Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR meets Machine Learning and Explanation

From Next Token Prediction to (STRIPS) World Models

Carlos Núñez-Molina[1], **Vicenç Gómez**[2], **Hector Geffner**[1]
[1]RWTH Aachen University, Germany
[2]Universitat Pompeu Fabra, Spain
{carlos.nunez,hector.geffner}@ml.rwth-aachen.de, vicen.gomez@upf.edu

Abstract

We study whether next-token prediction can yield world models that truly support planning, in a controlled symbolic setting where propositional STRIPS action models are learned from action traces alone and correctness can be evaluated exactly. We introduce two architectures. The first is the STRIPS Transformer, a symbolically aligned model grounded in theoretical results linking transformers and the formal language structure of STRIPS domains. The second is a standard transformer architecture without explicit symbolic structure built in, for which we study different positional encoding schemes and attention aggregation mechanisms. We evaluate both architectures on five classical planning domains, measuring training accuracy, generalization, and planning performance across domains and problem sizes. Interestingly, both approaches can be used to produce models that support planning with off-the-shelf STRIPS planners over exponentially many unseen initial states and goals. Although the STRIPS Transformer incorporates a strong symbolic inductive bias, it is harder to optimize and requires larger datasets to generalize reliably. In contrast, a standard transformer with stick-breaking attention achieves near-perfect training accuracy and strong generalization. Finally, standard transformers without stick-breaking attention do not generalize to long traces, whereas a symbolic STRIPS model extracted from a transformer trained on shorter traces does.

1 Introduction

A number of recent works have considered whether LLMs, and in particular transformer architectures, can learn world models as a result of learning to predict the next token autoregressively (Yildirim and Paul 2024; Vafa et al. 2024). This question is central because genuine understanding requires the ability to make meaningful predictions from an internal model of the world, and it remains unclear whether LLMs actually acquire such models, rather than relying solely on surface statistical regularities (Mitchell 2023).

In order to test the hypothesis that transformer networks can learn world models when predicting the next token, several works have examined controlled game environments such as Chess and Othello (Toshniwal et al. 2022; Li et al. 2023; Nanda 2023). Learning world models that support planning, however, requires learning both a representation of states and a model of how those states evolve, and while transformers have been shown to learn latent state representations, these representations have not been found to be good enough to support planning (Vafa et al. 2024).

This work is aimed at relating the tasks of next token prediction and world-model learning using transformers in the very concrete setting of propositional STRIPS world models. Propositional STRIPS world or action models have been standard in AI for many years (Russell and Norvig 2010; Ghallab, Nau, and Traverso 2004; Geffner and Bonet 2013), and they represent deterministic MDPs in compact form, where the states are given by sets of atoms, and actions add and delete atoms, provided that their preconditions are true. An atom in STRIPS stands for a Boolean variable, and an atom in the state, which is a collection of atoms, represents a Boolean variable that is true in the state.

For learning STRIPS models from action traces, we introduce two specialized architectures: the STRIPS Transformer and the Stick-Breaking (SB) Transformer. Both are motivated by recent results connecting hard-attention transformers and the B-RASP formalism to star-free languages (Yang, Chiang, and Angluin 2024).[1] The STRIPS Transformer incorporates an explicit symbolic alignment to STRIPS structure, whereas the SB Transformer retains a standard decoder-style architecture but removes positional encodings and replaces softmax with stick-breaking attention, a sequential normalization mechanism (Tan et al. 2025). Both models learn STRIPS action models from sequences of applicable (positive) and non-applicable (negative) actions obtained from random walks together with observations of action applicability. We show experimentally that both architectures generalize compositionally over long horizons and support planning over exponentially many unseen initial states and goals using off-the-shelf STRIPS planners.

The paper is organized as follows. We first review related work and background on STRIPS and transformers. Then, we introduce the learning task, describe the STRIPS and SB transformers, their training process, and how STRIPS models are extracted and used for planning. Finally, we present the results of our experiments and summarize our findings.[2]

[1]For the connections with B-RASP, see the Appendix.

[2]Code and data at github.com/TheAeryan/strips-transformer. Extended version with Appendix at arxiv.org/abs/2509.13389.

2 Related Work

Transformers and World Models. Transformer architectures trained for next-token prediction in the RL setting have motivated the question of whether sequence models can internalize aspects of environment dynamics. The common methodology in this line of work is to *probe* trained transformers in controlled environments to test whether latent state or causal structure can be inferred from the learned token embeddings (Li et al. 2023; Toshniwal et al. 2022; Nanda 2023; Rohekar et al. 2025; Vafa et al. 2025). By contrast, other approaches propose modified architectures or structured inductive biases to improve the usability of learned world models for planning and long-horizon reasoning, often by introducing explicit structure, abstraction, or symbolic components (Zhang, Yang, and Stadie 2021; Feng et al. 2025; Cano et al. 2025). None of these methods, however, are aimed at recovering exact symbolic models that can be used with off-the-shelf classical planners, as the methods proposed in this work.

Transformers and RASP languages. A simple programming language, called RASP, was introduced to capture the computations performed by transformers at a more abstract level (Weiss, Goldberg, and Yahav 2021), and a Boolean version of RASP, called B-RASP, has been proposed to characterize these computations in terms of a class of formal languages, specifically, the star-free languages (Yang, Chiang, and Angluin 2024). Interestingly, it has been shown that the language of valid action traces induced by a STRIPS planning domain is itself star-free (Lin and Bercher 2022), which suggests that B-RASP programs, and therefore hard-attention transformers, are expressive enough to recognize such domains. We build on this result in this work.

STRIPS Model Learning. Symbolic algorithms for learning STRIPS models from traces composed of actions and states have been considered in a number of works (Zhuo and Kambhampati 2013; Aineto, Celorrio, and Onaindia 2019; Lamanna et al. 2025). Other symbolic approaches, such as LOCM and SIFT, learn models from action traces alone (Cresswell, McCluskey, and West 2013; Gösgens, Jansen, and Geffner 2024), while others rely on action names together with the structure of the state graph (Bonet and Geffner 2020; Rodriguez et al. 2021). Fewer works address STRIPS model learning in the deep learning setting (Asai et al. 2022; Xi, Gould, and Thiébaux 2024), and none of them learn from action traces alone, as done in this work.

Pre-trained LLMs and Planning. Large language models (LLMs) have been increasingly explored as tools for planning. Early work investigated whether LLMs can act directly as planners by generating complete plans or plan fragments from task descriptions (Silver et al. 2022; Kambhampati et al. 2024; Pallagani et al. 2023). A complementary line of work incorporates LLMs into the planning process itself, using them to guide or structure combinatorial search (Hao et al. 2023; Yao et al. 2023; Besta et al. 2024; Katz et al. 2024).

Our work is most closely related to approaches that leverage LLMs to produce planning-related components, such as domain models or generalized planning programs, which are then input to classical planners or executed directly (Guan et al. 2023; Oswald et al. 2024; Gestrin, Kuhlmann, and Seipp 2024; Silver et al. 2024). Unlike these works, which rely on pre-trained LLMs, we learn a symbolic transition model from action-only traces, and evaluate generalization over exponentially many unseen initial states and goals.

