Towards Symbolic Planning via Diffusion

Arman Mohammadi, Markus Fritzsche, Jendrik Seipp

Department of Computer Science, Linköping University, Sweden
arman.mohammadi@liu.se, markus.fritzsche@liu.se, jendrik.seipp@liu.se

Abstract

Recent learning-based planners frame plan generation as next-token prediction with autoregressive models. We study masked diffusion as a non-autoregressive alternative. In this formulation the model learns the distribution of complete action sequences by reversing a forward process that gradually masks the plan. At each denoising step, the model predicts the entire plan, keeps high-confidence tokens and remasks the remaining positions for the next step. We train such a model from scratch on four classical planning domains and find coverage and plan quality comparable to an autoregressive baseline. We further analyze how the number of inference steps trades off against performance and identify limitations of the current formulation that motivate further work.

Introduction

Automated planning is concerned with generating sequences of actions that transform an initial state into a goal state. Planning tasks are typically defined over structured representations, where states are sets of facts about the world and actions are specified through preconditions and effects on those facts. Generating plans in this setting can be naturally viewed as a sequence generation task, where the objective is to predict a valid sequence of actions given the problem description. As generative modeling opens new directions for research, we explore the potential of conditional diffusion models for such sequence prediction tasks.

Diffusion models (Sohl-Dickstein et al. 2015; Ho, Jain, and Abbeel 2020; Song, Meng, and Ermon 2021; Karras et al. 2022) are a class of generative models that achieve strong results in image synthesis (Ho, Jain, and Abbeel 2020; Song, Meng, and Ermon 2021; Ramesh et al. 2022; Rombach et al. 2022) and have been adapted to molecule generation (Hoogeboom et al. 2022), temporal data modeling (Alcaraz and Strodthoff 2023; Rasul et al. 2021) and other sequence generation tasks (Yang et al. 2024). This broad success naturally raises the question of whether diffusion models can also be applied to automated planning.

Diffusion has been extended to natural language by mapping text into continuous embeddings (Li et al. 2022) or by extending the diffusion process directly to discrete tokens (He et al. 2023; Sahoo et al. 2024). Recent diffusion-based large language models match autoregressive ones on several benchmarks, suggesting that the strengths of modern language models do not depend on autoregressive generation (Nie et al. 2025). Two properties of diffusion are particularly relevant for planning. First, the model uses bidirectional attention and conditions each prediction on both past and future context, which has been shown to improve reasoning (Berglund et al. 2024). Second, diffusion models naturally capture multimodal trajectory distributions (Carvalho et al. 2025), which is particularly beneficial in planning tasks where multiple solutions may exist.

We develop a masked diffusion framework for classical planning and train it from scratch on four standard benchmark domains. Pre-trained large language models have been applied to planning problems (Pallagani et al. 2022; Huang et al. 2022; Silver et al. 2024; Kambhampati et al. 2024; Huang, Lipovetzky, and Cohn 2025), but models trained from scratch on formal problem descriptions remain underexplored, with the notable exception of *PlanGPT* (Rossetti et al. 2024), an autoregressive decoder which we compare our approach against. We evaluate coverage and plan quality on instances of comparable and slightly larger than training size, study how the number of denoising steps affects performance, and inspect the order in which tokens are revealed during inference.

Background

Classical Planning. We consider classical planning with full observability and deterministic actions. A (lifted STRIPS) planning task (Fikes and Nilsson 1971) is a tuple $\langle P, O, A, I, G\rangle$, where P is a set of predicates, O a set of typed objects, A a set of action schemas, I the initial state and G the goal condition. A state is a set of ground atoms over P and O. Each action schema $\mathcal{A} \in A$ has a precondition $pre(\mathcal{A})$, an add list $add(\mathcal{A})$ and a delete list $del(\mathcal{A})$. A ground action a is obtained by instantiating the variables of a schema with objects of matching type; it is applicable in state s if $pre(a) \subseteq s$ and yields the successor state $s' = (s \setminus del(a)) \cup add(a)$. A *plan* is a sequence of applicable actions that leads from I to a state s^* with $G \subseteq s^*$.

Deep Learning for Automated Planning. Deep learning in planning has mainly been used for heuristic prediction, where a model estimates the cost of reaching the goal from a given state (Toyer et al. 2020; Chen, Trevizan, and Thiébaux 2024; Ståhlberg, Bonet, and Geffner 2025). A more recent

line of work studies sequence-to-sequence formulations, in which a model generates a plan given a planning problem encoded as a sequence of state and goal atoms (Pallagani et al. 2022; Rossetti et al. 2024; Fritzsche, Gestrin, and Seipp 2026). These models are framed as plan generators, but their autoregressive prediction of actions conditioned on states can also be interpreted as learning a policy. In this work, we generate complete plans directly from the initial state, without explicitly learning a policy over intermediate states.

