Conditional Normalizing Flows for Modeling Environment Stochasticity

Using a MuZero-based learned model

Damian Radu Bogdan
Supervisor(s): Frans Oliehoek, Jinke He
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Bogdan Radu Damian
Final project course: CSE3000 Research Project
Thesis committee: Frans Oliehoek, Jinke He, Michael Weinmann

Abstract

Planning agents have demonstrated superhuman performance in deterministic environments, such as chess and Go, by combining end-to-end reinforcement learning with powerful tree-based search algorithms. To extend such agents to stochastic or partially observable domains, Stochastic MuZero leveraged a framework that models environment uncertainty by splitting transitions into agent actions and learned stochastic outcomes. In this paper, we propose a novel architecture, FlowZero, which builds on this idea but replaces the discrete latent modeling of environment stochasticity with Conditional Normalizing Flows (CNF). This allows the model to learn a rich, continuous probability distribution over possible future states conditioned on the afterstate. The key advantage of this approach is its ability to perform exact log-likelihood evaluation, offering more precise density estimation than the evidence lower bound (ELBO) used in Stochastic MuZero. We aim to verify the proposed CNF's **capacity to overfit data** and **generalize to similar** and **larger** data, and our novel agent FlowZero's **capacity to perform in a stochastic environment**.

1 Introduction

In the pursuit of better performance in complex tasks, planning has long been a focus in the domain of artificial intelligence. Algorithms that leverage tree-based methods have shown great promise in various environments such as card games [15], board games [22], and continuous control tasks [12].

The development of planning-based reinforcement learning agents that can operate at superhuman levels has become a major milestone in the field. These agents are particularly effective in domains that require long-term reasoning, precise planning, and strategic foresight, such as classic board games and complex control tasks. A central theme in the success of these agents has been their ability to combine powerful search algorithms with deep learning in an end-to-end framework.

Early approaches largely focused on model-free reinforcement learning due to the difficulties of model-based methods. Specifically, model-based agents struggled to simultaneously learn accurate transition dynamics and value functions. This disconnect limited their performance in high-dimensional or visually complex environments. However, the introduction of AlphaZero [23] marked a turning point. By leveraging a perfect simulator of the environment, AlphaZero combined deep neural networks with Monte Carlo Tree Search (MCTS) [14] to evaluate the value of states and propose action policies. This approach eliminated the need for explicit reward signals during search and demonstrated unmatched performance in games like chess, shogi, and Go.

Despite its success, AlphaZero's dependence on a known and accurate simulator limited its applicability to environments where such simulators are available. To address this, MuZero [2] proposed a more general framework that learns a model of the environment implicitly, without requiring access to the true dynamics. By learning a latent dynamics function from observations transformed into a rich latent space, MuZero was able to predict the policy, value, and reward necessary for planning with no previous knowledge of the environment. This extended the applicability of planning-based agents to more general settings, including visually rich domains such as Atari games.

However, MuZero assumes a deterministic environment model, which restricts its utility in many real-world applications where the environment is inherently stochastic or only partially observable. For example, in noisy physical environments, the same action may lead to different outcomes depending on hidden or random factors. Recognizing this limitation, Stochastic MuZero [2] introduced a framework that augments the MuZero architecture to handle stochastic transitions. By modeling the transition as a two-step process, first applying the agent's chosen action, then applying a learned stochastic outcome, the agent can plan over both decision and chance events. This stochasticity is modeled using a discrete latent representation trained via a Vector Quantized Variational AutoEncoder (VQ-VAE) [27], allowing the agent to predict a distribution over discrete possible environment outcomes.

While Stochastic MuZero is an important step forward, its reliance on discrete latent codes and approximate likelihood training (via evidence lower bounds) introduces limitations in modeling expressiveness and evaluation accuracy. In this paper, we propose an alternative approach that addresses these limitations by incorporating Conditional Normalizing Flows into the MuZero framework. Our method, FlowZero, retains the two-step planning structure of Stochastic MuZero but replaces the discrete latent stochastic model with a flow-based generative model capable of learning complex, continuous distributions over next states, conditioned on the afterstate. Conditional Normalizing Flows (CNF) provide exact log-likelihood evaluation, enabling more precise and expressive density modeling of environmental stochasticity. Under an offline supervised setting, we **proved that the CNF is capable of quickly overfitting various datasets, precisely generalizing to data similar to its training, and decently generalizing to vastly different data**. There is still work to be done to **achieve the desired performance in the online MuZero-style setting of FlowZero.**

2 Background

Autoregressive Models have proven to be an effective option for solving the issue of modeling highly expressive probability distributions in various domains like image [26] and speech [25] generation. They factorize the joint distribution of the target into conditional probabilities that create a direct dependency between the dimensions of a sample. This is done using the chain rule for probabilities, as shown in equation 1 where x_i represents a subset of dimensions of the target distribution. Although they are remarkably accurate, allowing exact likelihood maximization, they lack in sampling time, since one needs to sample each dimension subset sequentially.

$$p(x) = \prod_i p(x_i \mid x_{<i}) \tag{1}$$

Diffusion Models learn to denoise samples that represent pure noise into realistic samples of the target distribution. They do this by learning how to gradually reduce a regular sample into noise during training, and then reversing this process starting from noise for sampling. The formula for a noising (diffusion) step is shown in equation 2. Similarly to autoregressive models, diffusion models suffer from slow sampling, since the denoising process requires many steps for high resolution outputs. They have shown excellent performance in image [20], text-to-image [16], and video [17] synthesis.

