Translating Latent State World Model Representations into Natural Language

Matthew Barker
King's College London
London, United Kingdom
matthew.s.barker@kcl.ac.uk

Matteo Leonetti
King's College London
London, United Kingdom
matteo.leonetti@kcl.ac.uk

ABSTRACT

Recent successes in model-based reinforcement learning have stemmed from models that learn a latent representation of the world. However, these latent representations are unintelligible, meaning that we cannot interpret the agent's internal world representation, nor any plans made in this latent space. In this work we present Somniloquy, an algorithm that learns to translate latent state plans into a natural language description that captures what the plan means in terms of the agent's expected interaction with the world. We demonstrate that latent plan translations can be learned in tandem with the latent representations, whilst giving the latent representations an additional learning signal to be translatable, and that Somniloquy enables a deep model-based reinforcement learning agent to verbalise its latent plan in natural language prior to acting. In addition to interpretability, we show that Somniloquy enables defining desired behaviour in language by rewarding latent states whose translation matches the requested behaviour. Importantly, this requires no reward signal at any point from the environment. We demonstrate experimentally that, in deterministic environments, Somniloquy's plan translation accurately describes the plan's execution, and that in the stochastic setting the translations of multiple latent plan rollouts can approximate the true environment dynamics. Finally, we demonstrate that our translation reward function approach successfully trains policies to achieve goals specified in natural language, and achieves on-par performance with training an agent that has access to each natural language goal's unobservable extrinsic reward function.

KEYWORDS

Model-Based Reinforcement Learning; Natural Language Processing; World Models

ACM Reference Format:
Matthew Barker and Matteo Leonetti. 2026. Translating Latent State World Model Representations into Natural Language. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 10 pages. https://doi.org/10.65109/GUWU9511

1 INTRODUCTION

Recent advances in Model-Based Reinforcement Learning (MBRL) have been driven by algorithms that learn a latent representation of the environment's state-space, such as PlaNet [14] and the Dreamer family of models [13, 15, 16]. Unlike traditional approaches, learned latent representations enable efficient automated abstraction and compression, and afford the ability to model partially observable environments. However, the learned latent representation is unintelligible to us humans. When an agent plans a sequence of actions and latent states to achieve a given goal, we thus have no way of knowing (beyond the action choice) what the agent plans to do, nor what states it expects its actions to result in. For high-stake scenarios, ambiguous goals, or changeable environments, the ability to know what an agent plans to do before it acts is vital: knowing what dosage of medicine a hospital assistive robot is planning on giving a patient; which possessions a household robot tasked to "clean the room" is planning to throw away or store; and whether the same household robot plans on bringing you a glass of red wine via your newly re-carpeted room. Furthermore, based on the agent's plan verbalisation, we may wish to specify different behaviour to the agent in natural language, e.g., "please take the wine the long way round via the hallway".

Faithfully knowing what an agent plans to do and expects to happen necessitates an explicit link between the learned latent representation and natural language. Action verbalisation is not sufficient as it is lacking state information, that is, how the world is to be affected by the agent's actions, according to the model. Consider a household robot that states it will "wipe the table". Without state information, we cannot know if the robot expects objects on the table, what it would do with any objects it found, what dirt is on the table, is the correct cleaning tool to be used, etc. Thus, to fully understand an agent's latent plan, both state and action information needs to be incorporated into any translation.

In this work we present a method for directly translating an agent's latent state plan into a natural language description that captures both what the agent plans to do and how this plan is expected to impact the state of the world. We use Dreamer [16] as the underlying world model architecture, in which planning in latent state space is referred to as *dreaming*. Therefore, we dub our method Somniloquy (meaning sleep talking). An overview of Somniloquy can be found in Figure 1.

As well as demonstrating how Somniloquy enables an interpretability window into latent representations, we show that having a direct connection between natural language and latent representations naturally affords the ability to learn multiple policies for different goals inside of the latent model, by specifying goals in pure natural language, without any reward signal from the environment, something that is not possible with Dreamer, due to its dependence on learning a mapping from latent states to reward in order to learn a policy. As Somniloquy can describe latent states

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/GUWU9511

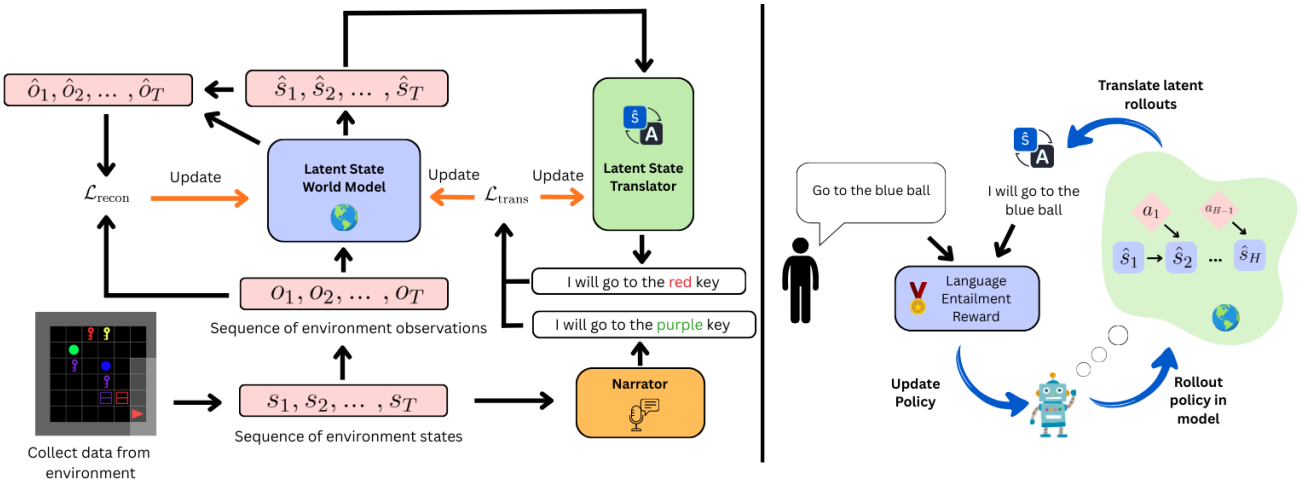

Figure 1: We introduce Somniloquy, an algorithm that augments a latent state world model with a translator component, that learns how to translate latent representations into natural language, whilst giving the representations an additional signal to be translatable (left). Having a way to describe latent states in natural language allows a deep RL agent to verbalise its latent state plan prior to acting, and allows learning of tasks specified by natural language inside of the world model (right).

in natural language, given a natural language goal, we show that a reward function can be created to turn policy search into the problem of seeking out latent states whose translation matches said goal. We demonstrate empirically that an agent trained under such a reward function achieves on par performance with an agent that has access to the natural language goal's corresponding unobservable extrinsic reward function.

