Appeared at: ICAPS'20 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS'20), Nancy, France, October 2020

NLtoPDDL: One-Shot Learning of PDDL Models from Natural Language Process Manuals

Shivam Miglani*[*] and Neil Yorke-Smith
Delft University of Technology, The Netherlands
shivam.miglani@pwc.com, n.yorke-smith@tudelft.nl

Abstract

Existing automated domain acquisition approaches require large amounts of structured data in the form of plans or plan traces to converge. Further, automatically-generated domain models can be incomplete, error-prone, and hard to understand or modify. To mitigate these issues, we take advantage of readily-available natural language data: existing process manuals. We present a domain-authoring pipeline called NLtoPDDL, which takes as input a plan written in natural language and outputs a corresponding PDDL model. We employ a two-stage approach: stage one advances the state-of-the-art in action sequence extraction by utilizing transfer learning via pre-trained contextual language models (BERT and ELMo). Stage two employs an interactive modification of an object-centric algorithm which keeps human-in-the-loop to one-shot learn a PDDL model from the extracted plan. We show that NLtoPDDL is an effective and flexible domain-authoring tool by using it to learn five real-world planning domains of varying complexities and evaluating them for their completeness, soundness and quality.

1 Introduction

Acquiring a computational domain model for real-world planning problem such as space mission (Chien et al., 1998) requires *manual hand-coding* via careful collaboration between Subject Matter Experts (SMEs) and Knowledge Engineers (KEs) in a time-consuming and error-prone effort. In the past two decades, the Automated Planning (AP) community has devised a variety of *automated approaches* that learn the domain models from historical plans or plan traces. However, these inductive learning approaches require a substantial amount of structured data to form meaningful models, which leads to a causality dilemma as a large plan dataset cannot be generated without a prior domain model. Moreover, since symbols from structured data are used, the learned domain models are hard to explain or modify.

This paper focuses on a recent automated paradigm that uses *Natural Language (NL)* input data to deal with the issues of availability and explainability prevalent in automated methods that use structured data. Fig. 1 summarizes these three major paradigms of acquiring domain models. Although unstructured, the NL data is easily available in the form of instructions, how-to guides or process manuals. Additionally, for an SME, a natural language provides an innate way to visualize and understand components of a domain model and for devising few example plans.

We present **NLtoPDDL**, a flexible domain authoring pipeline that learns a PDDL model from a single NL process manual containing an example plan. To achieve this, the pipeline tackles two kinds of sub-problems: 1) *action sequence extraction* to extract plans from the unstructured

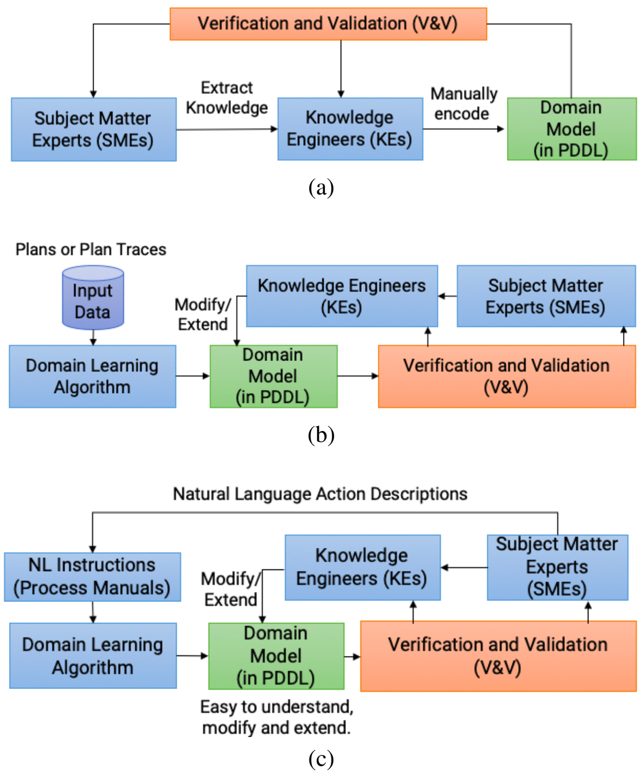

Verification and Validation (V&V)
Subject Matter Experts (SMEs)
Extract Knowledge
Knowledge Engineers (KEs)
Manually encode
Domain Model (in PDDL)
(a)
Plans or Plan Traces
Input Data
Modify/ Extend
Knowledge Engineers (KEs)
Subject Matter Experts (SMEs)
Domain Learning Algorithm
Domain Model (in PDDL)
Verification and Validation (V&V)
(b)
Natural Language Action Descriptions
NL Instructions (Process Manuals)
Modify/ Extend
Knowledge Engineers (KEs)
Subject Matter Experts (SMEs)
Domain Learning Algorithm
Domain Model (in PDDL)
Verification and Validation (V&V)
Easy to understand, modify and extend.
(c)

Figure 1: Domain model acquisition paradigms: (a) manual hand-coding, (b) automated using structured input (plan/-plan traces), (c) automated using unstructured NL input.

[*]Now at PricewaterhouseCoopers Advisory NV, Amsterdam

process manuals, and 2) our usual *automated domain acquisition* using the extracted structured data.

Specifically, we advance the state-of-the-art by:

- Assimilating transfer learning via contextual embeddings from pre-trained language models (e.g. BERT) into an action sequence extraction algorithm called EASDRL (Feng, Zhuo, and Kambhampati, 2018), to learn generalisable Deep Reinforcement Learning (DRL) models for extracting action sequences (Section 3.1 and 4).
- Designing interactive-LOCM; a human-in-the-loop version of LOCM2 algorithm (Cresswell and Gregory, 2011) which uses interactive graphs for easy visualization and user-modifications to learn PDDL models from a few examples (Section 3.2).
- Establishing the use-case of pre-trained language models for transfer learning concerning domain model acquisition, i.e., learning PDDL models from unseen, unrelated cross-domain *free text* NL process manuals (Section 5).
- Exemplifying the advantages of incorporating fast-paced current NL processing within AP tasks (Section 5.1).

The code of NLtoPDDL is available open-source for reproducibility at: https://github.com/Shivam-Miglani/contextual_drl.

2 Background and Related Work

A *planning domain* is a tuple $\langle \text{Ontology}, \text{Actions} \rangle$: the ontology encompasses *predicates* and *objects* of a domain and the actions represent the collection of action definitions. An action definition is an uninstantiated four-tuple: $\langle \text{Name}, \text{Parameters}, \text{Preconditions}, \text{Effects} \rangle$, where the parameters are object types, preconditions are set of necessary conditions (in the form of predicate values) which must be met before calling it and effects describe the changes in the environment (predicate values) after an action's execution (Segura-Muros, Pérez, and Fernández-Olivares, 2018). Thus, a *plan* can be defined as a sequence of instantiated actions that an intelligent agent takes to fulfil its objective. With captured state information, a *plan trace* is a sequence of interleaved actions and states. Given input in the form of plans, plan traces or a partial domain model, the task of domain acquisition is to learn a planning domain.

