Program Generation from Diverse Video Demonstrations

Anthony Manchin[1 2 3]
anthony.manchin@adelaide.edu.au
Jamie Sherrah[1 3]
jamie.sherrah@adelaide.edu.au
Qi Wu[1 3]
qi.wu01@adelaide.edu.au
Anton van den Hengel[1 3]
anton.vandenhengel@adelaide.edu.au

[1] The University of Adelaide, Australia
[2] Apollo Labs, Adelaide, Australia
[3] The Australian Institute for Machine Learning, Adelaide

Abstract

The ability to use inductive reasoning to extract general rules from multiple observations is a vital indicator of intelligence. As humans, we use this ability to not only interpret the world around us, but also to predict the outcomes of the various interactions we experience. Generalising over multiple observations is a task that has historically presented difficulties for machines to grasp, especially when requiring computer vision. In this paper, we propose a model that can extract general rules from video demonstrations by simultaneously performing summarisation and translation. Our approach differs from prior works by framing the problem as a multi-sequence-to-sequence task, wherein summarisation is learnt by the model. This allows our model to utilise edge cases that would otherwise be suppressed or discarded by traditional summarisation techniques. Additionally, we show that our approach can handle noisy specifications without the need for additional filtering methods. We evaluate our model by synthesising programs from video demonstrations in the Vizdoom environment achieving state-of-the-art results with a relative increase of 11.75% program accuracy on prior works.

1 Introduction

Inductive reasoning involves using logic to extract general rules from multiple observations and is a skill that is widely viewed as an indicator of intelligence. Humans (and many species of animals) are known to possess the ability to observe demonstrations and subsequently use inductive reasoning to acquire new knowledge and skills without requiring explicit instruction. An example of this behaviour would be children learning to play video games, wherein one is able to observe another player, and learn the rules of the game without having to play the game themselves.

Additionally, children are also able to abstract and generalise information they have induced from one video game and apply that same logical rule set to a completely different

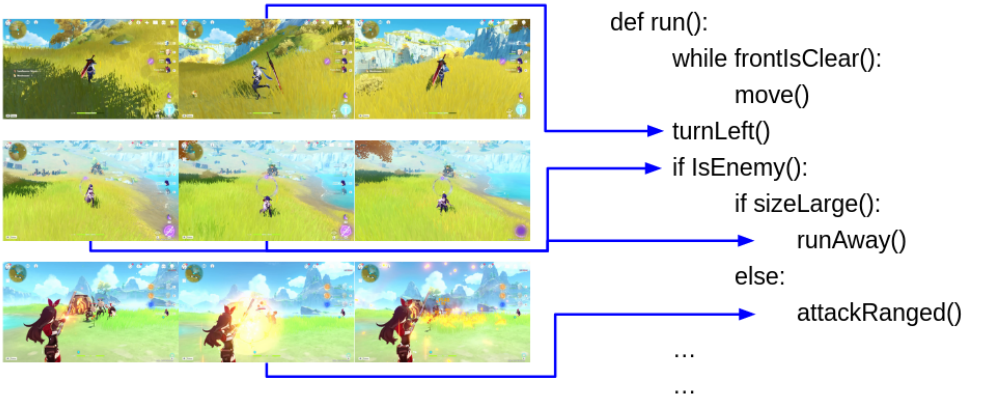

Figure 1: An illustration of the task of visual program synthesis on the game 'Genshin Impact'. Humans can infer a general rule set from simply observing examples of game play. The task of visual program synthesis involves training a machine that can synthesis a program that correctly captures a rule set simply from watching visual demonstrations of agent.

virtual domain. An example of this may be that someone observes that a red cross symbol indicates a health related bonus in one games, and subsequently uses that information to infer that similar symbols in a different game are also related to player health. Implying general rules from complex observations however has proven to be a difficult task for machines. In-fact until recently, very little work had been published on extracting general rules from a diverse range of visual observations. Advances in both computer vision and machine learning techniques however are changing this.

We seek to create a model that can generate executable code by inferring the specifications from multiple visual demonstrations. The goal of creating a model capable of generating executable code has long been a dream of artificial intelligence researchers [12, 27]. However, until recently research has primarily focused on the generation of code, *given* the desired specifications [2, 13]. However, the problem becomes much more difficult when the model is also required to infer the specifications for itself. This task, originally proposed by [24], presents a unique challenge as it requires a model to accurately detect the relevant semantics of a demonstration, understand the relationships between demonstrations, define a set of specifications that captures this information, and finally generate a program that satisfies these specifications.