$$q(x_t \mid x_{x-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{2}$$

MuZero [21] is a planning agent that learns a model of the game it is playing. Similarly to its predecessor AlphaZero [23], MuZero uses a powerful tree-based search planning algorithm that allows for precise and sophisticated look-ahead in games that require it, such as chess and go. They combine this powerful planning algorithm with an end-to-end reinforcement learning paradigm that trains the components of the model to accurately predict the value of the current state, the policy (probability distribution of being the best move over the set of valid next moves), and the estimated immediate reward of applying a certain move. These values that are necessary for the tree search to be accurate. MuZero revolutionized the algorithm by adding a new learned model that takes environment observations and transforms them into a rich latent representation. This way, the agent doesn't need knowledge of the environment dynamics (i.e. rules of the game), making it effective in any deterministic environment without access to a perfect simulator. Unfortunately, this versatility only applies to deterministic environments, since the planning step assumes each state-action pair will always lead to the same future state.

Stochastic MuZero [2] is a direct upgrade to base MuZero, aiming to bring stochastic planning into the agent's capabilities. The authors introduce a novel planning architecture that makes use of afterstates, splitting a 'move' into applying a user action a and applying environment stochasticity afterward, using another jointly trained model. Leveraging a variant of the Vector Quantized Variational AutoEncoder, they train a model that can effectively predict the probability of any one of m discrete possible stochastic effects happening at any timestep without supervision. They use these discrete codes in the same way as user actions, passing them to the same type of dynamics network to predict the future state.

VQ-VAE [27] is a variant of the VAE architecture that quantises the latent representations of inputs before they are passed to the decoder. After training both the encoder and decoder, a powerful autoregressive model effectively learns the probability distribution of the quantised embeddings of inputs, representing the prior of the model. Using ancestor sampling, they query the trained prior and feed the result to the decoder, forming an effective generative model. Stochastic MuZero uses this architecture to quantise environment representations into discrete stochastic events in the environment.

Normalizing Flows [18] provide a general mechanism for defining expressive and highly complex probability distributions. Through a sufficiently complex chain (or a flow) of transformations, any point on a particular target distribution can be mapped to a very simple same-dimensional distribution, usually a multivariate standard Gaussian. This technique becomes very strong when enforcing the restriction that all transformations are bijective. If this rule is satisfied, trivially sampling from the latent distribution and applying the inverse of the flow will output a highly accurate sample from the target distribution. Another benefit of using NF for probability modeling is exact log-likelihood evaluation. Since there is a one-to-one mapping between the target and latent distributions, the likelihood can be precisely computed using the change of variable formula, as shown in equations 3 and 4. $p_X(x)$ represents the likelihood that the current sample x is part of the distribution we are trying to model. $f(.)$ is the flow and thus $p_z(f(x))$ is the likelihood of the output of the flow to be part of the prior gaussian distribution. Finally, $\frac{\partial f(x)}{\partial x^T}$ is the jacobian of the flow. This term describes the partial derivative of all dimensions of the result with respect to all dimensions of the input. It represents the 'warping' of the distribution space and thus adding it helps us compute the real likelihood of our sample to our modeled distribution. The model does not have to be trained on likelihood estimations, but on maximising the likelihood directly. With careful choice for these transformations, the change of variable formula can be very easily computed. It requires taking the logarithm of the determinant of the jacobian of the transformations with respect to the input. If this determinant can be made triangular, this operation is simply a sum of the logarithm of the diagonal terms.

$$p_X(x) = p_z(f(x)) \left| det \left(\frac{\partial f(x)}{\partial x^T} \right) \right| \tag{3}$$

$$\log(p_X(x)) = \log(p_z(f(x))) + \log \left| det \left(\frac{\partial f(x)}{\partial x^T} \right) \right| \tag{4}$$

RealNVP [7] propose a very crafty solution to how to build a tractable and efficient normalizing flow for density estimation. Their idea for modeling transformations is as follows: you split input x into x_1 and x_2, leave one side (i.e. x_1) unchanged while plugging it into a deep residual network that generates a scale s and a shift t that is applied to x_2. Then we concatenate the unchanged x_1 and the $y_2 = s * x_2 + t$. Since the half that generates the scale and shift remains unchanged, it can be plugged back into the network in order to do a reverse pass of the network (i.e. latent variable to target distribution). Since the network does not need to be invertible with respect to the scale and shift factors, we can make these networks as complex as we want, ensuring a good density approximation. The negative log-likelihood is easily tractable under this paradigm because the log determinant of the jacobian for each of these layers (which summed up give the second term of the right side of equation 4) is equivalent to the sum of the elements of the scale vector s, as shown in equation 5 (which assumes the previous conditions i.e. x_1 stays unchanged and is passed to the s and t networks).

$$\log \left| det \left(\frac{\partial y}{\partial x^T} \right) \right| = \sum_j s(x_1)_j \tag{5}$$

Conditional RealNVP The authors of [29] suggest a natural extension to the Normalizing Flows concept, namely the capacity to learn a complex conditional probability. They hypothesized that concatenating the input to the scale and shift networks with the conditional element during training and sampling would effectively train the network to not only properly transform a latent variable into a realistic sample of the original distribution, but that it would also be accurately conditioned on the required element. The transformed change of variable formula is described in equation 6. This proved accurate as the performance found in the field of super-sampling was competitive with other modern solutions.

$$\log(p_X(x \mid c)) = \log(p_z(f(x;c))) + \log \left| det \left(\frac{\partial f(x;c)}{\partial x^T} \right) \right| \tag{6}$$

3 Related Work

In the field of reinforcement learning (RL), the Markov Decision Process (MDP) [19] is the format usually used to describe the agent's interactions with the environment, containing a set of world states and transitions from one to another, alongside with the rewards of these transitions. **Probabilistic Ensembles with Trajectory Sampling** [5] is an algorithm that seeks to to bridge the gap between the strength of model-based reinforcement learning methods and the efficiency of model-free alternatives in MDPs with inherently *stochastic* transitions. It trains a multitude of deterministic dynamics models, and then samples multiple trajectories through them to make planning robust to uncertainty. It combines this with Model Predictive Control (MPC) [4] to plan over the learned model instead of learned policies (distributions over possible future actions). At each timestep, the model plans a sequence of actions, runs it through the ensemble of dynamics networks to calculate expected return and chooses the action to play accordingly.

