A Derivations

A.1 $p^\star_{\boldsymbol{z}}$ is a valid density

Proposition A.1. *$p^\star_{\boldsymbol{z}}(\xi)$ is a proper probability density function.*

Proof. Recall that

$$p^\star_{\boldsymbol{z}}(\xi) = \frac{\mathbb{E}_{\substack{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\\ \boldsymbol{z}_{1:T}}}\left[\sum_{t=1}^{T}\delta(\boldsymbol{z}_t=\xi)\boldsymbol{m}_t\right]}{\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^{T}\boldsymbol{m}_t\right]}.$$

A function is a probability density function if it is non-negative, continuous, and integrates to 1. By construction, $p^\star_{\boldsymbol{z}}(\xi)$ is non-negative since $\boldsymbol{m}_t \geq 0$ and $\delta(\boldsymbol{z}_t = \xi) \geq 0$. Further, $\delta(\boldsymbol{z}_t = \xi)$ is continuous in ξ so the numerator $\mathbb{E}_{\substack{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\\ \boldsymbol{z}_{1:T}}}\left[\sum_{t=1}^{T}\delta(\boldsymbol{z}_t=\xi)\boldsymbol{m}_t\right]$, an expectation containing $\delta(\boldsymbol{z}_t = \xi)$, is also continuous in ξ. Finally,

$$\begin{aligned}
\int_\xi p^\star_{\boldsymbol{z}}(\xi)\,d\xi &= \frac{1}{\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^{T}\boldsymbol{m}_t\right]}\int_\xi \mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^{T}\delta(\boldsymbol{z}_t=\xi)\boldsymbol{m}_t\right]d\xi \\
&= \frac{1}{\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^{T}\boldsymbol{m}_t\right]}\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^{T}\left(\int_\xi\delta(\boldsymbol{z}_t=\xi)d\xi\right)\boldsymbol{m}_t\right] \quad \text{(Fubini's theorem)} \\
&= \frac{1}{\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^{T}\boldsymbol{m}_t\right]}\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^{T}1\cdot\boldsymbol{m}_t\right] \\
&= 1.
\end{aligned}$$

Therefore, $p^\star_{\boldsymbol{z}}(\xi)$ satisfies all 3 properties of a probability density function. □

Remarks. This result generalizes to the case for discrete $\boldsymbol{z}$ by replacing the delta with an indicator function and integration with summation.

A.2 Two expressions of $\mathcal{L}_{\text{CL}}$

Proposition A.2. *The expected coding length of a trajectory is equal to the expected number skills multiplied by the marginal entropy:*

$$\mathcal{L}_{CL}(\boldsymbol{\theta}) = -\mathbb{E}_{\substack{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\\ \boldsymbol{z}_{1:T}}}\left[\sum_{t=1}^{T}\log p^\star_{\boldsymbol{z}}(\boldsymbol{z}_t)\boldsymbol{m}_t\right] = n_s\mathcal{H}_{p^\star_{\boldsymbol{z}}}[\boldsymbol{z}].$$

Proof.

$$\begin{aligned}
&-\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^{T}\log p^\star_{\boldsymbol{z}}(\boldsymbol{z}_t)\boldsymbol{m}_t\right] \\
&= -\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^{T}\left(\int_\xi\delta(\boldsymbol{z}_t=\xi)\log p^\star_{\boldsymbol{z}}(\xi)d\xi\right)\boldsymbol{m}_t\right] \\
&= -\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\int_\xi\sum_{t=1}^{T}\boldsymbol{m}_t\delta(\boldsymbol{z}_t=\xi)\log p^\star_{\boldsymbol{z}}(\xi)d\xi\right] \quad \text{(Fubini's theorem)} \\
&= -\int_\xi\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^{T}\boldsymbol{m}_t\delta(\boldsymbol{z}_t=\xi)\right]\log p^\star_{\boldsymbol{z}}(\xi)d\xi \quad \text{(Linearity of expectation)} \\
&= -\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^{T}\boldsymbol{m}_t\right]\int_\xi\frac{\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^{T}\boldsymbol{m}_t\delta(\boldsymbol{z}_t=\xi)\right]}{\mathbb{E}_{\boldsymbol{\tau}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^{T}\boldsymbol{m}_t\right]}\log p^\star_{\boldsymbol{z}}(\xi)d\xi
\end{aligned}$$

$$= \underbrace{\mathbb{E}_{\boldsymbol{\tau}_{1:T}, \boldsymbol{m}_{1:T}, \boldsymbol{z}_{1:T}} \left[\sum_{t=1}^{T} \boldsymbol{m}_t \right]}_{n_{\mathrm{s}}} \underbrace{\int_{\xi} -p^{\star}_{\boldsymbol{z}}(\xi) \log p^{\star}_{\boldsymbol{z}}(\xi) d\xi}_{\mathcal{H}_{p^{\star}_{\boldsymbol{z}}}[\boldsymbol{z}]}$$

$$= n_{\mathrm{s}} \mathcal{H}_{p^{\star}_{\boldsymbol{z}}}[\boldsymbol{z}]$$

□

Remarks. This result generalizes to the case for discrete $\boldsymbol{z}$ by replacing the delta with an indicator function and integration with summation. Further, this rewriting not only offers an insightful interpretation of the objective but also has *different* finite-sample gradient compared to the original form. We discuss its implication further in Section 4.4.

