Neural Production Systems

Aniket Didolkar [1 *] **Anirudh Goyal** [1 *] **Nan Rosemary Ke** [1 2] **Charles Blundell** [2] **Philippe Beaudoin** [3]
Nicolas Heess [2] **Michael Mozer** [4] **Yoshua Bengio** [1]

Abstract

Visual environments are structured, consisting of distinct objects or *entities*. These entities have properties—both visible and latent—that determine the manner in which they interact with one another. To partition images into entities, deep-learning researchers have proposed structural inductive biases such as slot-based architectures. To model interactions among entities, equivariant graph neural nets (GNNs) are used, but these are not particularly well suited to the task for two reasons. First, GNNs do not predispose interactions to be sparse, as relationships among independent entities are likely to be. Second, GNNs do not factorize knowledge about interactions in an entity-conditional manner. As an alternative, we take inspiration from cognitive science and resurrect a classic approach, *production systems*, which consist of a set of rule templates that are applied by binding placeholder variables in the rules to specific entities. Rules are scored on their match to entities, and the best fitting rules are applied to update entity properties. In a series of experiments, we demonstrate that this architecture achieves a flexible, dynamic flow of control and serves to factorize entity-specific and rule-based information. This disentangling of knowledge achieves robust future-state prediction in rich visual environments, outperforming state-of-the-art methods using GNNs, and allows for the extrapolation from simple (few object) environments to more complex environments.

1. Introduction

Despite never having taken a physics course, every child appreciates that pushing a plate off the dining table will cause the plate to break. The laws of physics accurately characterize the dynamics of our natural world, but explicit knowledge of these laws is not necessary to reason. Humans can verbalize knowledge in propositional expressions such as "If a plate drops from table height, it will break," and "If a video-game opponent approaches from behind and they are carrying a weapon, they are likely to attack you." Expressing propositional knowledge is not a strength of current deep learning methods for several reasons. First, propositions are discrete and independent from one another. Second, propositions must be quantified in the manner of first-order logic; for example, the video-game proposition applies to any X for which X is an opponent and has a weapon. Incorporating the ability to express and reason about propositions should improve generalization in deep learning methods because this knowledge is modular— propositions can be formulated independently of each other— and can therefore be acquired incrementally. The proposition can also be applied consistently to all entities that match, yielding a powerful form of systematic generalization.

The classical AI literature from the 1980s can offer deep learning researchers a valuable perspective. In this era, reasoning, planning, and prediction were handled by architectures that performed propositional inference on symbolic knowledge representations. A simple example of such an architecture is the *production system* (Laird et al., 1986; Anderson, 1987; 1996; Anderson et al., 2004; Anderson, 2014), which expresses knowledge by *condition-action rules*. The rules operate on a *working memory*: rule conditions are matched to entities in working memory and such a match can trigger computational actions that update working memory or external actions that operate on the outside world.

Production systems were typically used to model high-level cognition, e.g., mathematical problem solving or procedure following; perception was not the focus of these models. It was assumed that the results of perception were placed into working memory in a symbolic form that could be operated on with the rules. In this article, we revisit production systems but from a deep learning perspective which naturally integrates perceptual processing and subsequent inference for visual reasoning problems. We describe an end-to-end deep learning model that constructs object-centric representations of entities in videos, and then operates on these

[*]Equal contribution [1]Mila, University of Montreal [2]Deepmind [3]Waverly [4]Google Brain. Correspondence to: Anirudh Goyal <anirudhgoyal9119@gmail.com>, Aniket Didolkar <adidolkar123@gmail.com>.

entities with differentiable—and thus learnable—production rules. The essence of these rules, carried over from traditional symbolic system, is that they operate on variables that are *bound*, or linked, to the entities in the world. In the deep learning implementation, each production rule is represented by a distinct MLP with query-key attention mechanisms to specify the rule-entity binding and to determine when the rule should be triggered.

We are not the first to propose a neural instantiation of a production system architecture. Touretzky & Hinton (1988) gave a proof of principle that neural net hardware could be hardwired to implement a production system for symbolic reasoning; our work fundamentally differs from theirs in that (1) we focus on perceptual inference problems and (2) we use the architecture as an inductive bias for learning.

1.1. Variables and entities

What makes a rule general-purpose is that it incorporates placeholder *variables* that can be bound to arbitrary *values* or—the term we prefer in this article—*entities*. This notion of binding is familiar in functional programming languages, where these variables are called arguments. Consider a simple function in C like `int add(int a, int b)`. This function binds its two integer operands to variables a and b. The function does not apply if the operands are, say, character strings. The use of variables enables a programmer to reuse the same function to add any two integer values. Analogously, the use of variables in the production rules we describe enable a model to reason about any set of entities that satisfy the selection criteria of the rule.

In order for rules to operate on entities, these entities must be represented explicitly. That is, the visual world needs to be parsed in a task-relevant manner, e.g., distinguishing the sprites in a video game or the vehicles and pedestrians approaching an autonomous vehicle. Only in the past few years have deep learning vision researchers developed methods for object-centric representation (Le Roux et al., 2011; Eslami et al., 2016; Greff et al., 2016; Raposo et al., 2017; Van Steenkiste et al., 2018; Kosiorek et al., 2018; Engelcke et al., 2019; Burgess et al., 2019; Greff et al., 2019; Locatello et al., 2020a; Ahmed et al., 2020; Goyal et al., 2019b; Zablotskaia et al., 2020; Rahaman et al., 2020; Du et al., 2020; Ding et al., 2020; Goyal et al., 2020; Ke et al., 2021). These methods differ in details but share the notion of a fixed number of *slots*, also known as *object files*, each encapsulating information about a single object. Importantly, the slots are interchangeable, meaning that it doesn't matter if a scene with an apple and an orange encodes the apple in slot 1 and orange in slot 2 or vice-versa.

A model of visual reasoning must not only be able to represent entities but must also express knowledge about entity dynamics and interactions. To ensure *systematic* predictions, a model must be capable of applying knowledge to an entity regardless of the slot it is in and must be capable of applying the same knowledge to multiple instances of an entity. Several distinct approaches exist in the literature. The predominant approach uses graph neural networks to model slot-to-slot interactions (Scarselli et al., 2008; Bronstein et al., 2017; Watters et al., 2017; Van Steenkiste et al., 2018; Kipf et al., 2018; Battaglia et al., 2018; Tacchetti et al., 2018). To ensure systematicity, the GNN must share parameters among the edges. In a recent article, Goyal et al. (2020) developed a more general framework in which parameters are shared but slots can dynamically select which parameters to use in a state-dependent manner. Each set of parameters is referred to as a *schema*, and slots use a query-key attention mechanism to select which schema to apply at each time step. Multiple slots can select the same schema. In both GNNs and SCOFF, modeling dynamics involves each slot interacting with each other slot. In the work we describe in this article, we replace the direct slot-to-slot interactions with rules, which mediate sparse interactions among slots.

In the next section, we first give an overview of classical production systems and then describe how we use them to structure a model that learns neural representations of entities and rules.

2. Production System

Formally, our notion of a production system consists of a set of entities and a set of rules, along with a mechanism for selecting rules to apply on subsets of the entities. Implicit in a rule is a specification of the properties of relevant entities, e.g., a rule might apply to one type of sprite in a video game but not another. The control flow of a production system dynamically selects rules as well as bindings between rules and entities, allowing different rules to be chosen and different entities to be manipulated at each point in time.

The neural production system we describe shares essential properties with traditional production system, particularly with regard to the compositionality and generality of the knowledge they embody. Lovett & Anderson (2005) describe four desirable properties commonly attributed to symbolic systems that apply to our work as well.

Production rules are modular. Each production rule represents a unit of knowledge and are *atomic* such that any production rule can be intervened (added, modified or deleted) independently of other production rules in the system.

Production rules are abstract. Production rules allow for generalization because their conditions may be represented as high-level abstract knowledge that match to a wide range of patterns. These conditions specify the attributes of relationship(s) between entities without specifying the entities

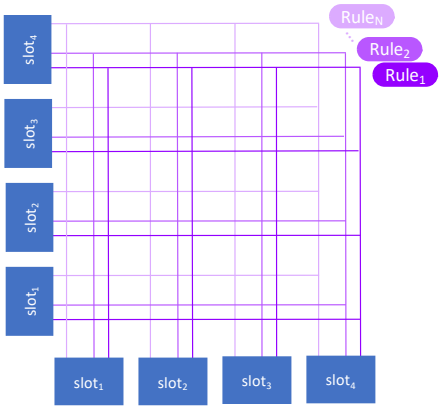


Figure 1. **Rule and slot combinatorics.** Condition-action rules specify how entities interact. Slots maintain the time-varying state of an entity. Every rule is matched to every pair of slots. Through key-value attention, a goodness of match is determined, and a rule is selected along with its binding to slots.

themselves. The ability to represent abstract knowledge allows for the transfer of learning across different environments as long as they fit within the conditions of the given production rule.

Production rules are sparse. In order that production rules have broad applicability, they involve only a subset of entities. This assumption imposes a strong prior that dependencies among entities are sparse. In the context of visual reasoning, we conjecture that this prior is superior to what has often been assumed in the past, particularly in the disentanglement literature—independence among entities (Higgins et al., 2016; Chen et al., 2018).

Production rules represent causal knowledge and are thus asymmetric. Each rule can be decomposed into a {condition, action} pair, where the action reflects a state change that is a causal consequence of the conditions being met.

