Neurosymbolic Programming Project Report

Seth Akins, Sarah Amini, Armin Ashrafi, Marianna Dehghan

April 2025

1 Problem Description

In this project, we explore an alternative approach to training the VAE model used in the LEAPS paper. When training the model with both reconstruction loss and the L^L loss, these objectives can sometimes conflict. For instance, minimizing reconstruction loss while encouraging programs with similar syntax to be close in the latent space may inadvertently harm the semantic similarity between programs, and the limited capacity of the neural network is used to represent features necessary for reconstruction. To address this, our project focuses solely on the L^L loss, aiming to emphasize the behavioral effects of the programs rather than their syntactic structure.

There are some significant changes between how we have implemented our project and the LEAPS VAE paper. There is no VAE here, the latent space is the input latent vector to the neural policy π. This means that there is no need for a reconstruction loss, and instead we use the L^L loss to the encoder to output. This neural policy is conditioned on the latent vector z, and different z vectors will correspond to different policies. (Figure 1) From here on, we use the same Cross Entropy Method (CEM) to do the search in the latent space, to find good policies. (Figure 2)

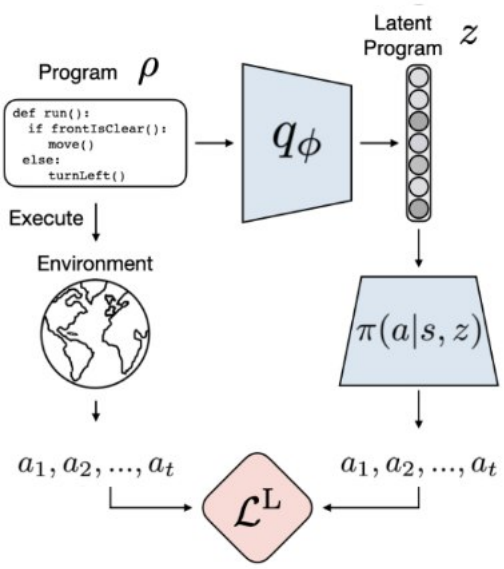

Figure 1: Learning Policy Embedding Stage

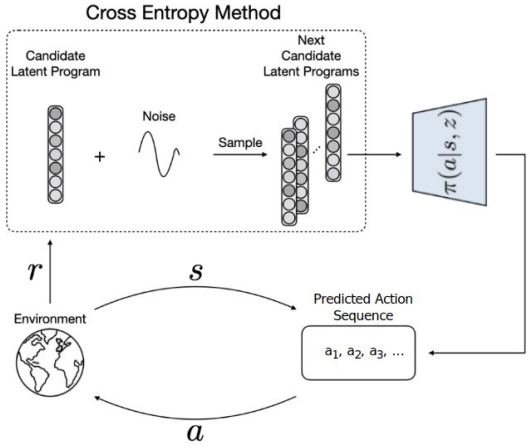

Figure 2: Latent Policy Search Stage

2 Experiments

In our experiments, we first trained 3 variants of the encoder-only model, with latent vector sizes 64, 128 and 256. Each of these models have been trained for 150 epochs. The training and validation loss of our models are highlighted in

Figure 3. We also introduced a truncation flag of 50 for each environment, to end episodes if they take too long, We however later realized that this value is too harsh of a limit for environments like Harvester. After training, we use the models by throwing away the decoder, and only using the conditioned neural policy to search in the latent policy space. For each of the 8 Karel tasks, we initialize a population of 256. We then use the mean elite CEM search method to find optimal policies. We are averaging the rewards for 16 different initialization points to generate the new population based on the top-k members of the population, and take their mean, and we get a mean elite latent vector to use as the new policy to evaluate and report its performance, starting the cycle again.