A.3 Connection between LOVE and Variational Inference

Is the compression objective a prior in the framework of variational inference (VI)? The answer to this question is surprisingly nuanced. A bridge that connects VI and source coding is the bits-back coding argument [31, 32]. Under this scheme, the additional cost of communicating a message is equal to the KL divergence between the approximate posterior and the chosen prior distribution. This is because the posterior often does not match the prior and the sender must use more bits to send the message compared to using the prior exactly. In modern deep latent variable models such as VAE [41], the prior is in general chosen to be an isotropic Gaussian distribution or a sequence of isotropic Gaussian distributions that decomposes over the time steps. The bit-back coding argument shows that VI as a coding scheme is asymptotically optimal (in the limit of sending large numbers of messages), but it does not immediately guarantee good *representation learning*. As a thought experiment, imagine we choose the dimension of the latent variable in a VAE to be equal to the dimension of the data itself. It is clear that the model is not incentivized to perform good representation learning, but the bit-backing coding scheme still guarantees asymptotic optimality. Instead, good representation can only emerge *if* the prior is chosen properly.

Contrary to this prevailing design choice, our latent variables $\boldsymbol{z}_{1:T}$ and $\boldsymbol{m}_{1:T}$ are not independent from each other and the timesteps are not independent from each other. In fact, the interaction between the latent variables is crucial for compressing the trajectory optimally in our framework. Therefore, our objective takes into account the *global* structure of the latent variables. Note that $\mathcal{L}_{\mathrm{CL}}$ also has the natural interpretation of measuring the average number of bits that is required to send a trajectory under the coding scheme of this work. As both quantities measure the cost of the communication, the philosophical parallel between KL divergence and $\mathcal{L}_{\mathrm{CL}}$ is likely not a coincidence. In practice, we observe that $\mathcal{L}_{\mathrm{CL}}$ also has comparable regularization effect on the latent code, and in some cases it is possible to learn a good model *without* any KL divergence if $\mathcal{L}_{\mathrm{CL}}$ is present. From this perspective, our compression objective is indeed a “prior" insofar as it specifies the desired properties of the posterior. Another interpretation of LOVE is that it is optimizing the prior *directly*.

Nonetheless, it is an open question whether $\mathcal{L}_{\mathrm{CL}}$ has a precise counterpart in Bayesian inference, that is, it is not immediately obvious if it can be written as the KL divergence between the posterior $q(\boldsymbol{z}_{1:T}, \boldsymbol{m}_{1:T} \mid \boldsymbol{x}_{1:T})$ and some prior distribution $p(\boldsymbol{z}_{1:T}, \boldsymbol{m}_{1:T})$. One complication, for example, is the fact that the log density of $\boldsymbol{m}_{1:T}$ does not participate in the computation of $\mathcal{L}_{\mathrm{CL}}$. It may be possible to construct some form of hierarchical priors that enables a fully Bayesian interpretation of $\mathcal{L}_{\mathrm{CL}}$, but it is also well-known that the MDL framework can accommodate codes that are not Bayesian (e.g., the Shtarkov normalized maximum likelihood code [76]). We do not offer a definitive answer to this question in this work, but the connection between Bayesian inference and $\mathcal{L}_{\mathrm{CL}}$ is certainly an interesting direction for future works.

B Evidence Lower Bound

B.1 VTA

We reproduce the data generating process of Kim et al. [39] here:

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{s}_t) p(m_t \mid \boldsymbol{s}_t) p(\boldsymbol{s}_t \mid \boldsymbol{s}_{1:t}, \boldsymbol{z}_t, m_{t-1}) p(\boldsymbol{z}_t \mid \boldsymbol{z}_{1:t}, m_{t-1}).$$

VTA [39] shows that the ELBO can be written as:

$$\begin{aligned}\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}) \approx \sum_{t=1}^{T} \quad & \log p_{\boldsymbol{\phi}}(\boldsymbol{x}_t \mid \boldsymbol{s}_t) + D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(m_t \mid \boldsymbol{x}_{1:T}) \,\|\, p_{\boldsymbol{\phi}}(m_t \mid \boldsymbol{s}_t)\right) \\ & + D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{s}_t, \boldsymbol{m}_{1:T}, \boldsymbol{x}_{1:T}) \,\|\, p_{\boldsymbol{\phi}}(\boldsymbol{z}_t \mid \boldsymbol{s}_{t-1}, m_{t-1})\right) \\ & + D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{m}_{1:T}, \boldsymbol{x}_{1:T}) \,\|\, p_{\boldsymbol{\phi}}(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{z}_t, m_{t-1})\right).\end{aligned}$$

Like most ELBO, this ELBO can be deconstructed into a reconstruction term:

$$\mathcal{L}_{\text{rec}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{t=1}^{T} \log p_{\boldsymbol{\phi}}(\boldsymbol{x}_t \mid \boldsymbol{s}_t),$$

and the remaining KL term $\mathcal{L}_{\text{KL}}(\boldsymbol{\theta}, \boldsymbol{\phi})$.

