Xuguang Duan, Qi Wu, Chuang Gan, Yiwei Zhang, Wenbing Huang, Anton Van Den Hengel, Wenwu Zhu

Watch, reason and code: Learning to represent videos using program

Proceedings of the 27th ACM International Conference on Multimedia (ACM Multimedia 2019), MM '19, 2019 / pp.1543-1551

Definitive Version of Record: http://dx.doi.org/10.1145/3343031.3351094

Watch, Reason and Code: Learning to Represent Videos Using Program

Xuguang Duan*
duan_xg@outlook.com
Tsinghua University

Qi Wu
qi.wu01@adelaide.edu.au
The University of Adelaide

Chuang Gan†
ganchuang1990@gmail.com
MIT-IBM Watson AI Lab

Yiwei Zhang
yw_zhangthu@163.com
Tsinghua University

Wenbing Huang
hwenbing@126.com
Tencent AI Lab

Anton van den Hengel
anton.hengel@adelaide.edu.au
The University of Adelaide

Wenwu Zhu†
wwzhu@tsinghua.edu.cn
Tsinghua University

ABSTRACT

Humans have a surprising capacity to induce general rules that describe the specific actions portrayed in a video sequence. The rules learned through this kind of process allow us to achieve similar goals to those shown in the video but in more general circumstances. Enabling an agent to achieve the same capacity represents a significant challenge. In this paper, we propose a Watch-Reason-Code (WRC) model to synthesise programs that describe the process carried out in a set of video sequences. The 'watch' stage is simply a video encoder that encodes videos to multiple feature vectors. The 'reason' stage takes as input the features from multiple diverse videos and generates a compact feature representation via a novel deviation-pooling method. The 'code' stage is a multi-round decoder that the first step leverages to generate a draft program layout with possible useful statements and perceptions. Further steps then take these outputs and generate a fully structured, compile-able and executable program. We evaluate the effectiveness of our model in two video-to-program synthesis environments, Karel and ViZdoom, showing that we can achieve the state-of-the-art under a variety of settings.

KEYWORDS

video understanding, video embedding, video to program translation

1 INTRODUCTION

The human ability to learn a skill by observing it being executed by another is fundamental to our individual development, and forms the backbone of our education process. Showing a video of 'making coffee using coffee machine' to a ten-year-old child is typically sufficient for them be able to operate a simple coffee machine, even if the appearance of the machine differs from that in the video. Showing a video of someone playing a game of 'Super Mario' (see Figure 1) similarly provides most of the information required to play it. Even better, the viewer will also do a better job of playing 'Sonic' because the two games have similar rules and a similar interface. Humans have such strong learning abilities because we can abstract over behaviours and situations to extract general rules that are applicable far more broadly. For example, rules like 'A mug needs to be placed under the outlet', 'press the full-cup button if you want a lungo' can be learned from coffee making videos, while rules like 'eat coins to gain points', and 'jump over items that have thorns' can be surmised from watching 'Super Mario'.

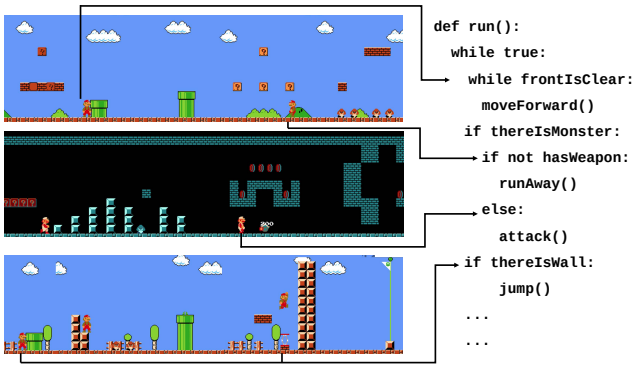

Figure 1: An illustration of the game of 'Super Mario'. Humans can imply general rules from these diverse videos. Our aim, known as video-to-program synthesis, is to train a machine to synthesise the underlying programs from several different video demonstration sequences, *i.e.* the model is required to summarise information from all input videos and predict the underlying logics, as shown in the right part of the figure.

Implying a general rule from a specific instance of a behaviour is an ill-posed problem, that humans solve on the basis of a lifetime's experience in observing and acting upon the world. For a machine to achieve the same task without the same lifetime's experience is complicated, but offers the prospect of machines that can learn to achieve a task by observation rather than instruction.

*Beijing National Research Center for Information Science and Technology (BNRist)
†Corresponding authors.

ACM ISBN . . . $
https://doi.org/

Although action recognition [Ali and Shah, 2010, Simonyan and Zisserman, 2014], description [Pan et al., 2017, Xu et al., 2016, Yu et al., 2016] and event prediction of further activities [Ryoo, 2011] in videos have progressed well thanks to the rapid development of deep learning, our task in this paper, is different. Our goal is to make a step towards a model that can generate executable programs by learning perception-based decision making logic rules from behaviours observed in multiple demonstrating videos, *i.e.*, program synthesis from videos. The video-based program synthesis empowers machines a more in-depth understanding ability on the diverse behaviours in videos, because a program is one of the most compact structured formal languages that can represent the decision making logic of different behaviours.

In contrast to conventional video understanding tasks [Ali and Shah, 2010, Pan et al., 2017, Ryoo, 2011, Simonyan and Zisserman, 2014, Xu et al., 2016, Yu et al., 2016], program synthesis from videos has two major challenges. First, the diversity of the demonstrating videos is large. A given collection of video demonstrations may only share the same final goal (for example, 'to survive' in the ViZdoom), but the action order or behaviour type might be totally different, due to the randomised setting of the perception environment. Hence, a sophisticated video summarisation model is required for synthesising underlying logic rules from diverse behaviours observed in videos. Since we have multiple video demonstrations that correspond to the same underlying program, this is a new multi-sequence-to-single-sequence problem, which is more challenging than the conventional sequence-to-sequence problem that only has a single input sequence. A powerful summarising model is thus required.

The second challenge is the strict constraints upon the output format. In contrast to the video classification task that only has a label output, and the video captioning task that only needs to generate a natural language description, the program synthesis problem is required to generate a piece of program code (with domain specific language) that can be executed in a domain-specific simulator. This requires that the generation process is able to strictly follow the grammar, for example, a 'while' statement must be followed by a 'condition', and there must be a 'then' if there is an 'if' token.