3 Background

3.1 STRIPS Models

Propositional (grounded) STRIPS problems P are expressed as tuples of the form $P = \langle F, A, I, G\rangle$ where F is a set of atoms or propositions, A is set of actions, $I \subseteq F$ is a subset of atoms defining the initial situation, and $G \subseteq F$ is a subset of atoms defining the goal (Russell and Norvig 2010; Ghallab, Nau, and Traverso 2004; Geffner and Bonet 2013). The actions $a \in A$ are defined in terms of three subsets of atoms from F: the precondition list $\mathrm{pre}(a)$, the add list $\mathrm{add}(a)$, and the delete list $\mathrm{del}(a)$.

A problem $P = \langle F, A, I, G\rangle$ defines a state model (deterministic MDP) $S(P) = \langle S, s_0, S_G, A, A(\cdot), f\rangle$ where the states $s \in S$ are subsets of F, the initial state s_0 is I, the goal states $s \in S_G$ include all the goal atoms, $A(s)$ is the set of applicable actions in s given by those in A whose preconditions $\mathrm{pre}(a)$ are all true in s, and a deterministic state-transition function $f(a, s)$ that maps s into the state $s' = (s \cup \mathrm{add}(a)) \setminus \mathrm{del}(a)$ when $a \in A(s)$.

The atoms p in a state s represent the subset of atoms in F that are true in s. An action sequence $\tau = (a_0, \ldots, a_{n-1})$ over P is a sequence of actions from A. The action sequence is applicable in a state s in $S(P)$ if there is a sequence of states $s_0, \ldots, s_n$ in $S(P)$ such that each action a_i is applicable in s_i, and s_{i+1} is the state that results from action a_i in s_i, i.e., $s_{i+1} = f(a_i, s_i)$, $i = 0, \ldots, n-1$. The action sequence τ is a plan for P if it is applicable in the initial state of P and the resulting state s_n is a goal state.

A propositional STRIPS *domain* or *model* M is given by the set of atoms F and the set of actions A, as $M = \langle F, A\rangle$. A model is compatible with a number of *problem instances* $P = \langle F, A, I, G\rangle$, all sharing the same atoms and the same actions but differing in the initial situation I or goal G. The number of instances P of a given model M is exponential in the number of atoms F in M.

3.2 Transformers

We will use transformers for learning propositional STRIPS models from action sequences. Transformers (Vaswani et al. 2017) are suited for processing sequences of tokens $\tau = \langle b_0, \ldots, b_{n-1}\rangle$, outputting an updated sequence $\tau' = \langle b'_0, \ldots, b'_{n-1}\rangle$. Input tokens b_i are represented by real vectors often containing information about their position i, in which case they are called *positional encodings/embeddings*. Token embeddings (positional or not) are updated across one or more layers. Transformer layers contain feed-forward nets and perform the operation known as *self-attention*, where tokens receive signals from other tokens. Often, each layer will perform several self-attention com-

putations in parallel (each as a separate *attention head*), in order to extract different types of relations between tokens.

Given an input sequence of length n, self-attention computes, for each position i, a weighted combination of representations at positions j in the same sequence. In its simplest *scalar* form, each position i is associated with a real *query* $Q(i)$, *key* $K(i)$, and *value* $V(i)$. The compatibility between positions i and j is measured by a score

$$S(i, j) = Q(i) \cdot K(j).$$

To ensure that predictions at position i depend only on preceding positions, a *strict future mask* is applied by setting $S(i, j) = 0$ whenever $j \geq i$.

The output at position i is obtained by aggregating the values $V(j)$ according to weights derived from the masked scores $S(i, j)$. Different attention mechanisms correspond to different ways of selecting or normalizing these weights.

In *masked hard attention*, each position i selects a single preceding position $j < i$ with maximal score, breaking ties by choosing the largest such index. This form of attention has been studied formally in the context of B-RASP, where it is shown that the Boolean operations of transformers with masked hard attention can be captured in symbolic B-RASP programs (Yang, Chiang, and Angluin 2024).

Stick-breaking attention (Tan et al. 2025) provides a differentiable mechanism that approximates masked hard attention while preserving its recency bias. Given masked scores $S(i, j) \in [0, 1]$, it defines attention weights as

$$S'(i, j) = S(i, j) \prod_{k=j+1}^{i-1} \big(1 - S(i, k)\big), \qquad j < i.$$

The weight assigned to position j is therefore reduced whenever a later position k with $j < k < i$ has a high score. As a result, attention concentrates on the rightmost high-scoring predecessor of i. When the scores are 0 or 1, stick-breaking attention behaves like masked hard attention, selecting the most recent position with score 1 (Tan et al. 2025). In the two architectures considered in this paper, attention is always strictly masked ($j < i$) and implemented using stick-breaking attention. For comparison, we also evaluate standard softmax-based attention mechanisms.

4 Learning Task

The learning task is to uncover the atoms F and the structure of the actions A (preconditions and effects) from a set T of *positive* and *negative* action traces over A drawn from instances $P = \langle F, A, I, G\rangle$ of a hidden STRIPS model $M = \langle F, A\rangle$. A trace over A is an action sequence $\tau = (a_0, \ldots, a_{n-1})$ made up of actions from A. For convenience and without loss of generality, we ignore the initial situation I and the goals G and define traces to be positive or negative according to their internal consistency, defined as follows:

Definition 1 (Positive and negative traces)**.** Given a propositional STRIPS model $M = \langle F, A\rangle$, a trace $\tau = (a_0, \ldots, a_{n-1})$ is *positive* if for every atom $p \in \mathrm{pre}(a_i)$, $i = 1, \ldots, n$, the following holds:

1. the last action a_j preceding a_i in τ that affects p makes p true; i.e., $p \in \mathrm{add}(a_j)$, or
2. no preceding action in τ affects p, i.e., there is no $j < i$ such that $p \in \mathrm{add}(a_j) \cup \mathrm{del}(a_j)$.

The action trace τ is negative in M if it is not positive.

From the perspective of next token prediction, if we assume tokens to be the actions in M, a trace $\tau = (a_1, \ldots, a_n)$ is positive iff each action a_i is a *possible next action* (applicable) given the previous actions in the trace, and negative otherwise. We also say that a negative trace τ is *inconsistent* with the model M, and capture this in the function $f_M(\tau)$ that is 1 if τ is inconsistent, and else is 0.

Definition 2 (Learning task)**.** Given a set of positive and negative traces T drawn from a hidden domain $M = \langle F, A\rangle$, the task is to learn a Boolean function f over all action sequences τ over A such that $f(\tau) = f_M(\tau)$.

Learning a function equal to $f_M(\tau)$ is equivalent to solving the next token prediction problem in the propositional STRIPS setting where actions are the tokens. Indeed, $b \in A$ is a *possible next action* after a positive trace τ in M iff $f_M(\langle \tau, b\rangle) = 0$, where $\langle \tau, b\rangle$ is the trace τ extended with action b. In the next two sections, we introduce two learning architectures that represent $f(\tau)$ as a function $f_\theta(\tau)$ with parameters θ that will be learned in a supervised manner from positive and negative traces.