The AP community has developed many successful automated domain acquisition methods. They are scoped for learning PDDL models from different environments: deterministic effects and full observability (Kučera and Barták, 2018; Zhuo et al., 2010), deterministic effects but partial observability (Zhuo and Kambhampati, 2013; Segura-Muros, Pérez, and Fernández-Olivares, 2018), probabilistic effects and full observability (Pasula, Zettlemoyer, and Kaelbling, 2007; Mourão et al., 2012), and finally probabilistic effects and partial state observability (Jiménez, Fernández, and Borrajo, 2008). Our approach, much like LOCM family of algorithms (Cresswell and Gregory, 2011; Cresswell, McCluskey, and West, 2013; Gregory, Lindsay, and Porteous, 2017), is scoped towards deterministic effects and no state observability.

The type of technique used in the automated domain acquisition method forms another taxonomy. The technique ranges from inductive (Cresswell, McCluskey, and West, 2013), genetic (Colledanchise, Parasuraman, and Ögren, 2019; Kučera and Barták, 2018), uncertainty-based (Zhuo and Kambhampati, 2013), MAX-SAT based (Yang, Wu, and Jiang, 2007), model-lite (Kambhampati, 2007; Zhuo and Kambhampati, 2017), classical planning (Bandres, Bonet, and Geffner, 2018; Aineto, Jiménez, and Onaindia, 2018) to transfer learning (Zhuo and Yang, 2014) and deep learning (Asai and Fukunaga, 2018; Arora et al., 2018b,a). Our approach has elements of inductive, transfer learning and model-lite types but also keeps human-in-the-loop to generate useful domain models from less data.

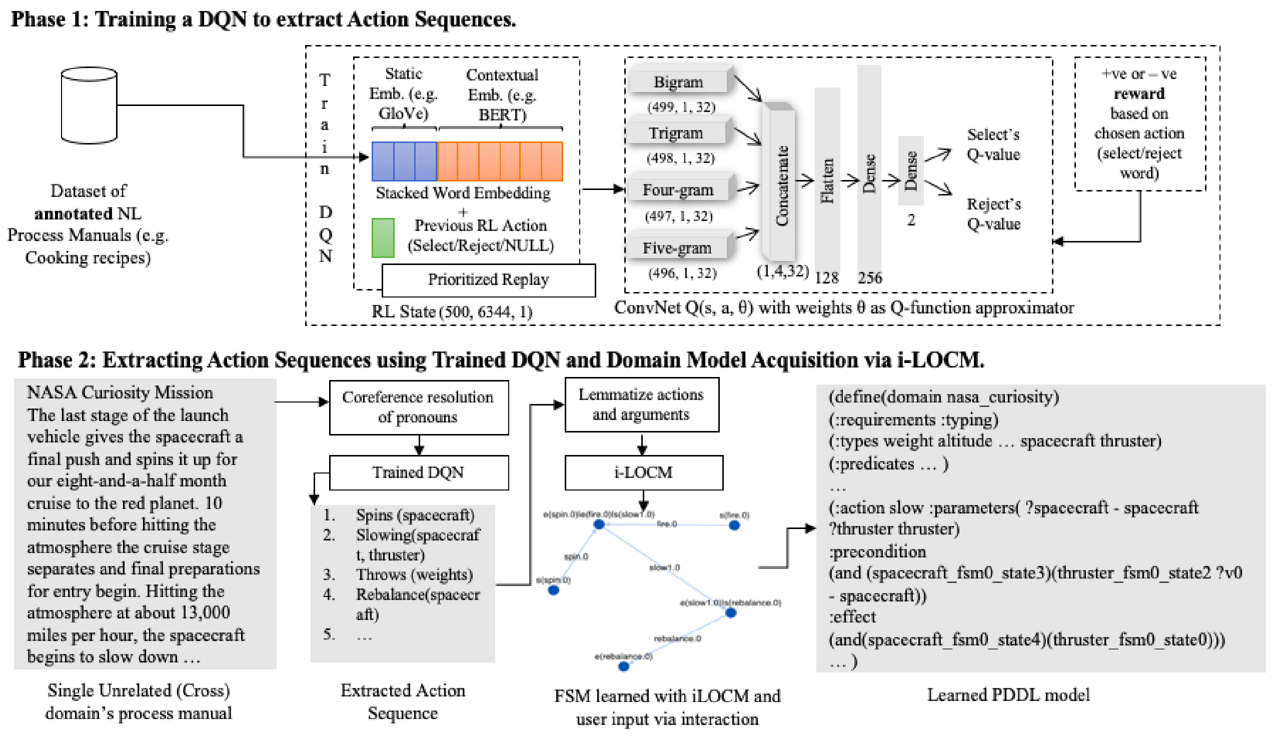
On the other hand, extracting action sequences from natural language instructions is an instance of sequence-to-sequence (*seq2seq*) class of problems (Sutskever, Vinyals, and Le, 2014). Most research in this area has been done on mapping navigational instructions (Mei, Bansal, and Walter, 2016). Mapping requires a prior finite set of actions as it performs sequence labelling instead of sequence extraction. EASDRL is an algorithm which uses DRL to achieve state-of-the-art results in extracting action sequences from *free* NL instructional data such as cooking recipes (Feng, Zhuo, and Kambhampati, 2018). We extend this approach with pre-trained language models (Devlin et al., 2019; Peters et al., 2018; Akbik, Blythe, and Vollgraf, 2018).

Considering the whole pipeline, i.e., generating domain models from NL data, very few approaches exist. NL instructions have been used to learn partial game dynamics by mapping them onto their logical action interpretations (Goldwasser and Roth, 2014). Framer uses a dependency parser to extract verbs and objects from small restricted sentences to formulate action templates and feeding the resulting sequences to LOCM (Cresswell, McCluskey, and West, 2013) to learn PDDL models (Lindsay et al., 2017). StoryFramer uses natural language stories to generate domain models in a mixed-initiative fashion (Hayton et al., 2017). In this paper, we extended the Framer's (Lindsay et al., 2017) architecture to work with unrestricted NL data.