These four properties are sufficient conditions for knowledge to be expressed in production rule form. These properties specify *how* knowledge is represented, but not *what* knowledge is represented. The latter is inferred by learning mechanisms under the inductive bias provided by the form of production rules.

3. Neural Production System: Slots and Sparse Rules

The Neural Production System (NPS), illustrated in Figure 1, provides an architectural backbone that supports the detection and inference of entity (object) representations in an input sequence, and the underlying rules which govern the interactions between these entities in time and space. The input sequence indexed by time step t, $\{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^t, \ldots, \boldsymbol{x}^T\}$, for instance the frames in a video, are processed by a neural encoder (Burgess et al., 2019; Greff et al., 2019; Goyal et al., 2019b; 2020) applied to each $\boldsymbol{x}^t$, to obtain a set of M entity representations $\{\boldsymbol{V}_1^t, \ldots, \ldots, \boldsymbol{V}_M^t\}$, one for each of the M slots. These representations describe an entity and are updated based on both the previous state, $\boldsymbol{V}^{t-1}$ and the current input, $\boldsymbol{x}^t$.

NPS consists of N separately encoded rules, $\{\boldsymbol{R}_1, \boldsymbol{R}_2, .., \boldsymbol{R}_N\}$. Each rule consists of two components, $\boldsymbol{R}_i = (\vec{\boldsymbol{R}}_i, MLP_i)$, where $\vec{\boldsymbol{R}}_i$ is a learned rule embedding vector, which can be thought of as a template defining the condition for when a rule applies; and MLP_i, which determines the action taken by a rule. Both $\vec{\boldsymbol{R}}_i$ and the parameters of MLP_i are learned along with the other parameters of the model using back-propagation on an end-to-end objective.

As we describe in detail in the next section, rules are selected and applied one at a time, with a sequence of K rule applications (*stages*) per time step. In the general form of the model, each rule has conditions and actions that are specified on a pair of entities, meaning that there are NM^2 possible rule-slot bindings to consider at each stage. To reduce the complexity of the search, we assume that the two slots are asymmetric: one slot is *primary* in that it is used both to match the rule condition and it is acted on by the rule; the other slot is *contextual* in that it determines how the primary slot is acted upon. With this set up, we perform selection in two operations, first considering all combinations of {rule, primary slot}, making a selection, and then conditioned on the selection, choosing a contextual slot. The resulting search is reduced to $NM + M$ combinations.

3.1. Computational Steps in NPS

Algorithm 1 formalizes NPS. The algorithm involves a loop over time steps t. Within a time step, exactly K rule applications occur. (Flexibility in the number of rule applications per step can be easily incorporated into the model.) Because each rule application can affect slot contents, we need to add notation to index the entity representation for slot j, $\boldsymbol{V}_j$, by both the time step t and the number of rules that have been applied during the time step, denoted h, i.e., $\boldsymbol{V}_j^{t,h}$. When we omit h, we imply that we are referencing the state following all rule applications, $h = K$.

Step 1 is external to NPS and involves parsing an input image, $\boldsymbol{x}^t$, into slot-based entities conditioned on the previous state of the slot-based entities. Any of the methods proposed in the literature to obtain a slot-wise representation of entities can be used (Burgess et al., 2019; Greff et al., 2019; Goyal et al., 2019b; 2020). The next three steps constitute the rule application procedure, which is iterated K times.

Step 2 selects the {rule, primary slot} pair, determined on-the-fly by an attention-based soft competition. This competition is analogous to rule matching and prioritization in traditional production systems. The current state $\boldsymbol{V}_j^{t,h}$ of a slot constructs a key that is matched to a query obtained from a rule embedding $\vec{\boldsymbol{R}}_i$; based on the goodness of the match, a rule and primary slot, indexed by r and p, respectively, are selected. Straight-through Gumbel softmax (Jang et al., 2016) is used for achieving a learnable hard decision.

Step 3 selects a contextual slot given the chosen primary slot and rule. The selected contextual slot is indexed by c. Again, straight-through Gumbel softmax is used for this competition among slots.

Step 4 applies the selected rule to the primary slot based on the rule and the current contents of the primary and contextual slots. The rule-specific MLP_r, takes as input the concatenated representation of the state of the primary and contextual slots, $\boldsymbol{V}_p^{t,h}$ and $\boldsymbol{V}_c^{t,h}$, and produces an output, which is then used to change the state of the primary slot $\boldsymbol{V}_p^{t,h+1}$ by residual addition.

Algorithm 1 Neural Production System model

Input: Input sequence $\{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^t, \ldots, \boldsymbol{x}^T\}$, set of embeddings describing the rules $\vec{\boldsymbol{R}}_i$, and set of MLPs (MLP_i) corresponding to each rule $\boldsymbol{R}_{1\ldots N}$. Hyper-parameters specific to NPS are the number of stages K, the number of slots M, and the number of rules N. W^k, $\widetilde{W}^k$, and $\widetilde{W}^q$ are learnable weights.

for *each input element* $\boldsymbol{x}^t$ *with* $t \leftarrow 1$ *to* T **do**

Step 1: *Update or infer the entity state in each slot* j, $\boldsymbol{V}_j^{t,0}$, *from the previous state,* $\boldsymbol{V}_j^{t-1,K}$ *and the current input* $\boldsymbol{x}_t$.

for *each stage* $h \leftarrow 0$ *to* $K-1$ **do**

Step 2: *Select {rule, primary slot} pair*

- $\boldsymbol{q}_i = \vec{\boldsymbol{R}}_i \quad \forall i \in \{1, \ldots, N\}$
- $\boldsymbol{k}_j = \boldsymbol{V}_j^{t,h}\boldsymbol{W}^k \quad \forall j \in \{1, \ldots, M\}$
- $r, p = \text{argmax}_{i,j}\,(\boldsymbol{q}_i \boldsymbol{k}_j + \gamma)$ where $\gamma \sim \text{Gumbel}(0, 1)$

Step 3: *Select contextual slot*

- $\boldsymbol{q}_{r,p} = \text{Concatenate}([\vec{\boldsymbol{R}}_r, V_p^{t,h}])\widetilde{\boldsymbol{W}}^q$
- $\boldsymbol{k}_j = \boldsymbol{V}_j^{t,h}\widetilde{\boldsymbol{W}}^k \quad \forall j \in \{1, \ldots, M\}$
- $c = \text{argmax}_j\,(\boldsymbol{q}_{r,p} \boldsymbol{k}_j + \gamma)$ where $\gamma \sim \text{Gumbel}(0, 1)$

Step 4: *Apply selected rule to primary slot conditioned on contextual slot*

- $\widetilde{\boldsymbol{R}} = \text{MLP}_r(\text{Concatenate}([\boldsymbol{V}_p^{t,h}, \boldsymbol{V}_c^{t,h}]))$
- $\boldsymbol{V}_p^{t,h+1} = \boldsymbol{V}_p^{t,h} + \widetilde{\boldsymbol{R}}$

end

end

4. Related Work

Key Value Attention. Key-value attention defines the backbone of updates to the slots in the proposed model. This form of attention is widely used in Transformer models (Bahdanau et al., 2014; Vaswani et al., 2017). Key-value attention selects an input value based on the match of a query vector to a key vector associated with each value. To allow easier learnability, selection is soft and computes a convex combination of all the values. Rather than only computing the attention once, the multi-head dot product attention mechanism (MHDPA) runs through the scaled dot-product attention multiple times in *parallel*. There is an important difference with NPS: in MHDPA, one can treat different heads as different rule applications. Each head (or rule) considers *all* the other entities as relevant arguments, and all the heads are applied in *parallel* as compared to *sequential* and *selective* application of rules in NPS.

Sparse and Dense Interactions. GNNs model pairwise interactions between all the slots hence they can be seen as capturing *dense* interactions (Scarselli et al., 2008; Bronstein et al., 2017; Watters et al., 2017; Van Steenkiste et al., 2018; Kipf et al., 2018; Battaglia et al., 2018; Tacchetti et al., 2018). Verbalizable interactions in the real world are sparse. The immediate effect of an action is only on a small subset of entities. In NPS, a selected rule only updates the state of one of the slots (per rule-application).

In NPS, one can view the computational graph as a GNN resulting from sequentially applying rules, where the states of the slots are represented on the different nodes of the graph, and different rules dynamically instantiate an edge between a set of slot, again chosen in a dynamic way. It is important to emphasize that the topology of the graph induced in NPS is dynamic, while in most GNNs the topology is fixed. Through a thorough set of experiments, we show that learning sparse and dynamic interactions using NPS indeed works better for the problems we consider than learning dense interactions using GNNs. We show that NPS outperforms state-of-the-art GNN-based architectures such as C-SWM (Kipf et al., 2019) and OP3 (Veerapaneni et al., 2019) while learning action conditioned world models.

Neural Program Induction. Neural networks have been studied as a way to address the problems of learning procedural behavior and program induction (Graves et al., 2014; Reed & De Freitas, 2015; Neelakantan et al., 2015; Cai et al., 2017; Xu et al., 2018; Trask et al., 2018; Bunel et al., 2018; Li et al., 2020). In this line of work, there's a controller which issues an instruction which has some pre-determined semantics over how it transforms state s. Such approaches don't deal with perceptual inference problems, and also only factorize knowledge as a set of experts (i.e., rules which only requires a single argument). Whereas in NPS there's a factorization of knowledge both in terms of slots as well as

rules which act on these slots. (Evans et al., 2019) impose a bias in the form of rules which is used to to define the state transition function, but we believe both the rules and the representations of the entities can be learned from the data with sufficiently strong inductive bias.