To this end, we propose a Watch-Reason-Code (WRC) model. The 'watch' stage is a recurrent neural network (RNN) based video encoder. However, in contrast to previous video encoders that encode each video independently, we use a peer-aware video encoder strategy to encode multiple diverse videos at the same time, through two correlated RNNs. The 'reason' module is responsible for summarising the output features from the last step, which we formulate as a pooling task. In order to reason and summarise diverse behaviours observed at different moments in different videos, we propose a novel deviation-pooling method which considers the feature similarity, deviation and the model complexity simultaneously. The last stage is a 'code' module that is formed by multiple RNNs. The first RNN focuses on producing a collection of possibly useful statements and perception conditions as a draft code statement. Subsequent steps (a sequence of RNNs) then refine the draft code repeatedly, to ensure it is executable and aligned with the input videos.

We evaluate the method's effectiveness on two video-based program synthesis datasets: Karel [Pattis, 1981] is a toy size dataset with a fully observable, third-person environment. ViZdoom [Kempka et al., 2016] is a large-scale shooting game dataset with partially observable, egocentric environment. Different settings of our model under different settings of environments are tested, and we outperform the baseline model in a large margin. Our final model achieves the state-of-the-art in both environments.

To summarise, our contributions are threefold:

- We propose a novel framework for a recently raised challenging problem, video-to-program synthesis. Our proposed framework divides the process into three stages which are 'watch', 'reason' and 'code', corresponding to a human programmer.
- We propose a novel deviation-pooling method that can effectively fuse features from multiple videos and domains. The proposed pooling strategy can reach similar performance with Relation-Network pooling[Santoro et al., 2017] method and cost much less resource.
- We show that in two datasets, whether fully or partially observable, our proposed Watch-Code-Deliberate model outperforms the comparison method, achieving the new state-of-art. Besides the performance, the results show that our model indeed learns to summarise the logic instead of memorise training examples, which is the most encouraging part of our work.

2 RELATED WORKS

2.1 Program Synthesis.

In addition to learning from video demonstrations, our work can be categorised as fulfilling the classic task of Program Synthesis, which aims to produce a program that describes the underlying logic of the given examples. Among program synthesis work, [Balog et al., 2017] makes use of search algorithms for inductive program synthesis, [Parisotto et al., 2017] proposes a Recursive-Reverse-Recursive neural network (R3NN) for string transformation, and [Bartoli et al., 2014] focus on the task of regular expression generation synthesis. Besides those works focusing on string transformation, [Bunel et al., 2018] uses a reinforcement learning framework and masks the decoder to address the importance of grammars in the KAREL [Pattis, 1981] environment, [Ellis et al., 2017] tries to infer the underlying graphics programs from hand-drawn images.

Most recently, [Sun et al., 2018] extends the task to a visual domain, that is, learning from video demonstrations. They introduce a 'summariser' module to integrate multiple demonstrations varying in behaviour with an RNN-based programme generator. We follow a similar encoder-decoder approach, but we extend it to three modules – a reasoning module that can effectively summarise diverse behaviours observed at different moments of different videos is inserted between the encoder and decoder. We also devise a sophisticated multi-round decoder which improves the program generation ability. Several experiments performed in Section 4 show the advantages of our proposed novel modules, compared to the models in [Sun et al., 2018].

2.2 Program Induction.

Instead of generating programs, Program Induction tries to discover the underlying logic of a certain task (*e.g.* sorting) and inducts a latent representation of their models. The Neural Turing Machine [Graves et al., 2014] and several other models [Kaiser and Sutskever,

2015, Kurach et al., 2015] try to solve the problem with the background of Turing Machines, and they work well at the tasks of sorting, memory-access, and long binary multiplication. Also, Stack-RNN [Joulin and Mikolov, 2015] makes use of an external stack-structured memory to learn algorithmic patterns of small description length. More recently, [Devlin et al., 2017] proposed an interesting approach to few-short program induction.

2.3 Video Understanding.

As our task tries to record the underlying logic of a set of video demonstrations using programming language, our work is also related to the task of video understanding. To understand videos, one common approach is to focus on discovering events and their correlations within videos, *e.g.* action recognition [Simonyan and Zisserman, 2014], and event prediction [Xu et al., 2016]. Some methods translate videos into other data modalities first [Song et al., 2016, Venugopalan et al., 2015]. Among the array of tasks related to video understanding, video captioning [Hori et al., 2017, Krishna et al., 2017, Shen et al., 2017, Venugopalan et al., 2015] is the most similar to ours, except in that case the output is natural language rather than an executable description of the actions observed. As natural language is flexible, and very robust, the process of generating and interpreting video captions is less demanding. However, in our program prediction scenario, the accuracy of the syntax and content is critical to developing an executable interpretation of the action, which increases the complexity of our task.

3 THE WATCH-REASON-CODE MODEL

Given k video demonstrations $V = \{\boldsymbol{v}_i\}_{i=1}^{k}$, the goal of the program synthesis is to generate an underlying program P that implies the behaviour logic in these videos. The program P is restricted to a Domain-Specific-Language(DSL) (see Figure 2) and is represented by a code: $P = \{\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_L\}$, where L is the length of the program and w is a code token. Each of the video demos $\boldsymbol{v}_i \in \mathbb{R}^{T_i \times H \times W \times C}$ is a video with length T_i, height H, width W, channel C generated with a simulation environment (*i.e.* a program executor), conditioned on the program P and a random initial state. The video frame rate is synchronous with the program, *i.e.* each executed action (*e.g.* 'move()') results in a specific frame in the video. Based on different initial state, the video demos would be different from each other, but all together traversal through all the branches and loops of the given program.

This problem can be seen as a new type of sequence-to-sequence prediction problem where the input sequence is a set of demonstrations V and the output is a code token sequence P. However, different from the conventional sequence-to-sequence problem, the input of this problem is more than one sequence. To solve the task, the model is required to be able to: 1) model every frame in every video demo, discover the implicit action (and condition) underlying the frame image; 2) discover and integrate the relationship between different video demos, find the underlying condition between different actions from different video demos; 3) decode the program correctly with the integrated information from previous stages.

To this end, we devise a Watch-Reason-Code (WRC) model (as shown in Figure 3) to address the above challenges. Our WRC model

```
def run():
  while isTargetDemon:        ------> "while" loop
    move()
  moveRight()                 ------> actions
  if isTargetHellKnight:      ------> perception
    attack()
  else:
    if isTargetRevenant:
      moveLeft()              ------> "if/else"
    else:                             statement
      turnRight()
```

Figure 2: A DSL program example in the ViZDoom, where all the typical components of modern program language are included, without variables.

consists of three modules: the 'watch' module is an encoder module that encodes multiple input videos simultaneously, mindful of their correlations and inter-dependencies; based on the output of the 'watch' module, the 'reason' module is required to summarise these features into a compact representation to avoid feature dimension explosion, and improve the performance at the same time. After obtaining the compact feature, the 'code' model is used to predict the program code. We thus propose a multi-stage decoding pipeline loosely based on the human reviewing process. We provide a detailed explanation of these three modules below.