3 NLtoPDDL Architecture

The task at hand is building a knowledge engineering tool which when given input in terms of a few plans (or a single plan) described in NL and no interleaved state information, should quickly output a valid and intuitive PDDL model. Moreover, the generated model should be easy to understand, extend or modify by the end-user. Fig. 2 shows the architecture of our NLtoPDDL pipeline, divided into two phases distinguished by the type of data used:

1. Training the DRL model with annotated training data. In Phase 1, a Deep Q-Network (DQN) based on EASDRL (Feng, Zhuo, and Kambhampati, 2018) trains on annotated datasets. This DQN learns to extract words that represent action names, action arguments, and the sequence of actions present in annotated NL process manuals. However, instead of originally-used Word2Vec embeddings (Mikolov et al., 2013), we incorporate dynamic contextual embeddings, namely BERT (Devlin et al., 2019), ELMo (Peters et al., 2018) and Flair (Akbik, Blythe, and Vollgraf, 2018)


Phase 1: Training a DQN to extract Action Sequences.
Dataset of **annotated** NL Process Manuals (e.g. Cooking recipes)
Train DQN
Static Emb. (e.g. GloVe)
Contextual Emb. (e.g. BERT)
Stacked Word Embedding
+ Previous RL Action (Select/Reject/NULL)
Prioritized Replay
RL State (500, 6344, 1)
Bigram (499, 1, 32)
Trigram (498, 1, 32)
Four-gram (497, 1, 32)
Five-gram (496, 1, 32)
Concatenate (1,4,32)
Flatten 128
Dense 256
Dense 2
Select's Q-value
Reject's Q-value
+ve or – ve **reward** based on chosen action (select/reject word)
ConvNet Q(s, a, θ) with weights θ as Q-function approximator
Phase 2: Extracting Action Sequences using Trained DQN and Domain Model Acquisition via i-LOCM.
NASA Curiosity Mission The last stage of the launch vehicle gives the spacecraft a final push and spins it up for our eight-and-a-half month cruise to the red planet. 10 minutes before hitting the atmosphere the cruise stage separates and final preparations for entry begin. Hitting the atmosphere at about 13,000 miles per hour, the spacecraft begins to slow down …
Coreference resolution of pronouns
Trained DQN
1. Spins (spacecraft)
2. Slowing(spacecraft, thruster)
3. Throws (weights)
4. Rebalance(spacecraft)
5. …
Lemmatize actions and arguments
i-LOCM
Single Unrelated (Cross) domain's process manual
Extracted Action Sequence
FSM learned with iLOCM and user input via interaction
Learned PDDL model

```
(define(domain nasa_curiosity)
(:requirements :typing)
(:types weight altitude … spacecraft thruster)
(:predicates … )
…
(:action slow :parameters( ?spacecraft - spacecraft
?thruster thruster)
:precondition
(and (spacecraft_fsm0_state3)(thruster_fsm0_state2 ?v0
- spacecraft))
:effect
(and(spacecraft_fsm0_state4)(thruster_fsm0_state0)))
… )
```

Figure 2: Pipeline architecture to learn PDDL domain models from natural language instructions. We build on elements from Lindsay et al. (2017), Akbik, Blythe, and Vollgraf (2018) and Feng, Zhuo, and Kambhampati (2018).

into EASDRL to boost its generalisability and performance. We call this variant *contextual-EASDRL* (cEASDRL).

2. Learning the domain model from unseen test data. In Phase 2, we extract the action sequences by feeding the unseen process manual to Phase 1's trained DQN. From this extracted sequence, LOCM2 (Cresswell and Gregory, 2011) algorithm learns a partial PDDL model in one-shot. The model can be completed with human input in an interactive fashion. We call this algorithm *interactive-LOCM* (iLOCM).

3.1 Training the contextual-EASDRL models

EASDRL uses DQNs to produce state-of-the-art results for action sequence extraction from unrestricted natural language texts; not requiring any prior list of actions (Feng, Zhuo, and Kambhampati, 2018). Since DQNs along with their improvements work well with discrete action spaces and are sample efficient (Hessel et al., 2018; François-Lavet et al., 2018), EASDRL exhibits similar properties. It represents action sequence extraction as a reinforcement learning problem and uses two DQNs. Each DQN can perform only two RL actions: *select* or *reject* a word. The first DQN $\mathcal{F}^1(actions|words;\theta_1)$ learns to extract Essential (ES), Exclusive-Or (EX) and Optional (OP) actions by *selecting* the words that represent these actions and rejecting other words. For this, the RL states consider 500 vectors (words) at a time and each vector is a concatenation of the word embeddings and RL action (NULL, select or reject) performed on them. The second DQN $\mathcal{F}^2(actions|words;\theta_2)$ uses the

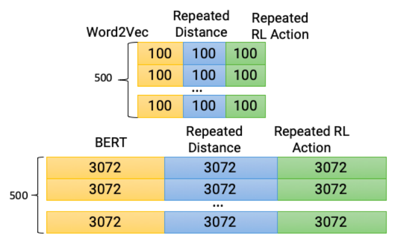


Figure 3: In DQN $\mathcal{F}^1$, RL states consider 500 words at a time. The repeat representation used in Feng, Zhuo, and Kambhampati (2018) leads to huge RL states in cEASDRL.

actions extracted by $\mathcal{F}^1$ to find arguments using the same strategy of selecting relevant words. Here, the RL state considers 100 features at a time, and each feature is $\mathcal{F}^2$ is a triple $\langle word\_vector, distance, RL\ action\rangle$, where the distance is the distance between the word in consideration and the action word. The static Word2Vec embeddings used in EASDRL renders it unable to handle out-of-vocabulary (OOV) words, multiple contexts (polysemy), and shared representations at the sub-word level. This restricts its generalisability beyond the training data.

Contextual-EASDRL extends EASDRL by incorporating contextual embeddings into it. These dynamic contextual embeddings include both the semantics of the word and other neural network parameters that describe their context. Since we already have a task-specific architecture, we use a *feature-based* instead of fine-tuning approach of the NL transfer learning to take advantage of following pre-trained language models: BERT, ELMo, Flair, and POS/NER embeddings (Trask, Michalak, and Liu, 2015). These resolve the mentioned issues of Word2Vec and produce generalisable DQN models which work with significantly different test distributions. Since word embeddings and contextual embeddings work well together, we use Flair library's stacked embeddings to combine them (Akbik et al., 2019). One side-effect of using contextual embeddings is the huge size of the RL state feature vector as shown in Fig. 3.

3.2 Acquiring PDDL models via iLOCM

In first step of Phase 2, we replace pronouns with nouns in the unseen process manual to reduce ambiguity using a neural network based technique (Huggingface, 2018). This coref-resolved unseen process manual is fed to the trained DQNs $\mathcal{F}^1$ and subsequently $\mathcal{F}^2$ of Phase 1 for sequence extraction. The extracted action sequences are normalized by doing Wordnet-based lemmatization (Bird, Klein, and Loper, 2009) to further reduce ambiguity, and are finally passed to the iLOCM for domain acquisition. The ambiguity reduction steps are necessary because they cluster similar looking actions and arguments under one template, resulting in higher quality PDDL models with precise action sets.