5. Experiments

In this section, we demonstrate the effectiveness of NPS on multiple tasks and compare to a comprehensive set of baselines. The experiments seek to address the following questions: (1) Can NPS learn to accurately extract operations that define the data generating distribution and learn to represent them as separate rules? (2) Are sparse interactions as encouraged by NPS a useful inductive bias for modelling dynamics, and how does NPS compare to GNNs that allow for dense interactions? and (3) Can the factorization of knowledge into rules scale up to complicated visually rich environments with abstract physical rules?

We first design simple experiments with a well-defined discrete set of operations and show that NPS learns to accurately discover and separate the underlying operations (into the separate rule MLPs in the NPS parametrization). We then show that using the NPS as a transition model in visually rich settings leads to considerable performance gains over GNNs. We further conduct ablation studies to assess the contribution of different components of NPS. Here we briefly outline the tasks on which we applied NPS and direct the reader to the appendix for full details on each task and details on hyperparameter settings for the model.

5.1. Learning intuitive rules with NPS: Toy Simulations

To demonstrate that NPS can learn intuitive rules, we design a couple of simple tasks, based on the following formulation: We define a dataset which contains entities that are closed under a set of well-defined operations. Each example in this dataset consists of a set of entities and operations to be performed on these entities. The learner needs to output the result of performing the corresponding operations on the given entities. The representation of *entities* and *operations* form the slots that are given as input to NPS. These operations are functions that can act on one or a few entities. For example, the *Bitwise AND* operation acts on multiple entities while the *complement* operation acts on only one entity. We show that NPS can learn a factorized representation of rules, assigning a separate rule to every operation and can apply these operations in novel scenarios.

Arithmetic task. The objective is to learn separate arithmetic operations (addition, subtraction, and multiplication) on numbers. When NPS is presented with three slots as 2 numerical arguments and an operation, it should interpret them in reverse polish notation to produce the numerical answer, but the overall learner can process a sequence of numbers and operations to obtain the final result composing all the operations. Each operation is represented with a 3-hot vector and encoded into a d-dimensional vector via a learned matrix multiplication. The numbers are encoded along with their positions like in transformers, also into d dimensions, and fed to NPS with $K = 1$ stage, i.e., a single rule application. Here, the slots presented to NPS consist of the 2 numerical arguments and the operation. NPS does not have preconceived knowledge of what is a number or an operation, so its main job is to discover that and apply the selected rule MLP, corresponding to the operation, to the appropriate numerical arguments. Step 2 of NPS can match the operation embedding to the appropriate rule. Step 3 can then be applied with two heads to select the two other slots (numerical arguments). Step 4 then applies the selected rule to the two numerical arguments (i.e. slots) and outputs the result of the operation, which can be used as final output or pushed on the stack of unprocessed inputs for the next rule application. We train the proposed model using the MSE loss. For the baseline, we form a concatenated representation of the two numerical representations of the arguments and the 3-hot operation vector and pass it to an unstructured net. See appendix for more details.

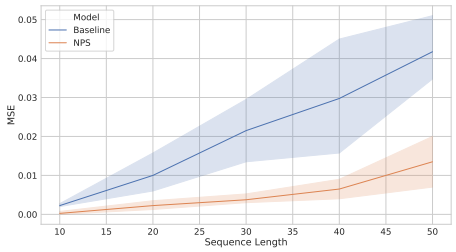


Figure 2. **Arithmetic Task.** Here, we compare NPS to the baseline model. We plot the MSE loss corresponding to various sequence lengths. As shown, NPS has a significantly lower MSE than the baseline.

With our experiments, **we see that NPS automatically learns to use a separate rule for each operation**. The results for this task are shown in figure 2. For testing, we use a sequential setting where the result of one operation becomes an argument for the next operation (according to reverse polish notation). We report MSE for various values of n (sequence length). We can see that NPS achieves a significantly lower MSE as compared to the unstructured baseline, especially for longer sequence lengths.

MNIST Transformation. In this task, we test whether the NPS rule learning abilities can scale to richer visual settings. We use a variation on the MNIST dataset for this task. In this case, the operations comprise performing transformations on MNIST digits. We generate data with four transformations: $\boldsymbol{O}$ = {Translate Up, Translate Down, Rotate Right, Rotate Left}. We feed the input image ($\boldsymbol{X}$) and the transformation ($\boldsymbol{o}$) to be performed as a one-hot

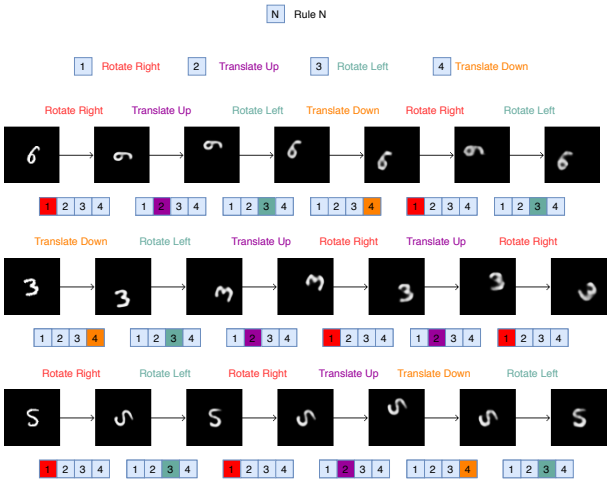


Figure 3. **MNIST Transformation Task**. Demonstration of NPS on the MNIST transformations task. The proposed model can be used to compose any novel combination of transformations on any digit by hand-picking the rule vector that was assigned to each corresponding operation during training.

vector to the model. The two entities are first encoded in two slots $M = 2$: the first entity (the image) is encoded with a convolutional encoder to a d-dimensional vector; the second entity (the one-hot operation vector) is mapped to a d-dimensional vector using a learned weight matrix. We feed these two slots to the NPS module. Step 2 will match the operation embedding to the corresponding rule. Step 3 can be used to select the correct slot for rule application (i.e. the image ($\boldsymbol{X}$) representation). Step 4 can then apply the MLP of the selected rule to the selected slot from step 3 and output the result, which is passed through a common decoder to generate the transformed image. We train the model using reconstruction loss.

Here, we use 4 rules corresponding to the 4 transformations with the hope that the correct transformations are recovered. Indeed, we observe that **NPS successfully learns to represent each transformation using a separate rule**. The performance of the proposed model is demonstrated in figure 3. We also show that we can *hand-pick* rules to compose novel combinations of transformations on the digits.

5.2. Learning Action-Conditioned World Models

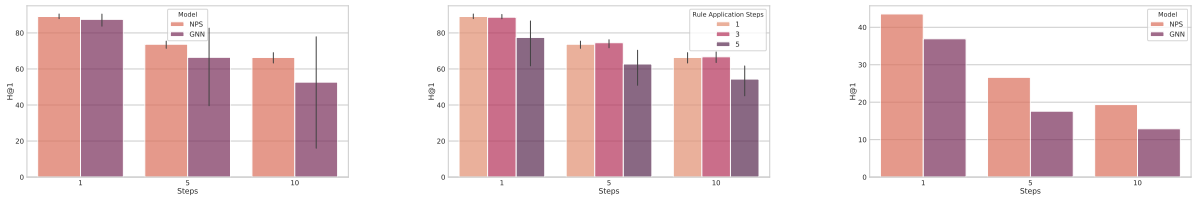
In this section, we present empirical results to support the discussion in section 4. For learning action conditioned world-models, we follow the same experimental setup as (Kipf et al., 2019). Therefore, all the tasks in this section are next-K step ($K = \{1, 5, 10\}$) prediction tasks, given the intermediate actions, and with the predictions being performed in the latent space. We use the following metrics described in (Kipf et al., 2019) for evaluation: **Hits at Rank 1 (H@1)** and **Mean Reciprocal Rank (MRR)**. H@1 is 1 for a particular example if the predicted state representation is nearest to the encoded true observation and 0 otherwise. We report the average of this score over the test set. Note that, higher H@1 indicates better model performance. For a description of MRR and the results on the MRR metric, we ask the reader to refer to the appendix. The proposed NPS is used as a drop-in replacement for the GNN used in the C-SWM model. We find that capturing sparse interactions using NPS is better suited for learning action-conditioned world models compared to GNNs in a wide range of settings as described below.

5.2.1. Physics Environment.

The physics environment (Ke et al., 2021), simulates a simple physics world. It consists of blocks of unique weights. The dynamics for the interaction between blocks is that the movement of heavier blocks pushes lighter blocks on their path. This rule creates an acyclic causal graph between the blocks. The weight of each block is observable through the intensity of its color. For an accurate world model, inferring the weights becomes essential. This environment is demonstrated in appendix Fig 7.