Previous approaches [10, 24] have considered framing this a sequence-to-sequence task and have utilised the once popular recurrent neural network architecture of long short term memory units as the backbone of their models. However, these approaches were restricted by the limitations of these recurrent based models to handle long sequences. This limiting factor resulted in both [10, 24] using summarised latent space representations, which undoubtedly limited their models ability to completely capture the entire specification constraints. Meanwhile [5] sought to use a rule-based solver to generate the desired code, which receives its specifications from a neural model. As rule-based solvers are very susceptible to noise, [5] proposed using a dynamic filtering method to de-noise specifications.

To address these limitations, we propose 'video-to-text transfer transformer' (VT4). We leverage the ability of attention based transformer networks to handle long and disjointed sequences, removing the need to summarise features. Specifically, we formulate a 'visual language' which encapsulates the relevant semantic information from each demonstration. We then simultaneously feed all the demonstrations (represented in a visual language) into our encoder-decoder transformer network which learns to summarise and translate the demonstrations into an executable program. Additionally we show that our approach is able to handle noisy specifications without the need for additional filtering methods. We evaluate our model in a partially observable virtual environment (Vizdoom) [16] and demonstrate that our approach is able achieve state-of-the-art results by out-performing previous summarisation and rule-based approaches. We observe a relative increase of 15.45% and 11.75% over summarisation and rule-based approaches respectively.

Contributions

- We present video-to-text transfer transformer for the task of executable program generation from video demonstrations. Addressing the limitations of previous summarisation and rule-based approaches, VT4 can generate programs from a long, disjointed set of demonstrations without the need for a summarised representation of the average demonstration.

- We evaluate the effects of noisy specifications on program accuracy and show that our model is robust to significant levels of erroneous detections. State-of-the-art results were achieved with a 10% error rate for perception primitives. Significantly out-performing previous rule-based models which strongly rely on dynamic filter for noise reduction.

- We evaluate our model's ability to generate programs from partially observable, visually complex demonstrations. We achieve increases of 9% and 11.75% for exact and aliased program accuracy relative to previous state-of-the-art. This translates to absolute increases of 5.3% and 7.7% respectively and represents the largest single increase in performance on this task to date.

2 Literature Review

The task of video understanding [18, 29] can be viewed as a subset task of program generation from video demonstrations (PGfVD). As with PGfVD, the understanding of videos requires the ability to extract and understand the correlations between events and features. This is often achieved through models that can perform tasks such as action and perception recognition [15]. However, unlike PGfVD, video understanding aims to describe *what* has been observed in a single demonstration, not *why* something has happened. For the most part, this is a straightforward translation task that can benefit from a large amount of acceptable ambiguity [8]. This is largely because for the task of video translation, multiple captions may be appropriate and considered semantically correct. However, for the task of PGfVD, the video demonstration may only display a single component of the overall rule set that one is trying to learn. Additionally, the task of PGfVD has much lower levels of ambiguity, as slight changes to a program can result in greatly different outcomes.

2.1 Program Induction

The task of extracting algorithmic representations from observations is known as program induction. Various works have contributed to this field with a diverse set of approaches aimed at solving this problem. [11, 14, 17] use memory based approaches such as Turing machines while [19] adopt end-to-end networks to solve and explain algebraic word problems. However, in contrast to these approaches, we aim to generate a fully defined executable program in a domain specific language.

2.2 Program Synthesis

The aim of program synthesis is to generate a program that captures the underlying logic of given examples. Typically, this task has restricted the programs to simple domain specific languages and has involved producing an abstract syntax tree. Examples of this work include [20], who proposed using Recursive-Reverse-Recursive neural networks (R3NN) for string transformation. Other work has paired neural models with search algorithms and rule based solvers [3, 5]. Additionally reinforcement learning has also been investigated as a possible way to solve the task of program synthesis [9, 22].

However, most of the work in this field does not consider the task of synthesising programs from visual observations. [24] identified this and proposed the task of generating a program from observing a diverse range of visual demonstrations. To achieve this goal [24] proposed using a sequence-to-sequence LSTM model. Their model consisted of convolutional neural network which fed an LSTM network encoded video frames. [24] introduced a combination of average pooling and a relational network to summarise the encoded demonstrations, which were subsequently passed to a LSTM decoder. [10] aimed to improve the computational efficiency of [24] approach by introducing a deviation-pooling summariser to replace the relation network. Additionally, [10] proposed using multiple decoding layers to refine the accuracy of the generated program which ultimately resulted in a slight improvement in performance.