3.1 Watch Module – A Peer-aware Encoder

Using Recurrent Neural Network (RNN) for video sequence encoding has been well studied [Krishna et al., 2017, Yuan et al., 2018]. Given the i-th input video $v_i = (\boldsymbol{v}_{i,0}, \boldsymbol{v}_{i,1}, ..., \boldsymbol{v}_{i,T_i})$, RNN would model it as:

$$\begin{aligned} \hat{\boldsymbol{v}}_{i,j} &= CNN(\boldsymbol{v}_{i,j}), j \in [0, T_i] \\ \boldsymbol{h}^a_{i,j} &= RNN^a(\hat{\boldsymbol{v}}_{i,j}, \boldsymbol{h}^a_{i,j-1}), j \in [0, T_i] \end{aligned} \tag{1}$$

where CNN is a convolution neural network for the j-th frame embedding and RNN^a is a recurrent neural network for sequence encoding, both of which shares parameters for all k video demos. The $h^a_{i,j}$ is the hidden state of v_i from the RNN^a at time step j , and $h^a_{i,-1} = \mathbf{0}$.

In our setting, since we have k correlative video sequence at the same time, ideally, the encoding process for each single video should consider other videos. Thus, following [Sun et al., 2018], we use a peer-aware video encoding strategy that uses another RNN_b to encode videos agian. Different from the RNN^a that is initialised with zero, the RNN^b is initialised by the average of the k video representation from the previous RNN_a. That means $\boldsymbol{h}^b_{init} = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{h}^a_{i,T_i}$, and

$$\boldsymbol{h}^b_{i,j}, = RNN^b(\boldsymbol{h}^a_{i,j}, \boldsymbol{h}^b_{i,j-1}), j \in [0, T_i] \tag{2}$$

i.e. a two-stage encoder is used, the information from different video demos is summarised in $\boldsymbol{h}^b_{init}$ and used in the second stage encoding. Besides, within each encoding stage, different videos are independent, which ensures efficient computing.

Then the last hidden state $\boldsymbol{h}^b_{i,T_i}$ from the RNN_b will be used as the feature representation for the i-th video (for simplicity, we denote it as $\boldsymbol{h}_i$ in the following).

3.2 Reason Module – A Deviation-pooling Net

The vector $\boldsymbol{h}_i$ is a good representation of video $\boldsymbol{v}_i$ considering its correlation with the other $k-1$ videos. However, in contrast to conventional single-sequence-to-single-sequence problems, under the multi-sequence-to-single-sequence setting, one of the biggest challenges is how to aggregate information from all input features. One straightforward way is concatenating all the input features together for decoding. However, as the dimension of $\boldsymbol{h}$ is usually hundreds or thousands of elements, and k is usually large, simple feature concatenation risks dimension explosion and thus model divergence.

To solve the dimension explosion problem a pooling strategy is typically applied. A simple pooling strategy would fail to preserve the complex information within the input features, however. Consider, for example, two input videos, containing the program segments 'move(), turnLeft(), move()', 'move(), turnRight(), move()'. The mean value can help us to decode the "move()" action, as that this action is shared. However, solely using mean pooling may fail to predict the condition branch 'turnLeft()', 'turnRight()', as that condition branch relies not only on the similarity of the inputs but also on the diversity.

To solve the above problem,Sun *et.al.* [Sun et al., 2018] propose to use RN-Pooling for feature aggregation, which, is a simplify of the Relation Network[Santoro et al., 2017]. Given a set of features $\{\boldsymbol{f}_i\}_{i=1}^K$, RN-Pooling conducts the following computation:

$$\hat{\boldsymbol{h}} = \frac{1}{K^2}\sum_{i,j} g_\theta(\boldsymbol{h}_i, \boldsymbol{h}_j), \tag{3}$$

within which the relationship between every input pair is modeled using g_θ(a MLP under parameter set θ), and then all the relationship features are averaged. Though achieved satisfying performance, RN-Pooling has its natural drawback: the computing complexity of RN-Pooling is extremely large compared with other simple pooling methods.

Based on the aforementioned requirement for a good pooling methods and the drawback of the RN-Pooling, we introduce a novel deviation-pooling strategy, which models the similarity and diversity of input features explicitly, and considers the pooling complexity at the same time. Formally, given a set of input features $\{\boldsymbol{h}_i\}_{i=1}^k$ from the encoder, where $\boldsymbol{h}_i \in \mathbb{R}^n$ is the corresponding feature of a certain input video. Given the similarity and deviations, the mean and max-min range of input feature $\{\boldsymbol{h}_i\}_{i=1}^k$ is computed:

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{k}\sum_i \boldsymbol{h}_i \\ \boldsymbol{\delta} &= \max\left(\{\boldsymbol{h}_i\}_{i=1}^k\right) - \min\left(\{\boldsymbol{h}_i\}_{i=1}^k\right) \end{aligned} \tag{4}$$

where $\max(\cdot)$ and $\min(\cdot)$ compute the point-wise maximum and minimum values respectively. Then, an MLP (denoted as g_θ as before) together with a residual connection is applied as follow:

$$\hat{\boldsymbol{h}} = g_\theta(\boldsymbol{\mu}, \boldsymbol{\delta}) + \boldsymbol{\mu} \tag{5}$$

Deviation-Pooling considers the deviation directly to address the importance of diversity in the program synthesis scenario. Compared with the relationship network-pooling (RN-pooling) method proposed in [Santoro et al., 2017] for multiple video encoding, our method is much more efficient. Specifically, the GPU memory usage is roughly 30% lower than RN-Pooling using the same number of parameters.

3.3 Code Module – A Multi-Round Decoder

Based on the features $\hat{\boldsymbol{h}}$ extracted by the previous 'reason' module, the goal of the 'code' module is to generate a sequence of statements that form a program.