The agent performs stochastic interventions (actions) in the environment to infer the weights of the blocks. Each intervention makes a block move in any of the 4 available directions (left, right, up, and down). When an intervened block A with weight W_A comes into contact with another block B with weight W_B, the block B may get pushed if $W_B < W_A$ else B will remain still. Hence, any interaction in this environment involves only 2 blocks (i.e., 2 entities) and the other blocks are not affected. Therefore, modelling the interactions between all blocks for every intervention, as is generally done using GNNs, may be wasteful. NPS is particularly well suited for this task as any rule application takes into account only a subset of entities, hence considering interactions between those blocks only. Note that the interactions in this environment are not symmetrical and NPS can handle such relations. For example, consider a set of 3 blocks: $\{A_0, A_1, A_2\}$. If an intervention leads to A_1 pushing A_0, then NPS would apply the rule to the slot representing entity A_0. Since the movement of A_0 would depend on whether A_0 is heavier or lighter than A_1, NPS would also select the slot representing entity A_1 as a contextual slot and take it into account while applying the rule to A_0. Therefore, NPS can represent sparse and directed rules, which as we show, is more useful in this environment then learning dense and undirected relationships (or commutative operations).

For training, we follow the same setup as (Kipf et al., 2019). We use the C-SWM model as baseline. For the proposed model, we only replace the GNN from C-SWM by the proposed NPS. Graph neural networks are generally implemented by sharing parameters across edges, we can also implement them using separate parameters per edge but that



(a) **Physics Env** (b) **Physics Env** (c) **Atari Games**

Figure 4. **Action-Conditioned World Models**, with number of future steps to be predicted for the world-model on the horizontal axes. (a) Here we show a comparison between the proposed NPS and GNNs on the physics environment using the H@1 metric (higher is better). (b) Effect of number of rule application steps used in NPS when using a single rule in the physics environment. (c) Comparison of average H@1 scores across 5 Atari games for the proposed model NPS and GNN.

would break their equivariance (as we add more entities). For the proposed NPS model we use separate parameters per rule. Therefore, for a fair comparison to the GNN baseline in terms of number of parameters, we run NPS with a single rule. Note that this setting is still different from GNNs as in GNNs at each step every slot is updated by instantiating edges between all pairs of slots, while in NPS an edge is dynamically instantiated between a single pair of slots and only the state of the selected slot gets updated.

The results of our experiments are presented in figure 4(a). We can see that for 1 step ahead prediction, there is little difference between the proposed NPS and GNN but the difference becomes more apparent for multi-step ahead predictions (i.e 5 steps and 10 steps). Multi-step settings are more difficult to model as errors may get compounded over time steps. The sparsity of NPS (only a single slot affected per step) reduces compounding of errors and enhances symmetry-breaking in the assignment of transformations to rules, while in the case of GNNs, since all entities are affected per step, there is a higher possibility of errors getting compounded. We can see that even with a single rule, we significantly outperform GNNs thus proving the effectiveness of dynamically instantiating edges between entities. Using a single rule is appropriate for the physics environment as this environment affords only a single rule, i.e., heavier blocks push lighter blocks. For more compositional environments such as those explored in the next section, we find that having multiple rules improves performance.

Effect of Multiple Rule Application. We analyse the effect of sequentially applying multiple rules (i.e., $K > 1$) in this setting (i.e. with 1 rule). The results of this analysis are presented in figure 4(b). We can see that while there is very little difference in performance between 1 and 3 rule application steps, the performance drops sharply with 5 rule application steps. This performance drop can be attributed to the density of interactions captured by 5 rule application steps. When we have a single rule which we apply K times to a set of entities, this is equivalent to instantiating K edges between dynamically selected pairs of variables. Therefore with multiple rule application steps, the interactions captured by NPS become more *dense*. This further shows the importance of sparse interactions and may eventually provide a cognitive science hypothesis regarding the difficulty of humans to handle long chains of reasoning.

5.2.2. Atari Games.

We also test the proposed model in the more complicated setting of Atari games. Even though environments in Atari do not have well defined discrete components like in the physics environment, interactions are still sparse. For instance, in pong, any interaction involves only 2 entities: (1) paddle and ball or (2) ball and the wall. Therefore, we conjecture and experimentally show that sparse interactions are useful here as well.

We follow the same setup for Atari as for the physics environment described in the previous section. We present the results for the Atari experiments in figure 4(c), showing the average H@1 score across 5 games: Pong, Space Invaders, Freeway, Breakout, and QBert. We can see that the proposed model achieves a higher score than the GNN-based C-SWM. We expect the explanation for this to be similar to that we developed for the physics environment. The results for the Atari experiments reinforce the claim that NPSis especially good at learning sparse interactions.

5.3. Learning Rules for Physical Reasoning

To show the effectiveness of the proposed approach for physical reasoning tasks, we evaluate NPS on 2 datasets: Sprites-MOT (He et al., 2018) and bouncing balls (Van Steenkiste et al., 2018).

Sprites-MOT. The Sprites-MOT dataset was introduced in (He et al., 2018). The dataset contains a set of moving objects of various shapes. This dataset aims to test whether a model can handle occlusions correctly. Each frame has consistent bounding boxes which may cause the objects to appear or disappear from the scene. A model which performs well should be able to track the motion of all

Model	MOTA ↑	MOTP ↑	Mostly Detected ↑	Mostly Tracked ↑	Match ↑	Miss ↓	ID Switches ↓	False Positives ↓
OP3	$89.1_{\pm 5.1}$	$78.4_{\pm 2.4}$	$92.4_{\pm 4.0}$	$91.8_{\pm 3.8}$	$95.9_{\pm 2.2}$	$3.7_{\pm 2.2}$	$0.4_{\pm 0.0}$	$6.8_{\pm 2.9}$
NPS	$90.72_{\pm 5.15}$	$79.91_{\pm 0.1}$	$94.66_{\pm 0.29}$	$93.18_{\pm 0.84}$	$96.93_{\pm 0.16}$	$2.48_{\pm 0.07}$	$0.58_{\pm 0.02}$	$6.2_{\pm 3.5}$

Table 1. **Sprites-MOT**. Comparison between the proposed NPS and the baseline OP3 for various MOT (multi-object tracking) metrics on the sprites-MOT dataset (↑: higher is better, ↓: lower is better). Average over 3 random seeds.

objects irrespective of whether they are occluded or not. We follow the same setup as (Weis et al., 2020). We use the OP3 model (Veerapaneni et al., 2019) as our baseline. To test the proposed model, we replace the GNN-based transition model in OP3 with the proposed NPS.

We use the same evaluation protocol as followed by (Weis et al., 2020) which is based on the MOT (Multi-object tracking) challenge (Milan et al., 2016). The results for this task are presented in table 1. We ask the reader to refer to appendix F.1 for more details. We can see that for almost all metrics, NPS outperforms the OP3 baseline. Although this dataset does not contain physical interactions between the objects, sparse rule application should still be useful in dealing with occlusions. At any time step, only a single object is affected by occlusions i.e. it may get occluded due to another object or due to a pre-specified bounding box, while the other objects follow their default dynamics. Therefore, a rule should be applied to only the object (or entity) affected (i.e. not visible) due to occlusion and may take into account any other object or entity that is responsible for the occlusion.

Method	Enc	Inter.	Schemata	Test	Transfer
SCOFF	Seq.	direct	2	$0.28_{\pm 0.04}$	$0.15_{\pm 0.02}$
SA++	Iterative	direct	1	$0.63_{\pm 0.12}$	$0.31_{\pm 0.05}$
SCOFF++	iterative	direct	2	$0.71_{\pm 0.04}$	$0.35_{\pm 0.03}$
SCOFF+NPS	iterative	rules	1	$0.81_{\pm 0.02}$	$0.47_{\pm 0.04}$
OP3	Iodine	GNN	NA	$0.32_{\pm 0.04}$	$0.14_{\pm 0.1}$
OP3+NPS	Iodine	rules	NA	$0.51_{\pm 0.07}$	$0.34_{\pm 0.08}$

Table 2. **Bouncing Balls.** Comparison between the proposed NPS model and the various baselines in terms of ARI scores. OP3+NPS replaces the GNN used in OP3 by rules. SCOFF+NPS replaces the direct entity-to-entity interactions by rules-mediated interactions. Results are an average over 5 random seeds.

Bouncing Balls. We consider a bouncing-balls environment in which multiple balls move with billiard-ball dynamics. We validate our model on a colored version of this dataset. We compare the following methods: (a) SCOFF (Goyal et al., 2020): factorization of knowledge in terms of slots (object properties) and schemata, the latter capturing object dynamics; (b) SA++: we extend the idea of iterative competition among slots as proposed in slot attention (SA) (Locatello et al., 2020a) for dynamics prediction; (c) SCOFF++: we extend SCOFF by using the idea of iterative competition as proposed in SA; (d) SCOFF+NPS: We extend SCOFF++ by replacing the pair-wise attention communication in slot communication by the proposed NPS. (e) OP3 (Veerapaneni et al., 2019): this extends (Greff et al., 2019) for dynamics prediction; (f) OP3+NPS: we replace the GNN responsible for slot interaction in OP3 by the proposed NPS. For comparing different methods, we use the Adjusted Rand Index or ARI (Rand, 1971). To investigate how the factorization in the form of rules allows for extrapolating knowledge from fewer to more objects, we increase the number of objects from 4 during training to 6-8 during testing. As shown in table 2, replacing the GNN with NPS in OP3, and using rule-mediated interactions in SCOFF instead of direct slot-to-slot interactions results in much better ARI scores in terms of performance on the training set as well as performance on a transfer set (with more number of balls).