B.2 LOVE

We reproduce the data generating process of LOVE here. Instead of using $\boldsymbol{\phi}$ for the parameters of priors and decoder, we will merge all parameters into $\boldsymbol{\theta}$:

$$p(\boldsymbol{a}_{1:T}, \boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T} \mid \boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{a}_t \mid \boldsymbol{s}_t) p(m_t \mid \boldsymbol{s}_t) p(\boldsymbol{s}_t \mid \boldsymbol{x}_t, \boldsymbol{z}_t, m_{t-1}) p(\boldsymbol{z}_t \mid \boldsymbol{x}_t, \boldsymbol{z}_{1:t}, m_{t-1}).$$

For LOVE, the ELBO is re-written as:

$$\begin{aligned}\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}) \approx \sum_{t=1}^{T} \quad & \log p_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t) \\ & + D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(m_t \mid \boldsymbol{x}_{1:t}) \,\|\, p_{\boldsymbol{\theta}}(m_t \mid \boldsymbol{s}_t)\right) + D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(z_t \mid m_t, \boldsymbol{x}_{1:T}, \boldsymbol{a}_{1:T}) \,\|\, p(z_t)\right) \\ & + D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid z_t, \boldsymbol{x}_t) \,\|\, p_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, m_{t-1})\right).\end{aligned} \tag{4}$$

Since z_t is a categorical distribution, we chose the uniform prior $p(z_t)$ similar to Oord et al. [65]. Similar to VTA, the loss is also decomposed into a reconstruction term and a KL term.

C Sensitivity to Compression Objective Weighting

Here we study how sensitive LOVE is to the choice of λ, the coefficient of $\mathcal{L}_{\text{CL}}$. We tested 5 values of $\lambda \in [0.01, 1]$ on *Simple* and *Cond. Colors* and applied dual gradient descent to automatically tune λ. Stopping point is selected based on the shortest code length achieved.

Table 3: The effect of different values of λ and dual GD on F1 scores of *Simple Colors* and *Conditional Colors* over 5 seeds.

	$\lambda = 0.01$	$\lambda = 0.03$	$\lambda = 0.1$	$\lambda = 0.3$	$\lambda = 1.0$	λ (dual GD)	VTA
Simple (F1)	.67 ± .27	.80 ± .21	.91 ± .02	.95 ± .06	.74 ± .08	.99 ± .00	.82 ± .13
Cond. (F1)	.84 ± .06	.92 ± .02	.90 ± .03	.82 ± .08	.87 ± .08	.99 ± .00	.83 ± .19

We see that while λ can change the quality of solutions found, the performance is stable within a $\pm\, 3\times$ regions around $\lambda = 0.1$. Further, using dual GD *consistently* outperforms handpicked λ with an ELBO threshold C slightly (e.g., 5%) higher than the $\mathcal{L}_{\text{ELBO}}$ achieved without compression, which can be obtained with minimal prior knowledge about the task. Hence λ likely can be tuned without strong prior knowledge. This contrasts with the other methods such as Kipf et al. [42] which require knowing the number of skills in each demonstration to perform well while achieving the similar or better performance.

D Architecture

D.1 Frame Prediction

Our base architecture is similar to Kim et al. [39][2] and we highlight the difference blue. The parameters of the posteriors are implemented with neural networks. First, the observation $\boldsymbol{x}_{1:T}$ are encoded into a lower dimension embedding $\hat{\boldsymbol{x}}_{1:T}$ with a `observation encoder`. The embedding is then passed through a `boundary posterior decoder` that outputs the parameters of $\boldsymbol{m}_{1:T}$. The embedding is further passed through a GRU [10], $f_{\text{s-rnn-fwd}}$ which runs on $\hat{\boldsymbol{x}}_{1:T}$ and give the cell state $\mathtt{h}_{1:T}$. For $\boldsymbol{z}_{1:T}$, we have two GRU's with cell, $f_{\text{z-rnn-bwd}}$ and $f_{\text{z-rnn-bwd}}$, which run on $\hat{\boldsymbol{x}}_{1:T}$ in different temporal direction (i.e., $f_{\text{z-rnn-bwd}}$ sees the future) to generate cell states $\mathtt{c}^{\text{fwd}}_{1:T}$ and $\mathtt{c}^{\text{bwd}}_{1:T}$. For each time step t, a candidate $\boldsymbol{z}'_t$ is sampled from a n_z-dimensional categorical distribution[3] whose log probability is a linear projection of $\mathtt{c}^{\text{fwd}}_t \| \mathtt{c}^{\text{bwd}}_t$. The sampled $\boldsymbol{z}'_t$ are embedded linearly into a vector of size 128. If $\boldsymbol{m}_1 = 1$, the $\boldsymbol{z}_t = \boldsymbol{z}'_t$; otherwise, $\boldsymbol{z}_t = \boldsymbol{z}_{t-1}$ [4]. Then we concatenate $\mathtt{h}_{t-1} \| \boldsymbol{z}_t$ and linearly project the vector into the mean and standard deviation of a isotropic Gaussian in $\mathbb{R}^8$. A $\hat{\boldsymbol{s}}_t$ is sampled from this Gaussian.