5 STRIPS Transformer

Our first proposed architecture builds on the connection between hard-attention transformers, the B-RASP language (Yang, Chiang, and Angluin 2024) and STRIPS (for more information, refer to the Appendix). We call this architecture the STRIPS Transformer. Indeed, checking whether a trace is positive reduces to identifying, for each action and each of its preconditions, the most recent preceding action that affects the corresponding atom, since only that action determines its current truth value. This recency-based dependency aligns naturally with masked hard attention. In the STRIPS Transformer, attention retrieves, for each atom and trace position, the rightmost prior action that modifies that atom and uses its effect to determine applicability.

We proceed constructively. Starting from an arbitrary propositional STRIPS model $M = \langle F, A\rangle$, we describe the set of computations that map a trace τ to the Boolean value $f_M(\tau)$, and show how these computations are realized as a parametrized transformer with parameters $\theta = \theta_M$ (to be defined next). Each step corresponds to a real-valued counterpart of a Boolean operation in B-RASP. The complete B-RASP program from which the architecture is derived is given in Section B of the Appendix.

Let $l = 1, \ldots, |F|$ index the atoms and $m = 1, \ldots, |A|$ index the actions in $M = \langle F, A\rangle$. The STRIPS Transformer is a single-layer, multi-head architecture that applies one attention head l per domain atom p_l. We define a real-valued tensor $\theta_M \in [0, 1]^{|F|\times|A|\times 3}$ to encode M, where each parameter $\theta_M(l, m, j)$ encodes whether atom p_l, associated with head l, is a precondition, a positive effect, or a negative effect ($j = 1, 2, 3$) of action $a_m \in A$. Formally,

$$\theta_M(l,m,1) := \begin{cases} 1 & \text{if } p_l \in \mathrm{pre}(a_m) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$\theta_M(l,m,2) := \begin{cases} 1 & \text{if } p_l \in \mathrm{add}(a_m) \cup \mathrm{del}(a_m) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

$$\theta_M(l,m,3) := \begin{cases} 1 & \text{if } p_l \in \mathrm{del}(a_m) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Let $\tau = (a_0, \ldots, a_{n-1})$ denote the input trace. Each head l operates as follows:

- **QKV projection.** For each action $a_i \in \tau$, let m be its index in θ_M. The vectors $Q_{p_l}, K_{p_l}, V_{p_l} \in [0,1]^n$ are computed as:

$$Q_{p_l}(i) := \theta_M(l,m,1), \quad (4)$$
$$K_{p_l}(i) := \theta_M(l,m,2), \quad (5)$$
$$V_{p_l}(i) := \theta_M(l,m,3), \quad (6)$$

- **Attention Scores.** The attention scores are computed as

$$S_{p_l} := Q_{p_l} \cdot K_{p_l}^\top, \quad (7)$$

  resulting in a matrix $S_{p_l} \in [0,1]^{n \times n}$, where $S_{p_l}(i,j) = 1$ if p_l is a precondition of a_i and a_j affects p_l, and $S_{p_l}(i,j) = 0$ otherwise.

- **Strict future masking.** We apply strict future masking by setting $S_{p_l}(i,j) := 0$ if $j \geq i$, which encodes that actions a_i can only be affected by preceding actions a_j ($j < i$).

- **Stick-breaking attention.** To implement hard-attention with scalar scores, we use stick-breaking attention (Tan et al. 2025). Given S_{p_l}, this computes:

$$S'_{p_l}(i,j) := S_{p_l}(i,j) \cdot \prod_{k=j+1}^{i-1} \big(1 - S_{p_l}(i,k)\big). \quad (8)$$

  This ensures that attention focuses on the rightmost high-scoring position preceding a_i. Note that no explicit $\max$ operation is required.

- **Value aggregation.** The output vector for head p_l, denoted $(y_0, \ldots, y_{n-1})_{p_l} \in [0,1]^n$, is computed as the matrix-vector product

$$y_{p_l} := S'_{p_l} \cdot V_{p_l}. \quad (9)$$

  Each entry $y_{p_l}(i)$ indicates the degree to which action a_i in the input sequence is *inapplicable* relative to the atom p_l. This will be the case if p_l is a precondition of a_i and the rightmost action preceding a_i deleted p_l.

Head outputs are combined via a real-valued OR, indicating for each position i the degree to which action a_i is *inapplicable* given its predecessors. We make explicit the dependence on parameters θ:

$$y_\theta(i) := 1 - \prod_{l=1}^{|F|} \big(1 - y_{p_l}(i)\big). \quad (10)$$

The final step aggregates the inapplicability degrees $y(i)$ via a real-valued OR. The result is 0 if and only if every action is applicable:

$$f_\theta(\tau) := 1 - \prod_{i=0}^{n-1} \left(1 - y_\theta(i)\right). \quad (11)$$

While $y_\theta(i)$ denotes the inapplicability of the action at position i, $f_\theta(\tau)$ aggregates these to represent the probability that τ is a negative trace. For $\theta = \theta_M$, $f_\theta(\tau) = f_M(\tau)$, meaning the model f_θ classifies traces in perfect agreement with M:

Theorem 3. *Let $M = \langle F, A \rangle$ be a propositional* STRIPS *model. Then for $\theta = \theta_M$, as in Eqs. 1–3, $f_\theta(\tau) = f_M(\tau)$ for any trace τ drawn from M.*

Since the operations in Eqs. 4–11 are end-to-end differentiable, the same architecture will allow us to learn M (encoded as θ_M) from positive and negative traces drawn from M. These traces contain actions from A but no information about the states. This learning process is described in Section 7.

6 Stick-Breaking Transformer

Our second proposed architecture is a standard decoder-style transformer (Vaswani et al. 2017) with one difference: we replace the standard softmax attention and positional encodings with stick-breaking attention (Tan et al. 2025). We refer to this model as the Stick-Breaking (SB) Transformer.

In contrast to the STRIPS Transformer, the SB Transformer does not embed a STRIPS model directly in its architecture, it does not align attention heads with domain atoms, and it does not represent action preconditions or effects explicitly in the parameters. As we show later, however, a STRIPS model can be extracted from a trained SB Transformer, provided that that the set of actions is extended with auxiliary actions that encode initial and final states, in a suitable way. The SB Transformer consists of:

(i) a learned token (action) embedding matrix of dimension d_{model}. This replaces the structured query, key, and value tensors used in the STRIPS Transformer to encode preconditions and effects (Eqs. 4–6);

(ii) L stacked transformer blocks with H attention heads per block. Each head implements stick-breaking self-attention with strict future masking, followed by additional transformations. In contrast, the STRIPS Transformer uses a single attention layer ($L = 1$) with one head per atom in F; and

(iii) a final linear layer with a sigmoid activation function that replaces the real-valued OR aggregation used in the STRIPS Transformer (10).

More precisely, given a trace $\tau = (a_0, \ldots, a_{n-1})$, the SB Transformer produces "contextual representations" (embeddings) $h_i^{(\ell)} \in \mathbb{R}^{d_{\text{model}}}$ for each position $i = 0, \ldots, n-1$ and layer $\ell = 0, \ldots, L$.

At layer $\ell = 0$, each token a_i is mapped to its learned embedding, yielding representations $h_i^{(0)}$. For $\ell = 1, \ldots, L$, the representations are updated by

$$h_0^{(\ell)}, \ldots, h_{n-1}^{(\ell)} = \text{Block}^{(\ell)}\big(h_0^{(\ell-1)}, \ldots, h_{n-1}^{(\ell-1)}\big),$$

where each block consists of multi-head self-attention using stick-breaking normalization and strict future masking, followed by a position-wise feed-forward network (MLP) with GELU activation (Hendrycks and Gimpel 2016). Layer normalization (pre-norm) and residual connections are applied in the standard way.