6. Discussion and Conclusion

Looking Backward. Production systems were one of the first AI research attempts to model cognitive behaviour and form the basis of many existing models of cognition. However, in traditional symbolic AI, both the key entities and the rules that operated on the entities were given. For AI agents such as robots trying to make sense of their environment, the only observables are low-level variables like pixels in images. To generalize well, an agent must induce high-level entities as well as discover and separate the rules which govern how these entities interact with each other. Here we have focused on perceptual inference problems and proposed NPS, a neural instantiation of production systems by introducing an important inductive bias in the architecture following the proposals of (Bengio, 2017; Goyal & Bengio, 2020; Ke et al., 2021).

Looking Forward. Our experiments on learning action-conditioned world models and extrapolation of knowledge in the form of learned rules in video prediction highlight the advantages brought by the factorization of knowledge into entities and sparse sequentially applied rules. Immediate future work would investigate how to take advantage of these inductive biases for more complex physical environments (Ahmed et al., 2020) and novel planning methods, which might be more sample efficient than standard ones (Schrittwieser et al., 2020). Humans seem to exploit the inductive bias in the sparsity of the rules and that reasoning about the application of these rules in an abstract space can be very efficient. For such problems, exploration becomes a bottleneck but we believe using rules as a source of *be-*

havioural priors can drive the necessary exploration (Goyal et al., 2019a; Tirumala et al., 2020; Badia et al., 2020).

7. Acknowledgements

The authors would like to thank Matthew Botvinick for useful discussions, and mentioning connections between RIMs and production systems which led to this work. The authors would also like to thank Alex Lamb, Stefan Bauer, Nicolas Chapados and Daniko Rezende for brainstorming sessions. We are also thankful to Dianbo Liu, Damjan Kalajdzievski and Osama Ahmed for proofreading.

References

Ahmed, O., Träuble, F., Goyal, A., Neitz, A., Wüthrich, M., Bengio, Y., Schölkopf, B., and Bauer, S. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020.

Anderson, J. R. Skill acquisition: Compilation of weak-method problem situations. *Psychological review*, 94(2): 192, 1987.

Anderson, J. R. *The architecture of cognition*, volume 5. Psychology Press, 1996.

Anderson, J. R. *Rules of the mind*. Psychology Press, 2014.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. An integrated theory of the mind. *Psychological review*, 111(4):1036, 2004.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pp. 507–517. PMLR, 2020.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Bengio, Y. The consciousness prior. *arXiv preprint arXiv:1709.08568*, 2017.

Bottou, L. and Gallinari, P. A framework for the cooperation of learning algorithms. In *Advances in neural information processing systems*, pp. 781–788, 1991.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.

Bunel, R., Hausknecht, M., Devlin, J., Singh, R., and Kohli, P. Leveraging grammar and reinforcement learning for neural program synthesis. *arXiv preprint arXiv:1805.04276*, 2018.

Burgess, C. P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M., and Lerchner, A. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

Cai, J., Shin, R., and Song, D. Making neural programming architectures generalize via recursion. *arXiv preprint arXiv:1704.06611*, 2017.

Chen, R. T., Li, X., Grosse, R., and Duvenaud, D. Isolating sources of disentanglement in variational autoencoders. *arXiv preprint arXiv:1802.04942*, 2018.

Ding, D., Hill, F., Santoro, A., and Botvinick, M. Object-based attention for spatio-temporal reasoning: Outperforming neuro-symbolic models with flexible distributed architectures. *arXiv preprint arXiv:2012.08508*, 2020.

Du, Y., Smith, K., Ulman, T., Tenenbaum, J., and Wu, J. Unsupervised discovery of 3d physical objects from video. *arXiv preprint arXiv:2007.12348*, 2020.

Engelcke, M., Kosiorek, A. R., Jones, O. P., and Posner, I. Genesis: Generative scene inference and sampling with object-centric latent representations. *arXiv preprint arXiv:1907.13052*, 2019.

Eslami, S., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Kavukcuoglu, K., and Hinton, G. E. Attend, infer, repeat: Fast scene understanding with generative models. *arXiv preprint arXiv:1603.08575*, 2016.

Evans, R., Hernández-Orallo, J., Welbl, J., Kohli, P., and Sergot, M. Making sense of sensory input. *Artificial Intelligence*, pp. 103438, 2019.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Goyal, A. and Bengio, Y. Inductive biases for deep learning of higher-level cognition. *arXiv preprint arXiv:2011.15091*, 2020.

Goyal, A., Islam, R., Strouse, D., Ahmed, Z., Botvinick, M., Larochelle, H., Levine, S., and Bengio, Y. Infobot: Transfer and exploration via the information bottleneck. *arXiv preprint arXiv:1901.10902*, 2019a.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. Recurrent independent mechanisms, 2019b.

Goyal, A., Lamb, A., Gampa, P., Beaudoin, P., Levine, S., Blundell, C., Bengio, Y., and Mozer, M. Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems. *arXiv preprint arXiv:2006.16225*, 2020.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Greff, K., Rasmus, A., Berglund, M., Hao, T. H., Schmidhuber, J., and Valpola, H. Tagger: Deep unsupervised perceptual grouping. *arXiv preprint arXiv:1606.06724*, 2016.

Greff, K., Kaufman, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., Matthey, L., Botvinick, M., and Lerchner, A. Multi-object representation learning with iterative variational inference. *arXiv preprint arXiv:1903.00450*, 2019.

He, Z., Li, J., Liu, D., He, H., and Barber, D. Tracking by animation: Unsupervised learning of multi-object attentive trackers. *CoRR*, abs/1809.03137, 2018. URL http://arxiv.org/abs/1809.03137.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., Hinton, G. E., et al. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Ke, N. R., Didolkar, A. R., Mittal, S., Goyal, A., Lajoie, G., Bauer, S., Rezende, D. J., Mozer, M. C., Bengio, Y., and Pal, C. Systematic evaluation of causal discovery in visual model based reinforcement learning, 2021. URL https://openreview.net/forum?id=gp5Uzbl-9C-.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

Kipf, T., van der Pol, E., and Welling, M. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

Kirsch, L., Kunze, J., and Barber, D. Modular networks: Learning to decompose neural computation. In *Advances in Neural Information Processing Systems*, pp. 2408–2418, 2018.

Kosiorek, A., Kim, H., Teh, Y. W., and Posner, I. Sequential attend, infer, repeat: Generative modelling of moving objects. *Advances in Neural Information Processing Systems*, 31:8606–8616, 2018.

Laird, J. E., Rosenbloom, P. S., and Newell, A. Chunking in soar: The anatomy of a general learning mechanism. *Machine learning*, 1(1):11–46, 1986.

Lamb, A., Goyal, A., Słowik, A., Mozer, M., Beaudoin, P., and Bengio, Y. Neural function modules with sparse arguments: A dynamic approach to integrating information across layers. *arXiv preprint arXiv:2010.08012*, 2020.

Le Roux, N., Heess, N., Shotton, J., and Winn, J. Learning a generative model of images by factoring appearance and shape. *Neural Computation*, 23(3):593–650, 2011.

Li, Y., Gimeno, F., Kohli, P., and Vinyals, O. Strong generalization and efficiency in neural programs. *arXiv preprint arXiv:2007.03629*, 2020.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020a.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention, 2020b.

Lovett, M. C. and Anderson, J. R. Thinking as a production system. *The Cambridge handbook of thinking and reasoning*, pp. 401–429, 2005.

McMillan, C., Mozer, M. C., and Smolensky, P. The connectionist scientist game: rule extraction and refinement in a neural network. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pp. 424–430, 1991.

Milan, A., Leal-Taixé, L., Reid, I. D., Roth, S., and Schindler, K. MOT16: A benchmark for multi-object tracking. *CoRR*, abs/1603.00831, 2016. URL http://arxiv.org/abs/1603.00831.

Neelakantan, A., Le, Q. V., and Sutskever, I. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

Rahaman, N., Goyal, A., Gondal, M. W., Wuthrich, M., Bauer, S., Sharma, Y., Bengio, Y., and Schölkopf, B. S2rms: Spatially structured recurrent modules. *arXiv preprint arXiv:2007.06533*, 2020.

Rand, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Raposo, D., Santoro, A., Barrett, D., Pascanu, R., Lillicrap, T., and Battaglia, P. Discovering objects and their relations from entangled scene representations. *arXiv preprint arXiv:1702.05068*, 2017.

Reed, S. and De Freitas, N. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

Ronco, E., Gollee, H., and Gawthrop, P. J. Modular neural networks and self-decomposition. *Technical Report CSC-96012*, 1997.

Rosenbaum, C., Klinger, T., and Riemer, M. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017.

Rosenbaum, C., Cases, I., Riemer, M., and Klinger, T. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019.

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

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Tacchetti, A., Song, H. F., Mediano, P. A., Zambaldi, V., Rabinowitz, N. C., Graepel, T., Botvinick, M., and Battaglia, P. W. Relational forward models for multi-agent learning. *arXiv preprint arXiv:1809.11044*, 2018.

Tirumala, D., Galashov, A., Noh, H., Hasenclever, L., Pascanu, R., Schwarz, J., Desjardins, G., Czarnecki, W. M., Ahuja, A., Teh, Y. W., et al. Behavior priors for efficient reinforcement learning. *arXiv preprint arXiv:2010.14274*, 2020.

Touretzky, D. S. and Hinton, G. E. A distributed connectionist production system. *Cognitive science*, 12(3):423–466, 1988.

Trask, A., Hill, F., Reed, S. E., Rae, J., Dyer, C., and Blunsom, P. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, pp. 8035–8044, 2018.

