Symmetry-Aware Transformer Training for Automated Planning

Markus Fritzsche*, Elliot Gestrin*, Jendrik Seipp

Linköping University, Sweden
firstname.lastname@liu.se

Abstract

While transformers excel in many settings, their application in the field of automated planning is limited. Prior work like PlanGPT, a state-of-the-art decoder-only transformer, struggles with extrapolation from easy to hard planning problems. This in turn stems from problem symmetries: planning tasks can be represented with arbitrary variable names that carry no meaning beyond being identifiers. This causes a combinatorial explosion of equivalent representations that pure transformers cannot efficiently learn from. We propose a novel contrastive learning objective to make transformers symmetry-aware and thereby compensate for their lack of inductive bias. Combining this with architectural improvements, we show that transformers can be efficiently trained for either plan-generation or heuristic-prediction. Our results across multiple planning domains demonstrate that our symmetry-aware training effectively and efficiently addresses the limitations of PlanGPT.

Code — https://github.com/mrlab-ai/symmetry-transformers
Data — https://doi.org/10.5281/zenodo.17591697
Appendix — https://arxiv.org/abs/2508.07743

1 Introduction

Transformer architectures (Vaswani et al. 2017) have revolutionized various fields, from natural language processing (e.g., Tunstall, Von Werra, and Wolf 2022) to computer vision (e.g., Zhai et al. 2022). While leveraging transformers for automated planning—complex reasoning tasks that require sequential decisions—holds significant promise, transformers specifically trained for this purpose remain rare. This is surprising, given that solving automated planning problems is inherently a sequence generation task: finding a sequence of actions that leads from the initial state to a goal state.

We posit that a key challenge lies in the inherent ambiguity of planning problem descriptions, combined with a lack of inductive bias in pure transformer architectures. While pretrained large language models have been explored for solving planning problems (e.g., Pallagani et al. 2022; Huang et al. 2022; Silver et al. 2024; Kambhampati et al. 2024; Huang, Lipovetzky, and Cohn 2025), models trained from scratch for generating plans from structured planning inputs are much less common. A notable exception is *PlanGPT* (Rossetti et al. 2024), a state-of-the-art GPT-2-based transformer (Radford et al. 2019) trained to predict plans from formal problem descriptions.

However, PlanGPT faces several limitations, particularly concerning extrapolation to more challenging instances beyond its training set. A primary reason for this is that the same planning problem can have a huge number of different input representations since 1) objects can be given arbitrary names, since they only serve as identifiers and carry no semantic meaning, and 2) the atoms of initial and goal states can be ordered arbitrarily. Pure transformer architectures must implicitly learn to ignore the differences between such *symmetric* planning problems, which is challenging and sample-inefficient. Furthermore, relying on an explicitly learned positional encoding can hinder generalization, as the model may struggle with novel input sizes and unseen positions.

We propose a symmetry-aware contrastive learning objective for transformers that explicitly accounts for the symmetries present in planning problems, guiding the model to learn representations that are equivariant to these symmetries (see Figure 1). While we focus on automated planning, this objective is applicable to any task using transformer tokens that primarily represent identifiers. To address remaining symmetries in the input representation, we highlight the importance of architectural choices, especially the omission of explicit positional encodings. Furthermore, we use a compositional tokenization scheme along with a transformer encoder, benefiting from its inherent permutation-equivariance. We denote our architectures as **Sym**metry-Aware **T**ransformers (SymT).

We evaluate our approach in two different settings: 1) *plan generation* using an encoder-decoder (SymT^{ED}) and 2) *heuristic prediction* using an encoder-only transformer (SymT^{E}). Our experiments show that our symmetry-aware contrastive learning objective and architectural innovations significantly improve planning performance compared to the PlanGPT baseline in three of the four evaluated domains, particularly in extrapolating to harder planning problems where our models find plans for many tasks that PlanGPT fails to solve. These findings showcase the importance of explicitly addressing input symmetries to unlock the full potential of transformers for automated planning and general reasoning tasks that use variable names.

*These authors contributed equally.