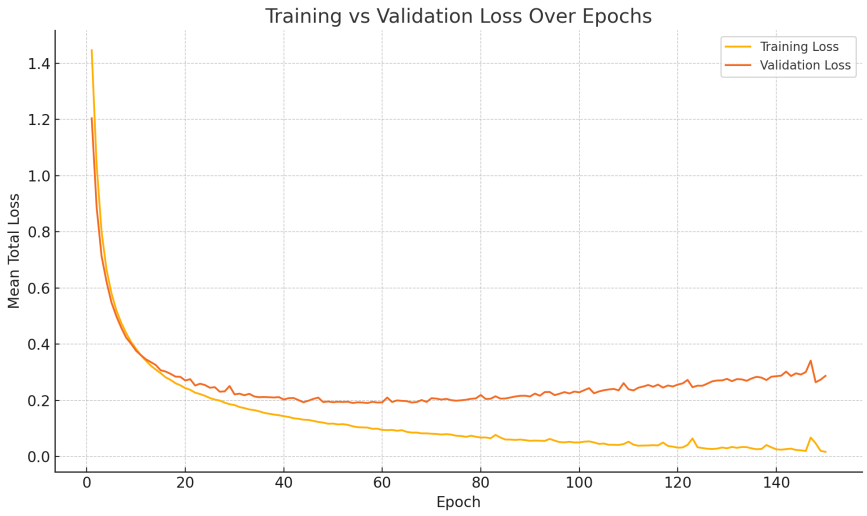

Figure 3: Training and Validation Loss for 150 epochs

In our graphs, we have shaded regions that correspond to the 95% bootstrap confidence interval over 10 seeds. These seeds represent different initialization in the latent space, and the sampling procedure when creating a new generation.

3 Results

Here are our results for our experiments, and the results run with the latent space constructed by the LeapsVAE paper. In general, our method learn the optimal policy faster, or at least reaches a much higher reward at the end of training. there are exceptions to this however, like the four corners environment. As mentioned before, the truncation is too short for environments such as Harvester, where we have failed to learn the optimal policy, as this is not

possible under our truncation method. Of course this decision was made due to time constraints.

As for the size of the latent space, we observe that harder environments such as Four corners or Harvester benefit from a larger latent space environment, and manage to get better policies. However, we do observe that in our case, 256 might be too large, as performance starts to drop off again.

Overall, the latent space size seems like a hyperparameter that needs to be tuned per environment. Another observation is that our method seems more responsive to changes in the latent size, while the LeapsVAE method does not respond as much.

Figures 4–6 illustrate the episodic return performance of VAE LEAPS compared to our method across six tasks in the KAREL and KAREL-HARD problem sets, evaluated at three different latent vector sizes: 64, 128, and 256. Results are reported as the mean and 95% confidence interval over 10 random seeds