After the final layer, each position i has an embedding $h_i^{(L)}$ that depends only on the prefix $\tau_{\leq i} = (a_0, \ldots, a_i)$. Then, a linear layer with learnable parameters w and b yields

$$y_\theta(i) = \sigma\big(w^\top h_i^{(L)} + b\big) \in [0, 1],$$

where σ is the sigmoid function. The value $y_\theta(i)$ represents the probability of action a_i being inapplicable given the previous actions a_j ($j < i$) in the trace τ.

The trace-level prediction is defined exactly as in Eq. 11:

$$f_\theta(\tau) = 1 - \prod_{i=0}^{n-1} \big(1 - y_\theta(i)\big),$$

where $f_\theta(\tau)$ aggregates the inapplicability degree $y_\theta(i)$ of each action $a_i \in \tau$ via a real-valued OR to obtain the probability that τ is a negative trace.

7 Learning

We train the STRIPS and SB transformers in a supervised manner by comparing the transformer predictions $y_\theta(i)$ with the given labels $z_i \in \{0, 1\}$, where $z_i = 0$ if action a_i is applicable according to the hidden, ground-truth model and $z_i = 1$ otherwise. We use the focal loss (Lin et al. 2017), a variant of the well-known binary cross-entropy loss:

$$\mathcal{L}(\theta) = \frac{1}{|T|} \sum_{\tau \in T} \frac{1}{|\tau|} \sum_{i=0}^{|\tau|-1} \ell(y_\theta(i), z_i)\,, \qquad (12)$$

where the normalization by $|\tau|$ ensures that all traces contribute equally regardless of length. The focal loss term $\ell(y, z)$ is:

$$-\alpha\, z(1-y)^\gamma \log y - (1-\alpha)(1-z) y^\gamma \log(1-y), \quad (13)$$

where parameter $\alpha \in [0, 1]$ controls the relative importance assigned to inapplicable versus applicable actions, while $\gamma \geq 0$ reduces the contribution of well-classified examples, focusing learning on harder cases. In our experiments, we use $\gamma = 1$ and $\alpha = 0.999$.

During training, we apply an additional masking rule: if $z_j = 1$, then actions a_i at later positions $i > j$ cannot attend to a_j when performing self-attention, reflecting the fact that inapplicable actions do not change the state. As this requires ground-truth labels, it is done only during training.

8 Planning

We now describe the extraction of a STRIPS model $M' = \langle F', A' \rangle$ from the trained transformers for its use in planning. A key challenge is that planning problems $P = \langle F, A, I, G \rangle$ are defined in terms of the set of atoms in F in the hidden ground-truth model $M = \langle F, A \rangle$, whereas the traces used in training convey information about A but not F. For solving planning problems P over the hidden model M it is thus necessary to align the two models M and M', so that problems P over M can be mapped into equivalent problems P' over the learned model M'. For this, we augment the set of actions A used during training with auxiliary *setup actions* that convey information about the initial and final states of the traces. No information is conveyed about intermediate states.

8.1 Initial and Final States in the Traces

There are three types of auxiliary (setup) actions: *init-false*, *init-p*, and *test-p* for $p \in F$. The first action sets all atoms p in F to false, the second has p as the single effect, and the third has no effects. The first two actions, in turn, have no preconditions, while *test-p* has precondition p. The action traces used in training, which are drawn from an initial state s_0 and end in a final state s_n, are extended with a *prefix* that conveys information about s_0, and a *suffix* that conveys information about s_n. More precisely, the prefix contains the action *init-false* followed by actions *init-p* for each p true in s_0, while the suffix contains all actions *test-p* for $p \in F$, in the case of the SB transformer, and all actions *test-p* for $p \in G$, where G is the goal of the instance, in the STRIPS transformer. The actions *test-p* have precondition p and hence, they are labeled as applicable in the state s_n and hence in the training trace iff p is true in s_n. The transformers are trained to predict the applicability of both the domain and the setup actions, and thus they must learn to predict the truth values of all or some atoms at the end of each trace, as they determine the applicability of the *test-p* actions.

In the STRIPS transformer, the model of the auxiliary actions, i.e., their preconditions and effects, can be hardwired and does not have to be learned. This is achieved by mapping the atoms $p \in F$ to different transformer heads, and setting the value of their θ parameters accordingly. In the SB transformer, this is not possible and the model of the auxiliary actions has to be learned like the model of the domain actions, as there is no one-to-one correspondence between model parameters and symbolic preconditions or effects.

8.2 Extracting the STRIPS Models

In the STRIPS Transformer, the model $M' = \langle F', A' \rangle$ is extracted from the trained parameters θ as follows. Let $l = 1, \ldots, |F'|$ index the attention heads, and $m = 1, \ldots, |A'|$ index the actions. We binarize the tensor of parameters θ as

$$\bar{\theta}(l, m, k) = [\![\theta(l, m, k) \geq 0.5]\!], \quad \text{for } k = 1, 2, 3, \quad (14)$$

and set the preconditions and effects of actions $a_m \in A'$ as

$$\begin{aligned}\mathrm{pre}(a_m) &= \{p_l \mid \bar{\theta}(l, m, 1)\},\\ \mathrm{add}(a_m) &= \{p_l \mid \bar{\theta}(l, m, 2) \wedge \neg\bar{\theta}(l, m, 3)\},\\ \mathrm{del}(a_m) &= \{p_l \mid \bar{\theta}(l, m, 2) \wedge \bar{\theta}(l, m, 3)\}.\end{aligned}$$

For the SB Transformer, the STRIPS model is extracted in two stages. First, a computationally intensive *state probing* process is performed, wherein the model predicts the applicability of each *test-p* action for every prefix $\tau_{\leq t}$ of the training traces. More precisely, if $\tau = (a_0^I, \ldots, a_r^I, a_0, \ldots, a_{n-1})$ is a training trace containing the initial setup actions $(a_0^I, \ldots, a_r^I)$ but no end setup actions,the reconstructed state s_t^τ at time $t \in \{-1, 0, \ldots, n-1\}$ is defined as:

$$s_t^\tau = \{\, p \in F \mid y_\theta(\langle \tau_{\leq t}, \textit{test-p} \rangle) < 0.5 \,\},$$

where $\tau_{\leq t} = (a_0^I, \ldots, a_r^I, a_0, \ldots, a_t)$. The model $M' = \langle F, A \rangle$ is then derived from these reconstructed states in a standard way, as follows.

Let $\mathrm{eff}(p, s, s')$ denote a function that returns add if $p \notin s \wedge p \in s'$, del if $p \in s \wedge p \notin s'$, and none otherwise. For each atom $p \in F$ and action $a \in A$, the associated preconditions and effects are defined by majority consensus over the training data:

- $p \in \mathrm{pre}(a)$ iff $p \in s_{t-1}^\tau$ for at least 95% of transitions where $a_t = a$;
- $p \in \mathrm{add}(a)$ (resp. $p \in \mathrm{del}(a)$) iff add (resp. del) is the most frequent output of $\mathrm{eff}(p, s_{t-1}^\tau, s_t^\tau)$ for $a_t = a$.

