Certified Guidance for Planning with Deep Generative Models

Francesco Giacomarra
University of Trieste
Trieste, Italy
francesco.giacomarra@phd.units.it

Mehran Hosseini
King's Colledge London
London, United Kingdom
mehran.hosseini@kcl.ac.uk

Nicola Paoletti
King's Colledge London
London, United Kingdom
nicola.paoletti@kcl.ac.uk

Francesca Cairoli
University of Trieste
Trieste, Italy
francesca.cairoli@units.it

ABSTRACT

Deep generative models, such as generative adversarial networks (GANs) and score-based diffusion models, have recently emerged as powerful tools for planning tasks and behavior synthesis in autonomous systems. Various guidance strategies have been introduced to steer the generative process toward outputs that are more likely to satisfy the planning objectives. These strategies avoid the need for model retraining but do not provide any guarantee that the generated outputs will satisfy the desired planning objectives. To address this limitation, we introduce *certified guidance*, an approach that modifies a generative model – without retraining it – into a new model guaranteed to satisfy a given specification with probability 1. We focus on Signal Temporal Logic (STL) specifications, which are rich enough to describe non-trivial planning tasks. Our approach leverages neural network verification techniques to systematically explore the generative models' latent spaces, identifying latent regions that are certifiably correct w.r.t. the STL property of interest. We evaluate the effectiveness of our method on four planning benchmarks using GANs and diffusion models. Our results confirm that certified guidance produces generative models that are always correct, unlike existing (non-certified) guidance methods.

KEYWORDS

Neural Network Verification, Deep Generative Models, Guided-sampling, Safe Planning, Certified Planning

1 INTRODUCTION

Deep generative models (DMG) have become increasingly powerful and popular for tasks like data augmentation, planning, and

behaviour synthesis. These models operate by transforming a standard distribution over a latent space into a more complex distribution over the target space, allowing for efficient sampling of high-dimensional data. Notable examples of state-of-the-art DGMs include Generative Adversarial Networks (GANs), Variational Autoencoders, and Score-based Diffusion Models.

A key advantage of DGMs is their ability to generate samples that meet desired criteria, e.g., a planning task specification, without explicitly encoding these constraints or retraining the model. This is achieved by so-called *guidance strategies*, which use a differentiable reward function to express the desired conditions and guide the generative process by optimizing the gradient of this reward. This technique parallels guided sampling, reinforcement learning, and planning algorithms, offering scalability, task composability, and effective non-greedy planning solutions [11].

Despite these advantages, guidance strategies are inherently limited by their inability to guarantee that the generated outputs will satisfy the desired constraints. When a requirement is unfeasible, the model may be pushed outside the data manifold, resulting in random or unreliable outputs that only coincidentally meet the specified conditions. This lack of robustness presents a significant challenge when using DGMs in safety-critical planning tasks.

To address these limitations, we introduce *certified guidance*, a method that constrains the latent space of a DGM to ensure that generated samples always satisfy a given specification while preserving the relative likelihood of the original latent inputs. Certified guidance modifies the generative model, without retraining it, into a new model guaranteed to satisfy the specification with probability 1. To do so, our method leverages neural network verification techniques to identify regions in the latent space of the DGM that are guaranteed to satisfy the specification.

In particular, we consider specifications expressed in Signal Temporal Logic (STL) [18], which provides a powerful formalism for defining complex temporal requirements, including planning tasks. In particular, our method uses STL both as a certification mechanism and as a differentiable reward function for guiding the generative process. To our knowledge, certified guidance is the first method to combine neural network verification with deep generative models to certify temporal logic properties.

We demonstrate the effectiveness of certified guidance on four planning tasks, showing that our approach can certify both GAN and diffusion models and provide stronger correctness guarantees than existing (gradient-based) guidance methods, while providing samples representative of the original DGM distribution.

Related Work

Generative Models for Planning. Generative models, particularly those based on diffusion models, have gained popularity recently. For instance, Janner et al. [12] introduced *diffuser* for flexible behaviour synthesis using a denoising diffusion probabilistic model, whereby diffusers plan by iteratively refining randomly sampled noise. Decision diffusers [1] are another popular approach where guidance is implemented by conditioning instead of latent space search. Dong et al. [7] put forward DiffuserLite, which employs a planning refinement process to generate fine-grained trajectories by reducing the modelling of redundant information and increasing the decision-making frequency.

Verification of Probabilistic Models. Closely related is the literature on single-step verification of probabilistic models. For instance, [5, 25] on verification of Bayesian neural network. Specifically, we will use the pure and gradient-guided iterative expansion approach from [5] to search the latent space of the generative model.

Safety of Planners. Most of the literature in this line of research focuses on the verification of sequential decision-making, where the decisions are usually made iteratively [20] using feedforward [2], recurrent [10], or Bayesian neural networks [26]. However, the safety verification of planners, especially those based on generative models, is an emerging line of research. Notably Xiao et al. [27] introduced SafeDiffuser, which ensures diffusion probabilistic models satisfy specifications using a class of control barrier functions. Other work similar work include [6, 15, 16, 23], which use conformal prediction for uncertainty-aware planning. Nonetheless, to the best of our knowledge, the work presented here is the first to specifically study the safety verification of generative models for planning. closest to this work is [13], which considers the verification of image-based neural network-based controllers, and to achieve this, it uses generative models for reducing the input dimension of such controllers. Nonetheless, this work does not consider the problem of verifying generative planners itself.

2 BACKGROUND

This section presents the key concepts needed to better understand the method detailed in §3 and 4.

2.1 Signal Temporal Logic (STL)

Signal temporal logic (STL) [18] is a formal language for specifying and monitoring the behaviour of physical systems, including temporal constraints between events. STL allows reasoning about dense-time, real-valued signals $\vec{s} : \mathbb{T} \to S$, where $\mathbb{T}$ is the (continuous) time domain and $S \subseteq \mathbb{R}^n$ is the signal's space. It does so by mapping $\vec{s}$ into a Boolean signal using atomic predicates of the form $\mu_g \equiv (g(\vec{s}(t)) > 0)$ where $g : S \to \mathbb{R}$. STL properties are defined by the following syntax:

$$\varphi := true \mid \mu_g \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathcal{U}_{[a,b]}\varphi, \tag{1}$$

where $[a, b] \subseteq \mathbb{T}$ is a bounded temporal interval. The *until* operator $\varphi_1 \, \mathcal{U}_{[a,b]}\varphi_2$ asserts that φ_1 must hold continuously until φ_2 becomes true at some point in the future. From this essential syntax, we can define as usual other operators as follows: $false := \neg true, \varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi), \Diamond_{[a,b]}\varphi := true \, \mathcal{U}_{[a,b]}\varphi$

and $\Box_{[a,b]}\varphi := \neg\Diamond_{[a,b]}\neg\varphi$, where $\Diamond_{[a,b]}\varphi$ and $\Box_{[a,b]}\varphi$ denote respectively the *eventually* and *globally* operator. The first asserts that condition φ will hold at some point in the time interval $[a, b]$, and the second that φ holds at all times in the interval $[a, b]$.