Van Steenkiste, S., Chang, M., Greff, K., and Schmidhuber, J. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2017.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J. B., and Levine, S. Entity abstraction in visual model-based reinforcement learning. *CoRR*, abs/1910.12827, 2019. URL http://arxiv.org/abs/1910.12827.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J., and Levine, S. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pp. 1439–1456. PMLR, 2020.

Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pp. 4539–4547, 2017.

Weis, M. A., Chitta, K., Sharma, Y., Brendel, W., Bethge, M., Geiger, A., and Ecker, A. S. Unmasking the inductive biases of unsupervised object representations for video sequences, 2020.

Xu, D., Nair, S., Zhu, Y., Gao, J., Garg, A., Fei-Fei, L., and Savarese, S. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2018.

Zablotskaia, P., Dominici, E. A., Sigal, L., and Lehrmann, A. M. Unsupervised video decomposition using spatio-temporal iterative inference. *arXiv preprint arXiv:2006.14727*, 2020.

Table 3. Learning representation of entities, dynamics of those entities and interaction between entities: Different ways in which the previous work has learned representation of different entities as a set of slots, dynamics of those entities (i.e., whether different entities follow the same dynamics, different dynamics or in a dynamic way i.e context dependent manner) and how these entities interact with each other. We note that the proposed model is agnostic as to how one learn the representation of different entities, as well as how these entities behave (i.e., dynamics of those entities).

Relevant Work	Entity Encoder	Entity Dynamics	Entity Interactions
RIMs (Goyal et al., 2019b)	Interactive Enc.	Different	Dynamic
MONET (Burgess et al., 2019)	Sequential Enc.	NA	NA
IODINE (Greff et al., 2019)	Iterative Reconstructive Enc.	NA	NA
C-SWM (Kipf et al., 2019)	Bottom-Up Enc.	NA	GNN
OP3 (Veerapaneni et al., 2020)	Iterative Reconstructive Enc.	Same	GNN
SA (Locatello et al., 2020a)	Iterative Interactive Enc.	NA	NA
SCOFF (Goyal et al., 2020)	Interactive Enc.	Dynamic	Dynamic

Appendices

A. Related Work

(McMillan et al., 1991) have studied a neural net model, called RuleNet, that learns simple string-to-string mapping rules. RuleNet consists of two components: a feature extractor and a set of simple condition-action rules – implemented in a neural net – that operate on the extracted features. Based on a training set of input-output examples, RuleNet performs better than a standard neural net architecture in which the processing is completely unconstrained.

Modularity and Neural Networks. A network can be composed of several modules, each meant to perform a distinct function, and hence can be seen as a combination of experts (Jacobs et al., 1991; Bottou & Gallinari, 1991; Ronco et al., 1997; Reed & De Freitas, 2015; Andreas et al., 2016; Rosenbaum et al., 2017; Fernando et al., 2017; Shazeer et al., 2017; Kirsch et al., 2018; Rosenbaum et al., 2019; Lamb et al., 2020) routing information through a gated activation of modules. The framework can be stated as having a meta-controller c which from a particular state s, selects a particular expert or rule $a = c(s)$ as to how to transform the state s. These works generally assume that only a single expert (i.e., winner take all) is active at a particular time step. Such approaches factorize knowledge as a set of experts (i.e a particular expert is chosen by the controller). Whereas in the proposed work, there's a factorization of knowledge both in terms of entities as well as rules (i.e., experts) which act on these entities.

Graph Neural Networks. GNNs model pairwise interactions between all the entities hence they can be termed as capturing *dense* interactions (Scarselli et al., 2008; Bronstein et al., 2017; Watters et al., 2017; Van Steenkiste et al., 2018; Kipf et al., 2018; Battaglia et al., 2018; Tacchetti et al., 2018). Interactions in the real world are sparse. Any action affects only a subset of entities as compared to all entities. For instance, consider a set of bouncing balls, in this case a collision between 2 balls a and b only affects a and b while other balls follow their default dynamics. Therefore, in this case it may be useful to model only the interaction between the 2 balls that collided (sparse) rather than modelling the interactions between all the balls (dense). This is the primary motivation behind NPS.

In the NPS, one can view the resultant computational graph as a result of sequential application of rules as a GNN, where the states of the entities represent the different nodes, and different rules dynamically instantiate an edge between a set of entities, again chosen in a dynamic way. Its important to emphasize that the topology of the graph induced in the NPS is dynamic, while in most GNNs the topology is fixed. Through thorough set of experiments, we show that learning sparse and dynamic interactions using NPS indeed works better than learning dense interactions using GNNs. We show that NPS outperforms state-of-the-art GNN-based architectures such as C-SWM (Kipf et al., 2019) and OP3 (Veerapaneni et al., 2019) while learning world-models.

Neural Programme Induction. Neural networks have been studied as a way to address the problems of learning procedural behavior and program induction (Graves et al., 2014; Reed & De Freitas, 2015; Neelakantan et al., 2015; Cai et al., 2017; Xu et al., 2018; Trask et al., 2018; Bunel et al., 2018; Li et al., 2020). The neural network parameterizes a policy distribution $p(a|s)$, which induces such a controller, which issues an instruction $a = f(s)$ which has some pre-determined semantics over how it transforms s. Such approaches also factorize knowledge as a set of experts. Whereas in the proposed work, there's a factorization of knowledge both in terms of entities as well as rules (i.e., experts) which act on these entities. (Evans

Type	Size	Activation
Linear	128	ReLU
Linear	Slot Dim. i.e size of $\boldsymbol{V}$	

Table 4. Architecture of the rule-specific MLP (MLP_r) in algorithm 1.

et al., 2019) impose a bias in the form of rules which is used to to define the state transition function, but we believe both the rules and the representations of the entities can be learned from the data with sufficiently strong inductive bias.

RIMs, SCOFF and NPS. (Goyal et al., 2019b; 2020) are a key inspiration for our work. RIMs consist of ensemble of modules sparingly interacting with each other via a bottleneck of attention. Each RIM module is specialized for a particular computation and hence different modules operate according to different dynamics. RIM modules are thus not interchangeable. (Goyal et al., 2020) builds upon the framework of RIMs to make the slots interchangeable, and allowing different slots to follow similar dynamics. In SCOFF, the interaction different entities is via direct entity to entity interactions via attention, whereas in the NPS the interactions between the entities are mediated by sparse rules i.e., rules which have consequences for only a subset of the entities.

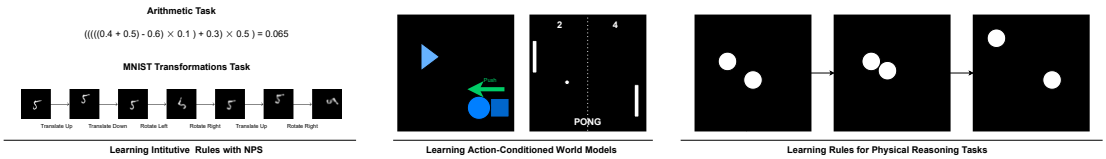


Figure 5. Demonstration of various experiments we use to prove the effectiveness of NPS. **Arithmetic Task**: NPS learns to solve mathematical equations by segregating each operation into a separate rule. **Learning Action-Conditioned World Models**: NPS successfully predicts the outcomes of various actions by learning sparse interaction rules between objects (or entities) in the environment. **Learning Rules for Physical Reasoning Tasks**: NPS succeeds in physical reasoning tasks in complex photo-realistic environments by learning physical rules that govern the dynamics of the objects.

B. NPS Specific Parameters

We use query and key size of 32 for the attention mechanism used in the selection process in steps 2 and 3 in algorithm 1. We use a gumbel temperature of 1.0 whenever using gumbel softmax. The architecture of the rule specific MLP is shown in table 4.

C. Arithmetic Task

This task is designed to test whether our model can learn rules corresponding to arithmetic operations. A concrete demonstration of this task can be found in figure 5. In our setup, all input numbers are sampled uniformly from $[0, 1]$. During training, the model learns to perform an operation on 2 numbers only. During test time, we present the model with longer sequences as those shown in figure 5. The model considers a single operation at a time during test time and uses the result of one operation as input for the next operation.

Setup.We use a 4-layered MLP to encode both the numbers to corresponding 100 dimensional vectors. The operation is encoded into a 3-hot vector for 3 operations: {addition, subtraction, and multiplication}. For the baseline, we concatenate the number representations with the operation vector and pass it through a unstructured model which is trained to output the result of the operation. For the proposed model, we feed the number and operation representations as slots to NPS algorithm 1. We use a batch size of 50 for our experiments. We use adam optimizer with a learning rate of $1e-4$. We generate 10000 examples for training and 2000 examples for testing.