Th model also keeps two "belief" vectors that keep track of the history of the computation. These vectors are modulated by two RNN, $f_{\text{h-rnn}}$ and $f_{\text{c-rnn}}$. The abstraction belief $\boldsymbol{c}_t = \boldsymbol{m}_t \cdot f_{\text{c-rnn}}(\boldsymbol{z}_t, \boldsymbol{c}_{t-1}) + (1 - \boldsymbol{m}_t) \cdot \boldsymbol{c}_{t-1}$. The observation belief $\boldsymbol{h}_t = \boldsymbol{m}_t \boldsymbol{c}_{t-1} + (1 - \boldsymbol{m}_t) \cdot f_{\text{h-rnn}}(\hat{\boldsymbol{s}}_t \| \boldsymbol{c}_t \| \boldsymbol{z}_t, \boldsymbol{h}_{t-1})$. Finally the state abstraction is computed as the linear projection of $\boldsymbol{s}_t = \text{proj}(\boldsymbol{h}_t \| \hat{\boldsymbol{s}}_t)$. The projection layer is 256×128. Finally, $\boldsymbol{s}_t$ is passed through a `observation decoder` and decode into the reconstruction of $\boldsymbol{x}_t$. Refer to Kim et al. [39] for more details and an illustration of the computation graph.

For each convolutional layer or transposed convolution layer, the tuple's values correspond to input channel, output channel, kernel size, stride, padding. With that notation in mind, the specific hyperparameters for each components are:

- `observation encoder`: 4-layer convolutional neural network with {(3, 128, 4, 2, 1), (128, 128, 4, 2, 1), (128, 128, 4, 2, 1), (128, 128, 4, 1, 0)} with ELU activation [11] and batch normalization [33]. The output is flattened linearly projected to vector of size 128.
- `boundary posterior decoder` is a 5-layer 1-D causal temporal convolution [64] with {(128, 128, 5, 1, 2) $\times$ 5, (128, 2, 5, 1, 2)} with ELU activation and batch normalization. The output is the logits for the $\boldsymbol{m}_{1:T}$.
- $f_{\text{z-rnn-bwd}}, f_{\text{z-rnn-fwd}}, f_{\text{s-rnn-fwd}}, f_{\text{c-rnn}}, f_{\text{h-rnn}}$: GRU with cell size 128
- `observation decoder`: 4-layer transposed convolutional layers {(128, 128, 4, 1, 0), (128, 128, 4, 2, 1), (128, 128, 4, 2, 1), (128, 3, 4, 2, 1)} with ELU activation and batch normalization on all but the last layer. The last layer has no normalization and has `tanh` activation.

Removing Boundary Regularization. An important difference from Kim et al. [39] is that we do not enforce maximum number of subsequence N_{max} and the maximum length of subsequence l_{max} through the boundary prior (This effectively amounts to setting both to a value larger than the sequence length). These two hyperparameters' effect are similar to that of picking the number of segments in CompILE [42] and provide strong training signal. This assumption is in general problematic since we cannot expect to know a priori what the good values are for these hyperparameters. Kipf et al. [42] demonstrates that the performance can suffer if this kind of supervision is wrong. On the other hand, we do not assume anything about the the duration of the subsequence or how many subsequences there are in each demonstration.

We do, however, enforce a minimum length on the skill. While it is largely an optimization choice, we believe this prior is significantly weaker since useful options are almost always temporally extended. Indeed, with only the minimum length constraint, VTA is unable to learn good skills conducive for downstream tasks.

---

[2] `https://github.com/taesupkim/vta`

[3] Kim et al. [39] uses a isotropic Gaussian.

[4] This is the COPY action in Kim et al. [39].

D.2 Multi-task Grid World Environment

For trajectories, we make some larger changes to the architecture to encode properties we want in a model for learning options, e.g., Markov property. Though similar in spirit, a large portion of the architecture is different from the base VTA model, so the difference will not be highlighted.

For the posterior, we have an `observation encoder` and an `action encoder` that embeds both the state observation $\boldsymbol{x}_{1:T}$ and actions $\boldsymbol{a}_{1:T}$ to get embedding $\mathtt{a}_{1:T}$ and $\mathtt{x}_{1:T}$ each of size 128. The concatenation $\mathtt{a}_t \| \mathtt{x}_t$ linear projected down to a vector of size 128. The embedding is then passed through a `boundary posterior decoder` that outputs the parameters of $\boldsymbol{m}_{1:T}$. The embedding is further passed through a GRU [10], $f_{\text{s-rnn-fwd}}$ which runs on $\hat{\boldsymbol{x}}_{1:T}$ and give the cell state $\mathtt{h}_{1:T}$. For $\boldsymbol{z}_{1:T}$, we have two GRU's with cell, $f_{\text{z-rnn-bwd}}$ and $f_{\text{z-rnn-bwd}}$, which run on $\hat{\boldsymbol{x}}_{1:T}$ in different temporal direction (i.e., $f_{\text{z-rnn-bwd}}$ sees the future) to generate cell states $\mathtt{c}^{\text{fwd}}_{1:T}$ and $\mathtt{c}^{\text{bwd}}_{1:T}$.

For sampling $\boldsymbol{z}$ is done differently for interaction data for better gradient and representation learning. Instead of doing straight-through estimator for the categorical random variable, we opt to use vector-quantization [65] with straight-through estimators. First, $\mathtt{c}^{\text{fwd}}_t \| \mathtt{c}^{\text{bwd}}_t$ is linearly projected into dimension 128. Then a ReLU is applied and another 128 by 128 linear layer is applied. The VQ codebook is of size $n_{\boldsymbol{z}} \times 128$. Instead of taking argmax of the codebook, we approximate the distribution to be proportional to the softmax over the distance over a temperature t_{VQ}. We can sample from this distribution and apply the straight-through gradient on the sampled code candidate $\boldsymbol{z}'_{1:T}$. If $\boldsymbol{m}_1 = 1$, the $\boldsymbol{z}_t = \boldsymbol{z}'_t$; otherwise, $\boldsymbol{z}_t = \boldsymbol{z}_{t-1}$.