Another model based RL agent that targets uncertain transitions within MDPs is **Dreamer** [9]. They managed to obtain high performance and efficiency in visual control tasks by formulating them as partially observable MDPs (POMDP), making every transition in their world model inherently stochastic. Dreamer operates on a similar conceptual plane as MuZero [21], the primary distinction coming down to the mechanism of choosing an action. While MuZero uses MCTS to decide the policy at the current timestep, Dreamer contains an action modeling component that directly predicts the next action given any latent state. This action model is then used to imagine trajectories which are then used to directly train the action model. Similarly to MuZero, Dreamer uses real life outcomes stored in a replay buffer to train the world dynamics model that is used in the imagined rollouts.

PlaNet [10] introduces a novel architecture for solving POMDPs which does not rely on explicit policy and value predictions. Similarly to other MBRL agents, PlaNet uses its real world experience to update the world model, using the concept of belief states. They are pairs of a deterministic state, preserving information from the trajectory that lead to it, and a stochastic state, modeling the local transition uncertainty. Thus, combining this pair of states encodes a distribution over possible future states based on history. It then samples this distribution to obtain imaginary trajectories and evaluate candidate action sequences via the

Cross-Entropy Method (CEM) [3] instead of MCTS to predict rewards. Unlike Dreamer and MuZero, which seek to learn an explicit policy, PlaNet uses MPC [4], similarly to PETS [5].

4 FlowZero

FlowZero combines a learned stochastic transition model of the environment dynamics with a variant of Monte Carlo Tree Search (MCTS). First, we describe the new model and subsequently how it is combined with MCTS planning.

4.1 Modeling Stochasticity

As mentioned earlier, our approach seeks to attempt more precise modeling of the environment uncertainty, but we build directly on the architecture of the Stochastic MuZero paper [2]. The principal idea lies in the differentiation of decision points and chance points in the tree nodes of the planning stage [6]. In the former, the agent is tasked with learning optimal player moves, while the latter focuses on estimating the real probability distribution of the uncertain environment events. This modified version of the MuZero MCTS builds on the concept of **afterstates**, introduced by [24]. Splitting a standard transition into a deterministic step and a stochastic step effectively allows the agent to focus on accurately modeling both aspects of the environment. The core difference in our approaches lies in how we model the stochastic transition, as follows: The Stochastic MuZero paper [2] assumes these stochastic events to be a fixed set of learned discrete possibilities. Then, they use an afterstate dynamics network identical as the one used in decision states to predict the event's impact on the environment. Thus, they boil down the modeling of stochasticity to learning those discrete chance codes.

FlowZero, on the other hand, seeks to learn the future state $S_{t+1} \sim P(s_{t+1} \mid z, as_t)$, where z is a simple distribution of the same dimension as the afterstate, namely a multivariate gaussian conditioned on the afterstate, as described in equation 7. The $\mu(.)$ and $\sigma(.)$ functions are not the mean and variance of the afterstate, but learned neural networks. This transformation is learned by taking the realized future state, putting it through the representation function, and running it through a Conditional Normalizing Flow using the afterstate to realistically map it to a latent standard gaussian. Thus, the prediction network should converge such that sampling a latent gaussian and passing it backwards through the flow using an afterstate as a condition, it would map back to a probabilistically accurate future state.

$$p(z \mid x) = \mathcal{N}(z; \mu(x), \sigma^2(x)) \tag{7}$$

4.2 Complete model

For the model description, the subscript value $_t$ represents the timestep of the real life observation that a rollout starts from, and the superscript value ^{k} represents the depth in the rollout itself. The differentiation is useful because the rollout happens with no knowledge of the outside world (i.e. only in the agent's latent state space). For example, latent state s_t^2 corresponds to the real life observation o_{t+2}. The FlowZero architecture contains 5 functions: a *representation* function h that generates a rich latent state s_t^0 from the observation $o_{<=t}$, a *dynamics* function ϕ which produces the latent afterstate as_t^k from the latent state s_t^k and the action a_{t+k}, an *afterstate dynamics* function g that produce the future latent state s_t^{k+1} and the immediate reward r_k^{t+1} from the current afterstate as_t^k, a *prediction* function f which given a latent state s_t^k predicts the value v_t^k and policy p_t^k, and an *afterstate prediction* function ψ that generates the afterstate value Q_t^k. The model equations are shown in equation 8. For any readers interested in cross-checking this description with the original Stochastic Muzero paper [2], keep in mind that we name the afterstate->future state network the *afterstate* dynamics, while they name it simply the dynamics.

$$\begin{aligned} \textit{Representation} \quad & s_t^0 = h(o_{<=t}) \\ \textit{Dynamics} \quad & as_t^k = \phi(s_t^k, a_{t+k}) \\ \textit{Afterstate Dynamics} \quad & s_t^{k+1}, r_t^{k+1} = g(as_t^k) \\ \textit{Prediction} \quad & p_t^k, v_t^k = f(s_t^k) \\ \textit{Afterstate Prediction} \quad & Q_t^k = \psi(as_t^k) \end{aligned} \tag{8}$$

During inference, given an initial observation $o_{<=t}$ and a sequence of actions $a_{t:t+k}$, we can generate trajectories from the above model by recurrently unrolling it and by transforming trivially sampled gaussian latents into future states such that stochasticity is accounted for.