Code and Supplementary Material can be found at https://m-barker.github.io/AAMAS-2026-Somniloquy/

2 RELATED WORK

Rationale Generation. The term rationale generation was first coined in the works by Ehsan et al. [8, 9] as "an approach for real-time explanation generation whereby a computational model learns to translate an autonomous agent's internal state and action representations into natural language" [9, p.263]. Their work generates a human-like explanation as to why an agent just took a single action in a given state, using a human annotated dataset of explanations, states, and actions, to train an encoder-decoder Recurrent Neural Network (RNN) in a supervised manner. Subsequent works have used task-success rate metrics to explain actions [6], used attention to ground the explanations to states the agent deems important [40], and extended the general framework to explain a past sequence of states and actions [28]. More recent works have demonstrated how LLMs can be used to generate narrations and explanations of an agent's behaviour [24, 42]. To the best of our knowledge, all prior works in this area generate explanations of *current or past*, non-latent, states and actions, as opposed to our work which aims to translate *future* states and actions in the form of a latent state plan. Whilst explanations are richer than translations as they focus on the *why* rather than the *what*, we hope that our work may lay groundwork for building towards explanations of an agent's latent state-action plan.

Plan Verbalisation. Some existing works allow an agent to state its plan in natural language prior to acting. One such body of research has demonstrated how symbolic plans can be mapped to natural language descriptions [3, 11, 35], and how LLMs can summarise robotic task plans [42]. However, these are constrained to plans output by symbolic planners, which severely limits their applicability. More recently, there have been many works that use an LLM to generate plans for an agent [2, 18, 19, 32, 37]. These LLM-based planners typically output a sequence of high-level natural language actions that are mapped to (sequences) of lower-level actions. Whilst an LLM can also output a description of the 'states' it expects its plan to result in, this prediction is not grounded in experience, beyond the information provided to the LLM in the prompt. This is fundamentally different to Somniloquy, whose latent plan translation is inherently grounded in the world as the states the plan is expected to result in come from predictions of a world model. It is also far from clear whether LLMs can actually plan [38], limiting the applicability of LLM-based planners, especially to novel and/or complex tasks. Other approaches have utilised learning from demonstration algorithms to simultaneously produce natural language sub-goals and learn a policy conditioned on them [17, 36]. However, there is no notion of 'state' for the sequence of natural language sub-goals and this is thus not a translation of a latent state plan. As discussed in the introduction, state information is a necessity to capture how the agent's actions are expected to impact the world. In concurrence with our work, Wu et. al. [44] fine tunes a Vision Language Model (VLM) to translate a pre-trained Dreamer world model's latent state representation into natural language, which enables plan verbalisation. However, their translation approach is done add hoc with a pre-trained, frozen, world model, as opposed to our method which couples learning the translation with learning the world model; their approach of fine-tuning a VLM uses orders of magnitude more parameters for the translation than our approach; and their downstream application for the translation is to

score action samples from a pre-trained policy, which differs from our downstream application which instead turns the translation into a reward function to learn new policies from scratch.

Language World Models. Recent works have explored integrating language directly into a world model. Most closely related to this paper are the works by Liu et al. [23] and Lin et al. [22]. The work by Liu et al. translate single latent state transitions into sentence embeddings describing the transition. This is used to assign an intrinsic reward based on embedding similarity with an LLM suggested exploration goal. Sentence embeddings themselves are not interpretable, so this prevents translating latent states into natural language. The work by Lin et al. embeds natural language into the latent representation learned by Dreamer [16] to form a 'multi-modal' world model called 'Dynalang'. They assume that the environment produces a single language token at each timestep, which is encoded into the learned latent representation. They use language primarily as an additional observation modality to better model the world and hence the language observed is not a representation of the latent state and thus does not allow an agent to verbalise their latent plan; instead, it focuses on providing instructions, rules, and hints to the Reinforcement Learning (RL) agent. Similarly, other works [31, 45] have integrated language into the observation space in order to improve the generalisability of world models and agents, but do not afford any plan translation or verbalisation.

Language to Reward in Latent World Models. Some recent works enable mapping from natural language to reward in latent state world models, without requiring extrinsic reward from the environment, for RL [23, 25, 41] and for Imitation Learning [27]. These works learn to map the natural language goal and latent states to a shared embedding space (often encoding the goal using a VLM, and then mapping VLM embeddings to latent state space [25, 41]). Policy search inside the model then becomes the question of 'reach the latent state that is closest to the natural language goal in this shared embedding space'. Our translation reward method takes a different approach, which enables both interpretability of latent states and turning natural language into reward, by turning policy search into the question of 'reach latent states whose translation entails that the natural goal as been reached', akin to the entailment as reward framework used for natural language processing tasks as discussed by Roit et. al. [34].

3 PRELIMINARIES - LATENT WORLD MODELS

Many RL environments are *partially observable*, in which the agent does not have access to the true environment state space and instead only has access to observations. Such environments are typically modelled as a Partially Observable Markov Decision Process (POMDP) which consists of a 7-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{X}, \mathcal{O}, \gamma \rangle$, where $\mathcal{S}$ denotes the set of true, unobservable, environment states, $\mathcal{A}$ denotes the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function that gives the probability that the environment state moves to another state s' given the previous state s and action a, i.e., $\mathcal{T}(s, a, s') = \Pr(\mathcal{S}_{t+1} = s' \mid \mathcal{S}_t = s, \mathcal{A}_t = a)$, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which gives an immediate scalar to an agent after it takes an action in a state, $\mathcal{X}$ denotes the set of observations, $\mathcal{O} : \mathcal{S} \times \mathcal{X} \to [0, 1]$ denotes the emission function that gives the the probability that the agent observes x given the true state s, i.e., $\mathcal{O}(s, x) = \Pr(\mathcal{X}_t = x \mid \mathcal{S}_t = s)$, and $\gamma \in [0, 1)$ denotes the discount factor.

The objective of the RL agent is to learn a policy that outputs the action that the agent should take at each timestep in order to maximise the expected discounted sum of rewards, $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1}]$. One model-based way to approach POMDPs is to learn a world model that outputs $\hat{\mathcal{S}}$, an approximate of the true state space $\mathcal{S}$. The policy learned by the RL agent is then conditioned on the latent approximation $\hat{s}_t \in \hat{\mathcal{S}}$, which is estimated by the world model at each timestep, i.e., $\pi(a_t \mid \hat{s}_t)$. We dub such an approach a 'latent state space world model'.