Diffusion with Masking for Text Generation. Diffusion models are commonly associated with continuous noise processes, such as Gaussian perturbations of images. Extending diffusion to discrete symbolic spaces requires a different formulation of the forward process. We adopt *masked diffusion* (Nie et al. 2025), where corruption replaces tokens with a special mask symbol rather than adding noise. This formulation is closely related to masked language modeling as used in BERT (Devlin et al. 2019).

In masked text diffusion, a parametric model $M_\theta(r_t)$ takes a partially masked response r_t as input and predicts the clean tokens at every position. Training minimizes a cross-entropy loss restricted to the masked positions (Shi et al. 2024; Sahoo et al. 2024; Ou et al. 2025):

$$\mathcal{L} = \min_{\theta} \; \mathbb{E}_{r_0,t,r_t}\left[\frac{1}{t}\sum_{i\in m} \ell_{\mathrm{CE}}(M_\theta(r_t)_i,\; r_{0,i})\right] \qquad (1)$$

Here, r_0 is a clean response, $t \in [0,1]$ is the masking probability drawn uniformly at training time, ℓ_{CE} is the cross-entropy loss and m is the set of masked indices in r_t. The $1/t$ factor rescales the loss so that each sampled noise level contributes comparably, independently of how many tokens happen to be masked. In the referenced work (Nie et al. 2025), the parametric model M_θ is a transformer architecture named *LLaDA*, whose design inspires the deep neural network used in this study.

Plan Diffusion

We represent a plan as a sequence of *tokens* $x_0 = \langle a_1, \ldots, a_L\rangle$ drawn from a fixed discrete vocabulary that contains symbols for action schemas, objects and a few control symbols. We condition generation on $c = (I, G)$, the initial state and goal of the planning task, also encoded as a sequence of tokens. Our objective is to model the conditional distribution $p(x_0 \mid c)$ with a masked diffusion process over the plan tokens.

The underlying principle of diffusion models is to progressively perturb the observed data with a forward process, then recover the original data through a reverse process. This formulation is commonly modeled using two Markov chains (see Figure 1 for an illustrative example in the context of this work). The forward process gradually replaces plan tokens with a special `<MASK>` symbol. We index the corruption by a continuous time $t \in [0,1]$. At $t = 0$ the plan is clean; at $t = 1$ every plan token has been replaced by `<MASK>`; and for intermediate t, each plan token is independently masked with probability t. We denote the resulting partially masked sequence by x_t. The reverse process recovers the data distribution by iteratively predicting the masked tokens as t moves from 1 to 0.

Our model M_θ is a *transformer*, a neural network that maps a sequence of input tokens to a sequence of output vectors. At each output position, the network produces a vector of unnormalized scores (*logits*) over the vocabulary; a softmax then turns these logits into a probability distribution over the next-token prediction at that position. Inside the network, every layer refreshes each position's representation by mixing in information from the other positions through a mechanism called *self-attention*. Autoregressive language models restrict this mixing so that a position can only attend to earlier tokens. Masked diffusion has no such directional bias. An unmasked plan token can appear anywhere in the sequence and is informative for predicting any masked position. We therefore use bidirectional self-attention, so each position attends to the entire sequence.

We form the input to the transformer by concatenating the tokenized condition c and the partially masked plan x_t into a single sequence $[c\,;\,x_t]$. The condition tokens are never masked and provide the problem-specific context that the model conditions on. Figure 2 illustrates the training procedure. We sample an optimal plan x_0 from the training set, draw a masking level $t \sim \mathcal{U}(0,1)$, and apply the forward process to obtain x_t. We then feed $[c\,;\,x_t]$ to M_θ, read off the predicted distributions at the masked plan positions and compute the loss from (1) against the ground-truth tokens.

The model operates on a discrete vocabulary including symbols for predicates, goal predicates, action symbols and object identifiers. Predicate and goal-predicate symbols occupy disjoint regions of the vocabulary, so even though the condition concatenates state and goal atoms in one sequence, the model can tell from a token's identity whether it describes the current state or the goal. The first tokens of the sequence encode the state and goal atoms. Following Fritzsche, Gestrin, and Seipp (2026), we represent each (goal) atom as a single token, e.g., the ground atom $on(o_1, o_2)$ becomes one symbol `on(o1,o2)` rather than three separate tokens for the predicate name and the two arguments. A `<BOS>` token marks the start of the plan. Inside the plan, we split each ground action into one action-schema token followed by one token per object argument, e.g., `load`, `pkg1`, `truck1`, `locA` instead of a single token for the whole ground action. This keeps the output vocabulary small since the model only needs to recognize the relatively few action schemas and objects of a domain, rather than every possible ground action. An `<EOS>` token terminates the plan, and we right-pad the sequence to a fixed maximum length with `<PAD>` tokens so that all training examples share the same length. Appendix A gives further details on the tokenization and on the positional encoding of the condition.