[5] took a different approach to this problem, proposing a hybrid model which combined a neural network to extract specifications and a rule-based solver to generate the program. In particular, [5] proposed using a convolutional neural network to encode the video frames, and two different decoder layers to predict the perceptions and actions observed in the videos. [5] correctly identifies the sensitivity of rule-based solvers to input specification noise [6], and proposed a dynamic filtering method to ignore certain demonstrations based on the confidence level of the neural networks predictions. The inclusion of a dynamic filter allowed [5] to surpass the performance of previous LSTM based program generators.

3 Method

This section first presents a formal definition for the task of generating programs from video demonstrations, as originally proposed by [24], before describing the proposed VT4 model, and our contributions in detail.

3.1 Program Generation

The goal of generating a program given k video demonstration can be considered a multi-sequence-to-sequence task. A domain specific language (DSL) is used to define a program

which consists of perception primitives, action primitives and control flow statements. Action and perception primitives define the way an agent can interact with and perceive the environment respectively. The control flow statements of the DSL language include while loops, repeat and if/else statements, and simple logic operations.

3.1.1 Program

A program $\pi_\theta(s_t) = a_t$ is defined as a deterministic function, which given an input of state $s \in \mathcal{S}$ at time t, returns an action $a \in \mathcal{A}$. For this task we limit the parameters of the program to a vectorised DSL, which we donate as $\theta \in \Theta$. The parameters are what would typically be referred to as the 'code' of the program.

3.1.2 Demonstrations

A demonstration is defined as a sequence of state-action pairs sampled from $\pi(\theta)$

$$\tau = ((s_1, a_1), (s_2, a_2), ..., (s_T, a_T)) \tag{1}$$

However, there is no guarantee that any particular demonstration generated by an agent following a program $\pi(\theta)$ will provide examples of all control flow statements present in $\pi(\theta)$. Therefore it is necessary to observe a set of demonstrations $\mathcal{D} = (\tau_1, \tau_2, ..., \tau_k)$ where $\mathcal{D} \subset D$ that contains transition examples of all control flow statements in $\pi(\theta)$.

3.1.3 Actions and Perceptions

Given the conditional structure of the DSL used to define the parameters θ for program π, we employ the use of perception primitives to simplify the high dimensionality of states $s_{1 \to T}$. Each perception primitive μ is given as a Boolean value, with q possible perception primitives. This allows for a high level representation of the state to be given by the vector $s = (\mu_1, \mu_2, ..., \mu_q)$.

For an agent interacting in discrete time steps with a deterministic environment of previously defined states $s \in \mathcal{S}$, we can define a finite set of possible actions $\mathcal{A}$. This allows us to model the deterministic transition between states as $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. In our case we only have one possible action per transition, and as such we can represent an action primitive as a one hot tensor of length m, where m is equal to the total number of possible actions.

3.1.4 Visual Language

Having defined both perception and action primitives, we can use these to define a visual language of semantic tokens $\psi \in \Psi$ with a deterministic function $F(x, y)$ such that,

$$\psi_t = F(s_t, a_t) \tag{2}$$

This approach of tokenizing high level semantic information for ease of use with transformer models has been shown to be very effective by [23]. With this we can also substitute our transition tokens into equation 1 giving,

$$\tau = (\psi_1, \psi_2, ... \psi_T) \tag{3}$$

3.2 VT4 Model

Our VT4 model can be separated into two main sections; i) the semantic encoder and ii) the program generator network. We approach this problem of generating an executable program from video demonstrations as a combined translation and summarisation task. In contrast to previous approaches which simplify the problem to a straight sequence-to-sequence task, (by creating a summarised expression of the multiple demonstrations) our approach frames the problem as the multi-sequence-to-sequence task that it is. This eliminates the inherent probability of information loss associated with summarising multiple diverse demonstrations.

3.2.1 Semantic Encoder

The Semantic Encoder itself can be separated into two parts: a neural module and a tokenizer. The neural module is a multi-layer convolutional neural network (CNN) which learns to detect the perception primitives present in each frame, and the actions taken between frames. The action prediction network consists of a five-layer convolutional neural network with two fully connected layers, and is trained from scratch. The perception prediction network utilises a pre-trained Efficientnet model [25].