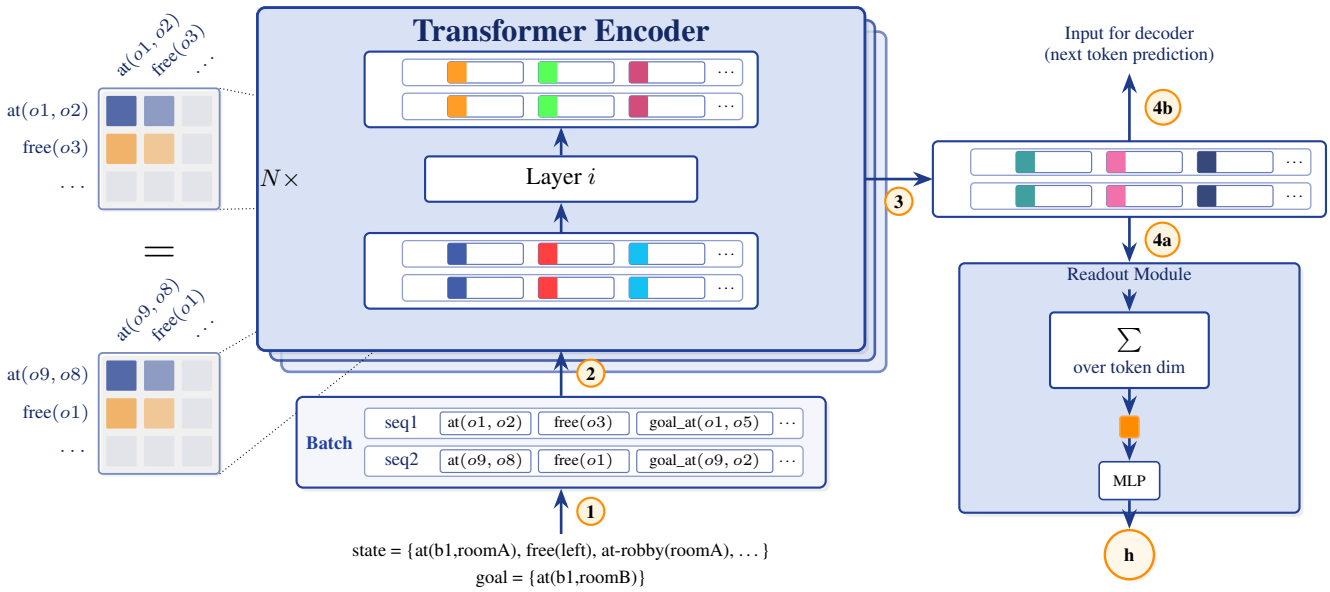

Figure 1: Illustration of our contrastive loss and the encoder part of our architecture. First, (1) we create two copies of the state and goal, each mapped into a token sequence, where each object is renamed to a random object token. While forwarding the sequences to the encoder (2), all attention scores and hidden states are collected for loss computations. The first objective is to align the attention scores for both samples in all attention-based modules (illustrated as equal attention scores). The second objective is to align fractions of the hidden states for all pairwise-corresponding tokens (illustrated by matching colors in the image). The output (3) can either be forwarded to the decoder (4b, Figure A.1 in the appendix) for next-token prediction or to a readout module for heuristic computation (4a). When predicting heuristics, we sum the hidden state fractions used in the contrastive loss and pass them through an MLP to obtain the prediction.

2 Background

Classical Planning. We consider classical planning, where the world is fully-observable and actions are deterministic. Formally, a (lifted STRIPS) planning problem (Fikes and Nilsson 1971) is a tuple $\langle \mathcal{P}, \mathcal{O}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$. Here, $\mathcal{P}$ is a finite set of predicates that describe properties of the world (e.g., *at*, *in*, *connected*) and the arity of predicate $p \in \mathcal{P}$ is its number of arguments. $\mathcal{O}$ is a finite set of objects that exist in the world (e.g., *pkg1*, *truck3*, *loc2*). Each object has a type (e.g., *Package*, *Truck*, *Location*). Using predicates and objects, we form (ground) atoms, which are basic propositions about the world (e.g., *at(pkg1, loc2)*). A state is a set of ground atoms that are true in a given situation and the initial state $\mathcal{I}$ describes which atoms are true initially. We denote the set of all possible states by $\mathcal{S}$. An atom can also contain variables (e.g., *at(?pkg, ?loc)*), which act as placeholders for objects. Such *lifted* atoms are used within action schemas $A \in \mathcal{A}$, which are templates for actions in the world. Each action schema $A \in \mathcal{A}$ consists of a precondition $pre(A)$, an add list $add(A)$ and a delete list $del(A)$. The precondition $pre(A)$ is a set of atoms that must be true in a state for the action to be applicable (e.g., *at(?pkg, ?loc)* and *at(?truck, ?loc)*). The add list $add(A)$ is a set of atoms that will be true after the action is applied (e.g., *in(?pkg, ?truck)*), and the delete list $del(A)$ is a set of atoms that will no longer be true after the action is applied (e.g., *at(?pkg, ?loc)*). We ground an action schema A to obtain a ground action by replacing all variables in $pre(A)$, $add(A)$ and $del(A)$ with objects from $\mathcal{O}$, respecting the types of the variables and objects. A ground action a is applicable in state s if $pre(a) \subseteq s$, and applying it results in the successor state $s' = (s \setminus del(a)) \cup add(a)$. The goal $\mathcal{G}$ is the set of atoms that must be true simultaneously to solve the planning problem. A sequence of ground actions is a *plan* if it is applicable from $\mathcal{I}$ and reaches a goal state s^* such that $\mathcal{G} \subseteq s^*$. Optimal planning is the task of finding a plan with minimum length. Here, we focus on satisficing planning, where any plan is acceptable, but short plans are preferred. Finally, a heuristic $h : \mathcal{S} \to R_0^+ \cup \infty$ is a function estimating the cost of reaching a goal state from a given state.

Transformer Architectures for Sequence Generation. Pure transformers (Vaswani et al. 2017) consist of an encoder and a decoder, each consisting of a stack of multiple layers. The input to each layer is a sequence of token embeddings, and the output is a sequence of hidden state vectors. Both types of layers consist of a multi-head attention (MHA) mechanism and a position-wise feed-forward network (MLP), with residual connections and optional layer normalization applied after each sub-layer. Multi-head attention and self-attention (A) are defined as

$$\alpha_i(Q, K) = \text{softmax}\left(Q_i K_i^T / \sqrt{d_k}\right)$$
$$\text{A}_i(Q, K, V) = \alpha_i(Q, K) V_i$$
$$\text{MHA}(Q, K, V) = (\text{A}_1(Q, K, V) | ... | \text{A}_L(Q, K, V)) W_O,$$

where $W_O \in \mathbb{R}^{d \times d}$ is a learned projection combin-

ing the L attention heads, and $|$ denotes concatenation. $Q, K, V \in \mathbb{R}^{N\times d}$ are the query, key, and value matrices, with $Q_i, K_i, V_i \in \mathbb{R}^{N\times d_k}$ the i-th head and $d_k = d/L$.