For both the STRIPS and SB transformers, the model extraction process described here results in STRIPS planning domains such as those shown in Listing 1.

8.3 Planning with the Learned Model

Once the STRIPS model $M' = \langle F', A' \rangle$ is extracted, it can be used to solve planning problems $P = \langle F, A, I, G \rangle$ over the hidden domain $M = \langle F, A \rangle$ by defining an equivalent problem $P' = \langle F', A, I', G' \rangle$, which like P does not include the auxiliary actions. For the SB Transformer, F' is F, as the former is obtained from the set of *test-p* actions used in training for all $p \in F$. As a result, I' and G' can be set in P' to I and G, respectively. In the STRIPS Transformer, on the other hand, we include in I' the learned atom $p' \in F'$ corresponding to the single (hardwired) add effect of the action *init-p*, for each $p \in I$, and in G', the learned atom $p' \in F'$ corresponding to the single (hardwired) precondition of the action *test-p*.

An off-the-shelf classical planner can then be used to solve the problem P'. If the learned model M' is correct, the resulting plan will be a plan for the target problem P too. In our experiments, we used the Mimir planner (Ståhlberg 2024) with greedy best-first-search (gbfs) and the FastForward (FF) heuristic (Hoffmann and Nebel 2001).

It is worth noticing that the role of the setup or auxiliary actions is different in the two architectures. In the STRIPS Transformer, setup actions are not required to learn the domain model $M' = \langle F', A' \rangle$, but to align F' with F, and thus to map the target planning problems $P = \langle F, A, I, G \rangle$ over the hidden model $M = \langle F, A \rangle$ into planning problems $P = \langle F', A', I', G' \rangle$ over the learned model. In the SB Transformer, on the other hand, setup actions are essential to learn the STRIPS model $M' = \langle F', A' \rangle$ in the first place. Without such auxiliary actions, the SB Transformer would just give us a Boolean trace classifier for determining whether an action trace is positive (applicable) or negative (not applicable). The gap is bridged in this case by learning to predict the applicability of the *test-p* actions, one for each $p \in F$, and hence, the truth value of p along any of the traces used in training. The full symbolic states along the traces are thus inferred, and the STRIPS model is obtained from the resulting interleaved sequences of actions and states in a standard way, as detailed in Section 8.2.

8.4 Alternative Initial State Encoding

For completeness, we also evaluate the learned STRIPS models using a different encoding of the initial states in the training traces. In the previous scheme, called *init-atoms*, an initial state s_0 was encoded by means of a prefix made up of *init-false* and *init-p* actions, one for each p in the initial situation I. The alternative encoding is simpler and assumes a predefined set of possible initial states S_I which is used in both training and testing. In this scheme, called *init-state*, there is one setup action *init-s* for each $s \in S_I$ with no preconditions, and with effects that make all the atoms in s true, and all other atoms false. The resulting trace prefixes contain in this case just one auxiliary action, an *init-s* action, and both the STRIPS and SB Transformers must learn its model, like the model of all domain actions. In this encoding of the initial states in the traces, and in contrast with the *init-p* scheme, there is no generalization to an exponential number of possible initial states, yet there is generalization to an exponential number of possible goals. In the experiments below, these two encoding schemes are evaluated experimentally for the two architectures.

9 Experiments

We evaluate the STRIPS and SB Transformers on trace classification and planning. This section details our experimental setup to address the following research questions:

- How do the STRIPS and SB Transformers compare against standard transformer baselines?
- Does the additional built-in structure of the STRIPS Transformer lead to better results when compared to the SB Transformer?
- Do the models exhibit compositional reasoning and generalize to an exponential number of unseen states and goals?

To evaluate the performance of our proposed architectures, we compare them against two standard baselines:

- **Sinusoidal:** The original transformer architecture using softmax attention and sinusoidal positional encodings (Vaswani et al. 2017).

	Blocksworld			
Model	Train	Test	Plan (M_S)	Plan (M_L)
SB	**1.0** (**1.0**)	.998 (.999)	**1.0** (**1.0**)	**1.0** (**1.0**)
Sinusoidal	.994 (.998)	.237 (.250)	**1.0** (**1.0**)	.00 (.00)
RoPE	.998 (.999)	.363 (.377)	**1.0** (**1.0**)	.00 (.00)
	Ferry			
Model	Train	Test	Plan (M_S)	Plan (M_L)
SB	**1.0** (**1.0**)	.999 (.998)	**1.0** (**1.0**)	**1.0** (**1.0**)
Sinusoidal	.999 (.999)	.258 (.258)	**1.0** (**1.0**)	.00 (.00)
RoPE	**1.0** (**1.0**)	.530 (.541)	**1.0** (**1.0**)	.25 (.04)
	Npuzzle			
Model	Train	Test	Plan (M_S)	Plan (M_L)
SB	**1.0** (**1.0**)	.997 (.997)	**1.0** (**1.0**)	**1.0** (**1.0**)
Sinusoidal	.999 (.999)	.253 (.253)	**1.0** (**1.0**)	.01 (.01)
RoPE	.998 (.999)	.402 (.445)	**1.0** (**1.0**)	.02 (.03)
	Maze			
Model	Train	Test	Plan (M_S)	Plan (M_L)
SB	**1.0** (**1.0**)	.958 (.967)	.98 (.98)	.93 (.93)
Sinusoidal	**1.0** (**1.0**)	.510 (.506)	.98 (.98)	.00 (.00)
RoPE	**1.0** (**1.0**)	.578 (.562)	.98 (.98)	.00 (.00)
	Logistics			
Model	Train	Test	Plan (M_S)	Plan (M_L)
SB	**1.0** (**1.0**)	.987 (.993)	**1.0** (**1.0**)	**1.0** (**1.0**)
Sinusoidal	.999 (**1.0**)	.281 (.284)	**1.0** (**1.0**)	.00 (.00)
RoPE	.999 (.999)	.412 (.402)	**1.0** (**1.0**)	.32 (.06)

Table 1: Comparison of next-token prediction accuracy and planning accuracy using extracted STRIPS models. We report results for each domain using the large problem size and the *init-atoms* encoding. M_S and M_L denote models extracted from short training traces and longer test traces, respectively. While the SB Transformer achieves near-perfect performance across all metrics, baseline models exhibit poor length generalization in both prediction and symbolic extraction from long traces. Notably, in all three cases, a symbolic model extracted from the trained transformer using short traces (M_S) is able to plan with perfect accuracy.

- **RoPE:** A transformer using Rotary Position Embeddings (Su et al. 2024). This model encodes relative token dependencies via rotation matrices, which has been shown to enhance generalization on long sequences (Spies et al. 2025).

Planning domains. We consider five domains, each with two problem sizes: *small* instances ($|F| < 50$ atoms) and *large* instances ($50 < |F| < 100$). We set the number of attention heads for the STRIPS Transformer to $H = 50$ and $H = 100$ respectively, ensuring $H \geq |F|$ without requiring prior knowledge about the exact number of atoms. Within each size, problems share the same objects and static atoms.