Given a set of video demonstrations $\mathcal{V} = \{v_i\}_{i=1}^{k}$, we desire the corresponding perceptions p and actions a sequences for each demonstration. This is a straight forward problem which is modelled as;

$$p_{i,j} = MLP(CNN(v_{i,j})) \tag{4}$$

$$a_{i,j} = MLP(CNN(v_{i,j}), CNN(v_{i,j+1})) \tag{5}$$

Where i and j refer to the i^{th} demonstration and j^{th} frame. While it is possible to predict all the observable perceptions from a single frame $v_{i,j}$ with purely spatial information, this is not the case for predicting actions. To predict the action taken at any time t temporal information in the form of a minimum of two frames is required. Thus, we concatenated sequential frames together as shown in equation 5 for action prediction.

Having now obtained the predicted actions $a_{i,j}$ and perceptions $p_{i,j}$ for each frame of every video, we are able to use these to create semantic tokens which completely encapsulates all the required information from each frame. We achieve this by passing our predictions through the second part of the semantic encoder which we refer to as a tokenizer. The tokenizer itself is a deterministic function that takes as input the predicted action and perceptions of each frame individually.

$$\psi_{i,j} = F(p_{i,j}, a_{i,j}) \tag{6}$$

As described in section 3.1, our perception prediction is a multiclass prediction problem given as $p_{i,j} = (\mu_1, \mu_2, ..., \mu_q)$, while our action prediction returns a single class prediction. Our tokenizer first concatenates our predictions into a single tensor pa, before calculating the finite sum of a basic power series.

$$\psi_{i,j} = \sum_{n=0}^{n} pa_n \times 2^n \tag{7}$$

Our demonstrations are now encoded into a sequence of semantic tokens which encapsulates all the semantic information derived from the original video frames.

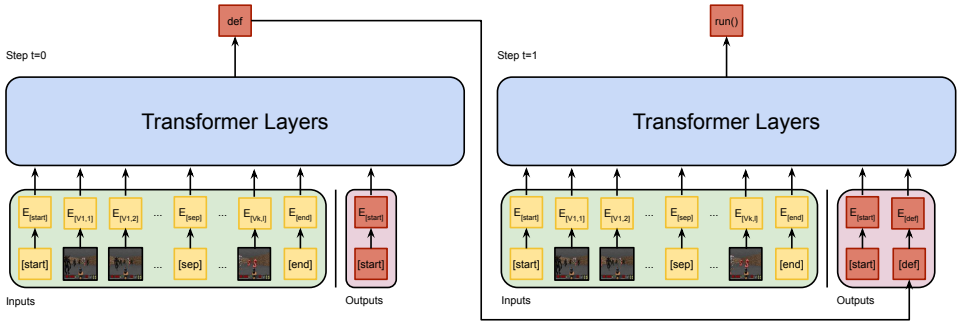

Figure 2: The Program Generator Module. The program generator consists of an encoder-decoder transformer network which receives as input the visual language tokens from the semantic encoder. The network iteratively decodes the program in a sequential fashion.

We pass every tokenised demonstrations into our program generator network as a set of disjoint sequences. This approach of creating 'visual words'[23] allows us to take inspiration from the field of natural language processing and concatenate our demonstrations together separating the different demonstrations with special tokens. $V = (< start >, \psi_{1,1}, \psi_{1,2}..., \psi_{1,l} < sep >, ..., \psi_{k,l}, < end >)$

3.2.2 Program Generator Network

Due to the wide success of Transformer based architectures [26] in various settings such as machine translation [1, 28], natural language processing [7, 21], and even more recently image classification [9], we hypothesis that they should also be highly effective for the task of program generation. As such our program generator is an encoder-decoder transformer network. By utilising the insights of [23] and converting our inputs into a visual language, we can leverage pre-trained language models. We choose to leverage a pre-trained 'Text-to-Text Transfer Transformer' (T5) network as proposed by [21].

The T5 model is particularly well suited to our task as it specifically casts all tasks as a text-to-text task. Due to this we can leverage pre-trained features from upstream natural language tasks such as summarisation and translation. The T5 implementation closely follows the originally proposed architecture of [26]. The encoder consists of a stack of 'blocks', wherein each block is comprised of a self-attention, layer normalisation, and a feed-forward network. The decoder has a similar structure, except that it includes an additional standard attention mechanism. This standard attention mechanism is applied after the self-attention layer and attends to the output from the encoder. Figure 2 gives an overview of generative sequence of the transformer network, while for full details we refer the reader to [21] and [26] respectively.

4 Experiments

In this section we present the experimental evaluation of our model. We include an overview of the dataset and metrics used for this evaluation. We then discuss the results of our exper-

iments along with providing a noise-ablation study to evaluate our model's ability to handle noisy inputs.