Our algorithm iLOCM is an interactive version of LOCM2 (Cresswell, McCluskey, and West, 2013; Cresswell and Gregory, 2011) which provides the option of correcting and modifying learned outputs (e.g. FSMs) of each step in an IPython (Pérez and Granger, 2007) environment. It substitutes the knowledge deficit caused by lesser data through these easily doable human corrections in an incremental fashion. The major changes include:

1. Interactive graphs of transition matrices and finite state machines for each object type visualized through Cytoscape.js (Franz et al., 2015) which helps in better visualization and faster modifications.
2. Optional user inputs to rename the object types (a.k.a. classes), edit the transition graphs and finite state machines, and enter static preconditions. The human-in-the-loop helps to achieve completeness from fewer data and also enhances explainability of the learned PDDL model.

The details of iLOCM are found at https://github.com/Shivam-Miglani/contextual_drl.

4 Evaluating the Trained DQN

Our evaluation of the trained DQN model considers:

Hypothesis 1. Dynamic contextual embeddings would integrate well with DQN of EASDRL and improve its performance and generalisability.

Hypothesis 2. Using dynamic contextual embeddings resolves the static embedding issues, namely, OOV words, polysemy, and shared representations at sub-word level.

Hypothesis 3. With a 20% unseen test set, we can have an unbiased estimate about the generalisation of our approach.

	WinHelp	Cooking	WikiHG
Labelled texts	154	116	150
Training pairs			
$\langle word, annotation \rangle$	1.5K	134K	34M
Action name rate(%)	19.47	10.37	7.61
Action argument rate(%)	15.45	7.44	6.30

Table 1: Annotated datasets used to train the DRL models. Values from Feng, Zhuo, and Kambhampati (2018).

Hypothesis 4. Transfer learning via contextual embeddings makes the DQN converge faster with the same amount of data, i.e., it converges in fewer epochs.

Annotated Datasets. For the training of the DRL models, the pre-annotated datasets are taken from Feng, Zhuo, and Kambhampati's repository[1] for three real-world domains:

1. 'Microsoft Windows Help and Support' (`WinHelp`) documents (Branavan et al., 2009; Feng, Zhuo, and Kambhampati, 2018)
2. 'CookingTutorial' (`Cooking`)[2]
3. 'WikiHow Home and Garden' (`WikiHG`)[3]

The datasets are of increasing complexity in view of the task of finding action names and arguments (Table 1).

Train–Val–Test Split. To avoid over-fitting, we held 20% of our data as unseen test data to test Hypothesis 3. The final ratio of training-validation-test data split was 64–16–20. We did not use cross-validation folds to avoid out-of-memory issues caused by large dimensionality of the embeddings.

Hyperparameters. We used the same hyperparameters for the ConvNet (Q-estimator of the DQNs) as mentioned in (Feng, Zhuo, and Kambhampati, 2018) except for changes in number of epochs and embedding dimensions. The authors of EASDRL took these parameters from MGNC-CNN (Zhang, Roller, and Wallace, 2016). We varied the input dimension according to the embedding used, for example, ELMo embeddings used (500 x 868 x 2) for action names and (100 x 868 x 3) for action arguments. The same architecture was used despite the large variation in embedding inputs of various baselines because the first Conv2D + MaxPool layer performs a dimensionality reduction leading to an equal-sized second layer irrespective of the size of input dimension. An instance of the architecture is shown in Fig. 2.

Evaluation Metrics. F_1-scores were computed the same way as in (Feng, Zhuo, and Kambhampati, 2018) for both validation and test sets. Specifically:

$$precision = \frac{\#TotalRight}{\#TotalTagged}, recall = \frac{\#TotalRight}{\#TotalTruth},$$

$$\text{and } F_1 = \frac{2 \times precision \times recall}{precision + recall}$$

[1]https://github.com/Fence/EASDRL

[2]http://cookingtutorials.com/

[3]https://www.wikihow.com/Category:Home-and-Garden

where $\#TotalRight$ is the number of correctly extracted action names or action arguments; $\#TotalTagged$ is the number of extracted action names or action arguments; and $\#TotalTruth$ is the number of ground truth action names or action arguments from the annotations.

Baselines. We compare these to the cEASDRL variants:

1. **StanfordCoreNLP**: Stanford CoreNLP's (Manning et al., 2014) parsing and parts-of-speech (POS) tagging was used in Framer (Lindsay et al., 2017) to extract verbs as action names and objects as action arguments.
2. **EASDRL** and **rEASDRL**: As EASDRL (Feng, Zhuo, and Kambhampati, 2018) reports cross-validation results, we compared it with the results of cEASDRL on our validation set. However, the results of EASDRL are not directly comparable as they are averaged out on 10 folds of the cross-validation but still give some indication of relative performance on F_1 score. For direct comparisons on both validation and test datasets, Reproduced-EASDRL (rEASDRL) was used. rEASDRL is same as EASDRL except that it employed our experimental setup of 64-16-20 train-val-test split without cross-validation but with hold-out set. In essence, rEASDRL trained on less data, i.e., 64% instead of 80% and so did our cEASDRL.
3. **GloVe + rEASDRL**: In GloVe + rEASDRL, static 100-dimensional GloVe embeddings (Pennington, Socher, and Manning, 2014) were used instead of Word2Vec.
4. **POS-GloVe + rEASDRL**: POS-GloVe + rEASDRL is inspired by Sense2Vec (Trask, Michalak, and Liu, 2015) and syntax-tree embeddings (Liu et al., 2017). We added some context to the words by appending parts-of-speech (POS) information extracted from the Stanford CoreNLP and then retrained the GloVe embeddings. For example, if the word was "cheese" is replaced by "cheese—NN", where NN stands for "Noun, singular or mass".

cEASDRL variants. We now specify the variants of our approach which are distinguished by the choice of contextual embedding employed. The following were our choice of stacked-embeddings based on empirical evidence of their performance (Peters et al., 2018; Devlin et al., 2019; Akbik, Blythe, and Vollgraf, 2018).