We construct directly $\boldsymbol{s}_t = \boldsymbol{z}_t \| \mathtt{x}_t$. and linearly project the vector into the mean and standard deviation of a isotropic Gaussian $\mathbb{R}^8$. $\boldsymbol{s}_t$ is sampled from this Gaussian and then decoded with the `action decoder` into the reconstructed action in $\mathcal{A}$.

For each convolutional layer or transposed convolution layer, the tuple's values correspond to input channel, output channel, kernel size, stride, padding. With that notation in mind, the specific hyperparameters for each components are:

- `observation encoder`: This takes as input the $10 \times 10 \times N_{\text{obj}} + 2$ and is implemented as a two 2D convolutional layers with ReLU activations, followed by a linear layer with a ReLU activation and a linear layer with no activation. The convolutional layers have hyperparameters $(12, 64, 3, 1, 0)$ and $(64, 64, 3, 1, 0)$ respectively. The first linear has input dimension $6 \times 6 \times 64$ and output dimension 128. The second linear layer has input and output dimensions 128.
- `action encoder`: The actions are embedded with an embedding matrix with embedding dimension 128.
- `boundary posterior decoder` is a 5-layer 1-D causal temporal convolution [64] with {(128, 128, 5, 1, 2) × 5, (128, 2, 5, 1, 2)} with ELU activation and batch normalization. The output is the logits for the $\boldsymbol{m}_{1:T}$.
- $f_{\text{z-rnn-bwd}}, f_{\text{z-rnn-fwd}}, f_{\text{s-rnn-fwd}}$: GRU with cell size 128
- `action decoder`: The decoder is implemented as three linear layers with ReLU activations, followed by a linear layer with no activation. The (input, output) dimensions of these layers are as follows: $(128, 128), (128, 128), (128, 128), (128, 5)$, where the last layer outputs logits over the actions.

D.3 3D Navigation Environment

The architecture used in the 3D navigation environment is identical to that of the multi-task environment detailed above, with the exception that the `observation encoder` is changed to handle visual inputs. Specifically, the `observation encoder` is a a 3-layer convolutional neural network parameters $(3, 32, 5, 2, 0), (32, 32, 5, 2, 0), (32, 32, 4, 2, 0)$, followed two linear layers $(7520, 128), (128, 128)$ with a ReLU activation in between.

D.4 Hierarchical Reinforcement Learning

For all approaches, we parametrize the policy as a double dueling deep Q-network [56, 86, 83]. The parametrization of the Q-function consists of a state embedding followed by two linear layer heads for the state-value $V_\theta(s)$ function and the advantage $A_\theta(s,a)$ function. Then the Q-function is computed as $Q_\theta(s,a) = V_\theta(s) + A_\theta(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a' \in \mathcal{A}} A_\theta(s,a)$.

The value function $V_\theta(s)$ and advantage function $A_\theta(s,a)$ are computed as linear layers with output dimension 1 and $|\mathcal{A}|$ respectively on top of the embedding $e(s)$ of the state s. Recall that the state s consists of two portions: the observation o of the grid or 3D environment, and the instruction i corresponding to the next object to pick up, where the instruction is not present in the demonstrations. The embedding $e(s)$ of the state s is computed by embedding the observation $e(o)$ with the `observation encoder` defined in the previous sections, and embedding the instruction $e(i)$ with a 16-dimensional embedding matrix. Then, the embedding $e(s)$ is a final linear layer with output dimension 128 applied to a ReLU of the concatenation of $e(o)$ and $e(i)$.

E Hardware

All our experiments are conducted on a GeForce RTX 2080 Ti or a GeForce RTX A6000.

F Hyperparameters

F.1 Frame Prediction

For these set of experiments, we use the same architecture and hyperparameters as Kim et al. [39]. The hyperparameters used for both *Simple Colors* and *Conditional Colors* are:

- KL divergence weight β : 1.0
- MDL objective weight λ (fixed) : 0.1
- Mini-batch size: 512
- Training iteration: 30000
- Number of skills $n_z = 10$

F.2 Multi-task Grid World Segmentation

We use the following hyperparameters for skill learning from demonstrations on the multi-task grid world environment experiments.

- KL divergence weight $\beta = 0$:
  - We found that the compression objective readily provides strong and sufficient regularization for the latent code. Adding additional KL divergence often results in the model not being able to reconstruct the actions properly.
- MDL objective weight λ:
  - We use a adaptive scheduling that approximate dual gradient descent. λ is initialized to 0. After every gradient step, λ is increased by 2.0×10^{-5} if $\mathcal{L}_{\text{ELBO}} \leq 0.05$ (Since $\beta = 0$, this is effectively $\mathcal{L}_{\text{rec}}$); otherwise, λ is decreased by 2.0×10^{-5}. After each update, the value of λ is clipped to $[0, 0.05]$. We find this setting provides the most stable training.
- Mini-batch size: 64
- Training iteration: 20000
- Number of skills $n_z = 10$
- $t_{\text{VQ}} = 0.1$
- Learning rate: 0.0005 with Adam Optimizer

F.3 3D Visual Navigation Segmentation

We use the same hyperparameter as Multi-task grid world experiments except that the maximum $\mathcal{L}_{\text{ELBO}}$ is capped at 0.001, *i.e.*, $\mathcal{L}_{\text{ELBO}} \leq 0.001$.