Given the features $\hat{\boldsymbol{h}}$ extracted from the previous watch module as the initial hidden state for the decoder. A vanilla decoder conducts the following action:

$$\begin{aligned} \boldsymbol{y}_i,\ \boldsymbol{h}_i &= RNN_d(\boldsymbol{h}_{i-1}, \boldsymbol{x}_{i-1}) \\ \boldsymbol{w}_i &= f_\theta(\boldsymbol{y}_i) \end{aligned} \tag{6}$$

where RNN_d; is a specific type of RNN cell (e.g. LSTM[Hochreiter and Schmidhuber, 1997]) for sequence decoding, f_θ is a linear mapping that maps $\boldsymbol{y}_i$ from feature space to the statement token space, and $\boldsymbol{w}_i$ is the i-th token in the decoded sequence. Note that $\boldsymbol{w}_0$ is usually a pre-defined begin-of-sentence token '*BOS*', and the decoding process ends when $\boldsymbol{w}_i$ is a predefined end-of-sentence token '*EOS*'.

We denote the decoding model in Eq. 6 as $\mathcal{D}_\theta$, then we rewrite the process as:

$$\{\boldsymbol{w}_i\}_{i=1}^L, \{\boldsymbol{h}_i\}_{i=0}^L = \mathcal{D}_\theta(\hat{\boldsymbol{h}}) \tag{7}$$

where L is the length of the decoded sequence.

The above procedure is the same as is used in many natural language generation tasks, such as machine translation and image captioning. However, one of the biggest differences between natural language and program generation is the constraints upon the acceptable output: a natural language sentence could be understood even with some mistakes, but even a small error in piece of code can make it fail to compile, thus rendering it unusable. Human programmers also make mistakes, that are eliminated repeatedly compiling and correcting.

One straightforward way to perform multi-round decoding is to make use of the attention mechanism as in [Xia et al., 2017]. However, in [Xia et al., 2017], it proposes that such a framework would be hard to optimise (see [Xia et al., 2017] for more details). Instead, we make use of the hidden state and yield an easy-to-train framework. As that the decoding process would last more than twice (decode, refine), we describe a multi-round decoding module here. Formally, denote the i-th decode model (A RNN decoder) as $D_\theta^{(i)}$ and its initial hidden state is defined as follow:

$$\boldsymbol{h}_0^{(i)} = \mathcal{M}_{\boldsymbol{h}}(\boldsymbol{h}_0^{(0)}, \boldsymbol{h}_0^{(1)}, ..., \boldsymbol{h}_0^{(i-1)}, \boldsymbol{h}_{N_0}^{(0)}, \boldsymbol{h}_{N_1}^{(1)}, ..., \boldsymbol{h}_{N_{i-1}}^{(i-1)}) \tag{8}$$

where $\mathcal{M}_{\boldsymbol{h};\theta}$ is a pooling method (we use the aforementioned Deviation-pooling method of here). $\boldsymbol{h}_0^{(j)}$ is the j-th decoder's initial hidden state and $\boldsymbol{h}_{N_j}^{(j)}$ is its final hidden state, wehre N_j is the length of code generated by the j-th decoder. Eq. 8 means that the initial hidden state relies on the initial hidden states and final hidden states of all the previous decoder. The results are easy to obtain following

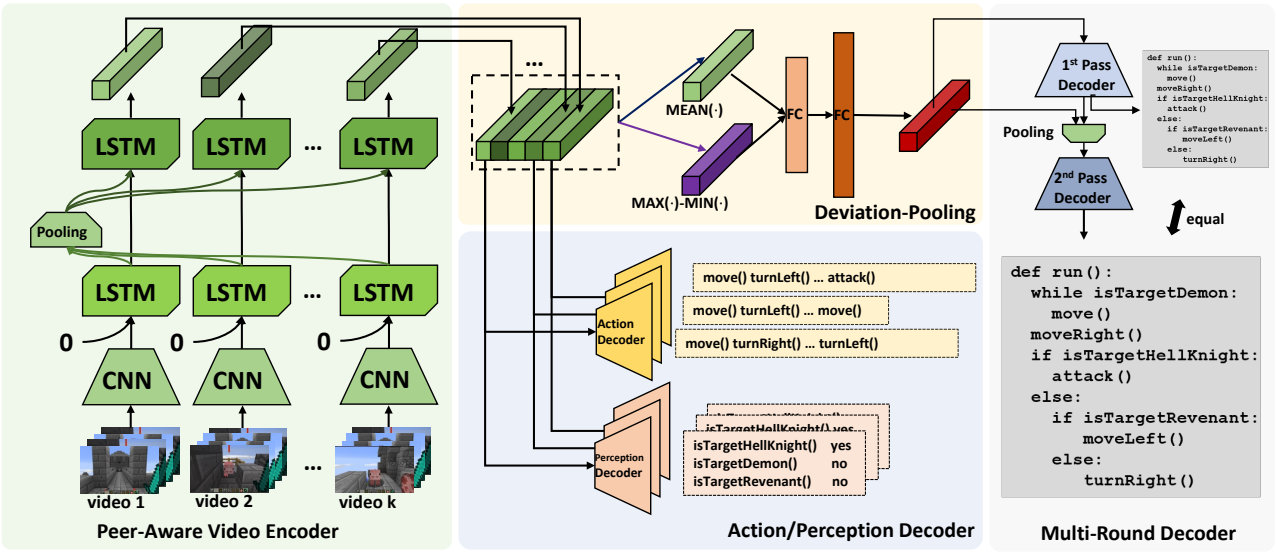

Figure 3: The overall model structure. There are 4 functional blocks of our model: 1) the peer-aware video encoder (the 'watch' module) that encodes k video demos in consideration of each other; 2) the deviation pooling net (the 'reason' module) that integrates all video features into a compact representation; 3) The Multi-Round program decoder (the 'code' module), which will refine the generated program round by round; 4) The Multi-task objective decoder for actions and perceptions.

standard decoding pipeline:

$$\{\boldsymbol{w}_j^{(i)}\}_{j=1}^{L_i}, \{\boldsymbol{h}_j^{(i)}\}_{i=0}^{L_i} = \mathcal{D}_\theta^{(i)}(\boldsymbol{h}_0^{(i)}) \tag{9}$$

When a two-layer decoding pipeline is used, the initial hidden state of the second decoder would become $\mathcal{M}_{\boldsymbol{h};\theta}(\boldsymbol{h}_0^{(0)}, \boldsymbol{h}_{N_0}^{(0)})$, which relies on the initial and final hidden state of the first decoder. The initial hidden state of the first decoder module is from the encoder which is zero-error, while the final hidden state stores the information of the decoding stage, using which the first decoder module can also be considered as an encoder encoding the information used for debugging. Under such a strategy, the information from previous decoding state is leveraged and such a process is very similar to a human programmer's debugging activity.