1. **GloVe + ELMo + cEASDRL**: This cEASDRL's variant uses 100 dimensional GloVe embeddings (Pennington, Socher, and Manning, 2014) and 768 dimensional ELMo embeddings (Peters et al., 2018). The pre-trained dataset used by these embeddings is 1B Word Benchmark (Chelba et al., 2013; Peters et al., 2018).
2. **GloVe + BERT + cEASDRL**: This variant uses *bert-base-uncased* version of BERT embeddings which are 3072-dimensional long vectors stacked with 100 dimensions of glove making it a 3172-dimensional stacked embedding. The datasets used in pre-training bert-base-uncased are BooksCorpus (800M words) and English Wikipedia (2,500M words) (Devlin et al., 2019).
3. **GloVe + Flair-f-b + cEASDRL**: This variant uses stacked *mix-forward* and *mix-backward* Flair character embeddings (Akbik, Blythe, and Vollgraf, 2018), each of which is 2048 dimensions. Stacked with GloVe, this makes a 4196-dimensional word embedding. The *mix-forward* and *mix-backward* versions are pre-trained on the mixed corpus (Web, Wikipedia, Subtitles) (Akbik et al., 2019). Due to high memory requirements, we set character limit to 128 characters per word.

4.1 Comparison with Baselines for Phase 1

Validation dataset results. Validation dataset allowed us to iterate over it repeatedly and get the best possible model weights and parameters. Table 2 shows that the validation results of the contextual-approaches are better than that of baselines in merely 1 epoch. Specifically, there is an improvement of 4-7% in the F_1-score from the best baseline for action names in all three datasets but barely any significant improvements in F_1-score for action arguments.

On the other hand, baseline POS-GloVe+rEASDRL had poor results as the average loss of DQN generally increased, and the RL agent became too conservative to act, neither selecting nor rejecting words to avoid negative rewards. This was caused by appended POS which created different entries in the lookup-table of GloVe and struggled to generalise when the same word was used in a different context.

Validation results give us an idea of the performance of cEASDRL variants compared to non-cEASDRL variants but is a biased estimate as it underestimates the true test error substantially. Thus, we look at the performance of all approaches on 20% of held-out test dataset to get an unbiased estimate that is closer to the real-world usage.

Test dataset results. In Table 3, we can clearly notice the advantage of using contextual embeddings. For the action names, we see an improvement over the best baseline by 5-8% in all datasets. For the action arguments, the biased estimate of the baselines on the validation is revealed, i.e., they were underestimating the true test error. Hence, the performance of baselines drops significantly, especially for complex datasets. In contrast, cEASDRL variants perform even better on the test sets than validation, making them more generalisable to real-world settings. In particular, GloVe+ELMo+cEASDRL beat the best baselines by 5-9% in Cooking and WikiHG datasets. This confirms our Hypotheses 1, 3 and 4.

Chosen model. Based on the test results, we select *GloVe+BERT+cEASDRL* for extracting action names and *GloVe+ELMo+cEASDRL* for extracting related action arguments as our final models.

4.2 Qualitative Analysis of Extracted Sequences

What does a 5% increase in F_1-score actually mean? To find out, we *qualitatively* assessed the extracted action sequences from unseen data. Fig. 4 shows an example of action descriptions taken from a fire safety manual to emulate the user-input. Comparing extracted action sequences of Word2Vec + rEASDRL and our chosen cEASDRL model, we see that cEASDRL produces coherent and correct action sequences, unlike Word2Vec + rEASDRL. In sentence no. 2, Word2Vec model initialises unseen words as zero vectors and is unable

			Action names			Action Arguments		
Method	Epochs	Cross-val	WinHelp	Cooking	WikiHG	WinHelp	Cooking	WikiHG
StanfordCoreNLP*	1	No	62.66	67.39	62.75	38.79	43.31	42.75
EASDRL	20	10-fold	93.46	84.18	75.40	**95.07**	74.80	75.02
Word2vec+rEASDRL	20	No	92.03	81.99	74.02	94.90	74.05	73.70
GloVe+rEASDRL	20	No	94.08	80.41	64.58	94.35	74.21	73.69
POS-GloVe+rEASDRL	20	No	32.33	0.0	0.0	73.24	38.82	42.66
GloVe+ELMo+cEASDRL	1	No	92.75	87.29	79.22	92.06	**75.81**	**76.99**
GloVe+BERT+cEASDRL	1	No	96.22	**89.18**	**82.59**	92.78	73.23	76.19
GloVe+Flair-f-b+cEASDRL	1	No	**97.32**	88.86	79.16	83.43	62.41	72.40

Table 2: F_1-scores on 16% of validation dataset in 64–16–20 train–val–test split. * indicates the result taken from Feng, Zhuo, and Kambhampati (2018). Note that extraction of action arguments uses ground-truth action names.

			Action names			Action Arguments		
Method	Epochs	Cross-val	WinHelp	Cooking	WikiHG	WinHelp	Cooking	WikiHG
Word2vec+rEASDRL	20	No	91.98	80.17	73.77	85.15	69.65	68.27
GloVe+rEASDRL	20	No	93.96	78.44	57.87	**94.02**	71.79	47.87
POS-GloVe+rEASDRL	20	No	32.68	0.0	0.0	74.55	38.63	51.27
GloVe+ELMo+cEASDRL	1	No	92.75	85.18	78.43	92.47	**76.50**	**77.12**
GloVe+BERT+cEASDRL	1	No	96.15	**88.42**	**82.95**	90.56	72.98	74.75
GloVe+Flair-f-b+cEASDRL	1	No	**97.46**	86.19	80.09	83.64	64.40	72.82

Table 3: F_1-scores on 20% of test dataset in 64–16–20 train–val–test split.

to extract essential actions. Contrarily, cEASDRL does extract correct action from unseen data and even recognises an exclusive-or between *hazardous experiments* and *procedures*. We see a safety-critical scenario in sentence no. 5, Word2Vec + rEASDRL, incorrectly extracts an action *go()* and misses the essential argument *heat*, whereas the chosen cEASDRL model correctly extracts the actions described in the statement. Some arguments are missed by both approaches, for example, *nature of emergency* in sentence no. 8 is missed because the models are trained to extract single word arguments. Also, cEASDRL correctly skips sentence no. 1 and 10 as they do not have any action names.

From these qualitative experiments, we deduce that the contextual embeddings do solve for problems of OOV, polysemy, and shared representation, confirming Hypotheses 2 and 3. This happens because these language models are pre-trained on either character level (ELMo, Flair) or sub-word level (BERT), and these algorithms take in the whole sentence as input to consider the context of the word before generating an embedding dynamically rather than just a dictionary lookup of a static vector. The language models being trained on large corpora also helps in disambiguating word senses (Peters et al., 2018; Devlin et al., 2019).

5 Evaluating Learned Domain Models

We use action sequences extracted from Phase 1 to learn PDDL models of five real-world domains through iLOCM. We bifurcate these into *related-domain transfer learning* where the domain to be learned is similar (but not same) to the NL instructions DQNs trained on; and *cross-domain transfer learning* where the domains to be learned are completely unrelated to the training data. Later on, we also evaluate learning of a non-classical (durative) domain.