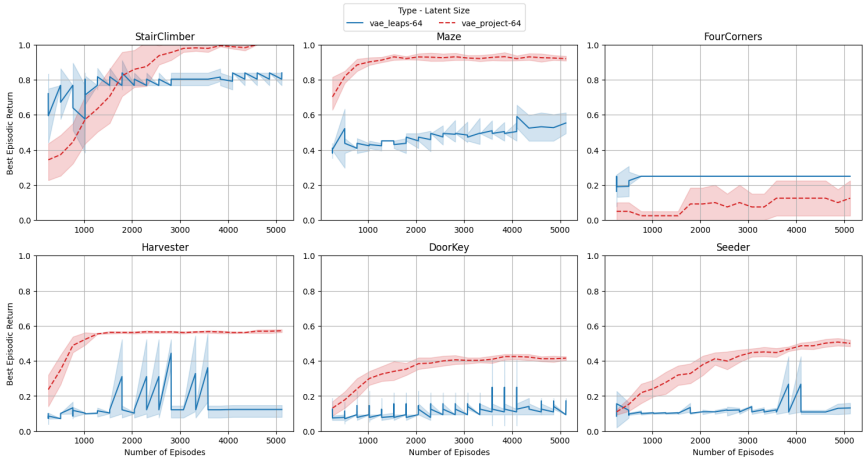

Figure 4: Best episodic return across tasks using a latent vector of size 64

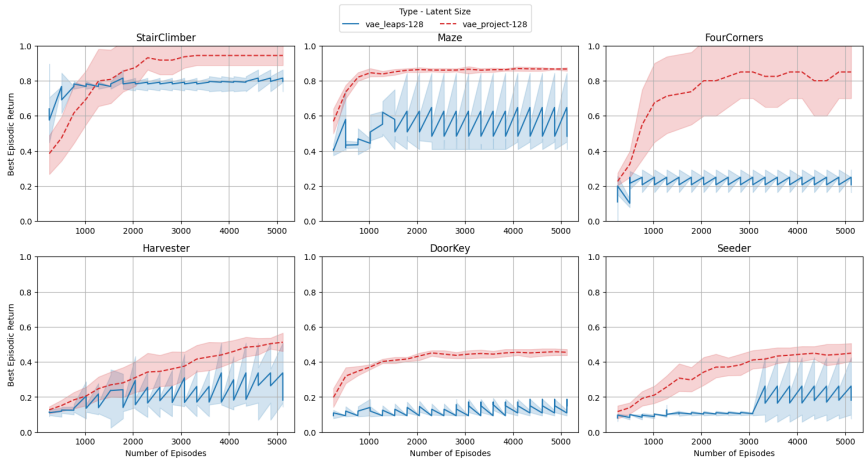

Figure 5: Best episodic return across tasks using a latent vector of size 128

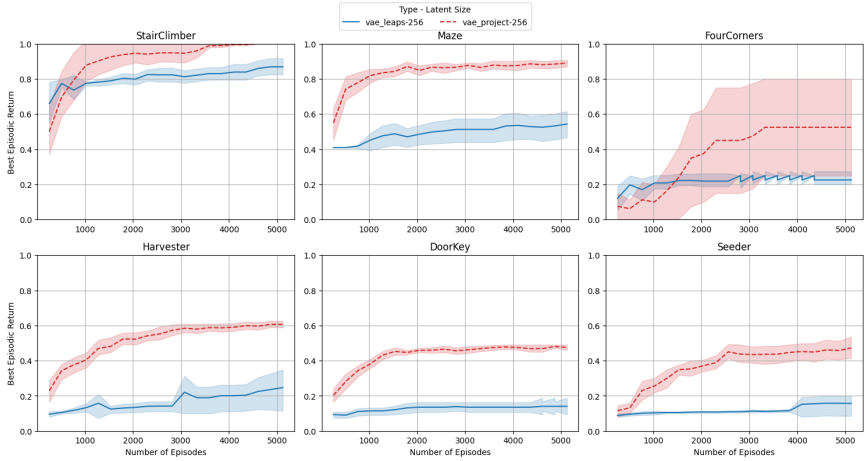

Figure 6: Best episodic return across tasks using a latent vector of size 256

4 Further Discussion and Future work

There are several directions one could explore to build on this work. One open question is how to interpret the validation loss in our setting—since our encoder is trained only with the L^L loss, a lower validation loss doesn't necessarily guarantee better downstream performance. It might even be beneficial to stop training earlier or add regularization to avoid overfitting to validation tasks while still learning useful latent representations. It would also be nice to test our method on environments that weren't part of the training set, as well as on

tasks with longer horizons, which tend to be more challenging. This would help us understand how well the method generalizes beyond the training distribution.

Another idea is to use our search process as a way to kickstart RL training—by first finding a promising latent vector using CEM, and then fine-tuning a policy on top of it. Finally, we'd like to explore how well the learned policies adapt to entirely new tasks and settings, which could give us deeper insight into the flexibility and robustness of the latent space.

5 Changes to the Code

We have made several changes to the code. We have created a policy class that works like a program would, we did this to take care of the forward passes of the policy, and manage changes in the world state. We also made significant changes to the Leaps VAE class, now called StrimmedVAE. We also created auxiliary classes like Collector, to streamline the process of data collection.