An encoder layer transforms the input $X^{(i)} \in \mathbb{R}^{N\times d}$ into the output $X^{(i+1)} \in \mathbb{R}^{N\times d}$ as follows:

$$Q^{(i)} = X^{(i)}W_Q^{(i)} \quad K^{(i)} = X^{(i)}W_K^{(i)} \quad V^{(i)} = X^{(i)}W_V^{(i)}$$
$$X_{\text{att}}^{(i)} = X^{(i)} + \text{MHA}^{(i)}\left(Q^{(i)}, K^{(i)}, V^{(i)}\right)$$
$$X^{(i+1)} = X_{\text{att}}^{(i)} + \text{MLP}^{(i)}\left(X_{\text{att}}^{(i)}\right)$$

Here, $W_Q^{(i)}, W_K^{(i)}, W_V^{(i)} \in \mathbb{R}^{d\times d}$ are learned weight matrices specific for the i-th layer. We omit layer normalization for simplicity. Optionally, a bias term can be added to the Q, K, and V matrices, but we omit it here for simplicity.

We let E denote the output of the last encoder layer and $Y \in \mathbb{R}^{M\times d}$ the decoder input sequence. A decoder layer transforms the input $Y^{(i)}$ into the output $Y^{(i+1)}$ as follows:

$$Q_Y^{(i)} = Y^{(i)}W_{Q,1}^{(i)} \quad K_Y^{(i)} = Y^{(i)}W_{K,1}^{(i)} \quad V_Y^{(i)} = Y^{(i)}W_{V,1}^{(i)}$$
$$Y_{\text{att}}^{(i)} = Y^{(i)} + \text{MHA}^{(i)}\left(Q_Y^{(i)}, K_Y^{(i)}, V_Y^{(i)}\right)$$
$$Q_{Y_{\text{att}}}^{(i)} = Y_{\text{att}}^{(i)}W_{Q,2}^{(i)} \quad K_E^{(i)} = EW_{K,2}^{(i)} \quad V_E^{(i)} = EW_{V,2}^{(i)}$$
$$\tilde{Y}^{(i)} = Y_{\text{att}}^{(i)} + \text{MHA}^{(i)}\left(Q_{Y_{\text{att}}}^{(i)}, K_E^{(i)}, V_E^{(i)}\right)$$
$$Y^{(i+1)} = \tilde{Y}^{(i)} + \text{MLP}^{(i)}\left(\tilde{Y}^{(i)}\right)$$

The first MHA module in the decoder masks out future tokens using a triangular mask, ensuring that the model can only attend to previous tokens in the sequence. A final MLP predicts the probability of the next token in the sequence, generating the output sequence one token at a time, referred to as autoregressive generation. Typically, the decoder is trained using the cross-entropy loss between the predicted token probabilities and the true next token in the sequence. Decoder-only variants, such as GPT-based models (e.g., Radford et al. 2018; Rossetti et al. 2024), focus solely on this sequential generation process and use only self-attention.

Transformers for Automated Planning. The primary target for Deep Learning in planning has been the prediction of heuristics, where the model is trained to predict the estimated cost of reaching the goal from a given state (e.g., Toyer et al. 2020; Ståhlberg, Bonet, and Geffner 2022a; Chen, Trevizan, and Thiébaux 2024). While transformers can be used for predicting heuristics, most work has focused on the sequence-to-sequence setting, where the model is trained to predict a plan (sequence of actions) given a planning problem, i.e., a sequence of state and goal atoms (Pallagani et al. 2022; Rossetti et al. 2024). Unlike architectures with strong built-in assumptions about local connectivity such as graph neural networks (GNNs), transformers possess limited inductive bias regarding sequence structure, relying heavily on data to learn dependencies. Handling sequence order explicitly requires positional information, commonly via positional encodings, as the attention mechanism itself is permutation-invariant. This data-driven nature and reliance on learned positional signals pose a challenge, particularly when input sequences can exhibit symmetries or structural variations not explicitly encoded, as is the case in planning problems.

PlanGPT. *PlanGPT* (Rossetti et al. 2024) is a decoder-only transformer architecture, trained to predict a plan for a given planning problem. The model processes the planning problem by segmenting it into an input token sequence. The input begins with a <*start*> token, followed by tokens representing the initial state, where each atom is decomposed into predicate and argument tokens (e.g., atom *at(truck5, loc3)* becomes the token sequence *at*, *truck5*, *loc3*). A <*goal*> token separates the state from the goal description, tokenized similarly. The input sequence concludes with an <*action*> token, signaling the start of the plan output. The output sequence comprises tokens representing the predicted actions, also segmented into action name and argument tokens, until the end-of-sequence token <*EOS*> is generated. An example input-output sequence for a successfully-solved simplified Logistics tasks is: <*start*> *at truck5 loc1 at pkg1 loc1* ... <*goal*> *at pkg1 loc2* <*action*> *load truck5 pkg1 loc1 move truck5 loc1 loc2 unload truck5 pkg1 loc2* <*EOS*>. To handle symmetries induced by arbitrary object names, PlanGPT randomizes object names based on their types from a fixed vocabulary, e.g., *truck5* becomes *truck2*. Training is supervised using a standard next-token prediction objective, namely cross-entropy loss. The authors use a training dataset that contains multiple (suboptimal) plans for each planning problem, allowing the model to learn from diverse solutions.