4.1 Dataset and Metrics

4.1.1 Dataset

We evaluate our model with the Vizdoom Program Dataset[24] which has the following structure. For every program label there exists multiple video demonstrations of an agent following said program in the deterministic virtual environment known as Vizdoom[16]. For every demonstration (of length T), there exists action and perception labels of lengths $T-1$ and T respectively. We conform with the previously established convention set by [24] and utilise a total of twenty-five demonstrations per program during testing and training. We also adhere to the assumption that action and perception labels are only available during training, and that at test time we only have access to the video demonstrations of the agent. The dataset contains 80,000 programs for training, and 8,000 programs for testing.

4.1.2 Aliased and Exact Accuracy

As the problem of verifying that two programs are in fact equal is an intractable problem, we evaluate the accuracy of our model by comparing the synthesised parameters $\hat{\theta}$ with the instantiated parameters of the ground truth program θ^*. We consider a program to be an exact match if, and only if, the synthesised parameters $\hat{\theta}$ is an exact match to that of the instantiated ground truth parameters θ^*. This is formally expressed as,

$$Acc_{exact} = \frac{1}{N}\sum_{n=1}^{N} 1_{exact}(\hat{\theta}, \theta^*) \tag{8}$$

While the exact accuracy is a simplistic measure of the performance of our model, it does not account for the ambiguity of the program space. As identified by [24], it is possible to exploit the simplistic syntax of our DSL and enumerate multiple variations of the code following a set of defined rules. Examples of this include decomposing control flow statements such as if/else statements, and unfolding repeat statements. With this we can formally express the aliased accuracy as

$$Acc_{alias} = \frac{1}{N}\sum_{n=1}^{N} 1_{alias}(\hat{\theta}, \theta^*) \tag{9}$$

4.2 Overall Performance

We report the results of our evaluation on the Vizdoom benchmark in table 1. We strictly adhere to the experimental settings originally proposed by [24] to provide a fair comparison. Our VT4 model significantly improves on the exact and aliased program accuracy of the prior state-of-the-art with relative 9% and 11.75% increases and 5.3% and 7.7% absolute increases respectively. These results provide clear evidence of the capabilities of our model to simultaneously perform summarisation and translation of disjointed sequences. These results also provide empirical evidence to support the use of visual languages to describe the semantics of complex scenes for tasks requiring translation. Additionally, we observe

Model	Exact	Aliased
demo2program[24]	53.2	62.5
watch-reason-code[10]	55.8	63.4
PLANS (dynamic) [5]	58.8 ±0.6	65.5 ±0.6
VT4 (ours)	**64.1**	**73.2**

Table 1: An exact comparison of our results compared to the results of previously published works.

Perception Noise	Exact	Alias
0%	66.0	75.1
10%	64.1	73.2
20%	62.7	71.6

Table 2: Exact and Alias program accuracy's for varying levels of perception noise.

the expected result of higher accuracy for aliased programs compared to exact programs as consistently observed across all prior works.

4.3 Noise Ablation Study

While our model is clearly capable of inductive reasoning over multiple demonstrations, we consider the implication of noise with respect to its ability to accurately generate programs. Previous approaches that utilise rule-based solvers [5] have been highly sensitive to noise. In-fact, even with high levels of perception accuracy the PLANS model required dynamic filtering. This indicates that even the slightest amount noise in the input causes the PLANS model to underperform.

To test our model's ability to deal with noise in its input we devise a noise ablation study. By separately predicting the actions and perceptions with two distinct networks we can vary the accuracy of the perception predictions independently of the actions. Our semantic encoder in this setup has independent action and perception encoders. Our action encoder easily obtains an accuracy of 98% with a simple five layer convolutional neural network (as described in section 3.2.1). We train a perception encoder with the same architecture which achieves an accuracy of 79.9%. We then utilise a pre-trained object detection network (efficientnet [25]) to improve this result. This time our prediction encoder achieves an accuracy of 90.2%. Our approach to utilise predicted perception primitives and actions to generate a visual language allows us to evaluate a 'perfect case' scenario. As we have access to ground-truth labels for these perception primitives and actions, we can use these directly to generate our visual language tokens. Doing so artificially creates a scenario in which our model has effectively no input noise.