4.3 Training

The fundamental end-to-end training style of MuZero is slightly changed in the implementation of FlowZero. Given a trajectory of length J with observations $o_{<=t:t+J}$, actions $a_{t:t+J}$, value targets $z_{t:t+J}$, policy targets $\pi_{t:t+J}$, rewards $u_{t+1:t+J}$, and represented future states $s_{t+1:t+J+1}$, the model is unrolled for J steps as shown in figure 1 and trained to optimize the sum of losses shown in equation 9: the standard MuZero loss and the tailored loss of the CNF network wholly described in equation 12 later on.

$$L^{total} = L^{MuZero} + L^{CNF} \tag{9}$$

The MuZero loss is the same as described in the paper [21]. The negative log-likelihood (NLL) loss is the negated log-likelihood that we compute using the change of variable formula (see equation 3 and 4; 6 for the conditional version). In this case, x is the future state that we are passing to the CNF network, and c is the afterstate that conditions the flow. Equation 10 shows the NLL loss for the model. This is only one component of the real CNF loss, as described in equation 12 under section 5.1.1.

$$L^{NLL} = \sum_{j=0}^{J-1} nll(s_{t+k+1}, as_{t+k}) \tag{10}$$

5 Experimental Setup and Results

Here we describe our experiments aimed to answer the four research questions proposed in the introduction of this paper. **Can the CNF overfit the data**, **generalize well both to similar** and **larger** data, and **does the FlowZero agent work**? Everything besides the CNF implementation is identical to their counterparts in the Stochastic MuZero paper [2].

5.1 Experimental Setup

In this section we will describe in detail the environment used, the implementation of the previously proposed CNF, and the data collected.

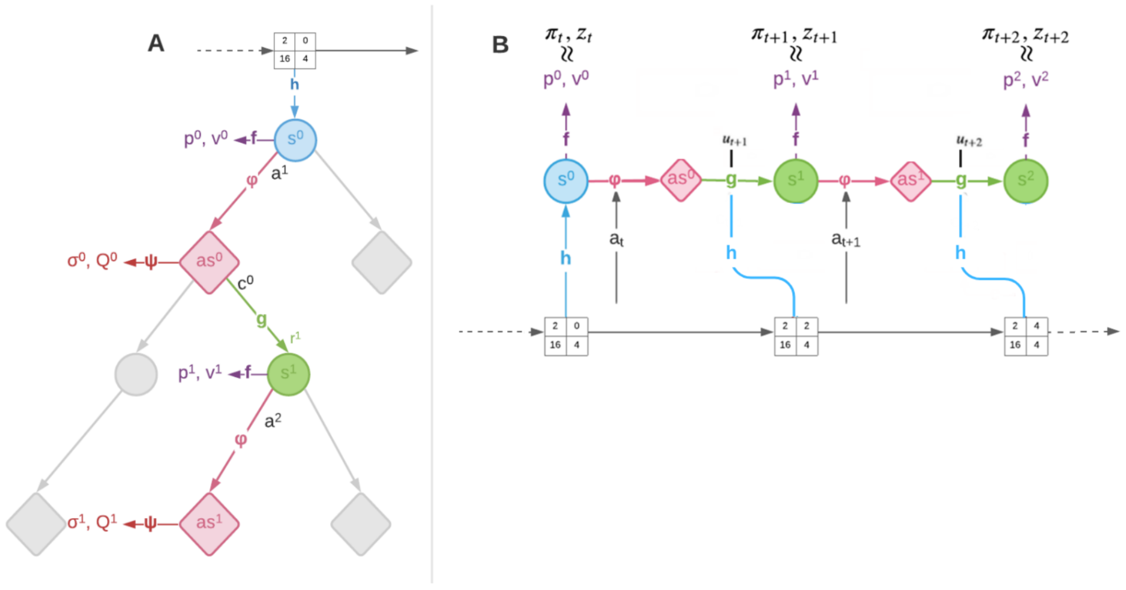

Figure 1: **FlowZero**. (**A**) Monte Carlo Tree Search used in FlowZero, which is identical to the approach used by Stochastic Muzero [2], where diamond nodes represent chance nodes and circular nodes represent decision nodes. During the selection phase edges are selected by applying the pUCT formula in the case of decision nodes, and by sampling the CNF network in the case of chance nodes (**B**) Training of the FlowZero model. In this example, given the trajectory of length $J = 2$ with observations $o_{<=t:t+2}$, actions $a_{t:t+2}$, value targets $z_{t:t+2}$, policy targets $\pi_{t:t+2}$, rewards $u_{t+1:t+3}$, the model is unrolled for 2 steps. During the unroll, the representation network h receives the observation $o_{<=t+k}$ as input and generates the estimated future latent state s_{t+k+1}, which is then fed into the negative log-likelihood loss function alongside with the afterstate as_{t+k}. The CNF is trained towards minimizing this loss. The policy, value and reward outputs of the model are trained towards targets π_{t+k}, z_{t+k} and u_{t+k} respectively.