Boolean semantics. The satisfaction of a formula φ by a signal $\vec{s}$ at time t is defined by:

- $(\vec{s}, t) \models \mu_g \iff g(\vec{s}(t)) > 0$;
- $(\vec{s}, t) \models \varphi_1 \wedge \varphi_2 \iff (\vec{s}, t) \models \varphi_1$ and $(\vec{s}, t) \models \varphi_2$;
- $(\vec{s}, t) \models \neg\varphi \iff \neg((\vec{s}, t) \models \varphi))$;
- $(\vec{s}, t) \models \varphi_1 \mathcal{U}_{[a,b]}\varphi_2 \iff \exists t' \in [t + a, t + b]$ s.t. $(\vec{s}, t') \models \varphi_2 \wedge \forall t'' \in [t, t'], (\vec{s}, t'') \models \varphi_1$.
- $(\vec{s}, t) \models \Diamond_{[a,b]}\varphi \iff \exists t' \in [t + a, t + b]$ s.t. $(\vec{s}, t') \models \varphi$;
- $(\vec{s}, t) \models \Box_{[a,b]}\varphi \iff \forall t' \in [t + a, t + b](\vec{s}, t') \models \varphi$.

Given formula φ and a signal $\vec{s}$ over a bounded time interval, we can define the Boolean satisfaction signal as $\chi^\varphi(\vec{s}, t) = 1$ if $(\vec{s}, t) \models \varphi$ and $\chi^\varphi(\vec{s}, t) = 0$ otherwise. Monitoring the satisfaction of a formula is done recursively, by computing $\chi^{\varphi_i}(\vec{s}, \cdot)$ for each sub-formula φ_i of φ. The recursion is performed by leveraging the tree structure of the STL formula, where each node represents a sub-formula, in an incremental fashion so that the leaves are the atomic propositions and the root represents the whole formula. Thus, the procedure goes bottom-up from atomic predicated to the top formula.

Quantitative semantics. A useful feature of STL is that it admits a quantitative notion of satisfaction, called (space) *robustness*, which quantifies how much a signal can be perturbed with additive noise before changing the truth value of a given property φ [8]. It is defined as a function R_φ such that:

$$R_{\mu_g}(\vec{s}, t) = g(\vec{s}(t));$$
$$R_{\neg\varphi}(\vec{s}, t) = -R_\varphi(\vec{s}, t);$$
$$R_{\varphi_1 \wedge \varphi_2}(\vec{s}, t) = \min(R_{\varphi_1}(\vec{s}, t), R_{\varphi_2}(\vec{s}, t));$$
$$R_{\varphi_1 \mathcal{U}_{[a,b]}\varphi_2}(\vec{s}, t) = \sup_{t' \in [t+a,t+b]} \left(\min \left(R_{\varphi_2}(\vec{s}, t'), \inf_{t'' \in [t,t']} R_{\varphi_1}(\vec{s}, t'') \right) \right).$$

The sign of R_φ indicates the satisfaction value. In particular, STL robustness is sound in the following sense [8]: $R_\varphi(\vec{s}, t) > 0 \implies (\vec{s}, t) \models \varphi$ and $R_\varphi(\vec{s}, t) < 0 \implies (\vec{s}, t) \not\models \varphi$. As for the Boolean semantics, it is possible to generate monitors for the quantitative semantics automatically. The algorithm follows a similar bottom-up approach over the syntax tree of the formula.

2.2 Deep Generative Models

Every dataset can be considered a set of observations $\mathbf{x}$ drawn from an unknown distribution $p(\mathbf{x})$. Generative models aim to learn a model that mimics this unknown distribution as closely as possible, i.e., learn a parametric distribution $p_\theta(\mathbf{x})$ that approximates $p(\mathbf{x})$ and allows for efficient sampling.

A generative model acts as a distribution transformer, i.e., a map $G_\theta : Z \to X$ transforming a simple distribution $p(\mathbf{z})$ over a latent space Z into a complex distribution $p_\theta(\mathbf{x}) \approx p(\mathbf{x})$ over the target space X. In other words, they generate data from noise. Training the generative model requires minimising a differentiable loss function $\mathcal{L}$, which measures the difference between true and generated samples for each condition $\mathbf{y}$ in D_t. The model architecture, its loss

and thus its training procedure depend on the generative model selected. In this paper, we focus on Generative Adversarial Networks (GAN) and Denoising Diffusion models (DIFF). Details about such models are provided in the Supplementary Material in [9].

For planning applications, we are interested in generating stochastic trajectories conditioned, for instance, on the starting position of the system. To approximate such conditional distribution $p(\mathbf{x} \mid \mathbf{y})$, we use *conditional deep generative models (CDGM)*, which can be seen as maps $G_\theta : Z \times Y \rightarrow X$, where Y is the conditioning space. The main difference lies in defining a conditional DGM $G_\theta : Z \times Y \rightarrow X$ that transforms a simple conditional distribution $p(\mathbf{z} \mid \mathbf{y})$ over the latent space Z into a complex distribution $p_\theta(\mathbf{x} \mid \mathbf{y})$ over the target space X that approximates $p(\mathbf{x} \mid \mathbf{y})$. Both GAN and DIFF support such a conditional formulation.

2.3 Guided sampling from DGM

As described before, the generative process of DGM is typically controlled through conditioning. Conditioning requires that the model is built from the ground up to accept a particular modality of conditions from the user, be it descriptive text, class labels, etc. While conditioning is a powerful tool, it results in models that are hand-cuffed to a single conditioning modality. If another modality is required, a new model needs to be trained, often from scratch. Unfortunately, the high cost of training makes this prohibitive for most users. A more flexible approach to controlling the generation process is *guidance* [4].

Guidance enables generative models to be controlled by arbitrary modalities without retraining any components. The generative model G_θ is paired with a real-valued reward function $\mathcal{R} : X \rightarrow \mathbb{R}$ measuring how much a criterion of interest is met by the generated sample. In our context, we could guide the model to generate trajectories that maximize the satisfaction of a planning specification.

Let us define the real-valued function $r_\theta = \mathcal{R}(G_\theta(\mathbf{z}))$, which associates a reward to each generated sample. The guidance algorithm aims to search for the optimal latent input that satisfies:

$$\mathbf{z}^* = \arg\max_{\mathbf{z}} r_\theta(\mathbf{z}). \tag{2}$$

If $\mathcal{R}$ is differentiable, one could compute $\nabla_{\mathbf{z}} r_\theta(\mathbf{z})$, the gradient of r_θ w.r.t. $\mathbf{z}$ via automatic differentiation and then leverage gradient-based optimization techniques to approximately solve (2), i.e. to search for the latent input $\mathbf{z}^*$ that generate the sample $\mathbf{x}^* = G_\theta(\mathbf{z})$ associated with the highest reward.

2.4 Neural Network Verification

Several methods for verifying the robustness of neural networks against input perturbations have been put forward in recent years. The goal here is to compute

$$f_L := \min_{\mathbf{x} \in \mathcal{B}(\mathbf{x}_0, \varepsilon)} f(\mathbf{x}) \quad \text{and} \quad f_U := \max_{\mathbf{x} \in \mathcal{B}(\mathbf{x}_0, \varepsilon)} f(\mathbf{x}).$$

which are lower and upper bounds for the output of a neural net f on an ε-ball (w.r.t. an ℓ_p-norm) $\mathcal{B}(\mathbf{x}_0, \varepsilon)$ around a given input $\mathbf{x}_0$.

Broadly speaking, approaches for solving this problem can be divided into *complete* and *incomplete* methods. Complete methods compute the exact values of f_L and f_U while incomplete methods compute a lower bound $\tilde{f}_L \leq f_L$ and an upper bound $\tilde{f}_U \geq f_U$.