3 Limitations of PlanGPT

PlanGPT is trained solely to minimize the next-token prediction loss. While this objective can in principle capture symmetries, the combinatorial explosion of input representations leads to poor sample efficiency in practice.

Limitation 1: Object Assignment Equivariance. Object names in planning are arbitrary and not intended to carry additional semantics. However, PlanGPT uses a fixed vocabulary of object names that are randomly assigned to objects in the planning problem. Assume a single object type with $|\mathcal{O}|$ objects in the training instance and $|\mathcal{V}|$ object names in vocabulary $\mathcal{V}$ with $|\mathcal{V}| \geq |\mathcal{O}|$. Then there are $\frac{|\mathcal{V}|!}{(|\mathcal{V}|-|\mathcal{O}|)!}$ different assignments of objects to vocabulary names. For example, a task with $|\mathcal{O}| = 8$ objects and $|\mathcal{V}| = 8$ names yields $\frac{8!}{(8-8)!} = 40320$ different assignments, all describing the same planning task. The model must learn to generalize across all assignments to predict plans equivariant to these name assignments, which is challenging without additional objectives.

Limitation 2: Leaking Information in Object Names. To mitigate Limitation 1, PlanGPT does not randomize object names of all types. In the Visitall domain, for instance, objects of type *Location* encode grid coordinates, e.g., *loc-x1-y2* corresponds to coordinate $\langle 1, 2\rangle$. Instead of randomizing these names, PlanGPT keeps the human-intended coordinates, which the model can memorize, preventing generalization to locations not seen during training. We demonstrate this in Appendix B1.

Limitation 3: Atom Order Invariance. The order of atoms in the initial and goal states of a planning problem is arbitrary. Assume there are $|\mathcal{I}|$ atoms in the state and $|\mathcal{G}|$ atoms in the goal description. Then there are $|\mathcal{I}|!$ different atom orders for the same state and $|\mathcal{G}|!$ different orders for the goal, yielding $|\mathcal{I}|! \cdot |\mathcal{G}|!$ equivalent representations for each object assignment from Limitation 1. PlanGPT is a decoder-only model that uses learned positional encodings to capture token order in the input sequence. To make atom order consistent across training instances, PlanGPT pre-sorts the atoms in the state and goal descriptions before passing them to the model. Nevertheless, the model may overfit to particular atoms being overrepresented in specific positions, which can hinder generalization to larger instances where atoms appear in different positions.

Limitation 4: Learned Positional Encodings. In contrast to state and goal atoms, the order of actions in a plan matters. Since transformer layers are permutation-equivariant functions,[1] PlanGPT uses learned absolute positional encodings (Radford et al. 2019) to capture the order of input tokens. This is problematic for state and goal tokens due to Limitation 3. Moreover, for any token type, learned positional encodings imply that embeddings for unseen positions are not learned, preventing generalization to harder planning problems with more objects, which typically require longer plans.

4 Symmetry-Aware Training

This section introduces our contrastive training objective and architecture choices for overcoming the limitations identified for PlanGPT. An overview of our objective and encoder-only architecture for heuristics is shown in Figure 1, while our encoder-decoder architecture for plan generation is shown in Figure A.1 (Appendix A).

Addressing Atom Order Symmetries with Encoders (Limitation 3). To directly address atom order symmetries, we use an encoder-only architecture for heuristic prediction and an encoder-decoder architecture for plan generation. The encoder processes the state and goal atoms, while the decoder predicts the action sequence forming the plan. By omitting positional encodings in the encoder, its processing is inherently permutation-equivariant with respect to the order of input atom tokens. This makes its output representation independent of their arbitrary order, yielding a $|\mathcal{I}|! \cdot |\mathcal{G}|!$ reduction of the input space.

PlanGPT tokenizes each atom into several tokens (predicate and arguments), each mapped to an individual learnable embedding and requiring an ordering. As this is incompatible with an encoder without positional encodings, we instead encode each atom as a single embedding. Let $E[\text{token}]$ denote the learnable embedding vector for a token. For an atom $p(o_1, \ldots, o_n)$ grounded from predicate p and objects $o_1, \ldots, o_n$, we concatenate $E[p]$, $E[o_1], \ldots, E[o_n]$, and $m - n$ padding token embeddings to obtain a vector of fixed length $|E[\text{token}]| \cdot (1 + m)$, where m is the maximum predicate arity in the domain. We then pass this vector through a single linear layer with parameters $\mathbf{W}$ and $\mathbf{b}$ to produce the atom embedding $T_{p(o_1,\ldots,o_n)}$:

$$T_{p(o_1,\ldots,o_n)} = \mathbf{W}(E[p]|E[o_1]|\ldots|E[o_n]|E[\text{pad}]|\ldots) + \mathbf{b}$$

This linear transformation maps the concatenated embedding to the dimensionality of the atom embeddings used by the encoder.[2] We further avoid splitting the input into state and goal subsequences by introducing new goal-predicates for every predicate. For example, the goal atom *at(truck3, loc2)* becomes *goal_at(truck3, loc2)*. By converting all atoms into these atom-level embeddings and operating without positional encodings, the encoder produces a representation that is guaranteed to be permutation-equivariant in the order of input atom tokens, addressing Limitation 3.