Figure 3 illustrates the inference stage of the diffusion process. At inference, we generate a plan by reversing the forward process in T discrete steps with $\Delta = 1/T$. We start at $t = 1$ from a sequence in which every plan position holds `<MASK>`, and we proceed $x_1 \to x_{1-\Delta} \to \cdots \to x_0$. At each step, we feed the current sequence and condition to the

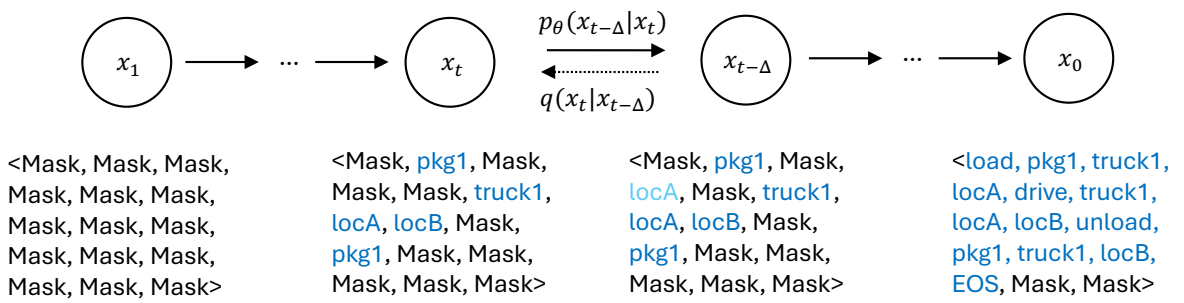

Figure 1: Illustration of the masked diffusion process. The model learns to transform a fully masked sequence into samples representative of the true data distribution, thereby generating feasible plans for a given problem instance.

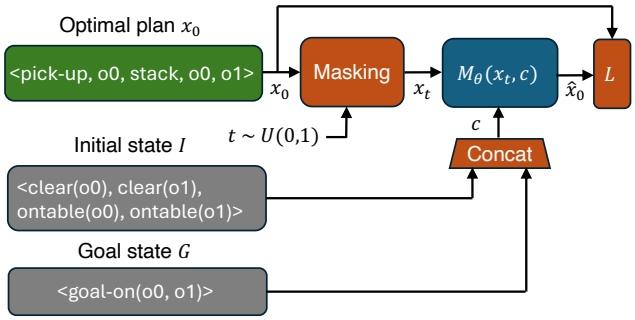

Figure 2: Illustration of the training process for plan generation. The model takes as input the condition $c = (I, G)$ and the corrupted plan x_t to predict the optimal plan x_0. The loss is computed using (1).

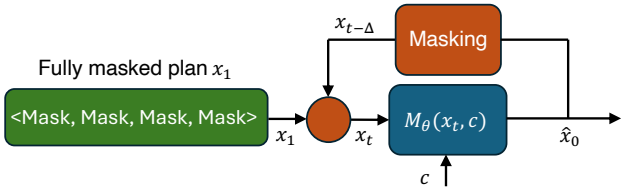

Figure 3: Illustration of the inference process for plan generation. Starting from a fully masked sequence, the model predicts the plan, keeps the tokens with highest confidence and remasks the rest for the next inference step at $t - \Delta$. This process repeats until all tokens before <EOS> are unmasked.

model, which predicts a distribution over the vocabulary at every plan position; we take the most probable symbol at each position as a tentative reconstruction $\hat{x}_0$ of the clean plan. We then form $x_{t-\Delta}$ by committing the tokens of $\hat{x}_0$ whose predicted probability is highest and remasking the remaining positions. The number of committed tokens per step follows a fixed schedule, so that by $t = 0$ all plan positions are unmasked. The clean plan $\hat{x}_0$ is predicted at every step of inference, and the number of inference iterations is a design choice.

Experiments

We evaluate the diffusion planner in terms of coverage and plan quality and inspect the inference process to understand how plans are generated.

Datasets. We use four planning domains: *Blocksworld*, *Gripper*, *Visitall* and *Logistics*. In *Blocksworld*, a set of blocks arranged in towers must be stacked into a single tower in a specific order. In *Gripper*, a robot with two grippers must move a set of balls from one room to another. In *Visitall*, an agent must visit all cells in a grid. In *Logistics*, trucks and airplanes must transport packages from their initial locations to their goal locations. To build the training set, we generate problem instances of varying sizes (e.g., number of blocks in *Blocksworld*) and fully expand each instance's state space. For every reachable state, we compute an optimal plan to the goal and use the resulting state-plan pairs as training examples. We evaluate each model on two sets of problems: *validation* contains instances slightly larger than those used for training, and *interpolation* contains instances whose parameter values lie between the training sizes. Appendix B provides the PDDL representations of the domains and the size and distribution of the problems.