3.4 Learning Details

Multi-Round Decoder Objective Loss. Though equipped with a very complex structure, the above model can still be organised in an end-to-end manner and be trained with the vanilla sequence-to-sequence loss. For the i-th decoder, with the final predicted program $\{\hat{\boldsymbol{w}}_i^{(i)}\}_{i=1}^{L}$ and ground truth program $\{\hat{\boldsymbol{w}}_i\}_{i=1}^{L}$, the program prediction loss is formulated as:

$$\mathcal{L}_p^{(i)} = -\sum_{t=1}^{L} \ln p(\boldsymbol{w}_t | \hat{\boldsymbol{w}}_0^{(i)} : \hat{\boldsymbol{w}}_{t-1}^{(i)}, \hat{\boldsymbol{h}}_o^{(i)}). \tag{10}$$

and the final loss is:

$$\mathcal{L}_p = \sum \mathcal{L}_p^{(i)} \tag{11}$$

Also, there are two tricks on learning the Multi-Round Decoder. Firstly, to train the i-th decoder, the final state of the previous decoder is obtained using greedy decoding strategy instead of using the ground truth, in another word, the previous decoder is in its testing mode, otherwise, the i-th decoder learns to repeat the previous results which are not what we want. Secondly, as the i-th decoder depends on the previous decoder's prediction, our model is trained gradually, *i.e.* we train the first decoder and then train the second one, and then the following ones.

Multi-Task Objective Loss. Followed Sun et.al. [Sun et al., 2018], we also use a multi-task objective loss function. Besides predicting the final program directly, the model is also required to predict the action sequence and perception sequence of every video demo, which corresponds to the action(e.g. 'move()') and condition(e.g.'frontIsClear()') in the underlying program logic. More specifically, given the final representation h_i of a video demon(See Section 3.1), we make uses of two extra single layer RNN decoder to predict the action sequence:

$$\mathcal{L}_{action} = -\frac{1}{K}\sum_{k=1}^{K}\sum_{t=0}^{T_k} \ln p(\hat{a}_{k,t} | a_{k,0} : a_{k,t-1}, \boldsymbol{h}_k) \tag{12}$$

$$\mathcal{L}_{per} = -\frac{1}{K}\sum_{k=1}^{K}\sum_{t=0}^{T_k} \ln p(\hat{p}_{k,t} | p_{k,0} : p_{k,t-1}, \boldsymbol{h}_k) \tag{13}$$

where $\{\hat{a}_{k,i}\}_0^{T_k}$ and $\{\hat{p}_{k,i}\}_0^{T_k}$ is the ground truth action and perception sequence, $\{a_{k,i}\}_0^{T_k}$ and $\{p_{k,i}\}_0^{T_k}$ is the predicted sequence given by the action decoder and perception decoder $\mathcal{D}_{action,\theta}(\boldsymbol{h}_k)$ and $\mathcal{D}_{per,\theta}(\boldsymbol{h}_k)$.

4 EXPERIMENTS

In this section, we will give the evaluation results of our method. Firstly, an introduction of implementation details, datasets and evaluation metrics we used will be given. We then discuss the overall

results on two datasets. To verify the effectiveness of our newly proposed Deviation-Pooling and Multi-Round Decoder module, several ablation studies are then performed.

4.1 Datasets and Metrics

Datasets We evaluate our methods on two datasets: ViZdoom [Kempka et al., 2016] and Karel [Pattis, 1981]. ViZdoom is a large-scale shooting game dataset with partially observable, egocentric environment, which is used as our main experiment dataset. Karel is a toy size dataset with a fully observable, third-person environment, which is used in most of the papers on Program Synthesis, and we will evaluate our final model on it. More statistics about the two datasets are given in Table 1, and see Figure 6 for visualisation on ViZdoom dataset.

Metrics Following [Sun et al., 2018], the metrics used in our experiments include sequence accuracy, program accuracy, and execution accuracy. **Sequence Accuracy** counts exact match between ground truth program $\hat{\boldsymbol{P}}$ and the generated program $\boldsymbol{P}$, which is formally written as:$Acc_{seq} = \frac{1}{N}\sum_{n=1}^{N}\mathbb{1}_{seq}(\hat{\boldsymbol{P}}_n, \boldsymbol{P}_n)$. **Program Accuracy** considers the program aliasing, i.e. different program code may indicates the same meaning(e.g. 'repeat (2):(move())' and '(move() move()'). Thus, a function to exploit the syntax of the DSL is used to identify program aliasing: e.g. unfolding repeat statements, decomposing if-else statement. This accuracy can be written as $Acc_{seq} = \frac{1}{N}\sum_{n=1}^{N}\mathbb{1}_{prog}(\hat{\boldsymbol{P}}_n, \boldsymbol{P}_n)$ where $\mathbb{1}_{prog}$ is an indicator function that returns 1 if $\boldsymbol{P}_n$ and $\hat{\boldsymbol{P}}_n$ have the same program meaning. **Execution Accuracy**. Also, we use the simulator to regenerate the video demos with our predicted program $\boldsymbol{P}$, and compare the generated video with the ground truth video, i.e. $Acc_{exe} = \frac{1}{N}\sum_{n=1}^{N}\mathbb{1}_{exe}(\{v_{n,i}\}_{i=0}^{K_n}, \mathcal{E}(\boldsymbol{P_n}, \{v_{n,i,0}\}_{i=0}^{K_n}))$, where $\mathcal{E}$ is the simulator environment, and $\mathbb{1}_{exe}$ is another indicator function comparing similarity between videos. Note that when the number of seen demo increase, the execution accuracy will converge to the program accuracy.

Implementation Details Our peer-aware encoder is a stack of five CNN layers and two LSTM layers (see Eq. 1). The basic component of our decoders is a dynamic LSTM decoder, and all video demos share the same action decoder and perception decoder (see Eq. 6). All the LSTM hidden states are 512 in our experiment. Besides, the train, test, validate split is 25,000:5,000:5000 for Karel, 80,000:8,000:800 for Vizdoom as [Sun et al., 2018] does. We use Adam optimiser with the initial learning rate of 0.01 to train our model on a TITAN XP(12G memory) GPU.

Table 1: Datasets Statistics.

Statistic	ViZdoom	Karel
aspects	First Person	Third Person
dataset size	88,800	35,000
seen video per program	10	25
max video length	20	20
max program length	43	20

4.2 Overall Performance

Table 2 displays the overall results on ViZdoom and Karel dataset. We compare our model with the previous state-of-art in [Sun et al., 2018] using the same experiment setting. On ViZdoom, we can see that the performance increases of all the three metrics with the adding of Deviation Pooling, and more decoding steps, which demonstrates the effectiveness of our model. On Karel, however, DV-Pooling + 2 Decoder achieves the best performance. The reason is that Karel is a toy dataset where two rounds decoding are totally enough, and more decodings would lead into over-fitting.