One of the state-of-the-art latent state space world models is the Dreamer family of algorithms, which have been shown to: solve diverse control tasks (Dreamer V1) [13]; master Atari video games (Dreamer V2) [15]; and achieve high performance across diverse domains with a fixed set of hyperparameters (Dreamer V3) [16]. Due to Dreamer's performance, popularity, and open source code base, we utilise the Dreamer V3 algorithm (Dreamer, hereafter) to learn the latent representation that Somniloquy translates, and also to learn the agent's behaviour.

The backbone of Dreamer is a Recurrent State Space Model (RSSM) [14] which learns a latent representation $\hat{s}$ that consists of two components: a deterministic component h, modelled using an RNN, and a stochastic component z, modelled by a Multi-Layer Perceptron (MLP) that parameterises a discrete distribution. Full details of Dreamer's architecture can be found in the Supplementary Material. In short, the RSSM is trained in an autoencoder-like manner, where environment observations x_t are encoded and used as input to learn z_t, and latent states (h_t, z_t) are decoded and trained to reconstruct observations $\hat{x}_t$, predict rewards $\hat{r}_t$, and predict non-terminal states $\hat{c}_t$. The RSSM also features a dynamics predictor that aims to mimic the 'true' stochastic component z_t *without* access to the environment observation, which enables 'dreaming' (planning) with the world model. Given a starting latent state (h_t, z_t) and a policy $\pi(a_t \mid (h_t, z_t))$, the RNN and dynamics predictor yield $(h_{t+1}, \hat{z}_{t+1})$, which gives a_{t+1} using the policy. This process can be repeated to yield a latent plan of arbitrary length. We refer to latent state plans inside the world model as *plan rollouts*, and we refer to carrying out a plan in the environment as *plan execution*.

4 TRANSLATING LATENT REPRESENTATIONS

Somniloquy learns how to translate sequences of latent states into natural language using an encoder-decoder transformer architecture [39], as often used in the field of machine translation. The encoder block takes in a fixed length sequence of latent states (the source language) and encodes them into a sequence of vectors. The decoder takes this embedding sequence and autoregressively generates the output sequence of natural language tokens (the target language).

Given that we have a sequence of latent states $\hat{s}_{0:M}$ from Dreamer, where $\hat{s}_{0:M} \doteq (\hat{s}_0, \cdots, \hat{s}_{M-1})$, we generate its natural language translation as follows: first, we pass the latent state sequence through a single, fully connected linear embedding layer $f_\phi(\cdot)$, to transform the continuous and discrete latent state vectors (as z is the mean of a discrete distribution) into a sequence of continuous vectors of

a given fixed dimension (d): $\hat{s}'_{0:M} = \langle f_\phi(\hat{s}_0), f_\phi(\hat{s}_1), \ldots, f_\phi(\hat{s}_{M-1}) \rangle$. We then pass the sequence of continuous latent states to an encoder transformer block, which encodes the sequence into a sequence of vectors $\epsilon_{0:M}$. The encoded sequence $\epsilon_{0:M}$ is then passed to a decoder transformer block, which autoregressively generates the sequence of natural language tokens $\hat{l}_{0:N}$, where N is the enforced maximum length of any generated sequence, which gives us the model's latent state translation.

More formally, the transformer blocks are trained to model the following:

$$\text{Transformer Encoder: } \epsilon_{0:M} = T_{enc}(\hat{s}'_{0:M})$$
$$\text{Transformer Decoder: } \hat{l}_n \sim T_{dec}(\hat{l}_n \mid \epsilon_{0:M}, \hat{l}_{<n})$$

Where T indicates a transformer block; $\hat{l}_n$ are the natural language tokens output by the decoder transformer; and n is the index of the latest generated token of the output sequence.

The encoder-decoder transformer is trained using *teacher forcing* [43], which gives the transformer decoder block the ground truth tokens $l_{<n}$, as opposed to the generated tokens $\hat{l}_{<n}$, when producing element n of the output sequence. At test time, the decoder does not have access to the ground-truth tokens, and thus is conditioned on $\hat{l}_{<n}$ when not in training. The loss function is the standard Cross-Entropy loss between generated and ground-truth tokens, summed across all tokens in a given translation, and averaged across batches.

For ease of reference, we refer to the whole translation procedure described above using a single function $f_{\text{trans}} : \hat{\mathcal{S}}^M \rightarrow \ell^N$, mapping from a sequence of latent states of length M to a sequence of natural language tokens of length N.

When updating the model parameters with the translation loss, we have two choices: (a) allow the translation loss to flow through the world model, in which case the new world model loss is as defined below:

$$\mathcal{L}_{\text{world\_new}} = \beta_{\text{trans}} \mathcal{L}_{\text{trans}} + \mathcal{L}_{world},$$

with β indicating the translation loss weighting; or (b) detach the latent states from the computational graph *before* passing them to the transformer, to prevent the translation loss impacting the learned latent representation. We conduct experiments under both choices and we present evidence that option (a) performs similarly to (b) in terms of latent state translation, whereas option (a) can also improve task performance, and thus by default Somniloquy uses option (a).

Training the transformer requires pairs of latent state sequences $\hat{s}_{0:M}$ and ground-truth translations $l_{0:N}$. To obtain the ground-truth translations, we need a way to obtain approximate descriptions of the true environment state sequence $s_{0:M}$. Somniloquy is agnostic to where these descriptions come from, e.g., human or Vision Language Model (VLM) descriptions of an agent's environment interactions. For proof of concept and ease of experimentation, we create our own rule-based narrator for each environment, that takes as input a sequence of environment states $s_{0:M}$, and outputs a corresponding natural language description that captures the details of the agent's interaction with the environment in this sequence. The assumption of a narrator is consistent with prior works that assume the existence of state captioners [7, 20, 23, 26, 28]. The narrator is only required for supervision during training and can be configured to output descriptions that contain whatever information the designer deems relevant, at any level of granularity. As almost all deep RL is trained in simulation, we are able to assume access to (approximate) state information during training, afforded to us by the simulator (e.g., robot, landmark, and object positions and semantics). Crucially, we do not assume that any of this information is available at test time (e.g., deployment on a real robot). No state information is an input to any model component and is used purely for supervision; the world model only has access to observations.