Implementation Details. Our denoiser $M_\theta(x_t, c)$ follows the LLaDA framework and uses a LLaMA-style transformer (Touvron et al. 2023) backbone with rotary positional embeddings (RoPE) (Su et al. 2024), pre-normalization, bidirectional self-attention and a gated feed-forward multilayer perceptron (MLP) with Sigmoid Linear Unit (SiLU) activation (Ramachandran, Zoph, and Le 2017). The model has 12 transformer layers with a hidden dimension of 768, 8 attention heads and an MLP intermediate size of 3072, yielding approximately 113 M trainable parameters. This is substantially smaller than the original LLaDA 8B model and reflects the comparatively small vocabularies of planning domains. We train one model per domain on a single NVIDIA A100 GPU for 20 hours. Each run uses a node with 16 CPU cores and 16 GiB of system memory. We repeat each experiment with three random seeds, and for each run we select the checkpoint with the best validation performance for final evaluation.

Domain	Dataset	PlanGPT		SymT		Diffusion	
		Coverage ($\mu \pm \sigma$)	Quality ($\mu \pm \sigma$)	Coverage ($\mu \pm \sigma$)	Quality ($\mu \pm \sigma$)	Coverage ($\mu \pm \sigma$)	Quality ($\mu \pm \sigma$)
Blocks	validation	$.00 \pm .00$	$.00 \pm .00$	$1.00 \pm .00$	$1.00 \pm .00$	$.11 \pm .19$	$.11 \pm .19$
	interpolation	$.56 \pm .16$	$.56 \pm .16$	$1.00 \pm .00$	$1.00 \pm .00$	$.56 \pm .51$	$.56 \pm .51$
Gripper	validation	$.00 \pm .00$	$.00 \pm .00$	$.17 \pm .24$	$.17 \pm .24$	$.00 \pm .00$	$.00 \pm .00$
	interpolation	$.00 \pm .00$	$.00 \pm .00$	$.67 \pm .00$	$.67 \pm .00$	$.00 \pm .00$	$.00 \pm .00$
Visitall	validation	$.00 \pm .00$	$.00 \pm .00$	$.33 \pm .09$	$.32 \pm .10$	$.00 \pm .00$	$.00 \pm .00$
	interpolation	$.05 \pm .04$	$.05 \pm .04$	$.87 \pm .01$	$.87 \pm .02$	$.11 \pm .00$	$.11 \pm .00$
Logistics	validation	$.00 \pm .00$	$.00 \pm .00$	$.00 \pm .00$	$.00 \pm .00$	$.00 \pm .00$	$.00 \pm .00$
	interpolation	$.07 \pm .05$	$.07 \pm .05$	$.22 \pm .31$	$.22 \pm .31$	$.00 \pm .00$	$.00 \pm .00$

Table 1: Normalized coverage and plan quality for PlanGPT, SymT and the proposed diffusion model. A coverage score of 1.00 means that all tasks are solved; a quality score of 1.00 means that all generated plans are optimal.

Quantitative Assessment. We measure performance with *coverage*, the fraction of problems for which the model generates a valid plan, and *plan quality*, the length of the generated plan compared to the best known plan length. We compare against the autoregressive decoder-only model *PlanGPT* (Rossetti et al. 2024), using the numbers reported by Fritzsche, Gestrin, and Seipp (2026) with comparable trainable parameters. PlanGPT generates plans by greedy decoding, where at each step the token with the highest predicted probability is appended until an end-of-sequence token is produced. Alternative decoding strategies, such as applicability-filtered plan generation, can change the picture substantially (Fritzsche, Gestrin, and Seipp 2026). We use greedy decoding because it matches the deterministic inference rule of our diffusion model, which always commits the tokens with the highest predicted probability. For reference, we additionally include results from Fritzsche, Gestrin, and Seipp (2026), denoted as SymT (Symmetry-aware Transformer), which adds symmetry-aware losses and architectural refinements to the autoregressive baseline with only 16M parameters.

Table 1 reports coverage and plan quality. On the interpolation set, the diffusion model is on par with PlanGPT on *Blocksworld* (.56 vs. .56), exceeds it on *Visitall* (.11 vs. .05), underperforms on *Logistics* (.00 vs. .07) and ties at zero coverage on *Gripper*, where neither approach solves any instance. On the (harder) validation set, both models almost always fail, with the diffusion model solving a small fraction of *Blocksworld* instances (.11) that PlanGPT does not solve. SymT is substantially stronger on every domain, showing that there is considerable room for improvement within the autoregressive family alone. We view this comparison as preliminary, since the three models differ in capacity, training time and vocabulary, which makes a fully controlled comparison difficult. In every row of the table, coverage and quality coincide, which means whenever a generated plan is valid, it is also optimal. We attribute this to training exclusively on optimal plans.

