrated into exploration through directly querying the LM with

> *The house has:* [$\underline{r}$].
> *You want to find a* [g]. *First, go to each* [r_0]. *If not found, go to each* [r_1]. *If not found, go to each* $\cdots$

whereby $\underline{r}$ is a list of all room types in the environment, for example, *3 bathrooms, 1 living room, 1 bedroom*. The LM returns the best room type r_0 to navigate to in order to find g. The agent visits all r_0 in order of proximity. If the object is not found, the LM is queried for the next best room type to visit, etc., until the object is found or we run out of rooms in the environment.

B.2 METHODS

Below we provide intuition for why our policy outlined in Section 4.1 follows from the generative story of the success score y_i depicted in Fig. 3. We assume objects are equally likely to be distributed across all rooms.

At a high level, an agent greedily optimizing for success should move to a room r that maximizes $p(y \mid x, r, g)$ which, following Fig. 3 is equivalent to maximizing:

$$p(y \mid g, r)p(x \mid y) \propto p(y \mid g, r)p(y \mid x)p(x)/p(y) \propto p(y \mid g, r)p(y \mid x) \tag{6}$$

The policy that maximizes Eq. (6) iterates between *navigation* and *selection* actions:

1. *Navigation*: We navigate to the unobserved room with the highest $p(y \mid r)$.[6] This follows directly from Eq. (6) and the assumption of uniform initial object locations.
2. *Selection*: Once we have some observations x of the room, we branch off into two cases:
   (a) *Deciding to continue:* We do not see g (low $p(y \mid x)$). If we take n ($= 25$) steps in the room and do not see any instance of g, we become relatively confident that the room does not contain g at all, so $p(y \mid x) \to 0$ for that room. (Consequently, the objective $p(y \mid x)p(y \mid r)$ also $\to 0$ for this room.) We eliminate this room from the list of unobserved rooms and return to step 1.
   (b) *Deciding to stop:* We see g (high $p(y \mid x)$), either on the way to the room or within the room. We navigate to exact location that maximizes $p(y \mid x)$ and decide to stop based on $p(y \mid x)p(r \mid y)$.

B.3 FULL RESULTS

See Table 5 for the full set of results over goal-object categories. Note that the goal objects that improved the most with the incorporation of LAMPP (TV monitor, sofa, toilet) are all ones that strongly co-occur with specific rooms, indicating the usefulness of incorporating room-object co-occurrence priors.

B.4 UNIFORM PRIORS MODEL VS. ORIGINAL STUBBORN AGENT MODEL

Recall that we modified the original STUBBORN agent in the navigation task to utilize the high-level map, by uniformly sampling a random (unvisited) room to visit. More specifically, the uniform priors model utilizes effectively the same high-level policy as LAMPP-based agent, but replaces LM priors over over object-room co-occurrences with uniform priors:

$$p(y = \text{True} \mid r, g) = \frac{1}{\#\text{ room types in environment}}. \tag{7}$$

[6] For rooms with the same $p(y \mid r)$, or if there are multiple rooms of the same type, we navigate in order of proximity to the agent's current location.

Base Model	SR		SPL		ΔSR by Goal Object Category
	Class	Freq.	Class	Freq.	
Orig STUBBORN	52.7	53.8	21.2	22.8	
Unif Prior (ZS)	52.1	51.7	20.2	21.5	
+ LAMPP	66.5	65.9	35.4	36.0	
+ SM	61.2	65.3	31.3	35.1	

Table 5: Navigation results, showing *class-averaged* success rate (SR; averaged over goal objects) and *frequency-averaged* success rate (averaged over episodes). We also report SPL, Success weighted by (normalized inverse) Path Length. Finally, we report the improvements to success rate over each object category when LAMPP vs. Socratic Models is applied.

Model	Class-Avg. SR	Freq.-Avg. SR
LAMPP	66.5	65.9
$-p(y \mid r)$ during selection	58.8	64.9
Socratic model	61.2	65.3

Table 6: Navigation results verification ablations. We ablate the LM uncertainties over $p(y \mid r)$ when computing the selection action, making LAMPP functionally similar to a Socratic model baseline. We find that having these uncertainties are crucial; without them, LAMPP actually *underperforms* the Socratic model baseline.

Note in the zero-shot case we have no additional information about $p(y \mid r, g)$, so we must assume it is uniform.

We report average success rates over object categories of the original STUBBORN agent vs. a uniform priors agent vs. an LAMPP-based agent vs. a Socratic-model-based agent in Table 5. We find that LAMPP is able to outperform both the original STUBBORN agent and the uniform priors agent.

B.5 Additional Analysis

Why does LAMPP outperform Socratic modeling? In the SM approach, high-level decisions from the LM and low-level decisions from observation models are usually considered separately and delegated to different phases (it is hard to combine these information sources in string-space): in our implementation, the SM baseline uses the top-down LM for *navigation*, and the bottom-up observation model for *selection*.

Because the policy dictated by the LAMPP probabilistic model also ignores bottom-up observation probabilities until the goal object is observed, the *navigation* step of both approaches is functionally equivalent. However, for the *selection* step, we find that combining bottom-up and top-down uncertainties is crucial (recall that LAMPP thresholds $p(y \mid x, r, g)$ at selection steps, which decomposes to $p(y \mid x)p(y \mid r, g)$).