Addressing Positional Encodings in the Decoder (Limitation 4). Predicting heuristic values only requires an order-invariant state representation and is therefore suitable for an encoder-only architecture without positional encodings. In contrast, the order of actions in a plan is essential and must be preserved. As noted in Limitation 4, learned positional encodings can struggle to generalize to unseen positions and thus limit extrapolation. To mitigate this issue in the decoder, we omit positional encodings entirely. This technique, *NoPE*, has shown successful length generalization in various sequence-to-sequence tasks, including reasoning problems (Kazemnejad et al. 2023).

Contrastive Training for Object Name Equivariance (Limitations 1 and 2). To obtain robust object name assignment equivariance (Limitation 1), architectural changes alone are insufficient. We therefore introduce a contrastive training objective that encourages the model to learn representations and processing patterns equivariant to arbitrary object name assignments.

The core idea is to train on pairs of symmetric planning problems. For a given problem, we generate two encoder input sequences, X and X', that share the same underlying state and goal structure but differ only in object names. We similarly generate two corresponding decoder output sequences, Y and Y', which are the plans predicted for X and X', respectively. Since X and X' represent the same planning problem, the model should process them in a way that reflects this equivalence. We propose two renaming modes: *Rename-One*, where object names in X are fixed across training instances and object names in X' are randomized, and *Rename-Both*, where object names in both X and X' are randomized. We then rename the objects in Y and Y' accordingly. Preliminary experiments showed that Rename-One works better for predicting heuristics, whereas Rename-Both is better for plan generation. We encourage equivalent behavior across both sequences using two complementary contrastive losses.

The first, the attention loss L_{att}, is motivated by the intuition that if the model encodes the same algorithm regardless

[1] If we reorder the input sequence of tokens according to some permutation, the output sequence will be the original output sequence reordered by the same permutation.

[2] Directly encoding atoms with unique tokens, i.e., $E[p(o_1, \ldots, o_n)]$, would induce a massive vocabulary that scales poorly in the number of supported objects, making it likely that some atoms are rarely seen during training and causing poor generalization.

	PlanGPT	SymT^{E}	SymT^{ED}
# Parameters	117M	7M	16M

Table 1: Parameter counts for PlanGPT and our architectures.

of names, the attention scores between corresponding parts of the input must match across layers and heads (Figure 1). We define L_{att} as:

$$L_{\text{att}} = \frac{1}{B} \sum_{\alpha \in Att} \sum_{i=1}^{\#\text{rows}(\alpha)} \sum_{j=1}^{\#\text{cols}(\alpha)} (\alpha_{i,j} - \alpha'_{i,j})^2$$

Here, B denotes the batch size and *Att* the set of all attention modules across layers and heads. α and α' are the attention scores produced by the transformer for (X, Y) and (X', Y'), respectively.

The second component, the hidden state loss L_{hid}, encourages similarity directly in the learned token representations. It is based on the idea that the hidden state vector for a token should encode problem-structure features that are independent of the specific object names. We compute L_{hid} as:

$$L_{\text{hid}} = \frac{1}{B} \sum_{H \in \{X,Y\}} \sum_{l=1}^{\#\text{layers}(H)} (H^{(l)}_{[:d_k]} - H'^{(l)}_{[:d_k]})^2$$

Here, $H^{(l)}$ denotes the hidden states after the l:th encoder layer when $H = X$ and the l:th decoder layer when $H = Y$, and $[:d_k]$ denotes a projection selecting the first d_k dimensions of these hidden states.

We combine these with the standard prediction loss: $L = w_1 \cdot L_{\text{pred}} + w_2 \cdot L_{\text{att}} + w_3 \cdot L_{\text{hid}}$. Here, L_{pred} is the next-token prediction loss (cross-entropy for plan generation, MSE for heuristic prediction), and w_1, w_2, and w_3 are hyperparameters controlling the importance of each component. We set all w_i to 1 in our experiments. Minimizing this objective incentivizes the model to predict correct actions or heuristic values while learning attention patterns and hidden state features that are equivariant to object name variations. For predicting heuristics with our encoder-only architecture, we feed $\sum_N X^{(N)}_{[:d_k]}$ instead of $\sum_N X^{(N)}$ to the readout MLP to obtain the final heuristic prediction (Figure 1). We additionally address Limitation 2 by always fully randomizing object names.

Summary of Our Architectures. We introduce two symmetry-aware transformers: SymT^{E}, an encoder-only architecture for heuristic prediction, and SymT^{ED}, an encoder-decoder architecture for plan generation. Inspired by the GNNs of Ståhlberg, Bonet, and Geffner (2022a), both share weights across layers to reduce the number of parameters, see Table 1. This is strictly necessary; otherwise L_{hid} would collapse to a trivial solution for slices of hidden states in non-final layers. Both models use the contrastive loss described above to ignore object names. Thus, our heuristic model is a permutation-equivariant, shared-weight encoder, and our plan generation model extends it with a shared-weight decoder using NoPE positional encodings for length generalization.

5 Plan Generation

There are many ways to use transformer models to generate plans, differing in how much they rely on the model versus search techniques such as backtracking or cycle detection. To assess the performance of the model rather than the search strategy, we consider three minimally supportive plan generation strategies.