Qualitative Assessment. To understand how the model arrives at a plan, we inspect its inference trajectory on a Blocksworld instance with a 20-step schedule (Table 2). Unlike an autoregressive decoder, the diffusion model reveals tokens in a non-left-to-right order, committing action schemas and object arguments at different positions across iterations. During the early reverse steps the plan portion of the sequence remains largely masked, while positions beyond the eventual plan are progressively committed to <PAD>, which means the model decides where the plan ends before it decides what the plan contains. In this run, the first action token appears at step 16; the model then fills in arguments and additional actions. The <EOS> token is revealed at step 19, one step before the last plan tokens, so the predicted plan length is fixed before all action details are resolved.

This ordering is a direct consequence of confidence-based unmasking combined with our fixed-length encoding, which does impose an upper bound on plan length. In each step, we commit the positions whose predicted distribution is most peaked, and we leave the uncertain ones for later. Whether a position lies beyond the plan is a structural property of the task that depends mainly on the goal and the initial state, so the model can become confident about <PAD> positions well before it has settled on individual actions. Once a contiguous block of <PAD> tokens is committed at the tail of the sequence, the location of <EOS> is essentially determined by the boundary between plan and padding. The action tokens themselves are the genuinely ambiguous part of the prediction and are therefore decided last.

Inference Step Effect. As noted in the previous section, the model predicts a full plan $\hat{x}_0$ at every denoising step, so a candidate solution is available already after a single forward pass. To assess how much the later iterations contribute, we evaluate coverage as a function of the number of denoising steps. Figure 4 shows that coverage rises sharply over the first few iterations and then saturates, while inference time grows linearly in the number of steps. Choosing a step budget that balances coverage and runtime is therefore a design parameter rather than a free knob, and a principled schedule is left to future work.

Conclusions and Future Work

We presented a masked diffusion framework for classical planning that generates an entire action sequence by iteratively denoising masked tokens, in contrast to the autore-

Step	Token sequence	#MASK	#PAD
1	<BOS>, <MASK>, <MASK>, ..., <MASK>, <PAD>, <MASK>, ...	209	11
10	<BOS>, <MASK>, <MASK>, ..., <MASK>, <PAD>, <MASK>, ...	110	110
15	<BOS>, <MASK>, <MASK>, ..., <MASK>, <PAD>, <MASK>, ...	55	165
17	<BOS>, unstack, <MASK>, <MASK>, put-down, o3, <MASK>, <MASK>, <MASK>, ..., <PAD>, <MASK>, ...	33	184
18	<BOS>, unstack, o3, o1, put-down, o3, <MASK>, o1, <MASK>, <MASK>, o1, <MASK>, ..., <PAD>, <MASK>, ...	22	191
19	<BOS>, unstack, o3, o1, put-down, o3, unstack, o1, o0, <MASK>, o1, <MASK>, o0, <MASK>, stack, o0, <MASK>, <MASK>, o1, <MASK>, o1, <MASK>, pick-up, <MASK>, stack, <MASK>, <MASK>, <EOS>, ...	11	192
20	<BOS>, unstack, o3, o1, put-down, o3, unstack, o1, o0, put-down, o1, unstack, o0, o2, stack, o0, o3, pick-up, o1, stack, o1, o0, pick-up, o2, stack, o2, o1, <EOS>, <PAD>, ...	0	193

Table 2: Inference trajectory of the diffusion plan generator on a Blocksworld instance with four objects. Blue tokens are newly revealed compared to the previous step. The last two columns report the number of remaining <MASK> and <PAD> tokens.

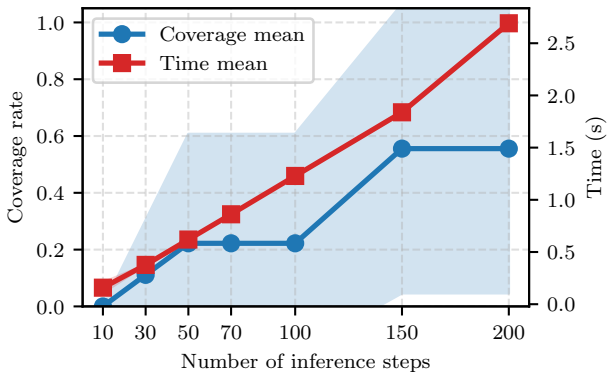

Figure 4: Coverage and inference time as a function of the number of denoising steps on *Blocksworld*. Shaded areas show the standard deviation across three independently trained diffusion models, evaluated on the interpolation dataset.

gressive token-by-token generation of prior plan-generation models. Trained from scratch on four standard benchmark domains, the model achieves coverage and plan quality comparable to the PlanGPT baseline but trails the symmetry-aware transformer of Fritzsche, Gestrin, and Seipp (2026) on all four domains. We view these results as preliminary evidence that diffusion is a viable approach for plan generation, and we identify three directions for follow-up work.