F.4 Hierarchical Reinforcement Learning

We use the same hyperparameters for training the policy for all approaches. Specifically, these hyperparameter values are as follows:

- ϵ-greedy schedule: We linearly decay ϵ from 1 to 0.01 over $500K$ timesteps for dense reward settings and over $5M$ timesteps for sparse reward settings in the multi-task grid world environment. In the 3D visual navigation environment, we decay over $250K$ timesteps.
- Discount factor γ: We use $\gamma = 0.99$ in all of the DQN updates.
- Maximum replay buffer size: 50K
- Minimum replay buffer size before updating: 500
- Learning rate: 0.0001 with the Adam optimizer [40]
- Batch size: 32
- Update frequency: every 4 timesteps
- Target syncing frequency: every 50K updates for multi-task grid world environment, every 30K updates for 3D navigation
- Gradient ℓ_2-norm clipping: 10
- Marginal threshold $\alpha = 0.001$

Since the demonstrations do not contain the instruction list observations, we set those to 0 in the demonstrations for the behavior cloning approach. Though we use $\alpha = 0.001$ in all of the experiments for consistency, we also found that slightly higher values of α yielded greater sample efficiency for LOVE.

G Algorithm

The Lagrangian for the unconstrained optimization problem is:

$$\min_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \ \lambda \mathcal{L}_{\text{CL}}(\boldsymbol{\theta}) + \left(\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}) - C\right).$$

In standard KKT condition, the dual variable is introduced on the ELBO, but the two problems are equivalent by inverting the dual variable. Since the only constraint on λ is that it is non-negative, this transformation does not change the optimal solution. Further note that we find that in many cases, choosing a fixed constant value for λ is sufficient for solving the problem.

Algorithm 1 Learning Options via Compression (LOVE): We highlight differences between our method and prior work (VTA) in blue.

```
Initialize model parameters $\boldsymbol{\theta}$
while not converged do
    Sample a trajectory $(\boldsymbol{x}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{x}_T)$ from the pre-collected experience $\mathcal{D}$
    for $t = 1, 2, \ldots, T$ do
        Sample boundary conditioned on entire trajectory $m_t \sim q_{\boldsymbol{\theta}}(m_t \mid \boldsymbol{x}_{1:t})$
        Sample skill from skill posterior $z_t \sim q_{\boldsymbol{\theta}}(z_t \mid m_t, \boldsymbol{x}_{1:t}, \boldsymbol{a}_{1:t})$
        Sample abstraction from abstraction posterior $\boldsymbol{s}_t \sim q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid z_t, \boldsymbol{x}_t)$
        Compute probability of the correct action from the decoder $p_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$
    end for
    Compute lower bound on maximum likelihood objective $\mathcal{L}_{\text{ELBO}}$ according to Equation 4
    Compute compression objective $\mathcal{L}_{\text{CL}}$ according to Equation 1
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla(\mathcal{L}_{\text{ELBO}} + \lambda\mathcal{L}_{\text{CL}})$ and $\lambda$
end while
```

Algorithm 2 Executing skill $\boldsymbol{z}$

```
while skill has not terminated do
    Compute state abstraction $\mu_{\boldsymbol{s}_t} = \mathbb{E}_{s \sim q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{z}, \boldsymbol{x}_t)}[s]$
    Take action $\boldsymbol{a}_t = \arg\max_{\boldsymbol{a}} \; p_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \mu_{\boldsymbol{s}_t})$
    Observe next state $\boldsymbol{x}_{t+1}$
    Terminate if $\mathbb{E}_{\boldsymbol{m}_{t+1} \sim q_{\boldsymbol{\theta}}(\boldsymbol{m}_{t+1} \mid \boldsymbol{x}_{1:t+1})}[\boldsymbol{m}_{t+1}] > 0.5$
end while
```

H Visualization of Learned Skills

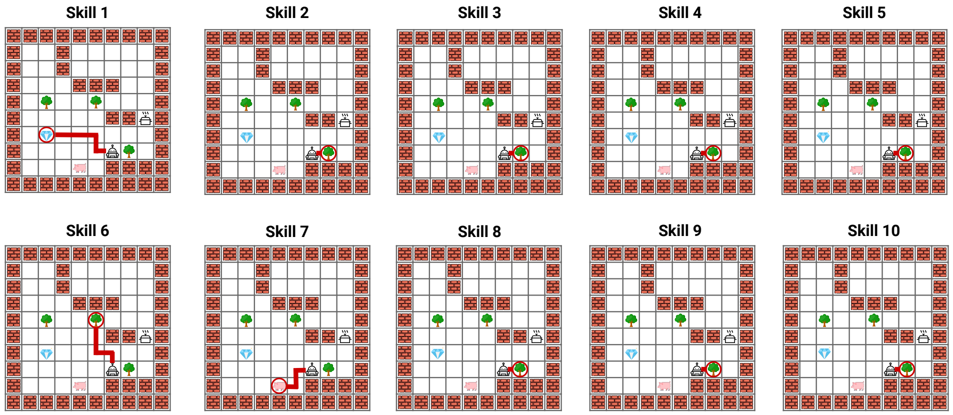

Figure 8: Skill visualization for all 10 skills of LOVE.