Greedy Plan Generation. At each step, we autoregressively select the token with the highest predicted probability and stop when an end-of-sequence token is generated.

Applicability-Filtered Plan Generation. This strategy again autoregressively selects the most probable token, but first masks the probabilities so that only tokens leading to an applicable action are considered. For example, if the last two tokens generated are *move* and *roomA*, then the model can only generate a token corresponding to a room reachable from *roomA* (e.g., *roomB*). Plan generation stops when the simulated state satisfies the goal. This strategy dominates the greedy plan generation strategy.

Regrounding Applicability-Filtered Plan Generation. This strategy filters applicable actions as above. After each action is generated, it is applied to update the state, and the model's input sequence is reset to this new state. Essentially, we alternate between predicting (partial) plans of length one and applying them. Again, we stop when the found state satisfies the goal. This strategy does not dominate either of the two above.

Greedy Heuristic Guidance. For the heuristic-prediction model, we generate plans by computing the heuristic value of each successor state and greedily selecting the action that leads to the state with the lowest heuristic value. This process is repeated until reaching a goal state.

For each strategy, we also stop plan generation early if the model generates more than 500 tokens. Pseudo-code for all strategies is in Appendix D.

6 Experiments, Results and Limitations

We now evaluate our symmetry-aware training, focusing on training efficiency and extrapolation capabilities.

Experimental Setup. Since the benchmarks used by PlanGPT are unsuitable for our purposes, due to significant training/test set overlap (see Appendix B2) and a lack of dedicated extrapolation data, we generated our own datasets for four widely used planning domains with characteristics relevant to PlanGPT's limitations: *Blocksworld*, *Gripper*, *Visitall* and *Logistics*. In Blocksworld, a set of blocks in multiple towers must be stacked in a specific order into a single tower. 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, a set of trucks and airplanes must transport packages from their initial locations to their goal locations. Appendix C details the domains, their PDDL representations, and the size and distribution of problems for each domain.

We train models for each combination of domain and architecture on an NVIDIA A100 GPU with 12 hours of training

		PlanGPT - Decoder (baseline)			SymT^{E} (ours)	SymT^{ED} (ours)		
		greedy	applicable	regrounding	greedy	greedy	applicable	regrounding
Blocks	validation	.00±.00	.00±.00	.00±.00	**1.00±.00**	**1.00±.00**	**1.00±.00**	.00±.00
	interpolation	.56±.16	.56±.16	.00±.00	**1.00±.00**	**1.00±.00**	**1.00±.00**	**1.00±.00**
	extrapolation	.00±.00	.00±.00	.00±.00	.05±.07	.07±.02	**.13±.05**	.00±.00
Gripper	validation	.00±.00	.00±.00	.00±.00	**1.00±.00**	.17±.24	**1.00±.00**	**1.00±.00**
	interpolation	.00±.00	.44±.16	.00±.00	.89±.16	.67±.00	**1.00±.00**	**1.00±.00**
	extrapolation	.00±.00	.00±.00	.00±.00	.02±.03	.00±.00	.15±.06	**.79±.16**
Visitall	validation	.00±.00	.14±.12	.00±.00	**1.00±.00**	.33±.09	.93±.04	.99±.02
	interpolation	.05±.04	.67±.18	.41±.22	**1.00±.00**	.87±.01	.99±.01	**1.00±.00**
	extrapolation	.00±.00	.02±.02	.00±.00	.42±.11	.00±.00	.15±.05	**.64±.12**
Logistics	validation	.00±.00	**.08±.12**	.00±.00	.00±.00	.00±.00	.00±.00	.00±.00
	interpolation	.07±.05	**.44±.09**	.19±.14	.11±.00	.22±.31	.26±.29	.22±.31
	extrapolation	**.00±.00**	**.00±.00**	**.00±.00**	**.00±.00**	**.00±.00**	**.00±.00**	**.00±.00**

Table 2: Normalized coverage scores $[(\mu \pm \sigma)]$ for all architectures when decoding with the various generation strategies. A coverage score of 1.00 means that the configuration solves all tasks. The best results are highlighted in bold.

per model; Appendix E gives further details. We then evaluate them using all plan generation strategies from Section 5 on three sets of problems: validation (slightly larger than training and used for checkpoint selection), interpolation (comparable but distinct sizes), and extrapolation (larger than validation). For robustness, we use three random seeds per configuration. For the PlanGPT baseline, we avoid leaking object names by always randomizing all object names. We additionally perform ablation studies to analyze the impact of our contrastive loss (adding it to PlanGPT and removing it from our models) and to examine the effect of the number of objects on the contrastive loss; see Appendix E2 and E3, respectively. Performance is primarily evaluated based on *coverage*, i.e., the percentage of problems for which a valid plan is generated, and *plan quality*, i.e., the length of the generated plans compared to best known plan lengths.

Results – Coverage. Coverage is shown in Table 2. PlanGPT can solve instances similar in size to the training problems (interpolation), but fails entirely on larger instances regardless of decoding strategy. In contrast, both our architectures solve larger instances to varying degrees. If regrounding extrapolates well for a domain, applicability-filtered plan generation tends to perform worse, and vice versa, with regrounding generally preferable. SymT^{ED} also consistently outperforms SymT^{E} in coverage. No approach extrapolates in Logistics, which aligns with known GNN results (Ståhlberg, Bonet, and Geffner 2022a) based on expressiveness limitations. Logistics is also the only domain where PlanGPT outperforms our models, likely due to its larger number of parameters allowing better fit to the training data. Even there, however, our SymT^{ED} model outperforms PlanGPT in both greedy and regrounding settings.