A key difference between complete and incomplete methods is that incomplete methods use relaxation to approximate non-linearities in f. In contrast, complete methods aim to compute exact upper and lower bounds for all neurons in the network. As such, complete approaches are usually more costly and less scalable than incomplete approaches. On the other hand, even though incomplete approaches tend to be more scalable, for larger models they result in extremely loose bounds that are not usable. Some state-of-the-art tools use a mixture of techniques from complete and incomplete methods to offer even more scalable approaches.

In our experiments, we use the Auto-LiRPA verifier [28], see § 5.2.1 for details. Nonetheless, the framework presented here is independent of the underlying neural network verification tool as long as it can derive sound bounds.

3 PROBLEM FORMULATION

Our focus is to certify that the realizations of a deep generative model (DGM) satisfy some planning goal expressed as a signal temporal logic (STL) requirement. More precisely, given a DGM $(G_\theta, p(\mathbf{z}))$ – uniquely identified by the latent distribution $p(\mathbf{z})$ and the pre-trained generator G_θ – we define the *probabilistic satisfaction* of an STL property ϕ as the probability that for $\mathbf{z} \sim p_\phi(\mathbf{z})$ the output of the DGM, $G_\theta(\mathbf{z})$, satisfies ϕ, i.e., $\chi_\phi(G_\theta(\mathbf{z})) = 1$, where χ_ϕ denote the Boolean STL semantics.

DEFINITION 1 (SATISFACTION PROBABILITY). *Let $(G_\theta, p(\mathbf{z}))$ be a DGM and ϕ an STL property. The* satisfaction probability *of ϕ is defined as*

$$P((G_\theta, p(\mathbf{z})) \models \phi) := Pr_{\mathbf{z} \sim p(\mathbf{z})} \left(\chi_\phi(G_\theta(\mathbf{z})) = 1 \right). \tag{3}$$

G_θ being a deterministic map, the stochasticity of the DGM is uniquely determined by the latent distribution $p(\mathbf{z})$. The goal of certified guidance is to derive from $p(\mathbf{z})$ a new property-specific latent distribution $p_\phi(\mathbf{z})$ such that every realization of the DGM $(G_\theta, p_\phi(\mathbf{z}))$ satisfies ϕ.

PROBLEM 1 (CERTIFIED GUIDANCE). *For a DGM $(G_\theta, p(\mathbf{z}))$, STL property ϕ, the* certified guidance *problem is finding a new latent distribution $p_\phi(\phi)$ for the DGM such that*

(1) ϕ is always satisfied, i.e.,

$$P\big((G_\theta, p_\phi(\mathbf{z})) \models \phi\big) = 1, \tag{4}$$

(2) the relative likelihood of the latent points is preserved, i.e.,

$$\forall \mathbf{z}_1, \mathbf{z}_2 \in \text{supp}(p_\phi), \ p_\phi(\mathbf{z}_1)/p_\phi(\mathbf{z}_2) = p(\mathbf{z}_1)/p(\mathbf{z}_2). \tag{5}$$

The first requirement, arguably the most important, states that the new DGM $(G_\theta, p_\phi(\mathbf{z}))$ is certified, i.e., it satisfies property ϕ with probability 1. The second requirement states that any two latent points under p_ϕ maintain the same relative importance between them as under p. This condition ensures that the original distribution, and so, the data distribution, is captured by p_ϕ. Note that 1 is formulated for a DGM $(G_\theta, p(\mathbf{z}))$, but it can be naturally extended to conditional DGMs $(G_\theta, p(\mathbf{z} \mid \mathbf{y}))$ for fixed values of the conditioning variable $\mathbf{y}$.

A natural solution is to define p_ϕ by restricting p to a region of the latent space that provably satisfies ϕ. Following a line of reasoning similar to [25], we now introduce (maximal) ϕ-satisfying sets as a tool to compute the satisfaction probability of ϕ.

Definition 2 (Maximal ϕ-satisfying set). *Given a DGM* $(G_\theta, p(\mathbf{z}))$, *the maximal set of latent inputs for which the corresponding generated trajectory satisfies property* ϕ, *namely the maximal* ϕ-*satisfying set, is defined as* $\overline{B} = \{\mathbf{z} \in \mathbb{R}^k | \mathcal{X}_\phi(G_\theta(\mathbf{z})) = 1\}$. *Furthermore, we say that B is a ϕ-satisfying set of latent inputs iff $B \subseteq \overline{B}$.*

The following proposition is a consequence of Definition 1 and 2.

Proposition 3.1. *Let $\overline{B}$ be the maximal ϕ-satisfying set and* $(G_\theta, p(\mathbf{z}))$ *be corresponding DGM. Then, it holds that*

$$P\big((G_\theta, p(\mathbf{z})) \models \phi\big) = \int_{\overline{B}} p(\mathbf{z}) d\mathbf{z}. \tag{6}$$

Prop. 3.1 simply translates the computation of the satisfaction probability from the function space to an integral computation on the latent space. Computing $P((G_\theta, p(\mathbf{z})) \models \phi)$ reduces to computing the maximal set $\overline{B}$ of latent inputs for which the corresponding generated trajectory satisfies property ϕ. However, we seek to find a latent distribution $p_\phi(\mathbf{z})$ such that $P((G_\theta, p_\phi(\mathbf{z})) \models \phi) = 1$.

For a set of latent points B, let us use the following notation

$$p(\mathbf{z}|\mathbf{z} \in B) = \frac{p(\mathbf{z}) \cdot \mathbf{1}(\mathbf{z} \in B)}{p(B)} \tag{7}$$

to denote the distribution obtained by restricting the support of p to B, where $\mathbf{1}(\cdot)$ is the indicator function. As a consequence of Proposition 3.1, if we define $p_\phi(\mathbf{z}) := p(\mathbf{z}|\mathbf{z} \in \overline{B})$ it follows that

$$P\big((G_\theta, p_\phi(\mathbf{z})) \models \phi\big) = \int_{\overline{B}} p(\mathbf{z}|\mathbf{z} \in \overline{B}) d\mathbf{z} = 1.$$

Identifying the maximal ϕ-satisfying set $\overline{B}$ is a non-trivial and generally infeasible task. Typically, we can identify a smaller ϕ-satisfying set $B \subseteq \overline{B}$. If we define $p_\phi(\mathbf{z}) := p(\mathbf{z}|\mathbf{z} \in B)$ it follows, as before, that

$$P\big((G_\theta, p_\phi(\mathbf{z})) \models \phi\big) = \int_B p(\mathbf{z}|\mathbf{z} \in B)) d\mathbf{z} = 1. \tag{8}$$

Corollary 3.2. *Let B be a ϕ-satisfying set. Then, it holds that*

$$P\big((G_\theta, p(\mathbf{z}|\mathbf{z} \in B)) \models \phi\big) = 1. \tag{9}$$

Moreover, by (7), for all $\mathbf{z}_1, \mathbf{z}_2 \in B$, we have that

$$p(\mathbf{z}_1|\mathbf{z}_1 \in B)/p(\mathbf{z}_2|\mathbf{z}_2 \in B) = p(\mathbf{z}_1)/p(\mathbf{z}_2).$$

Cor. 3.2 guarantees that certified guidance is equivalent to finding a certified ϕ-satisfying set $B \subseteq \overline{B}$ and restricting the support of the original latent distribution $p(\mathbf{z})$ to B resulting in a new latent distribution $p_\phi(\mathbf{z}) = p(\mathbf{z}|\mathbf{z} \in B)$.