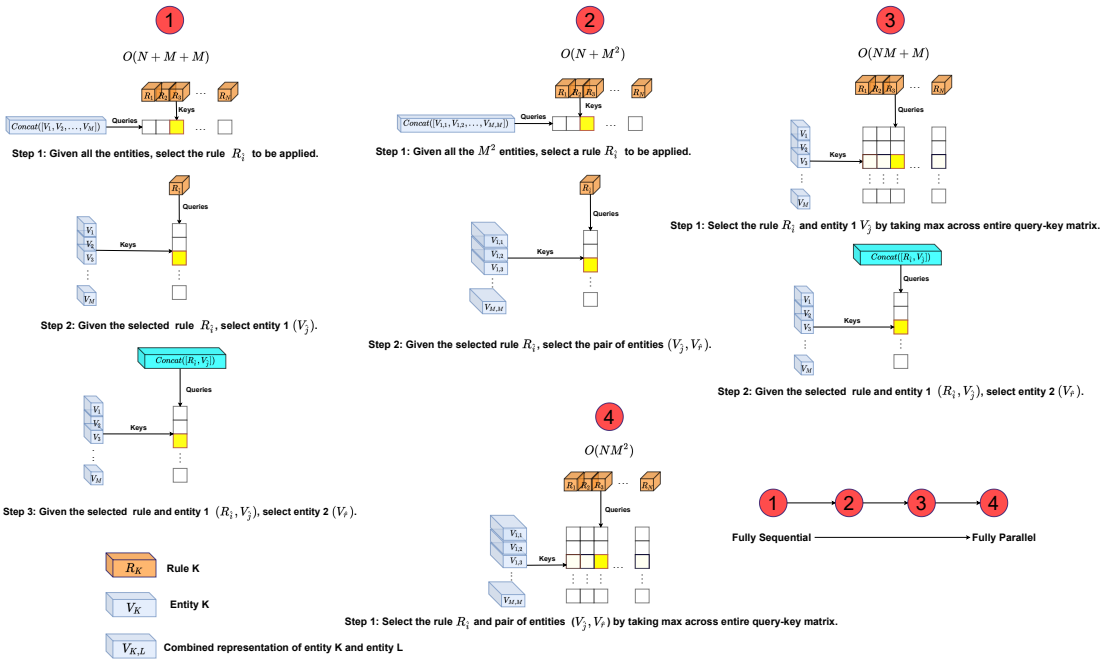


Figure 6. Different ways of parameterizing the search between different slots and rules: (1) The "simplest" way is to select everything greedily i.e., first select a rule, and then select the slot and then select another slot which acts as a contextual information. (2) Do a compositional search in the space of the slots by constructing M**2 slot pairs and then select the relevant production rule. (3) The third way is to do a joint search in the space of rules and slots, and then greedily select another entity which acts as a contextual information. (4) Do a joint search in the space of composition of slots and rules.

D. MNIST Transformation Task

In this task, we use images of size 64×64. There are 4 possible transformations that can be applied on a image: [Translate Up, Translate Down, Rotate Left, and Rotate Right]. During training, we present an image and the corresponding operation vector and train the model to output the transformed image. We use binary cross entropy loss as our training objective. As mentioned before, we observe that NPS learns to assign a separate rule to each transformation.

Setup. We first encode the image using the convolutional encoder presented in table 5. We present the one-hot transformation vector and the encoded image representation as slots to NPS (algorithm 1). NPS applies the rule MLP corresponding to the given transformation to the encoded representation of the image which is then decoded to give the transformed image using the decoder in table 5. We use a batch size of 50 for training.

E. Learning Action-Conditioned World Models

E.1. Physics Environment

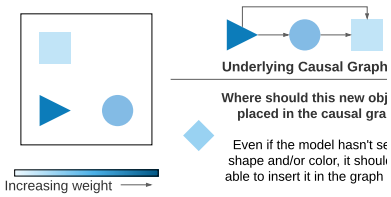


Figure 7. Demonstration of the physics environment.

A demonstration of this environment can be found in figure 7.

	Type	Channel	Activation	Stride
Encoder	Conv2D $[4 \times 4]$	16	ELU	2
	Conv2D $[4 \times 4]$	32	ELU	2
	Conv2D $[4 \times 4]$	64	ELU	2
	Linear	100	ELU	-
Decoder	Linear	4096	ReLU	-
	Interpolate (scale factor = 2)	-	-	-
	Conv2D $[4 \times 4]$	32	ReLU	1
	Interpolate scale factor = 2	-	-	-
	Conv2D $[4 \times 4]$	16	ReLU	1
	Interpolate scale factor = 2	-	-	-
	Conv2D $[3 \times 3]$	1	ReLU	1

Table 5. The architecture of the convolutional encoder and decoder for the MNIST Transformation task.

.

Data collection. For the data, we perform random interventions or actions in the environment and collect the corresponding episodes. We collect 1000 episodes of length 100 for training and 10000 episodes of length 10 for evaluation.

Metrics. For this task, we evaluate the predictions of the models in the latent space. We use the following metrics described in (Kipf et al., 2019) for evaluation: **Hits at Rank 1 (H@1)**: This score is 1 for a particular example if the predicted state representation is nearest to the encoded true observation and 0 otherwise. Thus, it measures whether the rank of the predicted representation is equal to 1 or not, where ranking is done over all reference state representations by distance to the true state representation. We report the average of this score over the test set. Note that, higher H@1 indicates better model performance. **Mean Reciprocal Rank (MRR)**: This is defined as the average inverse rank, i.e, MRR $= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{\text{rank}_n}$ where rank_n is the rank of the n^{th} sample of the test set where ranking is done over all reference state representations. Here also, higher MRR indicates better performance.

Setup. Here, we follow the experimental setup in (Ke et al., 2021). We use images of size 50×50. We first encode the current frame $\boldsymbol{x}^t$ using a convolutional encoder and pass this encoded representation using top-down attention as proposed in (Goyal et al., 2019b; 2020) to extract the separate entities in the frame as slots. We use a 4-layered convolutional encoder which preserves the spatial dimensions of the image and encodes each pixel into 64 channels. We pass this encoded representation to the object encoder which extracts the entities in the frame as M 64-sized slots ($\boldsymbol{V}^t_{1\ldots M}$). We then concatenate each slot with the action ($\boldsymbol{a}^t$) taken in the current frame and pass this representation to NPS which selects a rule to apply to one of the slots using algorithm 1. The following equations describe our model in detail. We use $M = 5$ since there are 5 objects in each frame. We use a rule embedding dimension of 64 for our experiments.

- $\hat{\boldsymbol{x}^t} = \text{Encoder}(\boldsymbol{x}^t)$
- $\boldsymbol{V}^t_{1\ldots M} = \text{Slot Attention}(\hat{\boldsymbol{x}}^t)$
- $\boldsymbol{V}^t_i = \text{Concatenate}(\boldsymbol{V}^t_i, \boldsymbol{a}^t)$ $\forall$i $\in \{1, \ldots, M\}$
- $\boldsymbol{V}^{t+1}_{1\ldots M} = \text{NPS}(\boldsymbol{V}^t_{1\ldots M})$

Here, NPS acts as a transition model. For the baseline, we use GNN as the transition model similar to (Kipf et al., 2019).

Training Details. The objective of the model is to make accurate predictions in the latent space. Mathematically, given the current frame $\boldsymbol{x}^t$ and a set of actions $\boldsymbol{a}^t, \boldsymbol{a}^{t+1}, \boldsymbol{a}^{t+2}, \ldots, \boldsymbol{a}^k$, the model performs these actions in the latent space as described in the above equations and predicts the latent state after applying these actions, i.e., $\boldsymbol{V}^{t+k}_{1\ldots M}$. Both the metrics, H@1 and MRR measure the closeness between the predicted latent state and $\boldsymbol{V}^{t+k}_{1\ldots M}$ and the ground truth latent state $\bar{\boldsymbol{V}}^{t+k}_{1\ldots M}$ which is obtained by passing frame x^{t+k} through the convolutional encoder and slot attention module. During training we use the contrastive loss which optimizes the distance between the predicted and the ground truth representations. We train the model for 100 epochs (1 hour on a single v100 gpu). Mathematically, the training objective can be formulated as follows:

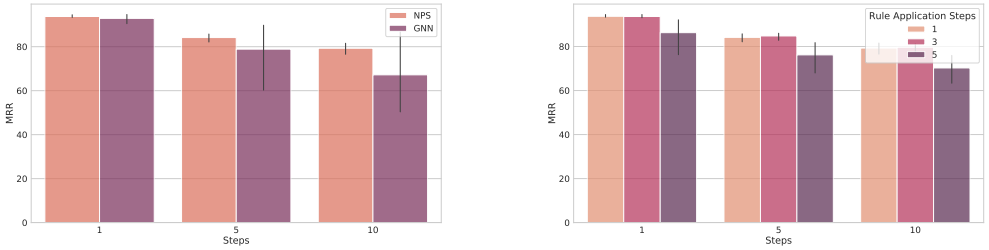
$$\textit{Contrastive Training} \; : \; \underset{\text{Encoder, Transition}}{\arg\min} \quad H + \max(0, \gamma - \tilde{H})$$

$$H = \text{MSE}(\hat{\boldsymbol{V}}^{t+1}_{1\ldots M}, \boldsymbol{V}^{t+1}_{1\ldots M})$$

$$\tilde{H} = \text{MSE}(\tilde{\boldsymbol{V}}^{t+1}_{1\ldots M}, \boldsymbol{V}^{t+1}_{1\ldots M})$$

$\tilde{\boldsymbol{V}}^{t+1}$: Negative latent state obtained from random shuffling of batch

$\hat{\boldsymbol{V}}^{t+1}$: Ground truth latent state for frame x^{t+1}



(a) **Performance on MRR** (b) **Effect of Rule Application Steps on MRR**

Figure 8. **Physics Environment performance on MRR**. (a) Here we compare the performance of NPS and GNN on the MRR metric for various forward-prediction steps.(We use 1 rule and 1 rule application step) (b) Here we analyse the effect of the rule application steps on MRR metric for NPS.