Moreover, the scores under three metrics are consistent with all of our experiments: the 'Program Accuracy' is higher than 'Sequence Accuracy' and the 'Execution Accuracy' is higher than the other two. These evaluation metrics can be seen as a kind of mutual verification. For example, If two program codes(e.q. 'move() move()' and 'repeat 2(move())') are not exactly matched, the 'Sequence Accuracy' would not take it into account. However, the two code segments are equal from the perspective of programming code, and is taken into consideration under 'Program Accuracy'. Reaching higher 'Program Accuracy' indicates that our model indeed has learned to discover the logic and express it using its own 'comfortable' way, instead of remembering training examples and repeating them. According to the 'execution' accuracy, our model achieves 68.1% successful rate, which is higher than our own baseline model, which is implemented based on the code provided in [Sun et al., 2018].

In Figure 4, we show one demo result under Karel dataset. In the left is the demo videos observed by the model. In the right, we show the ground-truth and synthesised program, which demonstrates the model's ability. In Figure 4, we give an illustration of results under ViZdoom dataset. We give two cases. On the top is an example that the model succeeds in predicting the program. On the bottom is a more convincing example that the model generates a different but totally correct program, which proves our above analysis.

4.3 Effectiveness of the Pooling Strategies

The key differentiator of our proposed Watch-Reason-Code model is the reason model, which is a deviation-pooling net that can learn information from diverse videos. To evaluate its effectiveness, in this section we compare our DV (Deviation) pooling strategy with 'Mean-Pooling' and 'RN-Pooling', where Mean-Pooling is the most common pooling strategy, while RN-Pooling is used in [Sun et al., 2018] and achieved the state-of-art performance.

In Table 3, we give the complexity of the aforementioned pooling strategies. Specifically, RN-Pooling is very expensive with respect to memory and computation, for that it will produce an intermediate tensor of shape $2K^2V$, which is hundreds of times costly than Mean-Pooling and our DV-Pooling (K is 10 or 25 in our experiment). In our experiment, the total GPU memory consumption of RN-Pooling is about 30% higher than other pooling methods.

In Table 4, the results using different pooing strategies are given, where a single-time program decoder is used, and the best performing model of each method on the validation set is chosen for evaluation. From the result, we can see that both RN-Pooling and DV-Pooling outperform the basic Mean-Pooling strategy. Mean-Pooling tries to find the average representation of all the input features, which is in conflict with the need of the decoder to make use of the diversity

Table 2: Overall results. We compare our model with [Sun et al., 2018]'s model with the same setting.

Model \ Dataset	Karel			ViZdoom		
	Sequence	Program	Execution	Sequence	Program	Execution
Induction [Sun et al., 2018]	-	-	62.8	-	-	35.1
Synthesis [Sun et al., 2018]	35.7	42.4	64.1	33.1	39.3	48.2
RN-Pooling [Sun et al., 2018]	41.0	48.9	72.1	53.2	62.5	**78.4**
Mean-Pooling	40.3	48.3	71.8	51.2	58.5	62.5
RN-Pooling(our)[1]	41.5	49.3	73.2	54.1	61.7	66.4
DV-Pooling + 1 Decoder	43.0	50.6	74.7	54.8	62.4	66.2
DV-Pooling + 2 Decoder	**43.3**	**51.2**	**74.7**	55.6	62.8	67.5
DV-Pooling + 3 Decoder	42.5	49.8	72.2	**55.8**	**63.4**	**68.1***

[1] We re-implement the model from [Sun et al., 2018] and we actually get competitive and even better results than the original ones on most of the metrics. However, we failed to reproduce the results for the 'execution' accuracy in the Vizdoom split.

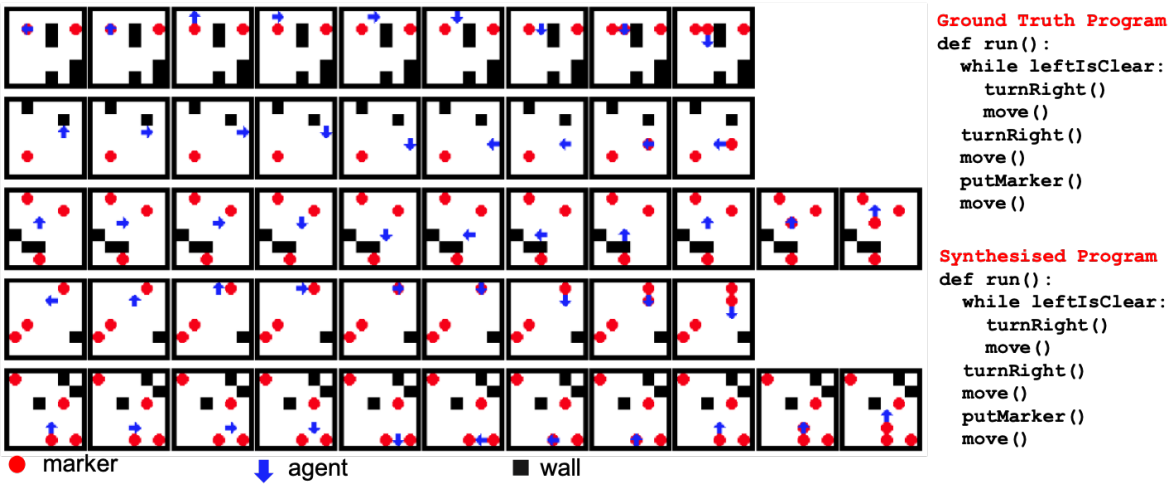

Figure 4: Selected results from Karel dataset. The model is able to synthesis correct program from given demo videos. In the demo videos, agent moves according a certain logic, and the model is required to find these underlying logic considering all given demos and synthesises a program to express the logic.

Table 3: Complexity of different pooling strategy. V is the dimension of feature and K is the number of demos;

Pooling	#parameter	Space	Computation
Mean	0	0	$O(KV)$
RN [Sun et al., 2018]	$O(V^2)$	$2K^2V$	$O(K^2V^2)$
DV(ours)	$O(V^2)$	$2V$	$O(KV+V^2)$

Table 4: Evaluation of pooling strategy using ViZdoom dataset.