5.1.1 CNF Implementation

For both experiments we use a single-scale variant of the architecture proposed in [7] since the environment is quite simple and would not benefit from that level of complexity. We use 4 coupling layers with alternating checkerboard masks. The $s(.), t(.), \mu(.), \sigma(.)$ functions are implemented using ResNets [11] of only one ResBlock with standard Batch Normalization [13] and ReLu activations [8]. To condition the $s(.)$ and $t(.)$ networks on the afterstate, we concatenate the input latent (either the future state for inference or the sampled z for sampling) with the afterstate and pass it to a common ResNet. Each function has a separate zero-initialized convolutional head, followed by a Batch Normalization layer. Since the shift factor can be negative, no activation is used. For the scale, the head also contains an activation and a tanh() layer for stability. Following the suggestions of [7], the $s(.)$ and $t(.)$ networks have learnable zero-initialized scale factors for stability. To incentivize the network to find the shortest path to the correct mapping (i.e. small volume distortion), an L2 regularization is applied to the scale factor of the $s(.)$ function, with a coefficient of $5 * 10^{-5}$. Another crucial aspect of the implementation is including Batch Normalization on the output of the coupling layers themselves. In this regard, the authors suggest using Batch Normalization with tracking averages of the mean and variance. Since this is yet another transformation of the Flow, we need to account for it in the computation of the log determinant of the Jacobian, which is fairly simple since the determinant is yet again triangular. Equation 11 shows what it boils down to. This leads us to the final Loss function, described in equation 12.

$$\log\left(\prod_i \frac{1}{\sqrt{\sigma^2+\epsilon}}\right) = -\frac{1}{2}\sum_i \log(\sigma^2+\epsilon) \tag{11}$$

$$L^{CNF} = L^{NLL} + \frac{1}{2}\sum_i \log(\sigma^2+\epsilon) + 5 * 10^{-5}\sum_k scale_k^2 \tag{12}$$

5.1.2 Environment

Both experiments are run using the single-player stochastic game **2048**. We chose it because of its simple observations (as seen in figure 2), since effectively learning around the stochasticity is relatively quick compared to complex games, allowing relatively shallow sub-networks while still achieving good performance. The game consists of a 4x4 grid whose tiles are initialized with either 0, 2, or 4, where a move up, down, left, or right moves all of the tiles in the specified direction as far as they can go. Whenever two tiles of the same denomination collide, they merge into a tile with double their value. After each move, a tile of value 2 or 4 appears randomly in one of the free squares on the grid, which confers the game its stochastic nature. The objective of the game is to create a tile of value 2048, although this is not a theoretical limit of the game. A legal move must lead to at least one tile moving or at least two tiles combining. Losing the game implies running out of legal moves before achieving the 2048 tile.

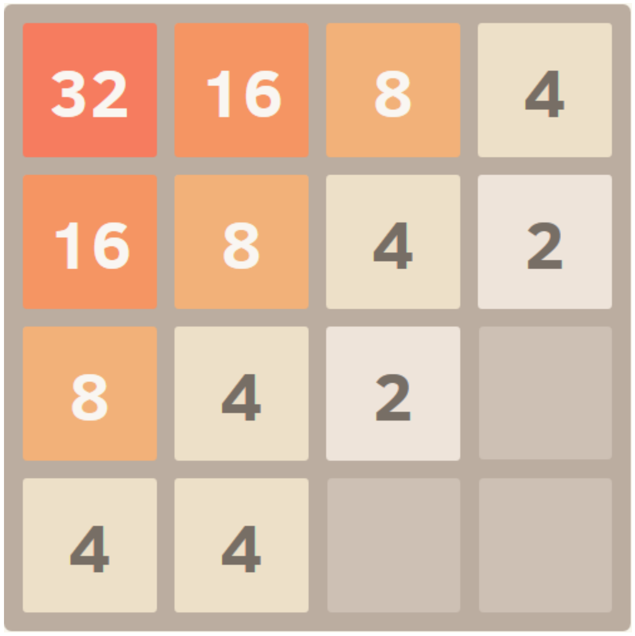

Figure 2: A game state in 2048

5.1.3 Data

The data used for the following experiments constitutes of 102400 data points. Each data point is an afterstate as_t^0 (reached in the first unroll step) and the latent representation of the corresponding future state $h(o_{t+1})$, where h is the representation function used. We collected this data using a well trained Stochastic MuZero [2] model to obtain realistic samples of sequential afterstate-future state pairs.

It is important to point out that there are two different modes of the CNF that should be tested. For the **forward pass**, we want to test how well the model maps the latent representation $h(o_{t+1})$ to our gaussian prior $\mathcal{N}(z; \mu(as_t^0), \sigma^2(as_t^0))$, using the afterstate as_t^0 as a condition of the transformations as well as the prior. We will do this using the log-likelihood computation formula for the gaussian prior as shown in equation 13, where z is the output of the network. For this metric it is paramount to discuss the expected value. For a gaussian prior, the log-likelihood of the mean of that gaussian is -0.9189, if the variance is 1. The prior space of our distribution is 1024 dimensions because the latent representation of observations in the model has shape 64x4x4. Since all of those dimensions are independent from one another, mapping to the *mean* of the prior gives a log-likelihood of $1024 \times -0.9189 = -940.953$. Correctly mapping to the whole distribution and not collapsing on the mean would give a log-likelihood of $1024 \times -1.4189 = -1452.9536$.

$$\log(p_Z(z)) = -\frac{(z - \mu(as_t^0))^2}{4\sigma^2(as_t^0)} - \frac{\log(2\pi\sigma^2(as_t^0)))}{2} \tag{13}$$

For the **backward pass**, we want to test how well the model samples our prior and generates a future state which seeks to approximate the real future state $h(o_{t+1})$, conditional on the afterstate as_t^0. We measure this accuracy using a Mean Squared Error (MSE) loss between the prediction and the real future state, as shown in equation 14, where s_t^1 is the prediction and N is the number of dimensions. We compare it to the MSE loss achieved by the Stochastic MuZero [2] agent we used for collecting this data, computed at collection time.

$$\frac{1}{N}\sum_{N}(s_t^1 - h(o_{t+1}))^2 \tag{14}$$

5.2 Experimental Results

The following contains four experiments tailored to the primary aims of this paper, ordered in ascending difficulty.