We construct B as the union of M disjoint hyper-rectangles $B = \bigcup_{i=1}^M B_i$, where each $B_i = [l_1^i, u_1^i] \times \cdots \times [l_k^i, u_k^i]$ is a ϕ-satisfying set. This choice has two main advantages. First, a hyper-rectangle can be expressed as an L^∞-ball, which can be certified by existing neural network verification tools. Second, the resulting distribution $p_\phi(\mathbf{z}) = p(\mathbf{z} \mid \mathbf{z} \in B)$ has an analytical form which makes sampling efficient. In contrast, implementing p_ϕ by rejection sampling would be highly inefficient if B has an overall low probability. Recall that the original latent distribution $p_\phi(\mathbf{z})$ is a multi-variate isotropic standard normal, i.e., $\mathcal{N}(0, \mathbf{I}; \mathbf{z})$ [3, 21], meaning that its density can be factored as $p_\phi(\mathbf{z}) = \prod_{j=1}^k \mathcal{N}(0, 1; z_j)$ where z_j is the j-th

component of $\mathbf{z}$. This allows us to express the density $p_\phi(\mathbf{z})$ as a mixture of truncated normal distributions, as follows:

$$p_\phi(\mathbf{z}) = \sum_{i=1}^M \frac{p(B_i)}{p(B)} \prod_{j=1}^k \mathcal{N}(0, 1, l_j^i, u_j^i; z_j), \tag{10}$$

where $\mathcal{N}(0, 1, a, b; x)$ is the density of the standard normal at x truncated within the interval $[a, b]$, and

$$p(B_i) = \prod_{j=1}^k \frac{1}{2} \left(erf\left(-l_j^i/\sqrt{2}\right) - erf\left(-u_j^i/\sqrt{2}\right) \right) \tag{11}$$

is the probability of the set B_i w.r.t. the original latent p (derived from the CDF of the standard normal), with $p(B) = p\left(\bigcup_{i=1}^M B_i\right) = \sum_{i=1}^M p(B_i)$ being the overall probability of B (because the sets B_is are mutually disjoint). Next, we present a method to generate ϕ-satisfying sets B of latent inputs.

3.1 Checking ϕ-Satisfying Sets

We now describe how to check whether a given set, B, in the latent space, is such that $B \subseteq \overline{B}$, that is, we want to check whether $\mathcal{X}_\phi(G_\theta(\mathbf{z})) = 1, \forall \mathbf{z} \in B$. This is equivalent to checking:

$$\min_{\mathbf{z} \in B} \mathcal{X}_\phi(G_\theta(\mathbf{z})) = 1. \tag{12}$$

The verifier can be used to find a lower bound on the solution of the problem posed by Eq. (12). Given a bounding box B in the latent space of a DGM, the NN verifier propagates B through the deterministic function $b_\theta^\phi := G_\theta \circ \mathcal{X}_\phi$ to obtain bounds over the Boolean satisfaction values, i.e. the set $\{0, 1\}$. If both the lower and the upper bound of the output box are equal to 1, then B is a ϕ-satisfying set of latent inputs.

In §4, we describe a strategy to efficiently search for ϕ-satisfying sets, which employs gradient-based search to identify candidate B's and iteratively expands them as long as they satisfy ϕ.

4 CERTIFIED GUIDANCE FOR PLANNING

Before discussing in detail our certified guidance algorithm, we briefly introduce how to represent a planning problem with DGMs. DGMs are used to predict the trajectories (state sequences) of a stochastic system evolving over discrete (or discretized) time. The state at a specific instant of time can be described as a n-dimensional vector $\mathbf{s} \in S \subseteq \mathbb{R}^n$. In particular, we use CDGMs (introduced in §2.2) to approximate the distribution $p(\mathbf{x} \mid \mathbf{y})$ of future state sequences $\mathbf{x} \in S^H$ given a conditioning prefix $\mathbf{y} \in S^h$, for some sequence lengths H and h. To approximate $p(\mathbf{x} \mid \mathbf{y})$, CDGMs use realizations of the stochastic systems (obtained via simulation or through direct measurements) as their training dataset: $D_t = \{(\mathbf{y}_i, \mathbf{x}_i) \mid i = 1, \ldots N_t\}$. We denote with $\vec{\mathbf{s}} \in S^{h+H}$ the trajectory obtained by concatenating the prefix and target sequences, with $\vec{\mathbf{s}}(t) \in S$ being the state of $\vec{\mathbf{s}}$ at time $t \in \{0, \ldots, h + H - 1\}$.

We now illustrate our solution method for the certified guidance problem, stated in Prob. 1. Our solution produces a new latent distribution $p_\phi(\mathbf{z}|\mathbf{y})$ of the generative model, such that the resulting CDGM $(G_\theta, p_\phi(\mathbf{z}|\mathbf{y}))$ offers verified satisfaction of ϕ for all its realizations. We need to define a strategy to build the certified ϕ-satisfying sets $B_1, \ldots, B_M$ to restrict the support of $p(\mathbf{z}|\mathbf{y})$. The
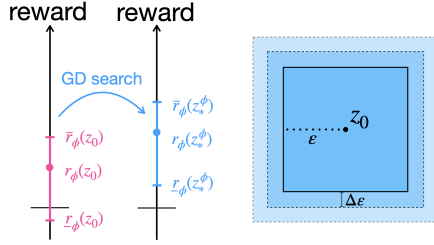
Figure 1: Certified Robust Generation: gradient ascent strategy (left) and iterative expansion of perturbation balls (right).

starting point is to consider a fixed condition $\mathbf{y}$, pick a latent input $\mathbf{z}_*$ and construct a bounding box $\mathcal{B}_p(\mathbf{z}_*, \varepsilon)$ around $\mathbf{z}_*$, where $\mathcal{B}_p(\mathbf{z}_*, \varepsilon)$ denotes an ε-perturbation around $\mathbf{z}_*$ with norm L^p. These bounding box in the latent space will be then certified by the neural network verification algorithm.

We leverage reward-guided sampling (introduced in § 2.3) to search for good candidates, or pivot points, for $\mathbf{z}_*$. To this purpose, we use the quantitative STL semantics R_ϕ (introduced in §2.1) as a differentiable measure of reward $\mathcal{R}$ to guide the DGM sampling. Let $R_\phi(\vec{\mathbf{s}})$ denote the reward associated to the target trajectory $\vec{\mathbf{s}}$ w.r.t. ϕ. High positive rewards indicate satisfaction of the property ϕ, whereas low negative rewards indicate violation. Given a condition $\mathbf{y}$ and a pre-trained generative model G_θ, we can define $r_\theta^\phi(\mathbf{z}, \mathbf{y}) := R_\phi(\tilde{G}_\theta(\mathbf{z}, \mathbf{y}))$, which, for a fixed $\mathbf{y}$, is a function of $\mathbf{z}$ alone. We stretch the notation a little and define $\tilde{G}_\theta(\mathbf{z}, \mathbf{y})$ as the model which outputs trajectories in S^{H+h} resulting from the concatenation of the condition $\mathbf{y} \in S^h$ and the generated target $G_\theta(\mathbf{z}, \mathbf{y}) \in S^H$. Searching for a pivot point can be seen as searching for the latent value $\mathbf{z}^*$ whose corresponding generated trajectory maximizes the satisfaction of ϕ. That is, we solve the following optimization problem:

$$\mathbf{z}^* = \arg\max_{\mathbf{z}} r_\theta^\phi(\mathbf{z}, \mathbf{y}), \tag{13}$$

which can be solved with gradient ascent techniques. In practice, we randomly choose L latent points, $\mathbf{z}_0^1, \ldots, \mathbf{z}_0^L$ and run gradient ascent long enough to find $\mathbf{z}_*^1, \ldots, \mathbf{z}_*^L$. We choose a small positive ε and build $\mathcal{B}_p(\mathbf{z}_*^i, \varepsilon)$. Then, for each i, we iteratively increase ε until $\mathcal{B}_p(\mathbf{z}_*^i, \varepsilon)$ remains a ϕ-satisfying set.