We augment the Dreamer training loop to include the above translation procedure, as given in Algorithm 1, with our additions in lines 14-20. At training time, Dreamer samples batches of fixed length sequences of observations $x_{0:T}$, rewards $r_{0:T}$, and continuation flags $c_{0:T}$ from the replay buffer (Line 10) , and uses this to produce the corresponding sequence of latent states $\hat{s}_{0:T}$ (Line 11). We include the environment states required by the narrator into the replay buffer, which additionally yields us $s_{0:T}$ (Lines 7 & 10). All sequences are then reshaped temporally to yield batches of sequence length M (Line 14), where M dictates the maximum latent state sequence length for translation.

The batches of sequences of environment states $s_{0:M}$ are then passed to the narrator to yield the set of (proxy) ground truth latent state sequence description natural language strings, one string per sequence (Line 15). The strings are then word tokenised (i.e., each unique word that the narrator can output is mapped to a unique integer) to yield the sequence of ground truth tokens for each batch $l_{0:N}$ where l denotes a language token and N denotes the maximum token sequence length, where appropriate padding is given to any sequence with a number tokens less than N. Finally, the batches of ground truth tokens $l_{0:N}$ and their corresponding latent state sequence $\hat{s}_{0:M}$ are passed to the translator component and used to compute the translation loss (Lines 16-20).

5 FROM LANGUAGE GOALS TO REWARD

In addition to making the latent state representations of world models interpretable, which enables understanding of the expected behaviour and effects of agent's plan prior to the agent acting in the world, Somniloquy also affords the ability to turn natural language goals into a reward function. In turn, as we will demonstrate, this language reward function enables training a policy inside of the world model, which, once deployed in the real environment, successfully achieves said goal.

Somniloquy provides a method for going from latent states to natural language, thus we can turn policy search inside of the model into the problem of 'choose actions to result in latent states whose translation matches the specified natural language goal/behaviour', and reward the policy for reaching such states accordingly. Importantly, this does not require any extrinsic reward signal from the environment, and allows specifying any goal or behaviour that the narrator (and hence latent translator) can describe.

5.1 Rewarding Latent States from Translations

The central idea is to reward latent states whose translation implies that the requested natural language goal/behaviour has been achieved, under the assumption that the natural language request

Algorithm 1 Somniloquy World Model Training

Parameters: plan length M, max language sequence length N
while acting **do**
  Step environment: $r_t, c_t, x_t, s_t \leftarrow \text{env}(a_{t-1})$
  Step sequence model: $h_t \leftarrow f_\phi(h_{t-1}, z_{t-1}, a_{t-1})$
  Encode observations: $z_t \sim q_\phi(z_t \mid h_t, x_t)$
  Sample policy: $a_t \sim \pi(a_t \mid h_t, z_t)$
  Add transition $(r_t, c_t, x_t, s_t, a_{t-1})$ to replay buffer
end while
while world model training **do**
  Sample batch $\{r_t, c_t, x_t, s_t, a_{t-1}\}$ from replay buffer
  Use world model to compute $\{h_t, z_t, \hat{z}_{t+1}, \hat{r}_t, \hat{c}_t, \hat{x}_t\}$
  Compute $\mathcal{L}_{\text{pred}}, \mathcal{L}_{\text{dyn}}, \mathcal{L}_{\text{rep}}$
  Define modelled latent state: $\hat{s}_t \leftarrow \text{concatenate}(h_t, z_t)$
  Chunk data into sequences of $\{s_m, \hat{s}_m\}_{m=0}^{M}$
  Compute ground truth plan descriptions: $\{l_n\}_{n=0}^{N} \leftarrow \text{narrator}(s_{0:M})$
  Compute continuous latent states: $\{\hat{s}'_m\}_{m=0}^{M} \leftarrow \{f_\phi(\hat{s}_m)\}_{m=0}^{M}$
  Encode: $\epsilon_{0:M} \leftarrow T_{\text{enc}}(\{\hat{s}'_m\}_{m=0}^{M})$
  Compute translated latent state plans: $\{\hat{l}_n \sim T_{\text{dec}}(\hat{l}_n \mid l_{<n}, \epsilon_{0:M})\}_{n=0}^{N}$
  Compute translation loss: $\mathcal{L}_{\text{trans}} \leftarrow \text{CrossEntropy}(l, \hat{l})$
  Update world model and translator parameters to minimize: $\mathcal{L}_{\text{model}} \leftarrow \beta_{\text{pred}}\mathcal{L}_{\text{pred}} + \beta_{\text{dyn}}\mathcal{L}_{\text{dyn}} + \beta_{\text{rep}}\mathcal{L}_{\text{rep}} + \beta_{\text{trans}}\mathcal{L}_{trans}$
end while

maps to a set of true environment states. This rewarding method brings up two questions: First, what does it mean for a natural language translation to imply the specified natural language goal or behaviour? And second, given we have ascertained that the translation of a sequence of latent states implies the requested behaviour, which state(s) in the sequence should get rewarded? The former can be viewed as the problem of textual entailment, which has its own rich literature as a sub-field of Natural Language Processing (NLP) [1, 30], which is beyond the scope of this paper. For our purposes, we use a simple definition of textual entailment that considers the latent state translation as entailing the given language goal if the language goal is a substring of the translation. Whilst this may appear limiting, since we have full control over the design of the environment's narrator, we can configure its output to ensure that this definition of entailment is correct. More flexible approaches, which we aim to explore in future work, could use a metric of similarity between the language goal and the latent state translation, such as the BLEU score [29], or semantic similarity metrics that use sentence embeddings [33], or Large Language Models (LLMs) as a judge [10].

For the temporal reward assignment problem, the simplest naive way is to translate the entire latent state plan for the given planning horizon, and give a positive reward to the final latent state in this sequence if it entails the goal. However, this method leads to credit being assigned to states that were of no relevance to reaching the goal. For example, for a horizon of length H, let the given goal be achieved at timestep $t < H$. Assigning a positive reward to state $\hat{s}_{H-1}$ leads to credit being incorrectly assigned to states $\hat{s}_{t+1:H}$, hindering policy learning (as we show empirically in the subsequent section). Instead, a better approach is to assign a non-negative reward to the first state $\hat{s}_t$ where the translation of states $\hat{s}_{0:t+1}$ entails the given language goal.