Pooling	Sequence	Program	Execution
Mean	51.2	58.5	62.5
RN	54.1	61.7	**66.4**
DV+1 Decoder (ours)	**54.8**	**62.4**	66.2

of all input feature, which, in our opinion, is the reason for its bad performance. Besides, compared with RN-Pooling, our DV-Pooing reaches a higher performance while consumes much less computation and memory resource. The reason, in our opinion, is that our pooling methods, though simple, models the average representation and max-min margin(deviation) of features which is sufficient for the decoding of programs.

Also, as the pooling strategy takes multiple videos' feature as input, one may concern whether the proposed method is sensitive to the number of input videos or not. In Figure 5, we evaluate the performance of our DV-Pooling model with different number of videos in the task. From the table, we can see that: 1) both our DV-Pooling and the RN-pooling[Santoro et al., 2017] outperforms the basic Mean-Pooling by a great margin for any number of demos; 2) the model performance increases stably with the number of input videos increases and reaches almost stable when the number of videos is more than 25, which means that 25 videos contain enough

information to decode the underlying logic most of the time; 3) the model performance increases with the number of decoding stage increases regardless of the number of 'K', which means that our model is stable towards 'K', and more decoding stage can always benefit the model.

4.4 Evaluation of the Multi-Round Decoder on Underlying Conditions

To evaluate the effectiveness of our Multi-Round Decoder, we vary the rounds of decoding and compare the results with a standard single-round decoder. The overall results in Table 2 show that with the number of rounds of the decoder increasing, we can get better results. This makes sense because the programs are reviewed again and again.

To evaluate the ability to infer underlying conditions of our Multi-Round Decoder, following [Sun et al., 2018] we perform evaluation only with programs containing a single if-else statement with two branching consequences. In Table 5, the results using a single round, two rounds, three rounds decoding strategy are given. We can see a dramatic increase in performance with the increase in decoding round. In Table 5, we can see that our DV-Pooling and RN-Pooling outperform Mean-Pooling strategy by a great margin, which indicates the significance of a pooling strategy for condition decoding. Also, compared with the single layer decoding pipeline, our multi-round decoding strategy outperforms the basic model by 2.4% in Program Accuracy, which proves the ability to debug. [Sun et al., 2018] considers a similar problem of refining (debug) the final result. They make use of edit-distance to distinguish how long is it from their results to the **ground truth** (which should not be provided in the testing time). They find that correcting 2 tokens will lead to a performance gain of 4.9% in sequence accuracy. For our model, without using the ground truth correction in the testing, still achieves similar improvement, which proves the debugging efficiency of our proposed Multi-Round Decoder module.

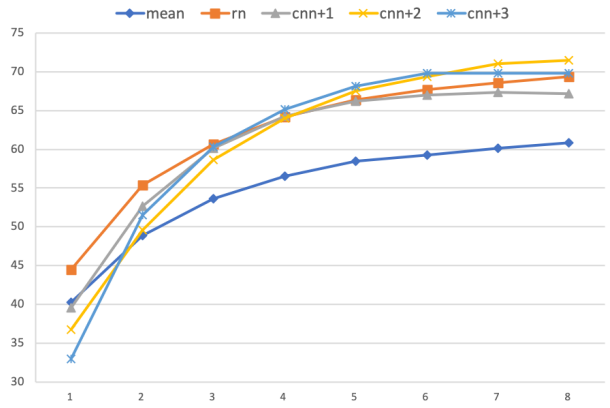

Figure 5: Model Sensitivity toward 'K' with respect to 'Execution Accuracy'. All model are trained using 25 videos, while tasted using different number of videos.

Table 5: Evaluation of Multi-round Decoder under 'ViZDoom if-else conditions'.

Decoding Time	Sequence	Program	Execution
Mean Pooling	46.7	57.7	69.8
RN-Pooling	55.1	65.3	**82.1**
DV + 1 decoder	55.7	65.4	81.7
DV + 2 decoder	**57.7**	67.3	79.9
DV + 3 decoder	57.6	**67.8**	81.6

5 CONCLUSION

Interpreting videos is a challenging task for a machine, not to mention implying rules from them. In this paper, we look into the task of synthesising program (a sequence of logic rules) from diverse video demonstrations. We proposed a novel Watch-Reason-Code (WRC) model to address two of its intrinsic problems: i) using a novel Deviation-Pooling strategy to integrate information from multiple input videos, which is known as a multi-input-single-output sequence to sequence problem. ii) using a multi-round decoding strategy to refine the program, which ensures its correctness and executability. This design is general enough to be extended to other domain and tasks, such as image captioning. The experiment results on two datasets demonstrate the effectiveness of our methods.