To determine whether $\mathcal{B}_p(\mathbf{z}_*, \varepsilon)$ is a ϕ-satisfying set, we need to find a lower bound $\underline{b}_\theta^\phi(\mathbf{z}_*, \mathbf{y})$ for $b_\theta^\phi(\mathbf{z}_*, \mathbf{y}) = \chi_\phi(\tilde{G}_\theta((\mathbf{z}_*, \mathbf{y})))$, where χ_ϕ denotes the Boolean semantics, i.e., $\chi_\phi(\vec{\mathbf{s}}) = 1$ if $\vec{\mathbf{s}}$ satisfies ϕ and $\chi_\phi(\vec{\mathbf{s}}) = 0$ otherwise. Formally, $\underline{b}_\theta^\phi(\mathbf{z}_*, \mathbf{y})$ is such that

$$\forall \mathbf{z} \in \mathcal{B}(\mathbf{z}_*, \varepsilon) \qquad \underline{b}_\theta^\phi(\mathbf{z}_*, \mathbf{y}) \leq b_\theta^\phi(\mathbf{z}, \mathbf{y}). \tag{14}$$

Then, we check whether $\underline{b}_\theta^\phi(\mathbf{z}_*, \mathbf{y}) = 1$, which implies that ϕ is always satisfied by all latent points in the ε-ball $\mathcal{B}(\mathbf{z}_*, \varepsilon)$. Otherwise, the region cannot be certified, and $\mathcal{B}_p(\mathbf{z}_*, \varepsilon)$ is disregarded. Importantly, when $\mathcal{B}_p(\mathbf{z}_*, \varepsilon)$ is instead certified, we can *iteratively increment* ε (Fig. 1-right) until the maximum perturbation we successfully certify. Indeed, we are interested in certifying the largest volume possible in the latent space.

Heterogenous increments of $\underline{\varepsilon}$. The perturbation $\underline{\varepsilon}$ can be incremented heterogeneously along the k latent dimensions. A similar approach has been introduced in [5] to certify the probabilistic robustness of Bayesian neural nets. The rationale behind this approach is to expand the hyper-rectangle more in the directions that provide a significant contribution to the increase of the STL robustness, $\underline{\varepsilon}' = \underline{\varepsilon} + \alpha \left(\frac{\nabla_\mathbf{z} r_\theta^\phi(\mathbf{z})}{\|\nabla_\mathbf{z} r_\theta^\phi(\mathbf{z})\|} \right)$, where α is some ad-hoc constant.

4.1 Algorithm

Algorithm 1 shows our approach to building a certified DGM. For simplicity of notation, we illustrate the algorithm in the non-conditional case, i.e., using $p(\mathbf{z})$ instead of $p(\mathbf{z} \mid \mathbf{y})$. In lines 1–13, the algorithm computes B, which is the union of ϕ-satisfying sets of latent inputs We build B as a union of hyper-rectangles (line 13), each of which is ϕ-satisfying. Each hyper-rectangle is generated as follows: we sample a random latent input $\mathbf{z}_0^i$ (line 6) from which we start a GD-search of a local optimum $\mathbf{z}_*^i$ (line 7). This process is repeated L times to generate multiple ϕ-satisfying hyper-rectangles in disconnected regions of the latent space. In lines 8-11, we check if the given hyper-rectangle is ϕ-satisfying by using NN verification, and we also check that the region does not overlap with others (lines 9 and 12). We iteratively increment ε as long as the ϕ-satisfying and the non-overlapping properties are met (lines 9-11). All the $M \leq L$ ϕ-satisfying non-overlapping hyper-rectangles are added to B (lines 12-13). We then use this set of hyper-rectangles to build our certified latent distribution $p_\phi(\mathbf{z}) = p(\mathbf{z}|\mathbf{z} \in B)$, which is defined as a truncation of the original latent distribution $p(\mathbf{z}|\mathbf{y})$ over the M disjoint hyper-rectangles, as explained in Eq. 10. To do so, our algorithm computes the probability $p(B_i)$ of each box and the overall probability of B, $p(B) = \sum_{i=1}^M p(B_i)$, in lines 14-16. For numerical stability, these probabilities are computed on a logarithmic scale.

5 EXPERIMENTAL RESULTS

We test the proposed certification procedure across a variety of planning problems, encompassing both 2D and 3D environments. Our experimental framework employs deep generative models, specifically Wasserstein Generative Adversarial Networks (GAN) and Diffusion (DIFF) models, which are trained on datasets generated using Probabilistic Road Map (PRM) [14] planners in MATLAB. The code used for all the experiments can be found here.

5.1 Case Studies

(i) **U-shaped Maze** – UMaze: an agent navigating from left to right in a U-shaped maze. The objective is to move across the maze without colliding with the walls. The model horizon is set to $\bar{H} = 24$ for GAN and 17 for DIFF, with $h = 1$. The STL specification encodes the safety constraints, i.e. always keep a safety distance from the walls: $\phi := \Box_{[0,\bar{H}]} \left[\left(\|\vec{\mathbf{s}} - (25, 15)\|_\infty \geq (5, 15) \right) \wedge \left((5, 5) \leq \vec{\mathbf{s}} \leq (45, 45) \right) \right]$, where $\vec{\mathbf{s}}$ denotes a trajectory.

(ii) **Urban Road Intersection** – Crossroad: an ego vehicle navigates an urban intersection with the option to turn left, proceed straight, or turn right, all while remaining within its designated lane. The model horizon is set to $\bar{H} = 23$ for GAN and 16 for DIFF, with $h = 1$. We choose a reach-avoid STL specification that encodes the safety constraints, i.e. always stay on the road and

Algorithm 1 Certified Guidance for DGMs

1: *Input:* DGM $(G_\theta, p(\mathbf{z}))$, STL property ϕ, max number of regions L, perturbation hyperparameters $\varepsilon_0, \Delta\varepsilon, \alpha$.
2: *Output:* Certified DGM $(G_\theta, p_\phi(\mathbf{z}) = p(\mathbf{z}|\mathbf{z} \in B))$.
3: $B \leftarrow \emptyset$
4: **for** $i = 1 : L$ **do**
5:     $\varepsilon \leftarrow \varepsilon_0$
6:     $\mathbf{z}_0^i \sim p(\mathbf{z})$
7:     $\mathbf{z}_*^i \leftarrow \text{GD}(-r_\theta^\phi(\mathbf{z}), \mathbf{z}_0^i, \gamma)$ ▷ Gradient-based search
8:     $\underline{b}_\phi, \overline{b}_\phi \leftarrow \text{NNVERIFICATION}\left(r_\theta(\mathbf{z}), \mathcal{B}(\mathbf{z}_*^i, \varepsilon)\right)$ ▷ Lower and Upper bounds over Boolean Satisfaction $\mathcal{X}_\phi$
9:     **while** $\underline{b}_\phi = 1 \wedge \mathcal{B}(\mathbf{z}_*^i, \varepsilon) \cap B = \emptyset$ **do**
10:         $\varepsilon \leftarrow \varepsilon + \Delta\varepsilon$ ▷ Heterogeneous: $\varepsilon \leftarrow \varepsilon + \alpha \nabla_\mathbf{z} r_\theta^\phi(\mathbf{z})$
11:         $\underline{b}_\phi, \overline{b}_\phi \leftarrow \text{NNVERIFICATION}\left(r_\theta(\mathbf{z}), \mathcal{B}(\mathbf{z}_*^i, \varepsilon)\right)$
12:     **if** $\underline{b}_\phi = 1 \wedge \mathcal{B}(\mathbf{z}_*^i, \varepsilon) \cap B = \emptyset$ **then**
13:         $B \leftarrow B \cup \mathcal{B}(\mathbf{z}_*^i, \varepsilon)$
14: **if** $B \neq \emptyset$ **then**   ▷ $p_\phi(\mathbf{z}) := p(\mathbf{z} \mid \mathbf{z} \in B)$
15:     **for** $i \leftarrow 1 : M$ **do**   ▷ M: # hyper-rectangles in B
16:         $p(B_i) \leftarrow \prod_{j=1}^k \frac{1}{2}\left(erf\left(-\frac{l_j^i}{\sqrt{2}}\right) - erf\left(-\frac{u_j^i}{\sqrt{2}}\right)\right)$ ▷ Eq. 11
        $p(B) \leftarrow \sum_{i=1}^M p(B_i)$