Domains and their Input Process Manual. We provide an unseen process manual (to DQN) for each of the domains:

- Bicycle Tyre[4]: An instruction set for fixing a flat bicycle tyre related to `WikiHG` dataset.
- ChildSnack[5]: A process manual fabricated from IPC 2014 domain which is about serving gluten and gluten-free types of sandwiches according to dietary requirements for customers (children).
- Fire Safety[6]: A fire safety procedure, for which we extracted an action sequence in Fig. 4.
- NASA Curiosity[7]: A transcript of a mission video.
- Tea Domain: A durative action domain (Talukdar, 2019).

Evaluation Metrics. Precision and Recall are respectively used as measures of robustness and completeness of the learned PDDL model (Aineto, Jiménez, and Onaindia, 2018) with reference to the available/hand-crafted domain model. As there can be many reference models, precision and recall are subjective. Thus, we lay more emphasis on the qualitative analysis on some of the important interactions in the iLOCM process.

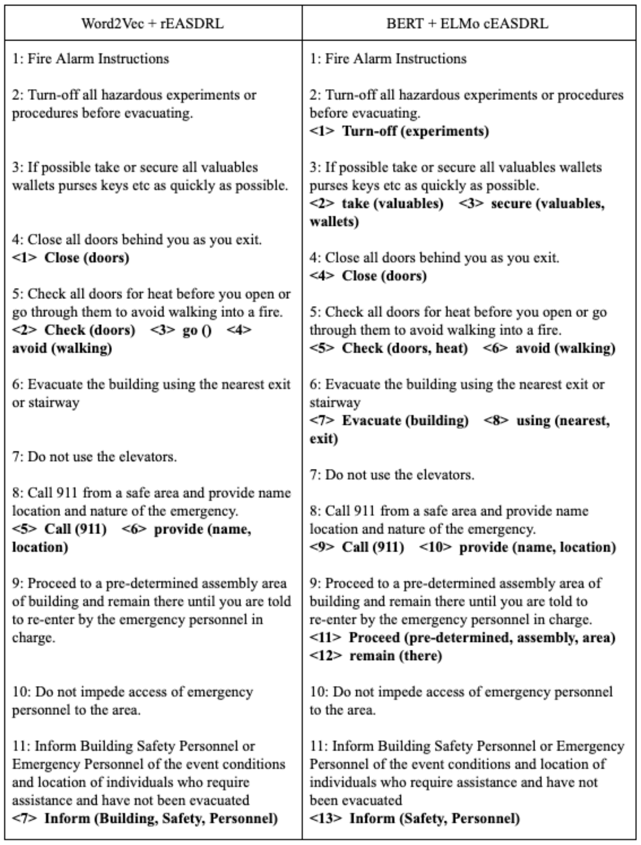
Bicycle Tyre and Childsnack – Related domains. We passed the lemmatized extracted action sequences of these

[4]https://theelectricbike.com/how-to-patch-an-e-bike-tire-2/

[5]https://github.com/potassco/pddl-instances/tree/master/ipc-2014/domains/child-snack-sequential-optimal

[6]http://www.fau.edu/ehs/info/fire-safety-manual.pdf

[7]https://mars.nasa.gov/resources/20024/next-mars-rover-in-action/?site=msl



Word2Vec + rEASDRL	BERT + ELMo cEASDRL
1: Fire Alarm Instructions	1: Fire Alarm Instructions
2: Turn-off all hazardous experiments or procedures before evacuating.	2: Turn-off all hazardous experiments or procedures before evacuating. **<1> Turn-off (experiments)**
3: If possible take or secure all valuables wallets purses keys etc as quickly as possible.	3: If possible take or secure all valuables wallets purses keys etc as quickly as possible. **<2> take (valuables) <3> secure (valuables, wallets)**
4: Close all doors behind you as you exit. **<1> Close (doors)**	4: Close all doors behind you as you exit. **<4> Close (doors)**
5: Check all doors for heat before you open or go through them to avoid walking into a fire. **<2> Check (doors) <3> go () <4> avoid (walking)**	5: Check all doors for heat before you open or go through them to avoid walking into a fire. **<5> Check (doors, heat) <6> avoid (walking)**
6: Evacuate the building using the nearest exit or stairway	6: Evacuate the building using the nearest exit or stairway **<7> Evacuate (building) <8> using (nearest, exit)**
7: Do not use the elevators.	7: Do not use the elevators.
8: Call 911 from a safe area and provide name location and nature of the emergency. **<5> Call (911) <6> provide (name, location)**	8: Call 911 from a safe area and provide name location and nature of the emergency. **<9> Call (911) <10> provide (name, location)**
9: Proceed to a pre-determined assembly area of building and remain there until you are told to re-enter by the emergency personnel in charge.	9: Proceed to a pre-determined assembly area of building and remain there until you are told to re-enter by the emergency personnel in charge. **<11> Proceed (pre-determined, assembly, area) <12> remain (there)**
10: Do not impede access of emergency personnel to the area.	10: Do not impede access of emergency personnel to the area.
11: Inform Building Safety Personnel or Emergency Personnel of the event conditions and location of individuals who require assistance and have not been evacuated **<7> Inform (Building, Safety, Personnel)**	11: Inform Building Safety Personnel or Emergency Personnel of the event conditions and location of individuals who require assistance and have not been evacuated **<13> Inform (Safety, Personnel)**

Figure 4: Extraction results of Word2Vec + rEASDRL and chosen BERT+ ELMo cEASDRL using the weights of `WikiHG` dataset.

two related-domains to the iLOCM algorithm. Fig. 5(a) shows learned parameterized state machine of the *tyre* class. It learns the correct behaviour of pinching tube, inserting lever, hooking lever, flipping bead and inflating tube of the tyre. The highlighted part shows the over-generalisation that iLOCM made due to *insert* action being used twice for both the *valve* and *lever* classes with similar action prototypes. This transition can be edited out by the end-user.

Fig. 5(b) shows one of the three state machines of the *sandwich* class of objects in the Childsnack domain. An object has to be present in each state of all state machines to justify its multiple behaviours. In this state machine, we see two disconnected components showing two types of *make-sandwich* actions. The top component has a state parameter *[Gluten-free]* to make only gluten-free sandwiches. Thus the correct behaviour is learned without user input.

Fire Safety and NASA Curiosity – Cross domains. The process manuals of these two cross-domains are significantly different from the data used to train DQNs. In both domains, the preconditions and effects of learned actions mostly reflect the previous action's end states and next action's start states respectively. It follows an intuitive notion of the linear sequence of actions prescribed in the corresponding process manuals.