To that end, given a natural language goal ℓ_g, we define a reward function as follows:

$$R(\hat{s}_t) = \begin{cases} 1 & \text{if } f_{Trans}(\hat{s}_{0:t+1}) \vDash \ell_g \ \wedge \ f_{Trans}(\hat{s}_{0:t}) \nvDash l_g \\ 0 & \text{otherwise} \end{cases}$$

Where we use the aforementioned sub-string definition of entailment for our experiments. The above reward function can then be used to train a policy to achieve some natural language goal ℓ_g. Importantly, this reward function requires no extrinsic reward from the environment: given an environment and its narrator, Somniloquy can be trained as detailed in Section 4, minus the reward and termination prediction heads of the world model, under any exploration policy that does not require extrinsic reward, to gather the necessary training data. After training, the world model is frozen, and, as we demonstrate empirically in the subsequent section, a policy can be trained entirely inside of this pre-trained model (i.e., without requiring any interaction with the real environment), under the above reward function, to achieve a given natural language goal. The training procedure is outlined in Algorithm 2, with the policy architecture, along with the updating of the policy parameters (line 14), following the same method as Dreamer [16].

Algorithm 2 Somniloquy Natural Language Goal to Policy

1: **Input:** batch size B, plan horizon H, policy π_θ, start state $\hat{s}_0$, language goal ℓ_g, pre-trained Somniloquy model $\mathcal{M}$
2: **while** π_θ not converged **do**
3: $\hat{\mathbf{s}}^{B\times H}, \mathbf{a}^{B\times H} \leftarrow \text{model\_rollout}(\mathcal{M}, \hat{s}_0, B, H, \pi_\theta)$
4: rewards, continues $\leftarrow \mathbf{0}^{B\times H}, \mathbf{1}^{B\times H}$ ▷ matrices of zeros and ones
5: **for** $b = 0 \ldots B$ **do**
6: **for** $t = 0 \ldots H$ **do**
7: **if** $f_{\text{Trans}}(\hat{s}^{(b)}_{0:t+1}) \vDash \ell_g$ **then**
8: $\text{rewards}^{(b)}_t \leftarrow 1$
9: $\text{continues}^{(b)}_{t:H} \leftarrow \mathbf{0}^{H-t}$ ▷ goal reached, episode ends
10: **break**
11: **end if**
12: **end for**
13: **end for**
14: $\theta \leftarrow \text{update\_policy}(\theta, \hat{\mathbf{s}}, \mathbf{a}, \text{rewards}, \text{continues})$
15: **end while**
16: **return** π_θ

6 EXPERIMENTS AND RESULTS

Our experiments seek to test: (a) whether Somniloquy faithfully translates latent state representations into natural language; and (b), the performance of a policy trained entirely inside the world model under our language-to-reward function. Specifically, we aim to answer the following research questions through our experiments:

(1) In deterministic environments, to what extent does Somniloquy generate a latent plan translation that matches the true description obtained from executing the plan?
(2) In stochastic environments, to what extent does the translation of multiple latent plan rollouts capture the environment's stochasticity?

(3) What impact does allowing the translation loss to shape the learned latent representation have on task and translation performance?
(4) Does training an agent fully inside of the world model under our language-to-reward function result in a policy that successfully reaches goals specified by natural language, once deployed for testing in the real environment?

6.1 Evaluation Procedures

6.1.1 Translation Evaluation Procedure. To evaluate the translation performance of Somniloquy, we need a surrogate ground-truth for the latent plan translation as we do not have access to the actual ground-truth description, as this requires knowledge as to what the latent states output by the world model represent. We propose using the environment's narrator output from the executed plan as the surrogate ground truth. For a given latent plan, we first compute its translation. Then, we execute the plan in the environment and pass the sequence of required environment states (afforded to us by the simulator) to the narrator to generate the ground-truth description of the agent's behaviour. The closer the similarity between the plan translation and the behaviour description, the higher our measure of translation efficacy. However, if there is a mismatch between the translation and the narration, we cannot distinguish between cases where the mismatch occurs due to errors in the translation and cases due to the errors in the model.

In deterministic environments, if we assume that the world model is correct, then our metric of efficacy does capture the first of our stated requirements: what the agent states that it will do, and how the world is expected to be impacted by its actions, is what actually happens. We thus utilise this metric of plan translation - plan execution description similarity as our translation performance score in deterministic environments, repeating the plan translation - plan execution loop until either the episode actually terminates, or the world model predicts the episode terminates, for multiple episodes at each evaluation epoch. The full evaluation procedure for deterministic environments can be found in Algorithm 1 in the Supplementary Material.

For stochastic environments, even if we assume a perfect world model, we cannot use the similarity metric, as differences between the plan translation and the plan execution description will occur due to the stochastic nature of environment transitions. Instead, we propose computing multiple plans, and their translation, from the same initial starting state. We can then compare the implied transition probabilities with the true transition probabilities. For example, if there is a 50% chance that taking an action moves the agent North, and a 50% chance the same action moves East, then one would expect 50% of plan translations to state "I will move North" and the other 50% to state "I will move East", assuming that the world model accurately estimates the environment dynamics. We use the distance between the true dynamics and the empirically estimated dynamics as a measure of efficacy in stochastic environments. For each unique stochastic transition in the environment that the agent plans to go through, we compute the Total Variation Distance (TVD) between the true transition distribution and the empirically estimated distribution, and then average this distance across transitions. The full evaluation algorithm can be found in Algorithm 2 in the Supplementary Material.

6.1.2 Language to Reward Evaluation Procedure. We consider a setting in which the environment offers no extrinsic reward signal. We train Somniloquy in such an environment for a fixed number of environment steps, as outlined in Section 4, minus the reward and continuation prediction heads, using an arbitrary exploration policy (uniform random for our experiments) to gather the data for training the model. The parameters of the world model are then frozen for the rest of the evaluation. We then use the same frozen model to train a policy from scratch inside of the model for a variety of goals specified in natural language, one policy per goal, as outlined in Algorithm 2.

We report the mean modelled reward (which corresponds to the modelled success rate, due to the sparse reward nature of our reward function) across batches of policy rollouts during training. Additionally, at fixed intervals during policy training, we execute the policy in the real environment, with a step-limit equal to the horizon length, and compute the mean success rate in reaching the ground truth state that the language goal is assumed to encode, across 10 evaluation episodes.

We compare the performance of our method to two other reward modelling approaches. The first we call 'naive-language-rewarding', which simply translates the entire sequence of rolled out latent states, and assigns a reward to 1 to the final latent state in the sequence if this translation implies that the goal has been reached, and 0 elsewhere. The second is a powerful baseline, dubbed 'extrinsic-rewarding' that, for each language goal, annotates the set of transitions used to train the world model with reward data capturing if the true corresponding goal state was reached, in essence assuming availability of the true extrinsic reward function during training. We then train a reward prediction head $f_{\text{reward}} : \hat{\mathcal{S}} \to \mathbb{R}$ for each language goal using the reward labelled dataset, and the corresponding latent states of each transition as input. Both methods utilise the same frozen world model as our method. The only difference in policy learning between the methods comes from how rewards are assigned to the latent world model states.