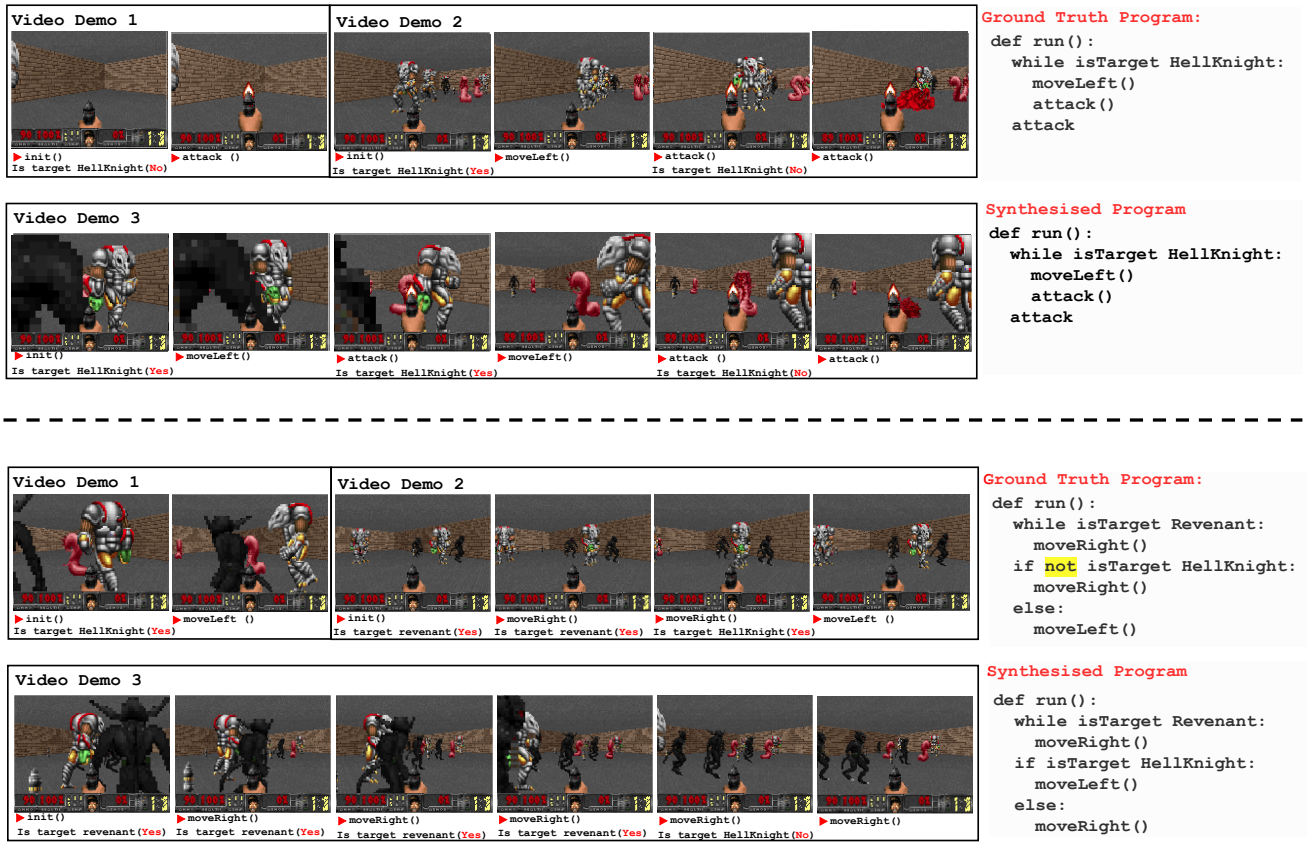

Figure 6: Selected results from ViZdoom dataset. The demo videos are in different length, and we annotate each video frames based on current action and key perception for better understanding. On the top is a success case where our model predicts the underlying program correctly. On the bottom, however, is a 'failure' case, where the prediction is not the same with the ground truth, but expresses the same meaning, which shows that our model does not try to repeat the training data, but finds the underlying logic.

REFERENCES

Saad Ali and Mubarak Shah. 2010. Human action recognition in videos using kinematic features and multiple instance learning. *IEEE transactions on pattern analysis and machine intelligence* 32, 2 (2010), 288–303.

M Balog, AL Gaunt, M Brockschmidt, S Nowozin, and D Tarlow. 2017. DeepCoder: Learning to Write Programs. In *International Conference on Learning Representations (ICLR)*. OpenReviews. net.

Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Eric Medvet, and Enrico Sorio. 2014. Automatic synthesis of regular expressions from examples. *Computer* 47, 12 (2014), 72–80.

Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. 2018. Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis. *International Conference on Learning Representations (ICLR)* (2018).

Jacob Devlin, Rudy R Bunel, Rishabh Singh, Matthew Hausknecht, and Pushmeet Kohli. 2017. Neural Program Meta-Induction. In *Advances in Neural Information Processing Systems (NIPS)*. 2080–2088.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B Tenenbaum. 2017. Learning to Infer Graphics Programs from Hand-Drawn Images. *arXiv preprint arXiv:1707.09627* (2017).

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401* (2014).

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

Chiori Hori, Takaaki Hori, Teng-Yok Lee, Ziming Zhang, Bret Harsham, John R Hershey, Tim K Marks, and Kazuhiko Sumi. 2017. Attention-based multimodal fusion for video description. In *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 4203–4212.

Armand Joulin and Tomas Mikolov. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems (NIPS)*. 190–198.

Łukasz Kaiser and Ilya Sutskever. 2015. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228* (2015).

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. 2016. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. In *IEEE Conference on Computational Intelligence and Games*. IEEE, Santorini, Greece, 341–348. http://arxiv.org/abs/1605.02097 The best paper award.

Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. 2017. Dense-Captioning Events in Videos.. In *Proceedings of the IEEE international conference on computer vision (ICCV)*. 706–715.

Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. 2015. Neural random-access machines. *arXiv preprint arXiv:1511.06392* (2015).

Yingwei Pan, Ting Yao, Houqiang Li, and Tao Mei. 2017. Video Captioning With Transferred Semantic Attributes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 6504–6512.

Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. 2017. Neuro-symbolic program synthesis. *International Conference on Learning Representations (ICLR)* (2017).

Richard E Pattis. 1981. *Karel the robot: a gentle introduction to the art of programming*. John Wiley & Sons, Inc.

Michael S Ryoo. 2011. Human activity prediction: Early recognition of ongoing activities from streaming videos. In *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 1036–1043.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Tim Lillicrap. 2017. A simple neural network module for relational reasoning. In *Advances in neural information processing systems (NIPS)*. 4967–4976.

Zhiqiang Shen, Jianguo Li, Zhou Su, Minjun Li, Yurong Chen, Yu-Gang Jiang, and Xiangyang Xue. 2017. Weakly supervised dense video captioning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 2.

Karen Simonyan and Andrew Zisserman. 2014. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems (NIPS)*. 568–576.

Yale Song, Miriam Redi, Jordi Vallmitjana, and Alejandro Jaimes. 2016. To click or not to click: Automatic selection of beautiful thumbnails from videos. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 659–668.

Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim. 2018. Neural Program Synthesis from Diverse Demonstration Videos. In *International Conference on Machine Learning (ICML)*. 4797–4806.

Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. 2015. Sequence to sequence-video to text. In *Proceedings of the IEEE international conference on computer vision (ICCV)*. 4534–4542.

Yingce Xia, Fei Tian, Lijun Wu, Jianxin Lin, Tao Qin, Nenghai Yu, and Tie-Yan Liu. 2017. Deliberation Networks: Sequence Generation Beyond One-Pass Decoding. In *Advances in Neural Information Processing Systems (NIPS)*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 1784–1794. http://papers.nips.cc/paper/6775-deliberation-networks-sequence-generation-beyond-one-pass-decoding.pdf

Jun Xu, Tao Mei, Ting Yao, and Yong Rui. 2016. Msr-vtt: A large video description dataset for bridging video and language. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5288–5296.

Haonan Yu, Jiang Wang, Zhiheng Huang, Yi Yang, and Wei Xu. 2016. Video paragraph captioning using hierarchical recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 4584–4593.

Yitian Yuan, Tao Mei, and Wenwu Zhu. 2018. To Find Where You Talk: Temporal Sentence Localization in Video with Attention Based Location Regression. *arXiv preprint arXiv:1804.07014* (2018).