GAN	UMaze	Crossroad	Obstacles	City
Original	0.08	0.27	0.11	0.47
Guidance	1.00	0.72	0.96	0.99
Certified	1.00	1.00	1.00	1.00

DIFF	UMaze	Crossroad	Obstacles	City
Original	0.26	0.36	0.09	-
Guidance	0.87	1.00	0.56	-
Certified	1.00	1.00	1.00	-

Table 1: Comparison of Acceptance Ratio: number of samples needed to obtain a satisfying trajectory across different methods and scenarios.

DGM (both a GAN and a DIFF). Upon successful training of the DGMs, we proceed to test our certified guidance method. The initial step involves defining a Signal Temporal Logic (STL) property ϕ that articulates the temporal behaviours we require our planner to enforce. This ensures that the outputs generated by the certified DGM, i.e. the DGM with a mixture of truncated Gaussians as latent distribution, will satisfy these requirements by design.

5.2.1 Implementation Details. For NN verification, we leverage the flexibility and scalability of linear relaxation verification algorithms. In particular, we use Auto-LiRPA [28] (Automatic Linear Relaxation-based Perturbation Analysis), a tool designed to provide provable guarantees on neural network behaviours under input perturbations. Auto-LiRPA applies a linear relaxation technique to automatically compute bounds for the outputs of neural networks in response to perturbations in the input space. In our methodology, we employ AutoLiRPA to assess how the satisfaction of STL properties varies inside a given hyper-rectangle in the latent space. In particular, we utilize the CROWN (Convex Relaxation Over Weighted Networks) strategy as Interval Bound Propagation (IBP) proved to be ineffective, yielding bounds that were not useful in practice due to their conservative nature. A considerable effort lies in encoding GAN and DIFF models together with the Boolean STL semantics into an AutoLiRPA-compliant format. The Boolean STL semantics is integrated into the computational graph of the DGM, allowing for perturbations in the latent space to propagate through to the satisfaction space, ensuring that the generated trajectories adhere strictly to the defined STL requirements. For STL guidance, we implemented a differentiable quantitative semantics for STL within the PyTorch framework. This approach enables the application of gradient ascent techniques to identify latent inputs that maximize the satisfaction of the STL properties.

DGM Architectural detail. For GANs, we use the same set of hyperparameters for all the case studies except for the dimension of the latent input that we choose to match the length of the considered trajectories. Similarly, we keep all the hyperparameters fixed also for the DIFF experiments. In the diffusion process, we set the minimum noise level $\beta_1 = 0.0001$ and the maximum noise level $\beta_T = 0.5$, while the other noise levels are set using the following quadratic policy, $\beta_\tau = (\beta_\tau(T - \tau)/(T - 1) - \beta_1(\tau - 1)/(T - 1))^2$, as it has been shown that this type of decay improves sample quality [19, 22, 24]. As it greatly affects the computational burden of the verification process, we choose a low number T of diffusion steps, in particular, we set $T = 6$ for all the experiments. After training

impose the generation of left turns:

$$\phi := \square_{[0,\bar{H}]}\left[\wedge_{i=1}^4 \|\vec{\mathbf{s}} - c_i\|_\infty \geq \ell_i\right]$$
$$\wedge \diamond_{[0,\bar{H}]}\square_{[0,\bar{H}]}\left[(0, 27) \leq \vec{\mathbf{s}} \leq (15, 35)\right],$$

where (c_i, ℓ_i) denotes the centre and the radius of road corners.

(iii) **2D Navigation** – Obstacles: an agent navigates from a fixed starting point to a predetermined destination point within a 2D space, avoiding any obstacles present. The task showcases multimodal behaviour, requiring the generative model to recognize and evaluate multiple potential paths that avoid obstacles. The model horizon is set to $\bar{H} = 15$, with $h = 1$. Once again, we choose a reach-avoid STL specification that lets the agent reach the goal while avoiding all the obstacles:

$$\phi := \square_{[0,\bar{H}]}\left[\wedge_{i=1}^4 \|\vec{\mathbf{s}} - c_i\|_\infty \geq \ell_i\right]$$
$$\wedge \diamond_{[0,\bar{H}]}\square_{[0,\bar{H}]}\left[\|\vec{\mathbf{s}} - (29, 29)\|_\infty \leq 1\right],$$

where (c_i, ℓ_i) denotes the centre and the radius of the obstacles.

(iv) **Drone Navigation Scenario** – City: a drone is required to fly over an urban landscape without ascending too high above the ground or crashing into buildings. The model horizon is set to $\bar{H} = 30$, with $h = 1$. The STL specification encodes the safety constraints, i.e. always keep a safety distance from the buildings: $\phi := \square_{[0,\bar{H}]}\left[\wedge_{i=1}^5 \|\vec{\mathbf{s}} - c_i\|_\infty \geq \ell_i\right]$, where c_i denotes the centre of each building (or group of buildings and $\ell_i = (\ell_{i,x}, \ell_{i,y}, \ell_{i,z})$ the measures of each building.

Fig. 2 shows a visualization of the considered planning problems.