5.2.1 Can the CNF overfit the data?

First of all, we seek to overfit the CNF to our data. For this reason, we tried overfitting on progressively more of the dataset, starting from 1%, going to 10%, and eventually 100% of the database. The results can be found in figure 3.

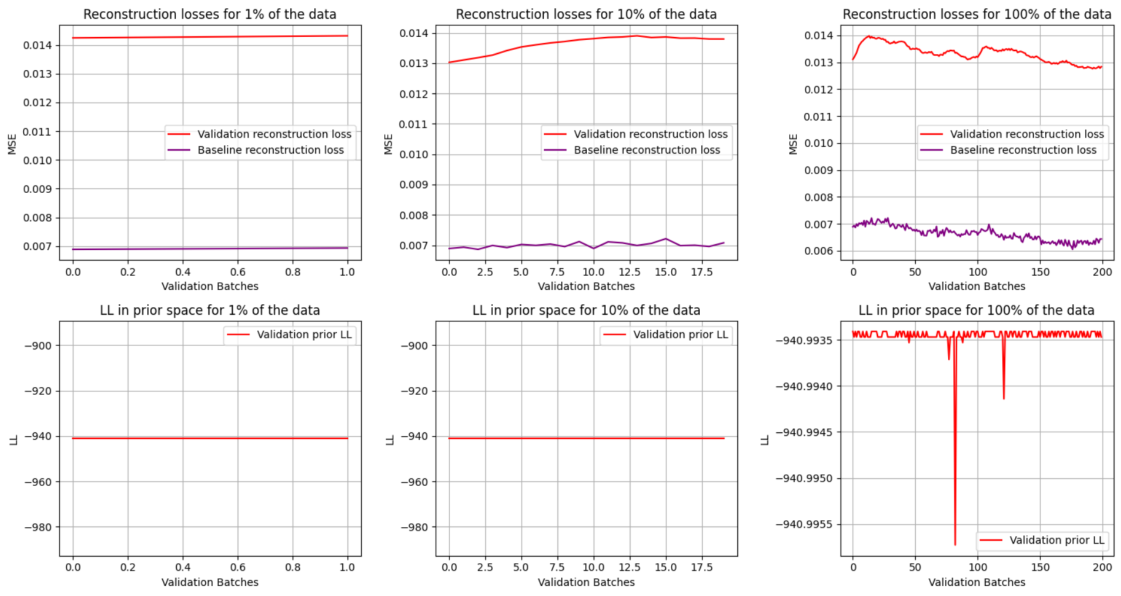

Figure 3: **Overfitting**. Each column represents the results obtained on a progressively larger data set

From these graphs we can conclude a few things. First of all, the performance is decidedly uniform across the different dimensions of overfit data, suggesting there is no limitation in this regard. Secondly, the log-likelihood measure of the forward pass seems to almost perfectly achieve the performance described earlier (i.e. the value of the mean value being passed). This decidedly shows that the CNF is mapping all of the future states to the mean of the prior distribution, such that it maximizes the log-likelihood. Unfortunately, this is incorrect, since it amounts to posterior collapse. During the development of this paper, we were unable to fix this issue. This heavily hinders the model's capacity to generate high quality samples, since in the backward pass, the sampled prior obviously does not only return the mean, as if it had 0 variance. Although this is the case, we find a very small reconstruction loss, barely double that of the baseline. This points to the rich modeling capability of the network since, despite all future states getting mapped on the same mean value, the generated samples are still decently close to a realistic future state. This is surely only due to the powerful modeling capacity of the conditional coupling layers.

5.2.2 Can the CNF generalize to similar data?

A natural next step in testing the capacity of the CNF is to overfit it to some percentage of the data, and validate it on the rest. Similarly to the previous experiment, we do this in 3 ratios: 80-20, 50-50, and 20-80. The results are presented in figure 4.