First, we assign object names randomly so the model does not rely on any semantic information carried by the names themselves. While this avoids overfitting to superficial naming patterns, prior work suggests that random naming can hurt generalization. A natural next step is to make the denoiser explicitly equivariant under object renamings, along the lines of the symmetry-aware training of Fritzsche, Gestrin, and Seipp (2026).

Second, our experiments show that coverage saturates as the number of denoising steps grows, but each additional step costs a forward pass through the transformer. One way to make few-step inference more accurate is to change the training-time noise schedule, where instead of sampling the masking probability uniformly from $[0, 1]$, we can use a distribution biased toward high masking probabilities, so the model sees more heavily corrupted inputs during training and learns to recover from them in fewer steps.

Finally, the formulation here corrupts plans by replacing tokens with a special mask symbol. An alternative is to embed plan tokens in a continuous space and apply Gaussian noise in that space, which mirrors the original diffusion processes used for images. This may be a good fit for planning, where the action vocabulary is discrete and small, but the semantic relations between actions, e.g., applicability and effect overlap, carry structure that a learned embedding can capture.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Computational resources were provided on the Berzelius system funded by the Knut and Alice Wallenberg foundation and operated by NAISS.

References

Alcaraz, J. M. L.; and Strodthoff, N. 2023. Diffusion-based time series imputation and forecasting with structured state space models. *Transactions on Machine Learning Research*.

Berglund, L.; Tong, M.; Kaufmann, M.; Balesni, M.; Stickland, A. C.; Korbak, T.; and Evans, O. 2024. The Reversal Curse: LLMs trained on "A is B" fail to learn "B is A". In *International Conference on Learning Representations*.

Carvalho, J.; Le, A. T.; Kicki, P.; Koert, D.; and Peters, J. 2025. Motion Planning Diffusion: Learning and Adapting Robot Motion Planning with Diffusion Models. *IEEE Transactions on Robotics*, 41: 4881–4901.

Chen, D. Z.; Trevizan, F.; and Thiébaux, S. 2024. Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning. In *Proc. ICAPS 2024*, 68–76.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 4171–4186.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *AIJ*, 2: 189–208.

Fritzsche, M.; Gestrin, E.; and Seipp, J. 2026. Symmetry-Aware Transformer Training for Automated Planning. In *Proc. AAAI 2026*.

He, Z.; Sun, T.; Tang, Q.; Wang, K.; Huang, X.; and Qiu, X. 2023. DiffusionBERT: Improving Generative Masked Language Models with Diffusion Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 4521–4534.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 6840–6851.

Hoogeboom, E.; Satorras, V. G.; Vignac, C.; and Welling, M. 2022. Equivariant diffusion for molecule generation in 3D. In *International Conference on Machine Learning*, 8867–8887.

Huang, S.; Lipovetzky, N.; and Cohn, T. 2025. Planning in the dark: LLM-symbolic planning pipeline without experts. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence*.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. arXiv:2201.07207 [cs.LG].

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and B Murthy, A. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Proceedings of the 41st International Conference on Machine Learning*, 22895–22907.

Karras, T.; Aittala, M.; Aila, T.; and Laine, S. 2022. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems*, 26565–26577.

Li, X.; Thickstun, J.; Gulrajani, I.; Liang, P. S.; and Hashimoto, T. B. 2022. Diffusion-LM Improves Controllable Text Generation. In *Advances in Neural Information Processing Systems*, volume 35, 4328–4343.

Nie, S.; Zhu, F.; You, Z.; Zhang, X.; Ou, J.; Hu, J.; Zhou, J.; Lin, Y.; Wen, J.-R.; and Li, C. 2025. Large Language Diffusion Models. In *Advances in Neural Information Processing Systems*.

Ou, J.; Nie, S.; Xue, K.; Zhu, F.; Sun, J.; Li, Z.; and Li, C. 2025. Your Absorbing Discrete Diffusion Secretly Models the Conditional Distributions of Clean Data. In *International Conference on Learning Representations*.

Pallagani, V.; Muppasani, B.; Murugesan, K.; Rossi, F.; Horesh, L.; Srivastava, B.; Fabiano, F.; and Loreggia, A. 2022. Plansformer: Generating Symbolic Plans using Transformers. *arXiv preprint arXiv:2212.08681*.

Ramachandran, P.; Zoph, B.; and Le, Q. V. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Ramesh, A.; Dhariwal, P.; Nichol, A.; Chu, C.; and Chen, M. 2022. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*.

Rasul, K.; Seward, C.; Schuster, I.; and Vollgraf, R. 2021. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International Conference on Machine Learning*, 8857–8868.

Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P.; and Ommer, B. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10684–10695.

Rossetti, N.; Tummolo, M.; Gerevini, A. E.; Putelli, L.; Serina, I.; Chiari, M.; and Olivato, M. 2024. Learning General Policies for Planning through GPT Models. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 500–508.

Sahoo, S. S.; Arriola, M.; Schiff, Y.; Gokaslan, A.; Marroquin, E.; Chiu, J. T.; Rush, A.; and Kuleshov, V. 2024. Simple and Effective Masked Diffusion Language Models. In *Advances in Neural Information Processing Systems*, volume 37, 130136–130184.

Shi, J.; Han, K.; Wang, Z.; Doucet, A.; and Titsias, M. 2024. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems*, volume 37, 103131–103167.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J. B.; Kaelbling, L.; and Katz, M. 2024. Generalized planning in PDDL domains with pretrained large language models. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence*.

Sohl-Dickstein, J.; Weiss, E.; Maheswaranathan, N.; and Ganguli, S. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2256–2265.

Song, J.; Meng, C.; and Ermon, S. 2021. Denoising diffusion implicit models. In *International Conference on Learning Representations*.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2025. Learning more expressive general policies for classical planning domains. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence*.

Su, J.; Ahmed, M.; Lu, Y.; Pan, S.; Bo, W.; and Liu, Y. 2024. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing*, 568: 127063.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *JAIR*, 68: 1–68.

Yang, L.; Zhang, Z.; Song, Y.; Hong, S.; Xu, R.; Zhao, Y.; Zhang, W.; Cui, B.; and Yang, M.-H. 2024. Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4): 105:1–105:39.

Appendix A Tokenization

Typically, <PAD> tokens are used to fill the remaining positions in the sequence when the actual plan is shorter than the maximum sequence length, ensuring that all input sequences have a consistent length for batch processing. In plan diffusion, however, we cannot predict the sequence length *a priori*, and we therefore predict <PAD> tokens as part of the generation process. We represent each action by an action symbol followed by its parameter object symbols. Padding to a fixed length limits the maximum plan length, but avoids the problem that learned positional encodings cannot operate on positions unseen during training. When applying autoregressive models trained with such encodings to instances larger than those seen during training, the generated plans can exceed the maximum trained position. In our setting, the model must instead predict <PAD> tokens for positions beyond the actual plan length, with the trade-off that it cannot generate plans longer than the maximum sequence length. Following Fritzsche, Gestrin, and Seipp (2026), we prevent the model from exploiting positional information among (goal) atoms, which form an unordered set. Since we encode each (goal) atom as a single token, we shift the positional encoding of RoPE by a dynamic offset to match the <BOS> token.

To support unknown objects at test time, we follow the approach of PlanGPT (Rossetti et al. 2024) and use a fixed number of object tokens, e.g., o0, o1, o2, etc., and we randomly permute the mapping between object tokens and actual objects during training.

Appendix B Planning Domains and Dataset Sizes

In this section, we provide high-level descriptions of the planning domains used in our experiments and their PDDL representations. In addition, we report the sizes of the training, validation and interpolation sets for each domain in Table 3. During training, we fully expand each training instance and sample from the complete state space. As such, our approach mitigates the risk of only observing limited regions of the state space, while also significantly reducing the number of problems required for training. We choose the number of training problems and instances to ensure a comparable range of training sizes to that of PlanGPT (Rossetti et al. 2024), which used 63,000 problems per domain.

Domain	Parameter	#Training	Training Sizes	Validation Sizes	Interpolation Sizes
Blocksworld	#blocks	9	4, 6, 7	8	5
Gripper	#balls	4	2, 4, 6, 8	9, 10	3, 5, 7
Logistics	#goals	12	1, 3, 5	6	2, 4
Visitall	#cells	207	1, 3, 4, 6, 10, 11, 12, 14, 16	18, 20	2, 5, 8, 9, 15

Table 3: Problem sizes for the four benchmark subsets. We select training problems that allow exhaustive exploration of their state spaces.

Blocksworld

A Blocksworld problem consists of a set of blocks that can be picked up and stacked on top of each other or placed on the table. The goal is to stack all blocks into a single stack in a specified order.

```
(define (domain blocks)
 (:requirements :strips)
 (:predicates (on ?x ?y) (ontable ?x) (clear ?x) (handempty) (holding ?x))

 (:action pick-up
  :parameters (?x)
  :precondition (and (clear ?x) (ontable ?x) (handempty))
  :effect (and (not (ontable ?x)) (not (clear ?x))
               (not (handempty)) (holding ?x)))

 (:action put-down
  :parameters (?x)
  :precondition (holding ?x)
  :effect (and (not (holding ?x)) (clear ?x)
               (handempty) (ontable ?x)))

 (:action stack
  :parameters (?x ?y)
  :precondition (and (holding ?x) (clear ?y))
  :effect (and (not (holding ?x)) (not (clear ?y))
```

```
               (clear ?x) (handempty) (on ?x ?y)))

 (:action unstack
  :parameters (?x ?y)
  :precondition (and (on ?x ?y) (clear ?x) (handempty))
  :effect (and (holding ?x) (clear ?y)
               (not (clear ?x)) (not (handempty))
               (not (on ?x ?y))))
)
```