5.2 Experimental Details

The first experimental step lies in generating a pool of planning realization using PRM and using these as a dataset to train our

GAN	UMaze	Crossroad	Obstacles	City
Original	-6452.1	-8783.7	-3981.3	-8530.5
Original Sat.	-6729.3	-8902.1	-4049.3	-8494.7
Guidance	-8955.7	-9436.7	-4244.8	-14152.7
Certified	-7062.9	-8295.7	-3740.2	-9469.2

DIFF	UMaze	Crossroad	Obstacles	City
Original	-8823.4	-8524.9	-7944.6	-
Original Sat.	-8853.7	-8452.9	-7885.4	-
Guidance	-8902.3	-8459.4	-7685.2	-
Certified	-9810.5	-8691.9	-8515.6	-

Table 2: Comparison of Log Likelihoods: sum of the log-likelihoods w.r.t. $p(z)$ over 200 samples across different methods and scenarios.
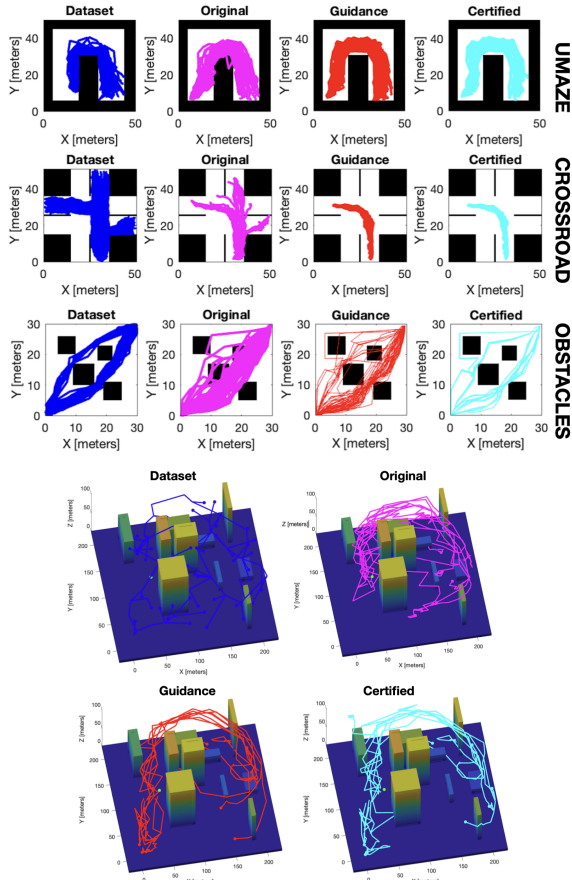
Figure 2: Visualizations of GAN-generated paths using each approach on different benchmarks.

the DDPM, we generate samples from the related DDIM by setting $\sigma_\tau = 0$ for all τ in the backward process; i.e., we remove the stochasticity in the generation process, thus making verification feasible. In all the experiments, the Rectified ADAM (RADAM) algorithm [17] with a learning rate of 0.0005 has been used to optimize the loss function. Training times depend heavily on the number of iterations, the number of training instances and their dimensionality.

Experimental settings. Each iteration of our experimental procedure begins with an initial perturbation where $\epsilon = 0.01$ and an

incremental increase $\Delta\epsilon = 0.005$. The iterative process is repeated for a total of $M = 20$ cycles for GAN and $M = 5$ for DIFF, implying that at most 20 and 5 hyper-rectangles are generated respectively throughout the experiment. This ensures a structured approach to exploring the robustness and variability of our model outputs under controlled perturbations. We experimented with heterogeneous increments of the hyper-rectangles but it was less effective at identifying satisfying regions. So, we report results only for the homogeneous increment version. We suspect that the gradient may cause abrupt increases in a latent space that is particularly stiff.

5.3 Results

Computational costs. Tbl. 1 computes, for each method, the ratio between 200 and the number of samples needed to obtain 200 satisfying trajectories. Samples from the original DGM show a low acceptance ratio, meaning that such samples have a low probability of satisfying ϕ. This happens also in scenarios where the DGM was trained over data that contained no violations, meaning that such violations were introduced by the generative model. This can be seen in the Obstacle Case Study (see Fig. 2), where all training trajectories (in blue) satisfy the planning requirements, whereas samples from the original DGM (in purple) often violate the property. This comes with no surprises as the DGM generalizes over the possible dynamics with no explicit information about the regions to avoid. STL guidance shows a much higher acceptance ratio over all the case studies; but it does not guarantee a 100% satisfaction rate, which is instead achieved by construction by our certified DGM.

From a computational point of view, the runtime needed to generate 200 ϕ-satisfying samples from the original models takes on average 3 seconds (15 seconds for DIFF), around half a second (2 seconds for DIFF) for our certified DGM and around 20 minutes (60 minutes for DIFF) for the guidance approach where we have to perform 200 gradient searches, i.e. solve 200 optimization problems. Add times for DIFF. Some offline costs need to be taken into account. The training time for the generative models is a cost shared by all the methods. However, our certification approach has a considerable computational overhead due to the search for good pivot points and the NN verification step. The cost of searching for the pivot is negligible; it takes less than 10 seconds. Performing one iteration of NN verification, however, takes ∼ 2 minutes for GANs and ∼ 120 minutes for DIFFs. This significative difference in verification time depends on the dimensionality of the latent spaces and on the different architectural complexity of the two models.

Diffusion models have a convoluted structure and the latent space is forced to have the same dimensionality of the output, whereas GAN does not have such a constraint. These offline costs accumulate as we perform an iterative expansion of our hyper-rectangles. However, §4.1's algorithm can be parallelized so that the M hyper-rectangles can be computed in parallel, checking for pair-wise intersections only at the end. The verification time for DIFF grows exponentially with the complexity of the model and the dimensionality of the associated latent space. DIFF faces a clear scalability issue. We were not able to perform experiments on the three-dimensional case study (City) due to the larger latent space (size $3 \cdot H$) and the increased number of diffusion steps needed to achieve a well-behaved generative model.
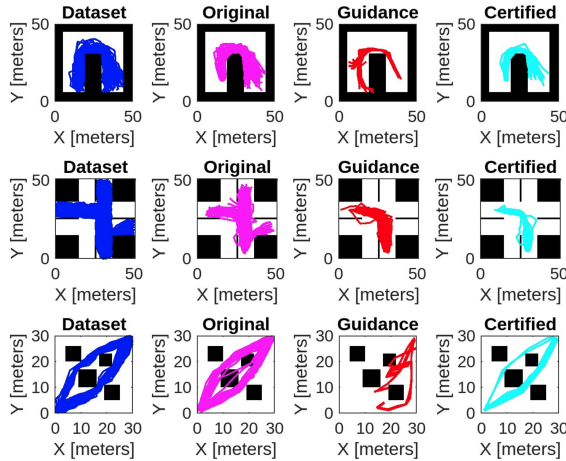
Figure 3: DIFF Certified paths in different benchmarks

Goodness of fit. Tbl. 2 shows a comparison of the cumulative log-likelihood w.r.t. original latent distribution $p(\mathbf{z} \mid \mathbf{y})$ of samples coming from the STL guidance and samples drawn from the new certified distribution $p_\phi(\mathbf{z} \mid \mathbf{y})$. As a baseline, we show the log-likelihood of samples taken from the original latent distribution and compare it with the likelihood of samples that satisfy ϕ. More specifically, we will compare the sum of the log-likelihoods over 200 samples. Despite our certified guidance method considers only a small subset of the support, it results in trajectories that are as likely or more likely than the simple guidance method.

Performances. For each case study, Fig. 2 for GAN and Fig. 3 for DIFF present a qualitative comparison of trajectories sampled either from the test dataset (in blue), from the original DGM (in purple), via STL guidance sampling (in red), and trajectories produced by our certified DGM (in cyan). These figures serve as a reference point for a qualitative evaluation of the performances of our certified guidance. In general, we notice how the original DGM introduces violations even when trained on a dataset that contained none. For instance, the UMaze and the Obstacles case studies show how none of the training trajectories (blue) crashed into an obstacle, whereas the DGM (purple) does. STL guidance stirs the sampling towards regions of satisfaction but does not guarantee that it produces violation-free trajectories. For instance, in the GAN Crossroad scenario, we had to reject 28% of the samples obtained via STL guidance because they did not satisfy the STL requirement.