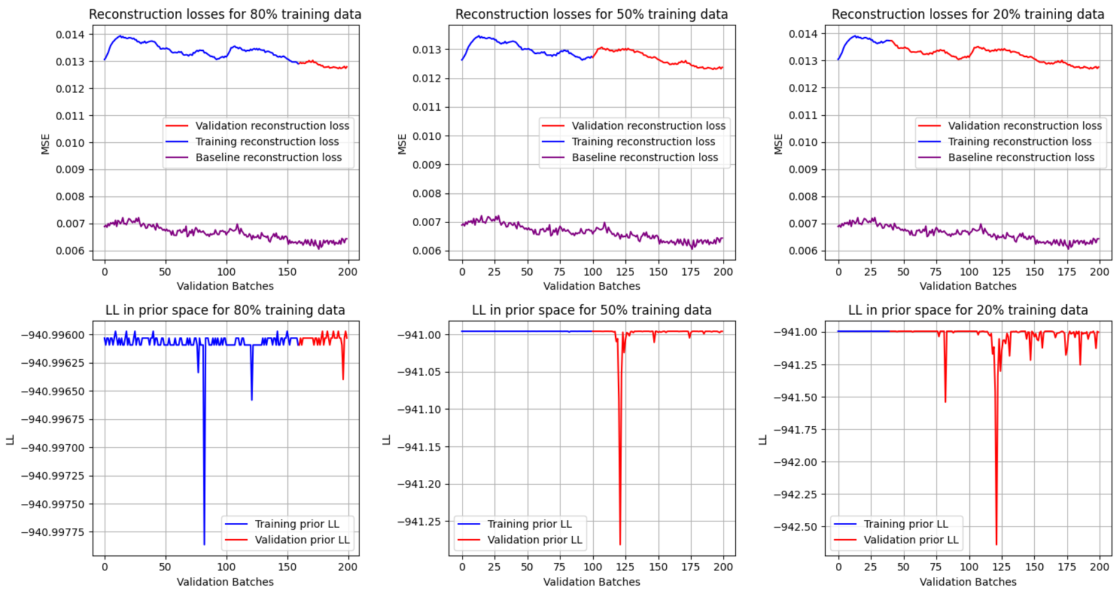

Figure 4: **Generalization on similar data**. Each column represents the results obtained on a progressively smaller training-to-validation set ratio

As expected, the same posterior collapse issue can be seen in this experiment as well, across all levels of testing. We can see a similar quality of reconstructions ar around two times worse than the baseline across the board, meaning that even though we have the problem of mapping everything to the mean, the flow still manages to output a decent quality sample approximating the future state. It is particularly revealing that for all three tests, the reconstruction loss curve looks practically identical, supporting the idea that the CNF generalizes very well to data similar to the one it used for training.

5.2.3 Can the CNF generalize to larger data?

A final test of the generalization power of the CNF constitutes on vastly different training and validation sets. For this purpose, we ordered the data with respect to the corresponding observation o_{t+1} size (i.e. the mean value of the tiles of the board). We then train on the first $X\%$ of this data and validate on the last $X\%$ of the data. As X decreases, the difference in the value of the observations widens, and thus becomes harder for the CNF to succesfully generate realistic samples on the meaningfully larger validation set. Once more, we executed this experiment on 3 values of X, namely 50, 12.5, and 3.125. The results can be found in figure 5. For clarity, Table 1 presents the difference in mean tile values as described above for the 3 different tests.

Although the posterior collapse is of course still present, we can see in these graphs that even though there is a significant drop-off in the quality of both reconstructions and mapping to the prior, even in the extreme case of $X = 3.125$, this difference remains sensible. It must also be noted that since there is less variance in

Table 1: Mean tile value in training and validation sets

X percentage	Training	Validation
50	15.01	31.8
12.5	9.9	43.2
3.125	5.85	52.22

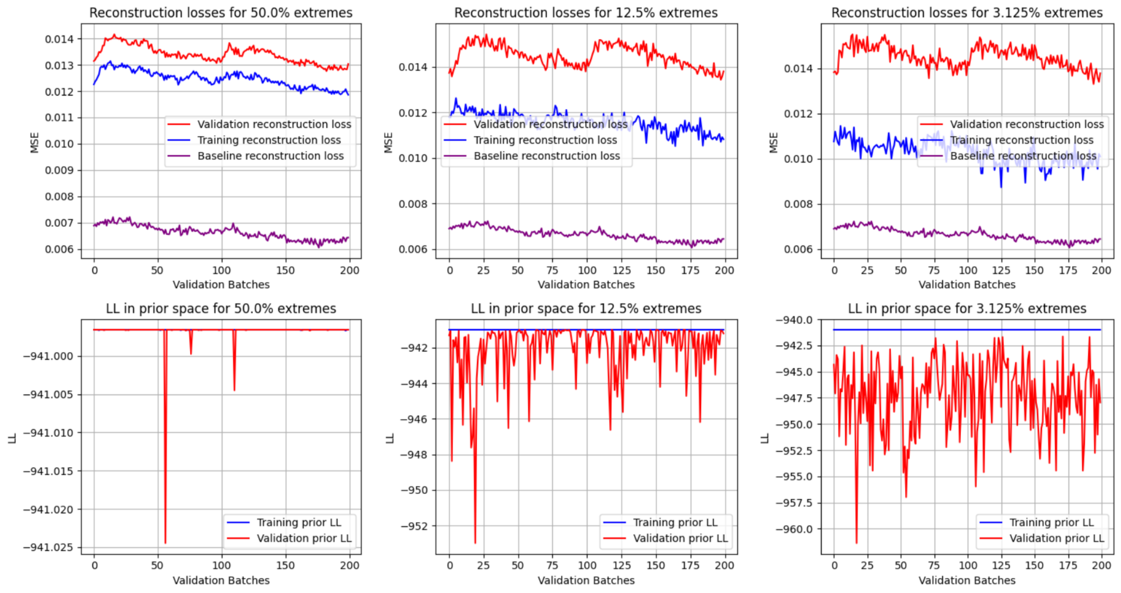

Figure 5: **Generalization on different data**. Each column represents the results obtained on progressively more extreme training and validation sets. The former tends towards very new games and the later tends towards very late games.

the data overfit during the training step (only the lower valued boards), the training reconstruction loss is lower than the previous experiments, which is to be expected. From these observations we can conclude that the CNF is capable of generalizing to vastly different afterstates with decent performance. This suggests that it would be well fit to realistically generate future states conditional on afterstates, which is its objective in the FlowZero network (the Afterstate Dynamics Network).