- **`blocksworld`**: Reassembling stackable blocks with a gripper. We use two grounded versions: `blocksworld-5b` and `blocksworld-8b`, containing five and eight blocks, respectively.
- **`ferry`**: Transporting cars between ports with a ferry. `ferry-5c` uses five cars and five ports; `ferry-8c` uses eight cars and seven ports.
- **`npuzzle`**: Sliding numbered tiles to goal positions. `npuzzle-5t` uses five tiles; `npuzzle-8t` uses eight.
- **`maze`**: Grid maze traversal with a $free$ predicate for traversable cells. `maze-5x5` is 5×5; `maze-7x7` is 7×7.
- **`logistics`**: Package delivery via trucks and airplanes. `logistics-3c` contains three cities and two packages; `logistics-5c` contains five cities and four packages. For both sizes, problems contain a single airplane, and each city contains an airport, a location and a truck.

Data generation. For each domain and size, we randomly generate training, test, and planning problems. Under *init-state* encoding, initial states are sampled from a predefined set S_I. Under *init-atoms* encoding, problems do not share initial states. Training and test problems are used to generate corresponding action traces. We generate 10^5 training traces of maximum length $D = 50$, and $2 \cdot 10^5$ test traces with $D = 200$ to evaluate length generalization. For `maze-7x7`, we use $D = 200$ for training to ensure successful learning, and we test using $D = 400$. Each trace contains d domain actions plus setup actions, with d sampled uniformly from $[0, D]$. For training, domain actions are sampled by picking applicable and inapplicable actions with equal probability. In contrast, in the test set, 50% of traces consist only of applicable actions, while the other 50% contain $d - 1$ applicable actions followed by a single inapplicable final action.

Training and evaluation. We train for $5 \cdot 10^5$ steps using the RAdam optimizer (Liu et al. 2019) with batches of 16 and 64 traces for small and large problem sizes, respectively. For the STRIPS Transformer, we use a learning rate of 0.01, except for `ferry-8c` with *init-atoms* where we use $5 \cdot 10^{-3}$. For the SB Transformer, we use a learning rate of $5 \cdot 10^{-5}$. The parameters θ of the STRIPS Transformer are initialized as follows: $\theta(:, :, 3) \sim U[0, 1)$; queries $\theta(:, :, 1)$ and keys $\theta(:, :, 2)$ are sampled from $U[0, 1)$ for *init-state* and $U[0, 0.1)$ for *init-atoms*. The latter effectively initializes actions with a minimal number of preconditions and effects. We also apply L1 regularization (10^{-4}) to queries and keys.

Every $2 \cdot 10^4$ training steps, we perform a validation epoch by computing model accuracy on the entire training set. After training, we load the checkpoint with the highest training accuracy to evaluate on the test dataset. A trace $\tau = (a_0, \ldots, a_{n-1})$ is considered correctly classified only if the model predicts the applicability of *every action* in the trace $a_{0..n-1}$, including domain and setup actions. Accuracy is defined as the percentage of correctly classified traces.

For the STRIPS Transformer, predictions are computed using the binarized parameters $\bar{\theta}$, whereas for the SB Transformer predictions are rounded to $\{0, 1\}$. For both architectures, we extract the STRIPS models from the learned parameters to evaluate their performance on the planning problems as described in Section 8.2. All experiments were conducted

Listing 1: STRIPS domains extracted from the STRIPS and SB transformers. Listings show the schema for action `stack_A_B` learned by (a) the STRIPS Transformer and (b) the SB Transformer in `blocksworld-8b`, for the *init-atoms* encoding and seed = 3. Constants `A` and `B` represent particular objects of type `block`. The domain of the STRIPS Transformer is encoded in terms of atoms `pX`, different to those of the ground-truth domain. For those `pX` appearing in initial states, we can obtain their associated ground-truth atom using the *init-p* actions. For those appearing in goals, we can use the *test-p* actions. For instance, in (a), the atom `p2` is translated to `clear_B`, whereas `p80` and `p99` cannot be translated as they encode learned atoms not present in initial states or goals. In contrast, since the SB Transformer requires one *test-p* action for each ground-truth atom p, the extracted domain uses the same atoms as the ground-truth domain (see (b)). For the SB Transformer, we can also substitute the objects (`A`, `B` in the example) in the learned action schemas by variables (`?x`, `?y` in the example) in order to obtain an equivalent set of lifted action schemas (see (c)).

(a) STRIPS Transformer

```
(:action stack_A_B
 :parameters ()
 :precondition (and
     (clear_B)
     (p80)
     (p99))
 :effect (and
     (handempty)
     (on_A_B)
     (not (clear_B))
     (not (p99))))
```

(b) SB Transformer

```
(:action stack_A_B
 :parameters ()
 :precondition (and
  (clear_B)
  (holding_A))
 :effect (and
  (clear_A)
  (handempty)
  (on_A_B)
  (not (clear_B))
  (not (holding_A))))
```

(c) SB Transformer (after lifting)

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

on a machine equipped with an Nvidia A10 GPU and an Intel Xeon Platinum 8352M CPU.

10 Results

Comparison with standard transformers. Table 1 compares the SB Transformer against the two baseline models, the standard transformer using softmax attention with sinusoidal positional encodings (Sinusoidal), and one using Rotary Position Embeddings (RoPe). While all models achieve high accuracy in training (1st column) for classifying traces, only the SB Transformer maintains high accuracy on the longer test traces (2nd column). The last two columns focus on the accuracy of plans obtained from the learned STRIPS models.

The results of the first two columns indicate that while *standard* transformers learn to classify the traces used in training, they fail to generalize to longer traces not seen during training. The stick-breaking attention transformer, on the other hand, bridges this gap, and yields robust generalization to new, long traces. Interestingly, the last two columns of the table show that while standard transformers show poor generalization over long test traces, the symbolic STRIPS models that can be extracted from them, following the same method as the ones used by the SB Transformer, are accurate and achieve nearly perfect generalization. More precisely, the STRIPS models obtained from standard transformers are very accurate when the training traces are short (M_s), but not when the training traces are long (M_L). Once again, the SB Transformer yields perfectly accurate STRIPS models in both cases.

Comparison between the STRIPS Transformer and the SB Transformer. Table 2 presents comprehensive results across all five planning domains (extended planning results are provided in Section G of the Appendix). We observe that while both models achieve strong overall performance, the SB Transformer consistently outperforms the STRIPS Transformer. This occurs despite the latter incorporating additional symbolic built-in structure, such as a one-to-one mapping between parameters θ and action preconditions and effects. Furthermore, the STRIPS Transformer exhibits greater variability across runs, and it occasionally fails to achieve high training accuracy despite possessing sufficient expressivity to represent the domains. We hypothesize that this performance gap is primarily due to the more complex optimization process that has to be solved via gradient descent. Nevertheless, Table 2 indicates that when the model does attain high training accuracy, the corresponding test and planning accuracies tend to be high as well. For further results regarding the scaling behavior of the two architectures with the amount of data available, see Section E of the Appendix.

Compositional Reasoning and Combinatorial Generalization. Listing 1 shows examples of STRIPS models extracted from our two proposed architectures after learning in `blocksworld-8b` under the *init-atoms* configuration. The model of the STRIPS Transformer is encoded in terms of atoms `pX` different to those of the ground-truth domain. To translate between learned atoms `pX` and ground-truth atoms $p \in F$, the *init-p* and *test-p* actions must be used. This means that only those `pX` appearing in initial states or goals can be aligned (i.e., translated) with the ground-truth domain. The SB Transformer assumes a one-to-one correspondence between ground-truth atoms $p \in F$ and *test-p* actions. Consequently, the learned model uses the same atoms as the ground-truth model. This comes at the cost of a larger set of *test-p* actions and the expensive state probing process used to extract the model.