GAN Performance. Our GAN-certified guidance approach offers good overall results in terms of variability of the generated trajectories. Its strengths lie in the sampling efficiency at runtime and in the insights we gain over how the latent space encodes information. For instance, in the Crossroad scenario, one could distinguish the areas of the latent space that generate left turns from those that generate right turns. Moreover, we expect these regions to correspond to robust encodings of the dynamics as contiguous latent points lead to similar outputs. Moreover, we preserve the original likelihood ratio so that latent inputs with low probability are not sampled too often. In general following the gradient of the robustness, as STL guidance does, may lead far from high probability regions. However, we can guarantee that we are not over-sampling from these

regions, whereas STL guidance cannot offer such guarantees. Moreover, Certified GAN also performs well in the three-dimensional planning problem (City) , showing a good level of scalability.

DIFF Performance. Due to the aforementioned scalability issues of NN verification, we must keep the latent space to a manageable dimension. Therefore, DIFF models are trained on trajectories shorter than the ones used by GANs. For all the considered Case Studies, we downsample the original PRM trajectories. The number of diffusion steps was deliberately kept low to ensure feasible verification times. Moreover, we are sampling from the implicit version of a diffusion model, which introduces stochasticity only in the initial step, drastically reducing the variance of the generated samples. We also had to impose shallower STL constraints to allow the STL guidance to converge to positive robustness values. Although this made verification possible, it still took considerably more time than GAN-based verification, and exploring the latent space remains challenging. This is further supported by the low acceptance ratios of the STL guidance. The gradient search tends to fall into the same regions consistently; and almost certainly due to the mentioned constraints, the latent space does not encode information robustly. Thus, finding acceptable hyper-rectangles is more difficult. While the results could be stronger, this is a first step in this direction, and our approach provided valuable insights into the behavior of the diffusion model w.r.t. its latent space.

6 CONCLUSION

We introduced certified guidance, a novel framework to ensure that the outputs of deep generative models satisfy specific planning constraints. Our method is inspired by gradient-based approaches to guidance but, unlike these, can provide rigorous correctness guarantees. Our experiments have shown promising results, demonstrating the reliability of our certified models in cases where "vanilla" guidance fails. We managed to certify both GAN and diffusion models, albeit the latter suffer from scalability issues. Despite that, to our knowledge, we are the first to successfully certify diffusion models w.r.t. temporal logic requirements. Additionally, this framework can be directly extended to VAEs, where a variational decoder can adopt the same structure as the GAN generator, further broadening its applicability. In future work, we aim to reduce the dependence on specific values in the conditioning space. We will focus on developing guarantees that generalize across the entire conditioning space, allowing for more flexible and reliable planning outputs. This direction will further enhance the robustness and practical usability of certified guidance in real-world scenarios.

REFERENCES

[1] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. 2023. Is Conditional Generative Modeling all you need for Decision-Making? arXiv:2211.15657 [cs.LG] https://arxiv.org/abs/2211.15657

[2] Michael Akintunde, Elena Botoeva, Panagiotis Kouvaros, and Alessio Lomuscio. 2022. Formal Verification of Neural Agents in Non-deterministic Environments. *Journal of Autonomous Agents and Multi-Agent Systems* 36, 1 (2022).

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875* (2017).

[4] Arpit Bansal, Hong-Min Chu, Avi Schwarzschild, Soumyadip Sengupta, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2023. Universal guidance for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 843–852.

[5] Ben Batten, Mehran Hosseini, and Alessio Lomuscio. 2024. Tight Verification of Probabilistic Robustness in Bayesian Neural Networks. In *The 27th International Conference on Artificial Intelligence and Statistics, AISTATS*, Vol. 238.

[6] Francesca Cairoli, Nicola Paoletti, and Luca Bortolussi. 2023. Conformal Quantitative Predictive Monitoring of STL Requirements for Stochastic Processes. In *International Conference on Hybrid Systems: Computation and Control, HSCC*. ACM, 1:1–1:11.

[7] Zibin Dong, Jianye Hao, Yifu Yuan, Fei Ni, Yitian Wang, Pengyi Li, and Yan Zheng. 2024. DiffuserLite: Towards Real-time Diffusion Planning. arXiv:2401.15443

[8] Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 92–106.

[9] Francesco Giacomarra, Mehran Hosseini, Nicola Paoletti, and Francesca Cairoli. 2025. Certified Guidance for Planning with Deep Generative Models. arXiv:2501.12815 [cs.LG] https://arxiv.org/abs/2501.12815

[10] Mehran Hosseini and Alessio Lomuscio. 2023. Bounded and Unbounded Verification of RNN-Based Agents in Non-deterministic Environments. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. ACM, 2382–2384.

[11] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. 2022. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991* (2022).

[12] Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. 2022. Planning with Diffusion for Flexible Behavior Synthesis. In *International Conference on Machine Learning, ICML*, Vol. 162. PMLR, 9902–9915.

[13] Sydney M. Katz, Anthony L. Corso, Christopher A. Strong, and Mykel J. Kochenderfer. 2022. Verification of Image-Based Neural Network Controllers Using Generative Models. *J. Aerosp. Inf. Syst.* 19, 9 (2022), 574–584.

[14] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12, 4 (1996), 566–580. https://doi.org/10.1109/70.508439

[15] Tom Kuipers, Renukanandan Tumu, Shuo Yang, Milad Kazemi, Rahul Mangharam, and Nicola Paoletti. 2024. Conformal Off-Policy Prediction for Multi-Agent Systems. arXiv:2403.16871

[16] Lars Lindemann, Matthew Cleaveland, Gihyun Shim, and George J. Pappas. 2023. Safe Planning in Dynamic Environments Using Conformal Prediction. *IEEE Robotics Autom. Lett.* 8, 8 (2023), 5116–5123.

[17] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2021. On the Variance of the Adaptive Learning Rate and Beyond. arXiv:1908.03265 [cs.LG] https://arxiv.org/abs/1908.03265

[18] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.

[19] Alex Nichol and Prafulla Dhariwal. 2021. Improved Denoising Diffusion Probabilistic Models. arXiv:2102.09672 [cs.LG] https://arxiv.org/abs/2102.09672

[20] Andreas Orthey, Constantinos Chamzas, and Lydia E. Kavraki. 2024. Sampling-Based Motion Planning: A Comparative Review. *Annu. Rev. Control. Robotics Auton. Syst.* 7, 1 (2024).

[21] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. arXiv:1503.03585 [cs.LG]

[22] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2020. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456* (2020).

[23] Jiankai Sun, Yiqi Jiang, Jianing Qiu, Parth Nobel, Mykel J. Kochenderfer, and Mac Schwager. 2023. Conformal Prediction for Uncertainty-Aware Planning with Diffusion Dynamics Model. In *Advances in Neural Information Processing Systems, NeurIPS*, Vol. 36. Curran Associates, Inc., 80324–80337.

[24] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. 2021. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems* 34 (2021), 24804–24816.

[25] Matthew Wicker, Luca Laurenti, Andrea Patane, and Marta Kwiatkowska. 2020. Probabilistic Safety for Bayesian Neural Networks. In *Proceedings of the Thirty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI*.

[26] Matthew Wicker, Luca Laurenti, Andrea Patane, Nicola Paoletti, Alessandro Abate, and Marta Kwiatkowska. 2024. Probabilistic reach-avoid for Bayesian neural networks. *Artif. Intell.* 334 (2024), 104132.

[27] Wei Xiao, Tsun-Hsuan Wang, Chuang Gan, and Daniela Rus. 2023. SafeDiffuser: Safe Planning with Diffusion Probabilistic Models. arXiv:2306.00148

[28] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems* 33 (2020), 1129–1141.