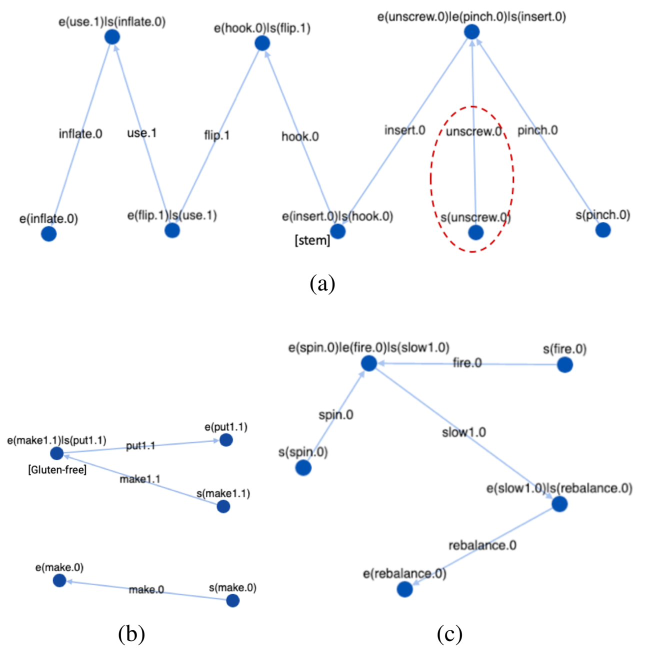
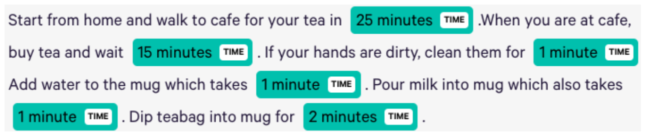


Figure 5: Learned state machines via iLOCM for: (a) Tyre class of domain Bicycle Tyre (b) Sandwich class of Childsnack, (c) Spacecraft class of NASA Curiosity domain. States/Nodes are represented in terms of start(transition) and/or end(transition); edges are transitions themselves on the domain objects. Entities inside '[]' denote state params.



Start from home and walk to cafe for your tea in 25 minutes TIME .When you are at cafe, buy tea and wait 15 minutes TIME . If your hands are dirty, clean them for 1 minute TIME Add water to the mug which takes 1 minute TIME . Pour milk into mug which also takes 1 minute TIME . Dip teabag into mug for 2 minutes TIME .

Figure 6: Time phrases extracted from NER module of SpaCy library (Honnibal and Montani, 2017).

Fig. 5(c) shows the state machine for the *spacecraft*. It learns the correct behaviour of spinning and firing the *thrusters* before slowing down and also re-balancing before landing. However, it is an incomplete state machine. It misses out on implicitly mentioned *spacecraft* in many sentences such as while deploying the parachute, dropping the back-shell, etc.

Summary. Table 5 shows precision and recall scores for names, arguments, preconditions and effects compared to an instance of manually constructed or existing PDDL model. The learned models are syntactically valid and capture valuable information about the domain from the limited information present in a process manual but they lack completeness demonstrated by the low recall in arguments, preconditions and effects. Since LOCM2 learns only the dynamic behaviour (Cresswell and Gregory, 2011), the learned model misses out on static conditions which can be provided through the interactive environment. Since the DQNs were

		Action Names		Action Params		Preconditions		Effects	
Domain	Similar to	P	R	P	R	P	R	P	R
Bicycle Tyre	WikiHG	1.00	1.00	0.94	0.84	0.80	0.64	0.85	0.72
Childsnack	Cooking	1.00	1.00	1.00	0.46	0.66	0.43	0.77	0.50
Fire Safety	–	0.92	1.00	1.00	0.66	0.73	0.50	0.75	0.56
NASA Curiosity	–	0.95	0.69	1.00	0.54	-*	-*	-*	-*

Table 4: Precision and recall compared to manually-constructed or existing PDDL models. * denotes not enough information in the process manual to construct a manual (meaningful) domain model to compare with.

trained to extract single words, the learned model extracts multiple words for the same argument and also misses out on implicit arguments. Thus, user input and modifications become necessary to make these models more robust and complete. Without the user input, these models can be termed as *shallow models* and can be used in approaches such as *model-lite planning* (Zhuo and Kambhampati, 2017).

5.1 Incorporating Durative Actions – Tea Domain

One of the advantages of using NL is that we can utilize existing research in the field to obtain information required for the non-classical PDDL models. To learn durative actions of the *tea* domain taken from Talukdar (2019), we used spaCy (Honnibal and Montani, 2017) library's statistical Named Entity Recognition (NER) to extract time phrases.

Fig. 6 shows that the time phrases are detected with high precision, however, we still need to assign these time phrases as action arguments for some action. We follow a simple heuristic of assigning the time phrase to the action appearing closest to the detected time phrase. This yielded perfect results for the tea domain. The learned model assigned correct duration to the actions but missed out preconditions and effects related to *mug* argument and static *hand* argument.

6 Conclusion and Future Work

We presented NLtoPDDL, a flexible domain authoring pipeline that learns a PDDL model from a single natural language process manual containing an example plan. NLtoPDDL enhances, combines and integrates algorithms for action sequence extraction and automatic domain acquisition into a single domain authoring framework. Contextual embeddings pave a way to train generalised sequence extraction DQNs and to do cross-domain transfer learning. Additionally, one-shot learning by keeping human-in-the-loop solves the causality dilemma. The generated PDDL models produced by NLtoPDDL are valid but shallow: thus, the generated models can be used in model-lite planning or, alternatively, a user or SME can extend them using the interactive interface or providing more data. Laying the foundations of NLtoPDDL in the active research field of NL processing means that we can learn more comprehensive and complex models than prior work in the automated planning literature.

Since NLtoPDDL decouples its components, we can individually enhance or replace them to boost its performance. We list the following ideas for future research. First, better-performing contextual embeddings or task-specific fine-tuned embeddings for larger number of epochs will yield a performance boost. One issue to resolve here is to find a smaller RL state representation than the repeat state version. Second, DQNs can be replaced by the asynchronous or theoretically sound alternatives (Mnih et al., 2016; Schulman et al., 2017). Third, focussing on extracting words with adjectives such as "cold milk" or compound nouns such as "training personnel" will generate a better argument F_1-score. Fourth, user studies should be done to assess the user-interactions that SMEs prefer and would like to do. Lastly, training on more NL data or learning the state representations can open up possibilities of better performing domain learners which use (partial) state information.