To further understand and how using LM probabilities contributes at this phase, we run a version of LAMPP where we simply change the decision rule at the selection action to $p(y \mid x)$. Results are reported in Table 6. Note that we actually *underperform* the SM baseline when we take away top-down uncertainties $p(y \mid r, g)$ — once again highlighting the importance of combining both sources of uncertainty.

C LAMPP for Video-Action Segmentation

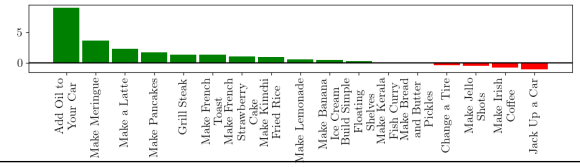
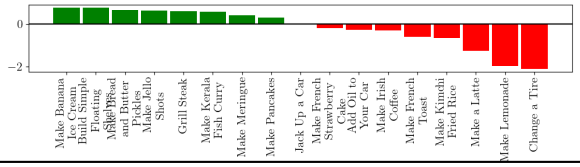
Base Model	Step Recall Class	Step Recall Freq.	Δ Step Recall by Task
HSMM (ZS)	44.4	46.0	
+ LAMPP	45.7	47.9	
HSMM (OOD)	37.6	40.9	
+ LAMPP	38.1	41.2	

Table 7: Action recognition and segmentation results, showing *class-averaged* step recall (averaged over tasks) and *frequency-averaged* step recall (averaged over videos). We also report the improvements to step recall over each object category when LAMPP is applied.

C.1 SELECTING THE LM PROMPT

We performed significant prompt engineering in this setting (on the training set), and found that with all of the prompts we tried, GPT-3 was biased towards outputting actions in the order that they were presented to the model. The following are some examples of other prompts we tried:

> "*Your task is to* $[t]$. *Your actions are:* $\{[Y]\}$
> *The step after* $[y]$ *is* $[y']$*:* [*plausible / implausible*] "

> "*Your task is to* $[t]$. *Your actions are:* $\{[Y]\}$
> *The step after* $[y]$ *is* $[y']$ "

We also tried having the LM produce a global ordering and setting probability from an action later in the sequence to earlier in the sequence to 0:

> "*Your task is to* $[t]$. *Your set of actions is:* $\{[Y]\}$
> *The correct ordering is:*

The prompt in 5.1 was slightly better at combating this effect, though still highly imperfect.

C.2 FULL RESULTS

See Table 7 for the full set of results over actions.

D TRAINING DETAILS

D.1 HYPERPARAMETERS

Since our method builds upon each base model, we used standard hyperparameters to train and run inference for each base model. For the components of LAMPP that we build on top of the base models, there are usually only a few hyperparameters to tune, which we tune based on performance on a held-out development set.

In image segmentation, we tune is a weighting term λ at the final decision layer that trades off the weight on the LM prior and the observation posterior.

$$\log p(y \mid x) = \lambda \log p(y) + (1 - \lambda) \log p(x \mid y)$$

We find he optimal λ to be 0.9. In the navigation case, we tune a threshold τ that determines when the agent stops ($p(y \mid x, r, g) > \tau$) and find the optimal τ is 0.2 For video-action segmentation, we have a hyperparameter α controlling the weight on the prior (see Eq. (4)), which we set to 10.

E Computational Details

We run all experiments on a single 32GB NVIDIA V100 GPU.

For image segmentation experiments, we used either: 1. a pretrained RedNet checkpoint, 2. a RedNet checkpoint trained for 1000 epochs on OOD data, 3. a pretrained zero-shot CLIPSeg model. 1 and 3 required no training, 2 required training for $\sim$ 3-4 days on the GPU. Inference for a single experiment takes several hours each on a single GPU.

For navigation experiments, we used the pre-trained STUBBORN model. A full inference run on our GPU takes about a full day to complete. (However, multiple inference experiments could be run in parallel using the same GPU – the bottleneck is simulating every step of every trajectory, rather than GPU inference.)

For video-action segmentation experiments, we use the HSMM model trained by Fried et al. (2020). On a single GPU, the model typically takes only a few minutes to fully train and a few minutes for inference. Several inference experiments can be run in parallel on the same GPU.

Finally, to derive LM priors, we query the OpenAI API. Using our method uses a budget of $\sim$\$12 (in totality, over *all experiments in all domains*), as it queries a fixed number queries ahead of time ($\sim$4k queries), which get reused over the entire test set. Meanwhile, running full SM experiments requires on the order of several hundred dollars in totality – frequently requiring multiple queries *per test sample*.

F Limitations

While flexible, LAMPP is not applicable to every machine learning task, and specifically requires (1) a formalization of the task as a probabilistic graphical model, in which (2) values of some variables (e.g. labels) can be represented as natural language strings, and (3) probabilistic relations between these variables are described (with some degree of precision) in large-scale text corpora. As shown above, many success cases involve common-sense knowledge about human environments or human task structures; more specialized tasks (e.g. fine-grained image classification or medical diagnosis) may not enjoy the same benefits.