6.2 Experimental Setups

6.2.1 Translation Performance. We evaluate Somniloquy across two deterministic environments and one stochastic environment. We use two diverse deterministic environments: Crafter [12], a procedurally generated environment in which an agent must survive, harvest resources, and craft items to receive reward, akin to a simplified 2D version of Minecraft; and AI2THOR [21], a 3D robotic home simulator with realistic graphics. Crafter and AI2THOR were chosen to demonstrate Somniloquy's applicability to diverse domains, along with the ease of which a variety of language can be generated by our narrators. We use the reward task for Crafter in which the agent is rewarded for unlocking unique achievements in an episode, such as harvesting wood, and we remove the stochasticity in the procedural generation by fixing the environment generation seed across all episodes in a given training run (we average over 5 runs), such that each episode in a given run has the same generated world. For AI2THOR, we use a kitchen environment and devise our own task where the agent is rewarded the first time it picks up a unique

object, in order to encourage interaction with a diverse range of objects, hence generating a diverse set of language. In the stochastic setting, we create our own environment in the MiniGrid [5] domain in which an agent is rewarded for successfully navigating to a goal, using teleporter objects that randomly move the agent to one of a set of pre-determined locations. For each environment, we also create a rule-based narrator to generate natural language descriptions from a sequence of environment states, as required for training and evaluation. Full environment, task, and narrator, details can be found in the Supplementary Material.

To answer our first research question, we train Somniloquy in our deterministic Crafter and AI2THOR environments, following the training procedure outlined in Algorithm 1. We set the translation loss weighting β_{trans} to 10 across all experiments, as empirically we found this to make the loss have a similar order of magnitude as the reconstruction loss. Full model details can be found in the Supplementary Material. At set intervals throughout training, we run the deterministic evaluation procedure for 15 evaluation episodes and a latent plan length of 16, to match the plan horizon used in Dreamer to train the agent's policy. We record the mean translation score across all plans in all evaluation episodes. To measure the similarity between translated plans and ground-truth rollout descriptions, we use the Bilingual Evaluation Understudy-4 (BLEU-4) score [29], consistent with prior works [8, 40]. The BLEU-4 score ranges from 0 to 1, with a value of 1 indicating a perfect translation. We also measured the similarity using a wide range of other commonly used machine translation metrics, and found small variations across scores, detailed in the Supplementary Material.

To answer our second research question, we train Somniloquy in our stochastic MiniGrid teleporter environment. At set intervals throughout training, we run the stochastic evaluation procedure to estimate the empirical distribution of the environment's transition function using only samples of latent plan translations, which we compare to the known ground-truth. We compute 30 samples for each plan starting state, and average results across 15 evaluation episodes.

To answer our third research question, we repeat our experiments in the deterministic setting using a slightly modified version of Somniloquy that detaches the latent states from the computational graph, which prevents the translation loss from propagating to the RSSM. All other training and evaluation is as before.

6.2.2 Language to Reward. We evaluate Somniloquy's ability to enable training a policy to achieve natural language goals in a gridworld environment from the BabyAI task suite [4]. The BabyAI suite is commonly used for evaluating an RL agent's ability to achieve goals specified in natural language, with a specific focus to test how it generalises to unseen goals. Generalisation testing is beyond the scope of our work, and thus we train the world model in a fixed environment. Specifically, we use the 'GoToLocal' environment, in which the agent is tasked with going to an object of a certain type and colour, with goals of the form 'go to the {colour} {type}', and with the goal considered reached if the agent is both facing, and in a cell adjacent to, the specified object.

When training our world model, we assume that there are no goals, and hence no extrinsic reward signal nor terminal states in the environment. Our own environment narrator describes state sequences in terms of what, if any, objects the agent went to, along with whether the agent moved or not.

After training the world model for 50K environment steps under a uniformly random exploration policy, we freeze the world model and train 4 policies from scratch inside of the world model, to achieve 4 language goals to go to the following objects: 'red key', 'blue ball', 'green ball', and 'purple box'. We also train the two comparison methods, 'naive-language-rewarding' and 'extrinsic-rewarding', following the procedures outlined in Section 5 and Section 6.2, training with a plan horizon length of 31, and for a total of 500 policy update steps (i.e., the while loop in Algorithm 2 is ran 500 times).

6.3 Results

6.3.1 Translation & Task Performance Results - Deterministic Environments. Figure 2 shows translation and task performance of Somniloquy in our deterministic Crafter and AI2THOR environments, both with (translation grads) and without (no translation grads) allowing the translation loss to shape the learned latent representation. In the Crafter environment, Somniloquy achieves a consistently high translation performance throughout training, even as the learned latent state representation and agent's policy evolve. In the AI2THOR environment, Somniloquy also performs well, with more variation and lower performance during training. One potential reason for the difference in performance between the AI2THOR and Crafter environments is that AI2THOR is a more challenging environment to translate, as evidenced by the translation loss curves given in the Supplementary Material.

In terms of task performance, there is no significant difference between the two methods in the Crafter environment. In the AI2THOR environment, allowing the translation loss to shape the learned representation appears to lead to higher, and more consistent, task performance, compared to vanilla Dreamer's latent state representation. This suggests that incorporating state-descriptive natural language information into the learned latent representation can offer an improved representation for policy learning. The increase in performance of the translation gradient method may also suggest why the translation performance appears slightly lower than in the AI2THOR environment, as an increased return may come from having seen more states that in turn need to be learned to be modelled, which may thus take longer to converge. Further work is needed to investigate these phenomena. In summary, Somniloquy yields high translation performance across both environments, meaning that what the agent describes will happen once its plan is executed corresponds closely to what actually happens when said plan is executed in the environment. Example plan translations, along with their corresponding ground truth narration, can be found in the Supplementary Material.