Gripper

A Gripper problem consists of a robot with two gripper arms that can pick up and drop balls and move between two rooms. The goal is to move all balls from room A to room B.

```
(define (domain gripper)
 (:requirements :strips :negative-preconditions)
 (:predicates (room ?r) (ball ?b) (gripper ?g) (at-robby ?r)
              (at ?b ?r) (free ?g) (carry ?o ?g))

 (:action move
  :parameters (?from ?to)
  :precondition (and (room ?from) (room ?to) (at-robby ?from))
  :effect (and (at-robby ?to) (not (at-robby ?from))))

 (:action pick
  :parameters (?obj ?room ?gripper)
  :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
                     (at ?obj ?room) (at-robby ?room) (free ?gripper))
  :effect (and (carry ?obj ?gripper)
               (not (at ?obj ?room))
               (not (free ?gripper))))

 (:action drop
  :parameters (?obj ?room ?gripper)
  :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
                     (carry ?obj ?gripper) (at-robby ?room))
  :effect (and (at ?obj ?room)
               (free ?gripper)
               (not (carry ?obj ?gripper))))
)
```

Logistics

A Logistics problem consists of a set of trucks, locations, airports, cities, airplanes, and packages. The goal is to move all packages from their initial locations to their goal locations. Cities contain locations, some of which are airports. Trucks can move packages between locations within a city, while airplanes can move packages between airports.

```
(define (domain logistics)
 (:requirements :strips)
 (:predicates (package ?obj) (truck ?truck) (airplane ?airplane)
              (airport ?airport) (location ?loc) (in-city ?obj ?city)
              (city ?city) (at ?obj ?loc) (in ?obj1 ?obj2))

 (:action load-truck
  :parameters (?obj ?truck ?loc)
  :precondition (and (package ?obj) (truck ?truck) (location ?loc)
                     (at ?truck ?loc) (at ?obj ?loc))
  :effect (and (not (at ?obj ?loc)) (in ?obj ?truck)))

 (:action load-airplane
  :parameters (?obj ?airplane ?loc)
  :precondition (and (package ?obj) (airplane ?airplane) (location ?loc)
                     (at ?obj ?loc) (at ?airplane ?loc))
```

```
  :effect (and (not (at ?obj ?loc)) (in ?obj ?airplane)))

 (:action unload-truck
  :parameters (?obj ?truck ?loc)
  :precondition (and (package ?obj) (truck ?truck) (location ?loc)
                     (at ?truck ?loc) (in ?obj ?truck))
  :effect (and (not (in ?obj ?truck)) (at ?obj ?loc)))

 (:action unload-airplane
  :parameters (?obj ?airplane ?loc)
  :precondition (and (package ?obj) (airplane ?airplane) (location ?loc)
                     (in ?obj ?airplane) (at ?airplane ?loc))
  :effect (and (not (in ?obj ?airplane)) (at ?obj ?loc)))

 (:action drive-truck
  :parameters (?truck ?loc-from ?loc-to ?city)
  :precondition (and (truck ?truck) (location ?loc-from)
                     (location ?loc-to) (city ?city)
                     (at ?truck ?loc-from) (in-city ?loc-from ?city)
                     (in-city ?loc-to ?city))
  :effect (and (not (at ?truck ?loc-from)) (at ?truck ?loc-to)))

 (:action fly-airplane
  :parameters (?airplane ?loc-from ?loc-to)
  :precondition (and (airplane ?airplane) (airport ?loc-from)
                     (airport ?loc-to) (at ?airplane ?loc-from))
  :effect (and (not (at ?airplane ?loc-from)) (at ?airplane ?loc-to)))
)
```

Visitall

A Visitall problem consists of a set of locations arranged in a four-connected rectangular grid and an agent that must visit all cells.

```
(define (domain grid-visit-all)
 (:requirements :strips :negative-preconditions)
 (:predicates (place ?x) (connected ?x ?y)
              (at-robot ?x) (visited ?x))

 (:action move
  :parameters (?curpos ?nextpos)
  :precondition (and (place ?curpos) (place ?nextpos)
                     (at-robot ?curpos)
                     (connected ?curpos ?nextpos))
  :effect (and (at-robot ?nextpos)
               (not (at-robot ?curpos))
               (visited ?nextpos)))
)
```