Results – Plan Quality. Appendix E2 reports plan quality tables, where we see that the generated plans are often optimal or close to optimal. This likely stems from training exclusively on optimal plans, encouraging the models to reproduce such policies. Although not our main focus, we also observe that the SymT^{ED} finds shorter plans in seconds than the classical planner LAMA (Richter and Westphal 2010) does in two hours for several problems (we use LAMA to obtain reference plan lengths), indicating that the learned policies are practically useful.

Results – Training Stability. Training with the contrastive objective yields lower validation losses, but the coverage gains are most pronounced for the heuristic prediction model. Following Abbe et al. (2024), we hypothesize that this is because predicting goal distances requires considering all input tokens, whereas autoregressive next-action prediction does not. For example, in Gripper predicting the next action only requires considering a few balls, while the goal distance depends on all balls.

Across many domains, losses plateau for SymT^{E} models without the contrastive term, while models with it continue to improve. We also observed frequent training instabilities without the contrastive loss for SymT^{E} and SymT^{ED}, leading to sudden loss divergence; standard stabilization techniques such as gradient clipping or reduced learning rates did not help. Across all trained models (including ablations and hyperparameter tuning), only a single model diverged with the contrastive loss, compared to 19 without it. We discuss the link between the contrastive objective and training stability in Appendix E.

Limitations. While we outperform PlanGPT on extrapolation, our models still have clear limitations. They extrapolate to some extent under greedy plan generation, but the applicability-filtered and regrounding strategies perform better, indicating that the learned algorithm is not perfect. In the hardest domain, Logistics, we cannot extrapolate to larger instances at all. Extrapolation success likely depends on several factors, such as the choice of positional encodings, and further work is needed to understand their impact on length generalization. Another limitation is that all transformer mod-

els, including PlanGPT, rely on a fixed-size vocabulary, so problems with more objects than vocabulary entries cannot be solved. As we show in Appendix E3, however, our contrastive loss allows stable training with much larger vocabularies, making it viable to choose a large vocabulary ahead of training as we do in our experiments.

7 Related Work

Learning-based approaches for automated planning increasingly complement traditional symbolic methods, aiming to improve performance, handle more complex domains, and enable faster planning.

Learning for Planning with Graph Neural Networks. GNNs (Scarselli et al. 2008) are the predominant deep learning architecture for learning directly from structured planning problem representations (Ståhlberg, Bonet, and Geffner 2022a,b; Chen and Thiébaux 2024; Ståhlberg, Bonet, and Geffner 2025a; Horčik et al. 2025). Their inductive bias towards processing graph-structured data, where nodes and edges can represent atoms and their relationships, aligns well with the relational nature of planning problems. However, standard GNNs are theoretically bounded by their expressive power, aligning with the 1-dimensional Weisfeiler–Lehman (WL) test (Xu et al. 2018), and may struggle to distinguish WL-indistinguishable graph structures. Approaches like higher-order GNNs (Ståhlberg, Bonet, and Geffner 2025b) have been explored to lift this expressiveness barrier for planning tasks. Despite their success, GNNs have not been exploited for generating plans in the context of next-token generation.

Transformers for Planning and Structured Data. Applying transformer architectures directly to structured planning problem inputs is less common than using GNNs. Notable examples include *PlanGPT* (Rossetti et al. 2024), which treats planning as a sequence generation task, and *Dualformer* (Su et al. 2024), which trains transformers on search traces for navigation and puzzle tasks. A key challenge for these sequence-based models is extrapolation to larger instances. Transformers, particularly those relying on learned absolute positional encodings, are often limited in their ability to generalize to sequence lengths significantly different from those seen during training (Su et al. 2024; Rossetti et al. 2024). Methods like iterative self-training (Lee et al. 2025) show that transformers can be trained to extrapolate on some tasks, including graph-encoded mazes, but typically require complex multi-round training strategies. To the best of our knowledge, training transformers for out-of-distribution extrapolation on STRIPS planning problems *without* such multi-stage fine-tuning remains an open challenge.

Large Language Models for Planning. Recent research increasingly leverages pre-trained large language models (LLMs) for solving planning problems. Some work fine-tunes LLMs specifically for planning tasks (Pallagani et al. 2022) or uses them to generate programs that solve planning problems (Silver et al. 2024; Katz et al. 2024). Other strategies employ LLMs to generate plans directly from natural language problem descriptions (Huang et al. 2022), with increasingly intricate prompting strategies (Yao et al. 2023; Sel et al. 2024; Gestrin, Kuhlmann, and Seipp 2024; Sel, Jia, and Jin 2025) and sometimes assisted by external tools (Kambhampati et al. 2024), or to translate natural language into PDDL tasks that can be solved with off-the-shelf planners (Liu et al. 2023; Oswald et al. 2024; Huang, Lipovetzky, and Cohn 2025). These approaches capitalize on the extensive knowledge embedded within LLMs and their inherent reasoning capabilities. In contrast, our work focuses on training models directly on structured planning data (like PDDL tokens) to obtain robust and generalizable plan generation models.