6.3.2 Translation & Task Performance Results - Stochastic Environments. In the stochastic setting, we find that the empirical transition estimate inferred purely from natural language translations of latent plan rollouts somewhat accurately captures true stochastic dynamics, as shown in Figure 3. Further work is needed to examine the cause of the remaining variation between the empirical and true dynamics, whether this is due to the world model being incorrect or the translation being incorrect. In terms of task performance,

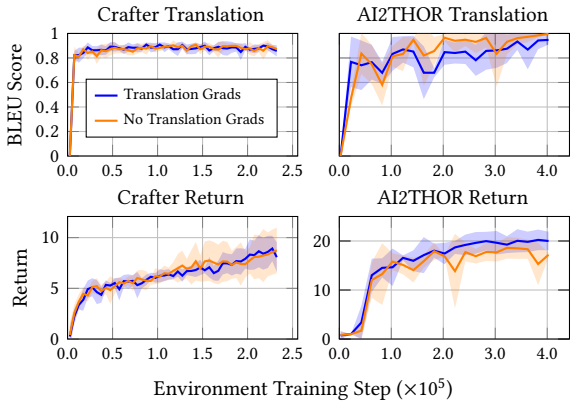

Figure 2: Latent state translation (top) and return (bottom) performance in Crafter and AI2THOR environments. Shaded regions show ±1 standard deviation across 5 seeds.

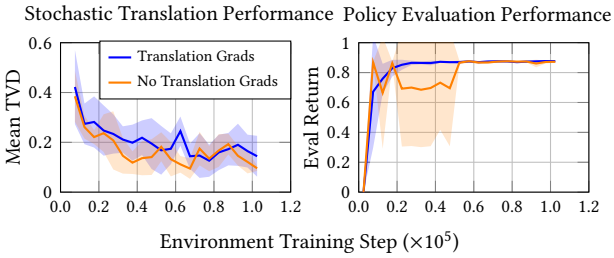

Figure 3: Stochastic MiniGrid plan translation evaluation. Left: mean Total Variation Distance (TVD) between empirical and true transition probabilities. Right: evaluation return of the learned policy. Curves depict the mean across 5 seeds, with shading showing ±1 standard deviation.

we again find evidence to suggest that allowing the translation loss to shape the learned representation leads to a latent representation that improves the stability of task performance, with lower variation than the representation learned by vanilla Dreamer.

6.3.3 Language to Reward Results. Figure 4 shows the success rates inside of the pre-trained model during training, according to different reward functions, for all four natural language goals. For all reward functions and language goals, the policy achieves 100% success rate in the environment after just 100 policy update steps. The training curves shown illustrate that (a) Somniloquy enables specifying a reward function in natural language, without access to any extrinsic reward function, that achieves on-par performance with a latent state reward function trained with access to the corresponding extrinsic reward function; and (b) as expected, the naive-language-rewarding method negatively impacts on learning, due to assigning reward (and thus credit) to latent states that had no causal impact in goal reaching.

7 LIMITATIONS AND FURTHER WORK

One of the main limitations of our work is that, when evaluating Somniloquy, we cannot distinguish between cases where the world model is wrong and cases where the translation is wrong. This is important as the translation could be a correct description of the latent plan in terms of the world model, but it appears to be incorrect due to inaccuracies in the world model. Despite this, we observe high translation performance in our experiments. Our work is also limited to evaluate translations of plan horizons of a fixed length of 16 steps, and so further work could explore varying this. Finally, further work needs to be done to explore the impact of different translation loss weightings in the world model when allowing the translation loss to impact the latent representation.

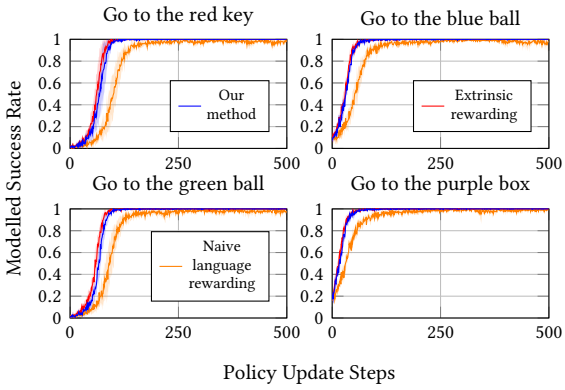

Figure 4: Modelled success rates across four BabyAI language goal subtasks. Shaded regions show ±1 standard deviation.

8 CONCLUSION

We introduce Somniloquy, an algorithm that simultaneously learns a latent representation of the world and how to translate sequences of these representations into natural language. We have demonstrated that in deterministic environments Somniloquy allows an agent to state, in natural language, what it plans to do before acting, along with how its plan impacts the world, that closely corresponds to what happens when the plan is executed. In stochastic environments, we have demonstrated that the translations of multiple plan rollouts can approximate the true stochastic environment dynamics, with further work needed to examine the cause of the remaining variation between the two. Finally, we have demonstrated that having a direct connection between latent states and natural language allows describing a desired goal in natural language, such that a policy can be trained to reach states whose latent state translation imply that the goal has been reached.

ACKNOWLEDGMENTS

This work was supported by UK Research and Innovation [grant number EP/S023356/1], in the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (www.safeandtrustedai.org).

King's Computational Research, Engineering and Technology Environment (CREATE) was used to run the experiments in this work. King's College London. (2026). Retrieved February 8th, 2026, from https://doi.org/10.18742/rnvf-m076

REFERENCES

[1] Ion Androutsopoulos and Prodromos Malakasiotis. 2010. A survey of paraphrasing and textual entailment methods. *Journal of Artificial Intelligence Research* 38 (2010), 135–187.

[2] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*. PMLR, 287–318.

[3] Gerard Canal, Senka Krivić, Paul Luff, and Andrew Coles. 2022. PlanVerb: Domain-Independent Verbalization and Summary of Task Plans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9698–9706.

[4] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019. BabyAI: First Steps Towards Grounded Language Learning With a Human In the Loop. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJeXCo0cYX

[5] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. 2023. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. In *Advances in Neural Information Processing Systems 36, New Orleans, LA, USA*.

[6] Francisco Cruz, Charlotte Young, Richard Dazeley, and Peter Vamplew. 2022. Evaluating human-like explanations for robot actions in reinforcement learning scenarios. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 894–901.

[7] Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. 2023. Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning*. PMLR, 8657–8677.

[8] Upol Ehsan, Brent Harrison, Larry Chan, and Mark O Riedl. 2018. Rationalization: A neural machine translation approach to generating natural language explanations. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. 81–87.

[9] Upol Ehsan, Pradyumna Tambwekar, Larry Chan, Brent Harrison, and Mark O Riedl. 2019. Automated rationale generation: a technique for explainable AI and its effects on human perceptions. In *Proceedings of the 24th international conference on intelligent user interfaces*. 263–274.