Despite these architectural differences, Table 2 demon-

	Blocksworld					
	5 blocks			8 blocks		
Model	Train	Test	Plan	Train	Test	Plan
STRIPS 50 states	.975 (**1.0**)	.964 (**1.0**)	.93 (**1.0**)	.982 (**1.0**)	.904 (**1.0**)	.94 (.98)
STRIPS 100 states	.967 (**1.0**)	.918 (**1.0**)	.98 (**1.0**)	**1.0** (**1.0**)	.996 (.996)	.91 (.96)
STRIPS atoms	.615 (.906)	.362 (.864)	.65 (**1.0**)	.544 (.979)	.286 (.648)	.53 (.94)
SB 50 states	.999 (**1.0**)	.984 (.991)	**1.0** (**1.0**)	**1.0** (**1.0**)	.982 (.997)	**1.0** (**1.0**)
SB 100 states	**1.0** (**1.0**)	.985 (.986)	**1.0** (**1.0**)	**1.0** (**1.0**)	.985 (.994)	**1.0** (**1.0**)
SB atoms	**1.0** (**1.0**)	.995 (.999)	**1.0** (**1.0**)	**1.0** (**1.0**)	.998 (.999)	**1.0** (**1.0**)

	Ferry					
	5 cars			8 cars		
Model	Train	Test	Plan	Train	Test	Plan
STRIPS 50 states	.985 (**1.0**)	.901 (**1.0**)	.95 (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.98 (.93)
STRIPS 100 states	**1.0** (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.992 (**1.0**)	.930 (**1.0**)	.88 (.97)
STRIPS atoms	.916 (**1.0**)	.894 (**1.0**)	.92 (**1.0**)	.742 (.821)	.538 (.742)	.28 (.70)
SB 50 states	.999 (**1.0**)	.911 (.922)	**1.0** (**1.0**)	**1.0** (**1.0**)	.950 (.971)	**1.0** (**1.0**)
SB 100 states	.999 (.999)	.898 (.921)	**1.0** (**1.0**)	**1.0** (**1.0**)	.968 (.977)	**1.0** (**1.0**)
SB atoms	**1.0** (**1.0**)	.988 (.992)	**1.0** (**1.0**)	**1.0** (**1.0**)	.999 (.998)	**1.0** (**1.0**)

	Npuzzle					
	5 tiles			8 tiles		
Model	Train	Test	Plan	Train	Test	Plan
STRIPS 50 states	.942 (**1.0**)	.810 (**1.0**)	.98 (**1.0**)	.953 (.969)	.810 (.907)	.94 (.98)
STRIPS 100 states	.943 (.954)	.820 (.801)	.98 (.98)	.964 (**1.0**)	.855 (**1.0**)	.69 (**1.0**)
STRIPS atoms	**1.0** (**1.0**)	.968 (.960)	.99 (.99)	.986 (.998)	.650 (.682)	.63 (.92)
SB 50 states	**1.0** (**1.0**)	.999 (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.998 (.999)	**1.0** (**1.0**)
SB 100 states	**1.0** (**1.0**)	.999 (.999)	**1.0** (**1.0**)	**1.0** (**1.0**)	.998 (.999)	**1.0** (**1.0**)
SB atoms	**1.0** (**1.0**)	.994 (.996)	**1.0** (**1.0**)	**1.0** (**1.0**)	.997 (.997)	**1.0** (**1.0**)

	Maze					
	5x5			7x7		
Model	Train	Test	Plan	Train	Test	Plan
STRIPS 50 states	.906 (.944)	.514 (.580)	.63 (.70)	.897 (.932)	.659 (.721)	.44 (.50)
STRIPS 100 states	.862 (.868)	.361 (.375)	.34 (.37)	.784 (.798)	.428 (.442)	.15 (.19)
STRIPS atoms	.889 (.902)	.431 (.412)	.11 (.16)	.158 (.166)	.021 (.011)	.00 (.00)
SB 50 states	**1.0** (**1.0**)	.997 (**1.0**)	.90 (.90)	**1.0** (**1.0**)	**1.0** (**1.0**)	.64 (.64)
SB 100 states	**1.0** (**1.0**)	.997 (.998)	**1.0** (**1.0**)	**1.0** (**1.0**)	.999 (**1.0**)	.99 (.99)
SB atoms	**1.0** (**1.0**)	**1.0** (.999)	.98 (.98)	**1.0** (**1.0**)	.958 (.967)	.98 (.98)

	Logistics					
	3 cities			5 cities		
Model	Train	Test	Plan	Train	Test	Plan
STRIPS 50 states	.985 (**1.0**)	.905 (**1.0**)	.95 (.97)	.988 (.999)	.821 (.891)	.54 (.73)
STRIPS 100 states	.988 (**1.0**)	.959 (**1.0**)	.91 (.97)	.989 (.999)	.851 (.895)	.38 (.37)
STRIPS atoms	.788 (.944)	.558 (.763)	.71 (.90)	.959 (**1.0**)	.902 (**1.0**)	.88 (**1.0**)
SB 50 states	**1.0** (**1.0**)	.997 (.999)	**1.0** (**1.0**)	**1.0** (**1.0**)	.978 (.984)	**1.0** (**1.0**)
SB 100 states	**1.0** (**1.0**)	.997 (**1.0**)	**1.0** (**1.0**)	**1.0** (**1.0**)	.978 (.980)	**1.0** (**1.0**)
SB atoms	**1.0** (**1.0**)	.990 (.999)	**1.0** (**1.0**)	**1.0** (**1.0**)	.987 (.993)	**1.0** (**1.0**)

Table 2: **Main Results.** Comparison of STRIPS and SB transformers across five planning domains. We evaluate three configurations: *init-state* encoding with 50 and 100 distinct initial states, and the *init-atoms* encoding (*atoms*). All configurations incorporate *test-p* end setup actions. Metrics represent average training, test, and planning accuracy across three random seeds. Values in parentheses indicate results for the seed achieving the highest training accuracy.

strates that the STRIPS models extracted from both transformers successfully interface with off-the-shelf planners and symbolic heuristics (e.g., FF), yielding high overall planning accuracy. This robustness persists even for complex problems where solution plans require a high number of actions (see Section G in the Appendix). Furthermore, the use of *test-p* actions allows these models to generalize to an exponential number of potential goals, distinct to those seen in training. Similarly, for the *init-atoms* encoding, the models generalize to an exponential number of previously unseen initial states. We provide additional comparative results with different setup configurations for the initial and final states in Section F of the Appendix.

Overall, the experiments show that both the STRIPS Transformer and SB Transformer produce STRIPS models that support effective planning over the hidden STRIPS domain and an exponential number of new initial states and goals not seen during training.