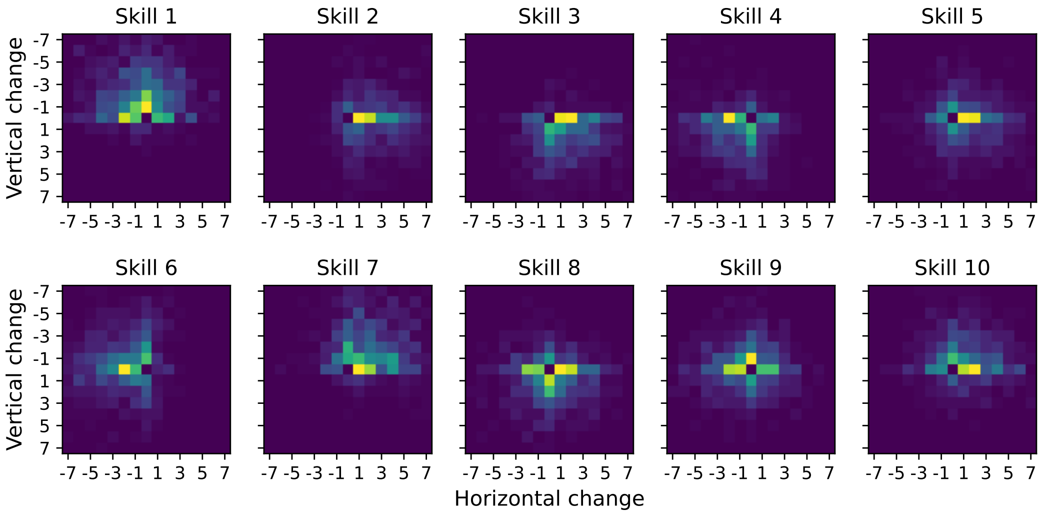

Figure 9: Heat map the agent's end position minus the agent's start position after applying each skill. In other words, this shows what direction the agent moves in after following each skill.

In Figure 8, we visualize the behavior of following each of LOVE's 10 skills on an example task. Each skill moves to and picks up an object, and the total set of skills covers 3 of the 4 object types in the task. These skills pick up both nearby objects, such as the tree in skill 2, as well as more distant objects, such as the diamond in skill 1. This contrasts DDO and VTA, which do not learn skills that move to and pick up objects.

To further understand what each of LOVE's skills does, we analyze whether there is a correlation between each skill and either the type of object it picks up, or the location of the object it picks up. We find that there appears to be little correlation between each skill and the type of object it picks up. Plotting the frequency of picking up each object type by skill shows that each skill picks up each object type with roughly the same frequency. Instead, there appears to be a correlation between each skill and the location of the object it picks up, illustrated in Figure 9. For example, skill 1 appears to specialize in moving to and picking up objects that are above the agent, while skill 7 tends to pick up objects that are up and to the right of the agent. It is also interesting to note that some skills are much more biased to move in certain direction (e.g., skill 1, 6, 7) while some appear to be more general (e.g., 4, 5, 8, 9) and move in any direction. Note in Figure 8, it may seem LOVE's skills seem short / redundant. This is due to the fact that LOVE's skills each pick up an object, so they naturally appear short when the agent is close to the objects, as in Figure 8. However, when the agent is far away from the objects, the skills still pick up the objects and are much longer. From Figure 9, we can see from the heatmap that even the skills that appear to do same thing in Figure 8 have very distinct behaviors depending on the state they are in. Also, recall LOVE imposes a sparse distribution over skills (Section 5). After filtering, skill 3, 7 and 10 in Figure 8 are dropped since they have low marginal probability and are not used to describe a significant part of the trajectories. Figure 8 includes redundant skills LOVE does not use; the used skills in the sparse distribution have little redundancy.

I Option Critic Results

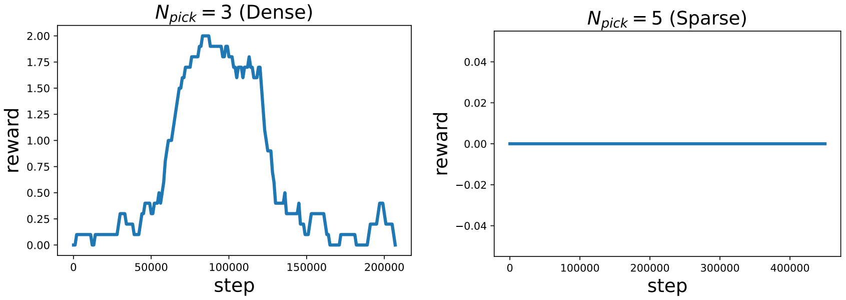

Figure 10: Results of running option critic [4] on 2 of the settings we considered in Figure 5.

Option critic [4] is a classical online HRL algorithm. We show the results of running option critic[5] on two of the four RL tasks we considered in Figure 10. The best return achieved over the training is shown in Table 4. We set the number of option to 8 following Bacon et al. [4] and leave other hyperparameters untouched. $N_{\text{pick}} = 3$ with dense reward is the easily setting and $N_{\text{pick}} = 5$ with sparse reward is the hardest setting. We see that in $N_{\text{pick}} = 3$, option critic is able to reach reward of 2 (maximum possible is 3) at around 1M environment steps but the performance deterioates afterwards. In $N_{\text{pick}} = 5$, the algorithm fails to make any meaningful progress. These observations may suggest online HRL algorithms like option critic may be insufficient for solving these tasks.