[10] Joseph Gatto, Omar Sharif, Parker Seegmiller, Philip Bohlman, and Sarah Preum. 2023. Text Encoders Lack Knowledge: Leveraging Generative LLMs for Domain-Specific Semantic Textual Similarity. In *Proceedings of the Third Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*. 277–288.

[11] Konstantinos Gavriilidis, Yaniel Carreno, Andrea Munafo, Wei Pang, Ronald PA Petrick, and Helen Hastie. 2021. Plan Verbalisation for Robots Acting in Dynamic Environments. In *ICAPS 2021 Workshop on Knowledge Engineering for Planning and Scheduling*.

[12] Danijar Hafner. 2022. Benchmarking the Spectrum of Agent Capabilities. In *International Conference on Learning Representations*. https://openreview.net/forum?id=1W0z96MFEoH

[13] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2020. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1lOTC4tDS

[14] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2019. Learning Latent Dynamics for Planning from Pixels. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 2555–2565. https://proceedings.mlr.press/v97/hafner19a.html

[15] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. 2021. Mastering Atari with Discrete World Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=0oabwyZbOu

[16] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2025. Mastering diverse control tasks through world models. *Nature* (02 Apr 2025). https://doi.org/10.1038/s41586-025-08744-2

[17] Shengran Hu and Jeff Clune. 2023. Thought cloning: Learning to think while acting by imitating human thinking. *Advances in Neural Information Processing Systems* 36 (2023), 44451–44469.

[18] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*. PMLR, 9118–9147.

[19] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2023. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *Conference on Robot Learning*. PMLR, 1769–1782.

[20] Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. 2024. Motif: Intrinsic Motivation from Artificial Intelligence Feedback. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=tmBKIecDE9

[21] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. 2017. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv* (2017).

[22] Jessy Lin, Yuqing Du, Olivia Watkins, Danijar Hafner, Pieter Abbeel, Dan Klein, and Anca Dragan. 2024. Learning to model the world with language. In *Proceedings of the 41st International Conference on Machine Learning*. 29992–30017.

[23] Zeyuan Liu, Ziyu Huan, Xiyao Wang, Jiafei Lyu, Jian Tao, Xiu Li, Furong Huang, and Huazhe Xu. 2025. World Models with Hints of Large Language Models for Goal Achieving. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 50–72.

[24] Lirui Luo, Guoxi Zhang, Hongming Xu, Yaodong Yang, Cong Fang, and Qing Li. 2024. End-to-end neuro-symbolic reinforcement learning with textual explanations. In *Proceedings of the 41st International Conference on Machine Learning*. 33533–33557.

[25] Pietro Mazzaglia, Tim Verbelen, Bart Dhoedt, Aaron Courville, and Sai Rajeswar. 2024. Genrl: Multimodal-foundation world models for generalization in embodied agents. *Advances in neural information processing systems* 37 (2024), 27529–27555.

[26] Jesse Mu, Victor Zhong, Roberta Raileanu, Minqi Jiang, Noah Goodman, Tim Rocktäschel, and Edward Grefenstette. 2022. Improving intrinsic exploration with language abstractions. *Advances in Neural Information Processing Systems* 35 (2022), 33947–33960.

[27] Iman Nematollahi, Branton DeMoss, Akshay L Chandra, Nick Hawes, Wolfram Burgard, and Ingmar Posner. 2025. Lumos: Language-conditioned imitation learning with world models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 8219–8225.

[28] X Phong Nguyen, Tho H Tran, Nguyen B Pham, Dung N Do, and Takehisa Yairi. 2022. Human language explanation for a decision making agent via automated rationale generation. *IEEE Access* 10 (2022), 110727–110741.

[29] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[30] Adam Poliak. 2020. A survey on Recognizing Textual Entailment as an NLP Evaluation. In *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*. 92–109.

[31] Rudra PK Poudel, Harit Pandya, Chao Zhang, and Roberto Cipolla. 2023. Langwm: Language grounded world model. *arXiv preprint arXiv:2311.17593* (2023).

[32] Shreyas Sundara Raman, Vanya Cohen, Eric Rosen, Ifrah Idrees, David Paulius, and Stefanie Tellex. 2022. Planning with large language models via corrective re-prompting. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

[33] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3982–3992.

[34] Paul Roit, Johan Ferret, Lior Shani, Roee Aharoni, Geoffrey Cideron, Robert Dadashi, Matthieu Geist, Sertan Girgin, Léonard Hussenot, Orgad Keller, et al. 2023. Factually consistent summarization via reinforcement learning with textual entailment feedback. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 6252–6272.

[35] Stephanie Rosenthal, Sai P Selvaraj, and Manuela Veloso. 2016. Verbalization: narration of autonomous robot experience. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. 862–868.

[36] Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. 2022. Skill Induction and Planning with Latent Language. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1713–1726.

[37] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 11523–11530.

[38] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 75993–76005. https://proceedings.neurips.cc/paper_files/paper/2023/file/efb2072a358cefb75886a315a6fcf880-Paper-Conference.pdf

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[40] Xinzhi Wang, Shengcheng Yuan, Hui Zhang, Michael Lewis, and Katia Sycara. 2019. Verbal explanations for deep reinforcement learning neural networks with attention on extracted features. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 1–7.

[41] Yucen Wang, Rui Yu, Shenghua Wan, Le Gan, and De-Chuan Zhan. 2025. FOUNDER: Grounding Foundation Models in World Models for Open-Ended Embodied Decision Making. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=UTT5OTyIWm

[42] Zihan Wang, Brian Liang, Varad Dhat, Zander Brumbaugh, Nick Walker, Ranjay Krishna, and Maya Cakmak. 2024. I Can Tell What I am Doing: Toward Real-World Natural Language Grounding of Robot Experiences. In *8th Annual Conference on Robot Learning*. https://openreview.net/forum?id=iZF0FRPgfq

[43] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

[44] Yilin Wu, Thomas Tian, Gokul Swamy, and Andrea Bajcsy. 2025. From Foresight to Forethought: VLM-In-the-Loop Policy Steering via Latent Alignment. In *Second Workshop on Out-of-Distribution Generalization in Robotics at RSS 2025*. https://openreview.net/forum?id=u3WDfeiWDX

[45] Alex Zhang, Khanh Nguyen, Jens Tuyls, Albert Lin, and Karthik Narasimhan. 2024. Language-guided World Models: A Model-based Approach to AI Control. In *Proceedings of the 4th Workshop on Spatial Language Understanding and Grounded Communication for Robotics (SpLU-RoboNLP 2024)*. 1–16.