11 Summary

We have shown that the STRIPS and SB transformers learn to classify positive and negative action traces in STRIPS, generalizing to longer traces. Moreover, propositional STRIPS models can be extracted from them in order to support planning via off-the-shelf planners. While the STRIPS Transformer follows an encoding of the STRIPS classification task in the B-RASP language (see Appendix), and thus has more built-in structure, the SB Transformer is easier to train and generalizes more robustly. Interestingly, standard transformers that do not use stick-breaking attention do not generalize to long traces, although the symbolic STRIPS models that can be obtained from them do when extracted on short traces. In the future, we plan to extend our approach to learn lifted STRIPS models from action traces, as done by the symbolic SIFT algorithm (Gösgens, Jansen, and Geffner 2024). We note, however, that SIFT is not suitable for learning propositional STRIPS models such as the ones considered in this work, as it runs in time that is exponential in the number of action schemas, which in the propositional case is equal to the number of actions (128 in `blocksworld-8b`, for example).

Acknowledgements

This research has been partially supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research, by the European Research Council (ERC), Grant agreement No. 885107, and by the Excellence Strategy of the Federal Government and the NRW state, Germany. This publication is also part of the action CNS2022-136178 financed by MCIN/AEI/10.13039/501100011033 and by the European Union Next Generation EU/PRTR.

AI Declaration

During the preparation of this work the authors used OpenAI's ChatGPT in order to enhance readability and perform grammar checking. In addition, ChatGPT Codex was used to support code development. All outputs generated by these tools were carefully reviewed by the authors, and modified where necessary. The authors take full responsibility for the content of the publication.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence* 275:104–137.

Asai, M.; Kajino, H.; Fukunaga, A.; and Muise, C. 2022. Classical planning in deep latent space. *Journal of Artificial Intelligence Research* 74:1599–1686.

Besta, M.; Blach, N.; Kubicek, A.; Gerstenberger, R.; Podstawski, M.; Gianinazzi, L.; Gajda, J.; Lehmann, T.; Niewiadomski, H.; Nyczyk, P.; et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, 17682–17690.

Bonet, B., and Geffner, H. 2020. Learning first-order symbolic representations for planning from the structure of the state space. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325, 2322–2329. IOS Press.

Cano, L. H.; Perroni-Scharf, M.; Dhir, N.; Ramamurthy, A.; and Solar-Lezama, A. 2025. Neurosymbolic world models for sequential decision making. In *Forty-second International Conference on Machine Learning*.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using locm. *The Knowledge Engineering Review* 28(2):195–213.

Feng, T.; Wu, Y.; Lin, G.; and You, J. 2025. Graph world model. In *Forty-second International Conference on Machine Learning*.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gestrin, E.; Kuhlmann, M.; and Seipp, J. 2024. Towards robust LLM-driven planning from minimal text descriptions. In *ICAPS 2024 Workshop on Human-Aware Explainable Planning*.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Elsevier.

Gösgens, J.; Jansen, N.; and Geffner, H. 2024. Learning lifted strips models from action traces alone: A simple, general, and scalable solution. *arXiv preprint arXiv:2411.14995*.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 79081–79094. Curran Associates, Inc.

Hao, S.; Gu, Y.; Ma, H.; Hong, J. J.; Wang, Z.; Wang, D. Z.; and Hu, Z. 2023. Reasoning with language model is planning with world model. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Hendrycks, D., and Gimpel, K. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Hoffmann, J., and Nebel, B. 2001. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

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 *Forty-first International Conference on Machine Learning*.

Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi, S. 2024. Thought of search: Planning with language models through the lens of efficiency. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Lamanna, L.; Serafini, L.; Saetti, A.; Gerevini, A. E.; and Traverso, P. 2025. Lifted action models learning from partial traces. *Artificial Intelligence* 339:104256.

Li, K.; Hopkins, A. K.; Bau, D.; Viégas, F.; Pfister, H.; and Wattenberg, M. 2023. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*.

Lin, S., and Bercher, P. 2022. On the expressive power of planning formalisms in conjunction with ltl. *Proceedings of the International Conference on Automated Planning and Scheduling* 32(1):231–240.

Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; and Dollár, P. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, 2980–2988.

Liu, L.; Jiang, H.; He, P.; Chen, W.; Liu, X.; Gao, J.; and Han, J. 2019. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.

Mitchell, M. 2023. LLMs and World Models. https://aiguide.substack.com/p/llms-and-world-models-part-1. Blog post.

Nanda, N. 2023. Actually, othello-gpt has a linear emergent world model.

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

Pallagani, V.; Muppasani, B.; Murugesan, K.; Rossi, F.; Horesh, L.; Srivastava, B.; Fabiano, F.; and Loreggia, A. 2023. Plansformer: Generating symbolic plans using transformers. In *NeurIPS 2023 Workshop on Generalization in Planning*.

Rodriguez, I. D.; Bonet, B.; Romero, J.; and Geffner, H. 2021. Learning First-Order Representations for Planning from Black Box States: New Results. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, 539–548.

Rohekar, R. Y.; Gurwicz, Y.; Yu, S.; Aflalo, E.; and Lal, V. 2025. A causal world model underlying next token prediction: Exploring GPT in a controlled environment. In *Forty-second International Conference on Machine Learning*.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.

Silver, T.; Hariprasad, V.; Shuttleworth, R. S.; Kumar, N.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. PDDL planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

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 and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press.

Spies, A. F.; Edwards, W.; Ivanitskiy, M.; Skapars, A.; Räuker, T.; Inoue, K.; Russo, A.; and Shanahan, M. 2025. Transformers use causal world models in maze-solving tasks. In *ICLR 2025 Workshop on World Models: Understanding, Modelling and Scaling*.

Ståhlberg, S. 2024. Mimir: A generalized planning library. https://github.com/simon-stahlberg/mimir.

Su, J.; Ahmed, M.; Lu, Y.; Pan, S.; Bo, W.; and Liu, Y. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing* 568:127063.

Tan, S.; Yang, S.; Courville, A.; Panda, R.; and Shen, Y. 2025. Scaling stick-breaking attention: An efficient implementation and in-depth study. In *The Thirteenth International Conference on Learning Representations*.

Toshniwal, S.; Wiseman, S.; Livescu, K.; and Gimpel, K. 2022. Chess as a testbed for language model state tracking. *Proceedings of the AAAI Conference on Artificial Intelligence* 36(10).

Vafa, K.; Chen, J. Y.; Rambachan, A.; Kleinberg, J.; and Mullainathan, S. 2024. Evaluating the world model implicit in a generative model. *Advances in Neural Information Processing Systems* 37:26941–26975.

Vafa, K.; Chang, P. G.; Rambachan, A.; and Mullainathan, S. 2025. What has a foundation model found? inductive bias reveals world models. In *Forty-second International Conference on Machine Learning*.

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.

Weiss, G.; Goldberg, Y.; and Yahav, E. 2021. Thinking like transformers. In *International Conference on Machine Learning*, 11080–11090. PMLR.

Xi, K.; Gould, S.; and Thiébaux, S. 2024. Neuro-symbolic learning of lifted action models from visual traces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 653–662.

Yang, A.; Chiang, D.; and Angluin, D. 2024. Masked hard-attention transformers recognize exactly the star-free languages. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. R. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yildirim, I., and Paul, L. 2024. From task structures to world models: what do LLMs know? *Trends in Cognitive Sciences* 28(5):404–415.

Zhang, L.; Yang, G.; and Stadie, B. C. 2021. World model as a graph: Learning latent landmarks for planning. In Meila, M., and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 12611–12620. PMLR.

Zhuo, H. H., and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 2444–2450. AAAI Press.