Our results from this study are presented in table 2. These results clearly show our models capacity to not only reason over multiple demonstrations, but to handle contradictory or noisy signals. In contrast to rule-based solvers, which as shown by [5] are strongly reliant on pre-defined filtering heuristics, our VT4 model is able to learn its own heuristics. These results also show that our model was able to exceed the previous state-of-the-art while enduring a 20% error rate in perception predictions.

5 Conclusions

The task of synthesising a program from multiple visual demonstrations involves solving many different tasks. These include learning spatial-temporal relationship from visually complex inputs and successfully translating these into a logical sequence in a different domain. Previous attempts at solving this problem have relied upon summarised representations of the spatial-temporal relationships and have been highly sensitive to input noise. In this paper we propose a video-to-text transfer transformer network that is able to perform multi-sequence-to-sequence translation without requiring summarised spatial-temporal embeddings. Our method is also highly robust to input noise, which is a problem that caused significant challenges for previous methods. On top of this, we achieve the largest increase in performance to date on this task.

References

[1] Karim Ahmed, Nitish Shirish Keskar, and Richard Socher. Weighted transformer network for machine translation. *CoRR*, abs/1711.02132, 2017. URL http://arxiv.org/abs/1711.02132.

[2] R. Alur, R. Bodik, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design*, pages 1–8, 2013. doi: 10.1109/FMCAD.2013.6679385.

[3] Matej Balog, Alexander Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. 11 2016.

[4] Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. *CoRR*, abs/1805.04276, 2018. URL http://arxiv.org/abs/1805.04276.

[5] Raphaël Dang-Nhu. Plans: Robust program learning from neurally inferred specifications. *ArXiv*, abs/2006.03312, 2020.

[6] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 990–998. JMLR.org, 2017.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

[8] Jiarong Dong, Ke Gao, Xiaokai Chen, Junbo Guo, Juan Cao, and Yongdong Zhang. Not all words are equal: Video-specific information loss for video captioning. *CoRR*, abs/1901.00097, 2019. URL http://arxiv.org/abs/1901.00097.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[10] Xuguang Duan, Qi Wu, Chuang Gan, Yiwei Zhang, Wenbing Huang, Anton van den Hengel, and Wenwu Zhu. Watch, reason and code: Learning to represent videos using program. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 1543–1551, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368896. doi: 10.1145/3343031.3351094. URL https://doi.org/10.1145/3343031.3351094.

[11] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

[12] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *SIGPLAN Not.*, 46(1):317–330, January 2011. ISSN 0362-1340. doi: 10.1145/1925844.1926423. URL https://doi.org/10.1145/1925844.1926423.

[13] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. Oracle-guided component-based program synthesis. ICSE '10, page 215–224, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587196. doi: 10.1145/1806799.1806833. URL https://doi.org/10.1145/1806799.1806833.

[14] Lukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.08228.

[15] Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017. URL http://arxiv.org/abs/1705.06950.

[16] Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski. Vizdoom: A doom-based AI research platform for visual reinforcement learning. *CoRR*, abs/1605.02097, 2016. URL http://arxiv.org/abs/1605.02097.

[17] Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random-access machines. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06392.

[18] Ji Lin, Chuang Gan, and Song Han. Tsm: Temporal shift module for efficient video understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[19] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *CoRR*, abs/1705.04146, 2017. URL http://arxiv.org/abs/1705.04146.

[20] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. *CoRR*, abs/1611.01855, 2016. URL http://arxiv.org/abs/1611.01855.

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019. URL http://arxiv.org/abs/1910.10683.

[22] Riley Simmons-Edler, Anders Miltner, and H. Sebastian Seung. Program synthesis through reinforcement learning guided tree search. *CoRR*, abs/1806.02932, 2018. URL http://arxiv.org/abs/1806.02932.

[23] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid. Videobert: A joint model for video and language representation learning. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7463–7472, 2019. doi: 10.1109/ICCV.2019.00756.

[24] Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim. Neural program synthesis from diverse demonstration videos. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4790–4799. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/sun18a.html.

[25] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/tan19a.html.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[27] Richard J. Waldinger and Richard C. T. Lee. Prow: A step toward automatic program writing. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, IJCAI'69, page 241–252, San Francisco, CA, USA, 1969. Morgan Kaufmann Publishers Inc.

[28] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. Learning deep transformer models for machine translation. *CoRR*, abs/1906.01787, 2019. URL http://arxiv.org/abs/1906.01787.

[29] Chao-Yuan Wu, Christoph Feichtenhofer, Haoqi Fan, Kaiming He, Philipp Krahenbuhl, and Ross Girshick. Long-term feature banks for detailed video understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.