Results on MRR. We show the results on the MRR metric for thr physics environment in figure 8. From figure 8(a), we can see that NPS outperforms GNN while predicting future steps. From figure 8(b) , we can see that increasing rule application steps decreases performance on MRR. This decrease in performance can be attributed to increasing density of interactions captured by increasing rule application steps as highligted before in section 5.2. We can also see that the performance trend on MRR metric is the same as that on H@1 metric shown in figure 4[a,b]. Note that we use 1 rule for these experiments since physics environment affords only one rule , i.e., heavier blocks push lighter blocks.

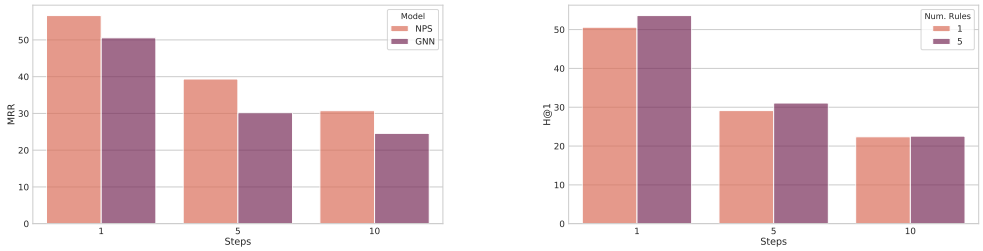
E.2. Atari

This task is also setup as a next step prediction task in the latent space. We follow the same setup as the physics environment for this task. We use the same H@1 and MRR metric for evaluation. We test the proposed approach on 5 games: Pong, Space Invaders, Freeway, Breakout, and QBert.

Data collection. Similar to the physics environment, here also we perform random interventions or actions in the environment and collect the corresponding episodes. We collect 1000 episodes for training and 100 episodes for evaluation.

Performance on MRR. We present the results on the MRR metric in figure 9(a). We can see that NPS outperforms on GNN for steps. We use a rule embedding dimension of 32 for our experiments.

Effect of Number of Rules. The physics environment is a very simple environment in which all kinds of interactions occur between similarly styled blocks and there is single rule governing these interaction. Hence using 1 rule in that environment is appropriate. The atari environment is more complicated than the physics environment and interactions in atari can take place between multiple different entities. The nature of these interactions are also more complex than in the physics environment. For example, in pong when the ball hits a paddle, the resulting direction of the ball depends on the angle of interaction between the ball and paddle, the direction in which the paddle is moving, and the direction in which the ball is moving. Therefore, we hypothesize that using multiple rules will be useful in atari to capture all these complicated interactions as compared to 1 rule. We present our results in figure 9(b). We can see that NPS with 5 rules outperforms NPS with 1 rule this showing the effectiveness of multiple rules in capturing complex interactions.



(a) **Performance on MRR** (b) **Effect of Number of Rules**

Figure 9. **Atari**. (a) Here we compare the performance of NPS and GNN on the MRR metric for various forward-prediction steps. (b) We study the effect of number of rules on the atari environments. We can see that more rules help in this environment as 5 rules outperform models with 1 rule. For this comparison, we use 1 rule application step. The results shown in these plots are averaged across 5 games.

F. Using Rules for Physical Reasoning

F.1. Sprites-MOT

Setup. We use the OP3 model (Veerapaneni et al., 2019) as our baseline for this task. We follow the exact same setup as (Weis et al., 2020). To test the proposed model, we replace the GNN-based transition model in OP3 with the proposed NPS. The output of the model consists of a seperate binary mask for each object in a frame along with an rgb image corresponding to each mask which contains the rgb values for the pixels specified by the mask.

Evaluation protocol. We use the same evaluation protocol as followed by (Weis et al., 2020) which is based on the MOT (Multi-object tracking) challenge (Milan et al., 2016). To compute these metrics, we have to match the objects in the predicted masks with the objects in the ground truth mask. We consider a match if the intersection over union (IoU) between the predicted object mask and ground truth object mask is greater than 0.5. We consider the following metrics:

- **Matches (Higher is better)**: This indicates the fraction of predicted object masks that are mapped to the ground truth object masks (i.e. IoU > 0.5).
- **Misses (Lower is better)**: This indicates the fraction of ground truth object masks that are not mapped to any predicted object masks.
- **False Positives (Lower is better)**: This indicates the fraction of predicted object masks that are not mapped to any ground truth masks.
- **Id Switches (Lower is better)**: This metric is designed to penalize *switches*. When a predicted mask starts modelling a different object than the one it was previously modelling, it is termed a an *id switch*. This metric indicates the fraction of objects that undergo and id switch.
- **Mostly Tracked (Higher is better)**: This is the ratio of ground truth objects that have not undergone and id switch and have been tracked for at least 80% of their lifespan.
- **Mostly Detected (Higher is Better)**: This the ratio of ground truth objects that have been correctly tracked for at least 80% of their lifespan without penalizing id switches.
- **MOT Accuracy (MOTA) (Higher is better)**: This measures the fraction of all failure cases i.e. false positives, misses, and id switches as compared to the number of objects present in all frames. Concretely, MOTA is indicated by the following formula:

$$MOTA = 1 - \frac{\sum_{t=1}^{T} M_t + FP_t + IDS_t}{\sum_{t=1}^{T} O_t} \tag{1}$$

  where, M_t, FP_t, and IDS_t indicates the misses, false positives, id switches at timestep t and O_t indicates the number of objects at timestep t.

- **MOT Precision (MOTP) (Higher is better)**: This metric measures the accuracy between the predicted object mask and the ground truth object mask relative to the total number of matches. Here, accuracy is measured in IoU between

the predicted masks and ground truth mask. Concretely, MOTP is indicated using the following formula:

$$MOTP = \frac{\sum_{t=1}^{T}\sum_{i=1}^{I} d_t^i}{\sum_{t=1}^{T} c_t} \tag{2}$$

where, d_t^i measures the accuracy for the i^{th} matched object between the predicted and the ground truth mask measured in IoU. c_t indicates the number of matches in timestep t.

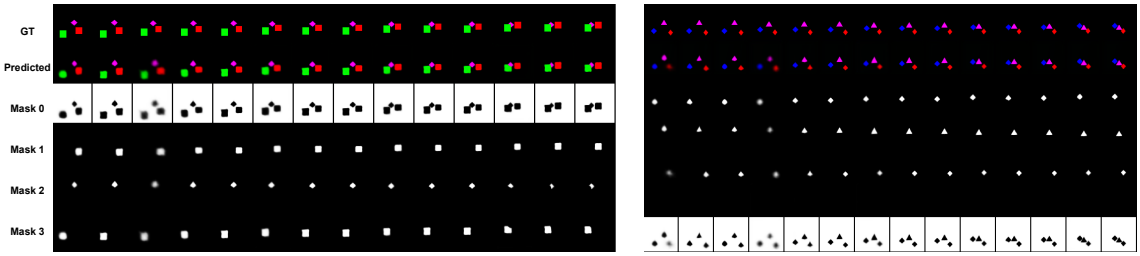


Figure 10. This figure shows the predictions of the OP3 model using the proposed NPS as a transition model. We can see that the proposed model succeeds in segregating each entity into separate slots and predicting the motion of each individual entity.

Model output. We show the predictions of the proposed model in figure 10. We use 10 rules and 3 rule application steps for our experiments. We use a rule embedding dimension of 64 for our experiments. Each rule is parameterized by a neural network as described in Tab. 4.

F.2. Bouncing Balls

Setup. We consider a bouncing-balls environment in which multiple balls move with billiard-ball dynamics. We validate our model on a colored version of this dataset. This task is setup as a next step prediction task where the model is supposed to predict the motion of a set of balls that follow billiard ball dynamics. During training each of the balls can have one of the four possible colors, and during testing we increase the number of balls from 4 to 6-8.

We use the following baselines for this task:

- **SCOFF (Goyal et al., 2020)**: This factorizes knowledge in terms of object files that represent entities and schemas that represent dynamical knowledge. The object files compete to represent the input using a top-down input attention mechanism. Then, each object file updates its state using a particular schema which it selects using an attention mechanism.
- **SA++**: Here we use SCOFF with 1 schema. We replace the input attention mechanism in SCOFF with an iterative attention mechanism as proposed in slot attention (Locatello et al., 2020b). We note that slot attention proposes to use iterative attention by building on the idea of top-down attention as proposed in (Goyal et al., 2019b). We note that slot attention was only evaluated on the static images. Here, the query is a function of the hidden state of the different object files in SCOFF, which allows temporal consistency in slots across the video sequence.
- **SCOFF++**: This model is same as SA++, but in this case we use 2 schemas instead of 1.
- **OP3**: We use the same setup for this model as for the Sprites-MOT dataset (Weis et al., 2020; Veerapaneni et al., 2019).

We use two instantiations of NPS for this task. (1) **SCOFF + NPS**: We augment SCOFF with iterative input attention (SA++) with the proposed NPS module. The states of the object files act as slot that are input into NPS. The NPS algorithm is applied after every object file update step. (2) **OP3 + NPS**: We replace the GNN transition model in OP3 with NPS. We use the MSE loss for training our model. We run each model for 20 epochs with batch size 8 which amounts to 24 hours on a single v100 gpu. We use 5 rules and 3 rule application steps for our experiments. We use a rule embedding dimension of 32 for our experiments.