	$N_{\text{pick}} = 3$ (Dense)	$N_{\text{pick}} = 5$ (Sparse)
Option-Critic	2.0	0.0
LOVE (Ours)	**3.0**	**0.7**

Table 4: Comparison between LOVE and Bacon et al. [4] on the two RL tasks we consider in this work. Each entry is the maximum average return achieved during the course of training for both algorithms. For $N_{\text{pick}} = 3$ (Dense), 3.0 is the maximum possible return. For $N_{\text{pick}} = 5$ (Sparse), 1.0 is the maximum possible return.

J Comparison to Zhang et al. [90]

	Simple Colors		*Conditional Colors*		*Navigation*	
	MDL	**LOVE**	**MDL**	**LOVE**	**MDL**	**LOVE**
Precision	0.87	**0.99**	0.84	**0.99**	0.79	**0.90**
Recall	0.78	**0.85**	0.82	**0.83**	0.34	**0.94**
F1	0.82	**0.91**	0.83	**0.90**	0.48	**0.92**

Table 5: Comparison between LOVE and Zhang et al. [90] on the three segmentation tasks we consider in this work. We refer to Zhang et al. [90] as **MDL** in the table.

---

[5] `https://github.com/lweitkamp/option-critic-pytorch`

Zhang et al. [90] learn open-loop skills, which do not condition on the state and therefore cannot adapt to different states. Further, the MDL used in Zhang et al. [90] is equivalent to the variational inference used by VTA (on a different graphical model), which can be seen as greedily compressing each skill independently, rather than compressing a whole trajectory as LOVE does. Table 5 reports segmentation results on the Color domain and grid world, akin to Tables 2 and 3. Zhang et al. [90] performs the same as VTA on the Color domains – as they both use variational inference and there is no state – which achieves lower precision / recall than LOVE. On the grid world navigation, Zhang et al. [90] fails to recover the boundaries, unlike LOVE, because skills that navigate to objects require observing the state. This problem is shared by all open-loop approaches.

K Number of Initial Skills Ablation

	$K = 2$	$K = 5$	$K = 10$	$K = 15$	$K = 20$	$K = 30$	$K = 50$
Precision	0.27	0.80	0.90	0.96	0.96	0.95	0.95
Recall	0.53	0.91	0.94	0.96	0.95	0.96	0.93
F1	0.35	0.86	0.92	0.96	0.95	0.95	0.94

Table 6: Performance of LOVE on the grid navigation task with varying number of initial skills (K).

The number of skills to extract from demonstrations is often not known a priori, so choosing an appropriate value of the number of initial skills before filtering K is important. Intuitively, we hypothesize that K can be set conservatively: LOVE requires at least a minimum number of skills in order to fit the behaviors in the demonstrations, but may be able to gracefully prune out skills if K is set too high via the filtering described in Section 5. We test this intuition by varying $K \in \{2, 5, 10, 15, 20, 30, 50\}$ and measuring the segmentation performance on the grid world multi-task domain, and find that it appears to hold true in Table 6. For smaller values of K, such as $K = 2$ and $K = 5$, LOVE's F1 scores significantly degrade. However, performance remains high for all values where K is sufficiently large. Hence, we conservatively sizing K to be a large number.

L Comparison to Different Regularizers

	`entropy`	`num switch`	`VTA(3,5)`	`VTA(3,10)`	`VTA(5,5)`	`VTA(5,10)`	LOVE
Precision	0.26	0.80	0.34	0.40	0.34	**0.95**	0.90
Recall	0.53	0.93	0.46	0.60	0.50	0.43	**0.94**
F1	0.35	0.86	0.39	0.48	0.40	0.37	**0.92**

Table 7: Comparison of different kinds of regularizers that have been used in the literature for skill segmentation. LOVE outperforms all significantly.

Several prior methods encourage skills to act for more timesteps or prevent skills from switching too frequently, which can similarly help avoid degenerate solutions like LOVE. We note that such prior methods do not necessarily yield skills that help learn new tasks, while the maximum of LOVE's objective achieves the information-theoretic limit of compressing the trajectories (under a fixed function class), which is not done in other works. We empirically compare LOVE with several such methods on the segmentation of the multi-task grid world domain, including:

1. *Entropy*: This approach regularizes the entropy of skill marginal akin to Shankar et al. [72], which they refer to this as *parsimony*. Specifically, this approach adds a regularization term $\mathcal{H}_{p_z^\star}[z]$ to the objective $\mathcal{L}_{\text{ELBO}}$.
2. *Num switches*: This approach penalizes switching between skills, similar to Harb et al. [27], a variant of the Option-Critic framework [4]. Specifically, this approach adds a penalty n_{s} to the objective $\mathcal{L}_{\text{ELBO}}$ equal to the number of switches (i.e., the number of times $m_t = 1$) in a demonstration.

3. VTA($N_{\max}, l_{\max}$): VTA includes a prior on its boundary variables, which effectively sets the maximum number of skills per episode to be $N_{\max}$ and sets the maximum number of actions a skill can take to $l_{\max}$. We experiment with several settings of $N_{\max}$ and $l_{\max}$, including those that use prior knowledge about the domain not used by LOVE.

The results are summarized in Table 7. We find that regularizing the number of switches performs fairly well, but LOVE achieves higher F1 than other approaches that regularize the skills.