Acknowledgments We thank W. Feng and J. Porteous for sharing the code for EASDRL and LOCM1 respectively, and we thank the anonymous reviewers for their suggestions.

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS action models with classical planning. In *Proc. of ICAPS'18*, 399–407.

Akbik, A.; Bergmann, T.; Blythe, D.; Rasul, K.; Schweter, S.; and Vollgraf, R. 2019. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proc. of NAACL-HLT'19*, 54–59.

Akbik, A.; Blythe, D.; and Vollgraf, R. 2018. Contextual string embeddings for sequence labeling. In *Proc. of ACL'18*, 1638–1649.

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018a. A review of learning planning action models. *Knowledge Eng. Review* 33:e20.

Arora, A.; Fiorino, H.; Pellier, D.; and Pesty, S. 2018b. Action model acquisition using LSTM. *CoRR* abs/1810.01992.

Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proc. of AAAI'18*, 6094–6101.

Bandres, W.; Bonet, B.; and Geffner, H. 2018. Planning with pixels in (almost) real time. In *Proc. of AAAI'18*, 6102–6109.

Bird, S.; Klein, E.; and Loper, E. 2009. *Natural Language Processing with Python.* O'Reilly Media.

Branavan, S. R.; Chen, H.; Zettlemoyer, L. S.; and Barzilay, R. 2009. Reinforcement learning for mapping instructions to actions. In *Proc. of ACL'09*, 82–90.

Chelba, C.; Mikolov, T.; Schuster, M.; Ge, Q.; Brants, T.; and Koehn, P. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR* abs/1312.3005.

Chien, S. A.; Muscettola, N.; Rajan, K.; Smith, B. D.; and Rabideau, G. 1998. Automated planning and scheduling for goal-based autonomous spacecraft. *IEEE Intell. Syst.* 13(5):50–55.

Colledanchise, M.; Parasuraman, R.; and Ögren, P. 2019. Learning of behavior trees for autonomous agents. *IEEE Trans. Games* 11(2):183–189.

Cresswell, S., and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *Proc. of ICAPS'11*, 42–49.

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.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL-HLT'19*, 4171–4186.

Feng, W.; Zhuo, H. H.; and Kambhampati, S. 2018. Extracting action sequences from texts based on deep reinforcement learning. In *Proc. of IJCAI'18*, 4064–4070.

François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M. G.; and Pineau, J. 2018. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning* 11(3-4):242–253.

Franz, M.; Lopes, C. T.; Huck, G.; Dong, Y.; Sumer, O.; and Bader, G. D. 2015. Cytoscape.js: A graph theory library for visualisation and analysis. *Bioinformatics* 32(2):309–311.

Goldwasser, D., and Roth, D. 2014. Learning from natural instructions. *Machine Learning* 94(2):205–232.

Gregory, P.; Lindsay, A.; and Porteous, J. 2017. Domain model acquisition with missing information and noisy data. In *Proc. of ICAPS'17 KEPS Workshop*.

Hayton, T.; Porteous, J.; Ferreira, J.; Lindsay, A.; and Read, J. 2017. Storyframer: From input stories to output planning models. In *Proc. of ICAPS'17 KEPS workshop*.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proc. of AAAI'18*, 3215–3222.

Honnibal, M., and Montani, I. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. https://github.com/explosion/spaCy. Last accessed: 2019-09-14.

Huggingface. 2018. NeuralCoref 4.0: Coreference Resolution in spaCy with Neural Networks. https://github.com/huggingface/neuralcoref/. Last accessed: 2020-07-30.

Jiménez, S.; Fernández, F.; and Borrajo, D. 2008. The PELA architecture: integrating planning and learning to improve execution. In *Proc. of AAAI'08*, 1294–1299.

Kambhampati, S. 2007. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proc. of AAAI'07*, 1601–1605.

Kučera, J., and Barták, R. 2018. LOUGA: Learning planning operators using genetic algorithms. In *Pacific Rim Knowledge Acquisition Workshop*, 124–138.

Lindsay, A.; Read, J.; Ferreira, J. F.; Hayton, T.; Porteous, J.; and Gregory, P. 2017. Framer: Planning models from natural language action descriptions. In *Proc. of ICAPS'17*, 434–442.

Liu, R.; Hu, J.; Wei, W.; Yang, Z.; and Nyberg, E. 2017. Structural embedding of syntactic trees for machine comprehension. In *Proc. of EMNLP'17*, 815–824.

Manning, C. D.; Surdeanu, M.; Bauer, J.; Finkel, J. R.; Bethard, S.; and McClosky, D. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proc. of ACL'14 System Demos.*, 55–60.

Mei, H.; Bansal, M.; and Walter, M. R. 2016. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Proc. of AAAI'16*, 2772–2778.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. In *Proc. of ICLR'13 Workshop Track*. CoRR abs/1301.3781.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *Proc. of ICML'16*, volume 48 of *JMLR*, 1928–1937.

Mourão, K.; Zettlemoyer, L. S.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS operators from noisy and incomplete observations. In *Proc. of UAI'12*, 614–623.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29:309–352.

Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In *Proc. of EMNLP'14*, 1532–1543.

Pérez, F., and Granger, B. E. 2007. IPython: a system for interactive scientific computing. *Computing in Science and Engineering* 9(3):21–29.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. In *Proc. of NAACL-HLT'18*, 2227–2237.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *CoRR* abs/1707.06347.

Segura-Muros, J. Á.; Pérez, R.; and Fernández-Olivares, J. 2018. Learning numerical action models from noisy and partially observable states by means of inductive rule learning techniques. *Proc. of ICAPS'18 KEPS Workshop* 46–53.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Proc. of NeurIPS'14*, 3104–3112.

Talukdar, A. 2019. *Inference in Temporal Planning to Enhance Planning Performance for Problems with Required Concurrency*. PhD thesis, Kings College London.

Trask, A.; Michalak, P.; and Liu, J. 2015. sense2vec - A fast and accurate method for word sense disambiguation in neural word embeddings. *CoRR* abs/1511.06388.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2-3):107–143.

Zhang, Y.; Roller, S.; and Wallace, B. C. 2016. MGNC-CNN: A simple approach to exploiting multiple word embeddings for sentence classification. In *Proc. of NAACL-HLT'16*, 1522–1527.

Zhuo, H. H., and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proc. of IJCAI'13*.

Zhuo, H. H., and Kambhampati, S. 2017. Model-lite planning: Case-based vs. model-based approaches. *Artificial Intelligence* 246:1–21.

Zhuo, H. H., and Yang, Q. 2014. Action-model acquisition for planning via transfer learning. *Artificial Intelligence* 212:80–103.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence* 174(18):1540–1569.