Equivariant and Invariant Learning. Training models to be equivariant or invariant to specific input transformations is a fundamental theme in deep learning, notably in computer vision (e.g., CNNs for translation invariance) and GNNs (permutation-equivariance). For planning problems, symmetries such as the arbitrary assignment of object names introduce complex permutations that ideal learned models should handle. Some work explores architectural modifications to achieve equivariance for structured data, e.g., Graphormer (Ying et al. 2021). Relevant to object name symmetries, *Renamer* (Ankner, Renda, and Carbin 2023) proposes transformer architectural changes for semantics-preserving variable renaming. Our work complements such architectural efforts by exploring training objectives.

Objectives on Intermediate Representations. Standard training often focuses on the final output loss (e.g., prediction error), but some methods introduce auxiliary losses on intermediate layer outputs, such as hidden states or attention weights, to encourage desired properties in the learned representations. We found no prior work that specifically introduces a contrastive learning objective applied directly to attention scores or targeted hidden state projections *between symmetrically equivalent inputs* with the explicit goal of inducing object name assignment equivariance in a transformer model. However, there is work that uses losses on attention for other purposes (Patro et al. 2021).

8 Conclusions and Future Work

By introducing a novel contrastive training objective designed to encourage object name equivariance in the learned representations, combined with architectural choices to address planning-related symmetries, we showed improved inter- and extrapolation performance compared to the state-of-the-art PlanGPT baseline across multiple planning domains. However, despite significant gains over the baseline, our model is still unable to consistently generate valid plans for problems *significantly* larger than those encountered during training. This outcome reinforces that achieving robust, out-of-distribution extrapolation on complex symbolic reasoning tasks remains a substantial challenge for existing transformer architectures and training paradigms.

Future work should identify the architectural or representational bottlenecks that limit generalization. Also, addressing the practical limitation of a fixed vocabulary size by exploring representation learning techniques independent of token vocabularies is crucial for tackling very large tasks.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

Abbe, E.; Bengio, S.; Lotfi, A.; Sandon, C.; and Saremi, O. 2024. How far can transformers reason? the globality barrier and inductive scratchpad. *Advances in Neural Information Processing Systems*, 37: 27850–27895.

Ankner, Z.; Renda, A.; and Carbin, M. 2023. Renamer: A Transformer Architecture In-variant to Variable Renaming.

Chen, D.; and Thiébaux, S. 2024. Graph learning for numeric planning. *Advances in Neural Information Processing Systems*, 37: 91156–91183.

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.

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.

Gestrin, E.; Kuhlmann, M.; and Seipp, J. 2024. NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions. In *ICAPS Workshop on Human-Aware and Explainable Planning*.

Horčik, R.; Šír, G.; Šimek, V.; and Pevnỳ, T. 2025. State Encodings for GNN-Based Lifted Planners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 26525–26533.

Huang, S.; Lipovetzky, N.; and Cohn, T. 2025. Planning in the Dark: LLM-Symbolic Planning Pipeline without Experts. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 26542–26550.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In *Proc. ICML*, 9118–9147. PMLR.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Proc. ICML*, 22895–22907. PMLR.

Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi Araghi, S. 2024. Thought of search: Planning with language models through the lens of efficiency. *Advances in Neural Information Processing Systems*, 37: 138491–138568.

Kazemnejad, A.; Padhi, I.; Natesan Ramamurthy, K.; Das, P.; and Reddy, S. 2023. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36: 24892–24928.

Lee, N.; Cai, Z.; Schwarzschild, A.; Lee, K.; and Papailiopoulos, D. 2025. Self-Improving Transformers Overcome Easy-to-Hard and Length Generalization Challenges. *arXiv preprint arXiv:2502.01612*.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. arXiv:2304.11477.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large language models as planning domain generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 423–431.

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:2212.08681.

Patro, B. N.; GS, K.; Jain, A.; and Namboodiri, V. P. 2021. Self supervision for attention networks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 726–735.

Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I.; et al. 2018. Improving language understanding by generative pre-training.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language Models are Unsupervised Multitask Learners.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *JAIR*, 39: 127–177.

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.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

Sel, B.; Al-Tawaha, A.; Khattar, V.; Jia, R.; and Jin, M. 2024. Algorithm of thoughts: enhancing exploration of ideas in large language models. In *Proceedings of the 41st International Conference on Machine Learning*. JMLR.org.

Sel, B.; Jia, R.; and Jin, M. 2025. LLMs Can Plan Only If We Tell Them. *arXiv preprint arXiv:2501.13545*.

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 AAAI conference on artificial intelligence*, volume 38, 20256–20264.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Proc. ICAPS 2022*, 629–637.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In *Proc. KR 2022*, 474–483.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2025a. Learning More Expressive General Policies for Classical Planning Domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 26697–26706.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2025b. Learning More Expressive General Policies for Classical Planning Domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 26697–26706.

Su, D.; Sukhbaatar, S.; Rabbat, M.; Tian, Y.; and Zheng, Q. 2024. Dualformer: Controllable fast and slow thinking by learning with randomized reasoning traces. In *The Thirteenth International Conference on Learning Representations*.

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

Tunstall, L.; Von Werra, L.; and Wolf, T. 2022. *Natural language processing with transformers*. O'Reilly Media, Inc.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36: 11809–11822.

Ying, C.; Cai, T.; Luo, S.; Zheng, S.; Ke, G.; He, D.; Shen, Y.; and Liu, T.-Y. 2021. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34: 28877–28888.

Zhai, X.; Kolesnikov, A.; Houlsby, N.; and Beyer, L. 2022. Scaling Vision Transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 12104–12113.