5.2.4 Does the FlowZero agent work?

Unfortunately, testing the FlowZero agent itself has proven unsuccessful. At best, the agent made effectively random moves, since the return it achieved never increased, even after 250k environment steps. For reference, the Stochastic MuZero agent achieved five times better results than the random baseline in the same timespan. Since the rather large failure of the CNF component to map the future states to the whole prior distribution, and not just its mean, has not been fixed during this project, we believe it is inadequate to conclude that there is any conceptual block that would disallow a CNF to successfully model the required stochastic transitions. It is rather more likely that the posterior collapse introduces a hard cap on the expressiveness of the network's predictions which leads to the loss of crucial details for the prediction of value, reward, and policy that is paramount to the MuZero strategy of learning. Thus, it is our strong suggestion for future work to tackle this issue and check the performance of the CNF in the FlowZero architecture as a whole.

6 Conclusions and Future Work

In this paper, we proposed an alternative to [2]'s solution to the problem of enabling MuZero to work in stochastic environments through the novel architecture of FlowZero. Our agent replaces the Afterstate Dynamics Network of Stochastic MuZero with a Conditional Normalizing Flow that seeks to model future states as a distribution conditional on the current afterstate that seeks to directly maximize log-likelihood instead of maximizing a lower bound. This led the model to disregard the realistic mapping to a prior distribution and thus collapsed to the mean. A strong suggestion for the future is to introduce a loss term that incentivizes accurate variance modeling as well as modal. This is the principal limitation of this work, since we believe it is the largest contributor to our failure to obtain results in the context of the whole model. However, we proved the CNF's capacity to overfit realistic data, generalize to similar data as its training, and decently generalize to vastly different data. Another major limitation is the testing of a singular -relatively small observation- environment. If we were to run our generalization experiments on a Backgammon-based dataset for instance, we would certainly require a more complex CNF to obtain low reconstruction losses.

7 Responsible Research

Our work introduces a novel alternative to stochastic MuZero [2] for planning in environments with high uncertainty. While this model may benefit applications such as robotics and games, it also carries potential risks.

Enhanced decision-making under uncertainty could be misused in surveillance [1] or military AI applications [28] where safety and ethical boundaries are less regulated.

Reproducibility is a significant consideration in such work, to facilitate easy verification and responsible reuse. The code developed under this project is available on demand. Perfect reproduction of our results is impossible, since the core of the agent works around stochasticity, randomly sampling from a gaussian distribution at every single step. Additionally, there are differences in hardware architecture and approximations in float calculations, but the overarching trend highlighted by our experiments is undoubtedly verifiable independently. A list of hyperparameters is not necessary since all of the implementation details can be found in section 5.1.1.

During the work described in this paper we made use of the Large Language Model (LLM) ChatGPT. In the beginning, it sped up research by suggesting useful starting points in wider domains (density estimation, stochastic MBRL agents). During the development of the agent itself, the model was prompted for specific methods of the PyTorch framework given some circumstance, replacing the cumbersome browsing of the classic documentation when needing some specific functionality.

References

[1] Sabrina Aberkane and Mohamed Elarbi. Deep reinforcement learning for real-world anomaly detection in surveillance videos. In *2019 6th International Conference on Image and Signal Processing and their Applications (ISPA)*, pages 1–5, 2019.

[2] Ioannis Antonoglou, Julian Schrittwieser, Sherjil Ozair, Thomas K Hubert, and David Silver. Planning in stochastic environments with a learned model. In *International Conference on Learning Representations*, 2021.

[3] Zdravko I. Botev, Dirk P. Kroese, Reuven Y. Rubinstein, and Pierre LâEcuyer. Chapter 3 - the cross-entropy method for optimization. In C.R. Rao and Venu Govindaraju, editors, *Handbook of Statistics*, volume 31 of *Handbook of Statistics*, pages 35–59. Elsevier, 2013.

[4] E. F. Camacho and C. Bordons. *Introduction to Model Predictive Control*, pages 1–11. Springer London, London, 2007.

[5] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *CoRR*, abs/1805.12114, 2018.

[6] Adrien Couetoux. *Monte Carlo Tree Search for Continuous and Stochastic Sequential Decision Making Problems.* Theses, Université Paris Sud - Paris XI, September 2013.

[7] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp, 2017.

[8] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav DudÃk, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[9] Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *CoRR*, abs/1912.01603, 2019.

[10] Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *CoRR*, abs/1811.04551, 2018.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[12] Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces. *CoRR*, abs/2104.06303, 2021.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[14] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[15] Matej Moravcík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael H. Bowling. Deepstack: Expert-level artificial intelligence in no-limit poker. *CoRR*, abs/1701.01724, 2017.

[16] OpenAI. DallÂ·e 3 technical overview. `https://openai.com/dall-e-3`, 2023. Accessed: 2025-06-17.

[17] OpenAI. DallÂ·e 3 technical overview. `https://openai.com/sora`, 2023. Accessed: 2025-06-17.

[18] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

[19] M L Puterman. *Markov decision processes.* Wiley Series in Probability & Mathematical Statistics: Applied Probability & Statistics. John Wiley & Sons, Nashville, TN, May 1994.

[20] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *CoRR*, abs/2112.10752, 2021.

[21] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604â609, December 2020.

[22] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484â489, January 2016.

[23] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[24] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[25] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[26] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016.

[27] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018.

[28] Ning Wang, Zhe Li, Xiaolong Liang, Yueqi Hou, and Aiwu Yang. A review of deep reinforcement learning methods and military application research. *Math. Probl. Eng.*, 2023(1), January 2023.

[29] Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows, 2023.